INTERNATIONAL DOCTORAL SCHOOL OF THE USC

Francisco Javier
Cardama Santiago

PhD Thesis

Distributed Quantum Computing integrated into High-Performance Computing environments

Santiago de Compostela, 2026

Doctoral Programme in Information Technology Research

ESCOLA DE DOUTORAMENTO
INTERNACIONAL DA USC

Ph.D. Thesis

Distributed Quantum Computing integrated into High-Performance Computing environments

Francisco Javier Cardama Santiago

Supervisor: Prof. Dr. Tomás Fernández Pena.

Tutor: Prof. Dr. Tomás Fernández Pena.

DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY RESEARCH

SANTIAGO DE COMPOSTELA
2026

Declaration of conflicts of interest

The doctoral candidate, Francisco Javier Cardama Santiago, declares no conflicts of interest regarding the doctoral thesis presented in this document, entitled: "*Distributed Quantum Computing integrated into High-Performance Computing environments*".

Santiago de Compostela, 24 de marzo de 2026

"God does not play dice with the universe."
— Albert Einstein, *Letter to physicist Max Born.*

"It is those who possess wisdom who are the greatest fools.
History has shown us this."
— Rintaro Okabe, *Steins;Gate.*

Para miña nai,
unha muller á que non lle deixaron cumprir os seus soños,
pero fixo o imposible por verme cumprir os meus.

Á memoria de todas as galegas e galegos anónimos
que loitaron a reo por un mañá que non chegaron a ver,
sementando as liberdades que hoxe recollemos.

Agradecementos

Sen lugar a dúbidas, todos os que temos redactado unha tese doutoral sabemos de primeira man a complexidade de escribir cada liña de texto: moitas preguntas ás cales temos que fabricar as nosas propias respostas. Parece algo de suma complexidade... ata que comezas a escribir os agradecementos. Plasmar nun papel todas esas palabras capaces de facer aflorar sentimentos nos futuros lectores... Que poderosas son as palabras.

"Las palabras son, en mi no tan humilde opinión, nuestra más inagotable fuente de magia, capaces de infligir daño y de remediarlo."

— Albus Dumbledore, *"Harry Potter y las Reliquias de la Muerte"*.

E si, resulta difícil buscar as palabras para agradecer dende o corazón porque traballar coas matemáticas, aínda que abstracto, é exacto, doado de demostrar e de convencer. Neste sector científico vivimos nunha constante voráxine por buscarlle unha xustificación a todo o que plasmamos nun papel. Tal vez por ese sinxelo motivo pode que nos sexa tan complexo guiarnos por un sentimento que non damos comprendido de todo.

"El amor es lo único que podemos percibir que trasciende las dimensiones del tiempo y el espacio. Quizás deberíamos confiar en ello, aunque no podamos comprenderlo."

— Dra. Amelia Brand, *"Interstellar"*.

Non podemos esquecer, de cando en vez, pararnos a reflexionar sobre os motivos polos que nos dedicamos a facer ciencia, quen nos trouxo ata aquí e os "ombros de xigantes" sobre os que camiñamos. Síntome exquisitamente afortunado de ter tanto que escribir nesta sección. **Ter tantísimo que agradecer faime unha persoa moi privilexiada.**

Gustaríame comezar agradecendo poder redactar parte deste documento na miña lingua nai, o galego. Pode parecer algo moi banal ou mesmo sinxelo, especialmente

para aqueles que temos nacido en democracia. Se hoxe teño a posibilidade de redactar isto na miña lingua é porque houbo un pobo que non deixou de usala, incluso cando iso che podía custar a propia vida. Infinitas grazas a esa xente que loitou para que hoxe puidésemos vivir a nosa cultura e lingua en liberdade.

"A liberdade é a única reserva coa que contan os pobos para crearen o seu futuro."

— Daniel Castelao, *"Cousas"*.

Eu estudei dende ben pequeniño nun cole público, nun instituto público, cursei o bacharelato na pública, gradueime en Enxeñaría Informática na Universidade pública de Santiago de Compostela, finalicei dous mestrados na mesma institución pública e a día de hoxe redacto esta tese doutoral na mesma universidade pública onde comecei, nun centro de investigación público e con recursos públicos. Veño dunha familia humilde, si, desas que non poderían hipotecarse para que os seus fillos estudasen unha carreira universitaria. Eu non podería ter sido enxeñeiro, e moito menos ter finalizado unha tese doutoral, se non fose polo compromiso solidario de contar cunha educación pública, gratuíta e de calidade que garanta o acceso igualitario á formación de toda a sociedade.

"Somos analfabetos que criamos fillos enxeñeiros. Agora tócavos non criar fillos analfabetos."

— Miña nai, *nalgunha charla de terraza, dunha noite de cálido verán, tomando un licor café.*

Sinto un profundo sentimento de gratitude cara a todas as persoas que loitastes para que a nosa xeración tivese acceso a un sistema solidario, onde todos teñamos as mesmas oportunidades de medrar profesionalmente, garantindo un sistema público sólido e robusto. Moitos deles sen teren ningún tipo de formación académica, o cal demostra que os títulos carecen de sentimento e conciencia de clase. Tamén sinto unha forte responsabilidade; non todo está feito, queda moito camiño por percorrer e, sobre todo, debemos ter coidado: todo isto escorréganos entre os dedos. Non podemos dar un só paso atrás na defensa dos nosos dereitos, non só polo futuro, senón tamén polo respecto ás loitas do pasado. Que non nos caiba ningunha dúbida de que poderán vencer, pero nunca convencer, carecen de todas as ferramentas para facelo.

"Venceréis, pero no convenceréis. Venceréis porque tenéis sobrada fuerza bruta, pero no convenceréis porque convencer significa persuadir."

— Miguel de Unamuno, *resposta ao militar José Millán-Astray o 12 de outubro do 1936.*

Ao longo destes 23 anos de formación continuada, como calquera persoa, atopeime con incontables profesoras e profesores de todo tipo: algúns deles tiveron impactado de forma moi directa na persoa na que me convertería hoxe en día, ben impactando dunha forma positiva ou ben dunha forma negativa.

Xa dende ben pequeno me gustou coñecer o mundo que me rodeaba, especialmente no mundo dos ordenadores, curioso por natureza. O resultado final foi rematar a miña carreira de enxeñaría informática con premio extraordinario, algo que a todo o mundo lle gusta ler e escribir—ata incluso fardar—, pero non foi algo sinxelo, durante gran parte deste longo percorrido eu non fun máis que un mero estudante que non destacaba en nada, pasaba desapercibido e, incluso, se me chegou a considerar un fracaso académico de libro con nulas opcións de acabar a educación secundaria obrigatoria. Durante os meus primeiros anos de adolescencia tiven que soportar profesores que tiñan a necesidade de sacarme da cabeza a idea de facer algo relacionado coa informática, cortándome as ás e conseguindo que xerara un profundo sentimento de decepción e de pasotismo polo mundo académico. De súpeto a miña curiosidade por coñecer o mundo, as matemáticas, a ciencia, a informática desvanecéronse en cuestión de meses, instaurando nun rapaz de trece anos a sensación de ter fallado e fracasado na curta vida que levaba vivida.

Non foi ata que en terceiro curso da educación secundaria obrigatoria, por aló no 2014, houbo un profesor de matemáticas que foi capaz de ver algo en min que eu xamais vira, non só consigueu volver espertar ese interés, traballou a reo para ensinarme todo o que xa deixara atrás cursos anteriores. Recordo como si fora onte as clases individuais ensinándome ecuacións de primeiro grao mentres outros celebraban o Nadal, mentres el xa lle explicaba ao resto da clase as de segundo grao. Escasos meses despois diso, tiñasme facendo exames de cursos superiores sinxelamente por diversión. En cuestión dun ano pasara de sacar ceros e uns en matemáticas a noves e deces. Gustaríame agradecer a Rafael Urrutia todo o esforzo que me dedicou de forma tan altruistra, para conseguir volver a espertar en min esa paixon polas matemáticas, as cales foron, a partir dese día, o meu pasatempos favorito, o meu "as" na manga, a miña máis forte ferramenta e sen dúbida a que mellor se me da. Quero corrixir as miñas palabras de "profesor" a verdadeiro "mestre", porque é moi sinxelo centrarte nos que sacan deces, é unha aposta segura, o que é complicado de corazón e conseguir apostar por unha persoa—de catorce aniños— que non é "excelente", un neno que tiña abandonado os seus soños, e conseguir sacar o mellor del. Grazas ás túas aprendizaxes aquel curso, o resto foi pan comido, espero que sintas un pedaciño de este traballo como propiamente teu.

A continuación chegaron docentes que continuaron o traballo, empurrándome sempre a volver a ser curioso e gozar da aprendizaxe, graciñas a Libertad, Paula, Moncho, Natalia, Elena e tamén en especial a Joserra por facerme namorar da historia, teño por seguro as palabras que me terías dito ao ler esta tese.

"Se conseguimos que unha xeración, unha soa xeración, creza libre, xa ninguén lles poderá arrancar nunca a liberdade, ninguén lles poderá roubar ese tesouro."

— O mestre Don Gregorio, *"A lingua das bolboretas"*.

Da carreira, pouco teño que contar, teño a gran sorte de poder contar cos dedos dunha única man aos docentes que non aportaron nada á miña formación. O Grao en Enxeñaría Informática da Universidade de Santiago de Compostela está plagado de excelentes profesionais que compaxinan a súa tarefa docente cunha investigación de altísima calidade e prestixio a nivel internacional. Teño aprendido moito con vosco, sígoo facendo e teño pensado seguir aprendendo no futuro nesta área de Arquitectura e Tecnoloxía de Computadores. Moitas grazas pola vosa dedicación tan altruísta a facer desta esquina de Europa un sitio excelente para estudar e investigar nunha universidade pública.

Mentres estudas o grao ou un mestrado, tes unha cantidade lixeiramente cercana a infinito de docentes arredor. Pero cando te mergullas nunha tese, ese universo encóllese de golpe. Nese momento decátaste de que un só nome o cambia todo. No meu caso, ese nome é o de Tomás Fernández Pena.

Quero agradecerche, Tomás, con total sinceridade e honestidade, a oportunidade e a confianza que me deches para iniciar este traballo. Fixeches do doutoramento un camiño cheo de ledicia, onde equivocarse era tan valioso como acertar, onde nunca faltaron as boas palabras e onde sempre me tratacheches coma un igual.

Grazas polas horas infinitas que lle dedicaches, polas recomendacións certeiras, por escoitarme sempre, e, sobre todo, por ser honesto e humilde. Síntome privilexiado por poder pechar esta etapa e levala comigo para sempre como algo que funcionou, que foi divertido e da que non cambiaría absolutamente nada. Ti es o culpable directo diso.

Grazas por facelo todo tan doado, por chegar cada día cunha idea nova e por pensar sempre na miña formación—non podo dicir que foron poucos os congresos e cursos aos que me mandaches nestes dous anos e medio—. Esta universidade precisa moita máis xente coma ti, Tomás. Se algún día chego a dirixir unha tese sequera a metade de ben do que ti o fixeches comigo, podereime dar por máis que satisfeito.

Quero agradecer tamén ao grupo de computación cuántica do CESGA, por ser xente tan agradable como intelixente, foi moi sinxelo integrarse neste grupito, especialmente nos cafés... Grazas especiais a Jorge e Marta, por todos os consellos, a axuda e as risas intercambiadas, esta tese tería sido moi distinta de non haber estado vós por aquí. Espero que sigamos sendo compis de congresos (e demais viaxes) durante moitos anos máis.

Tamén quero agradecer a todos os meus compañeiros do CiTIUS, dende que entrei no 2021, pasou moita xente por aquí, de todos vós aprendín unha gran cantidade de cousas. Sempre é un pracer ter xentiña coa que tomarse un bo café das 11 e intercambiar experiencias vividas. Moitas grazas tamén a todo o persoal de apoio do centro, que fai que as nosas vidas sexan moito máis sinxelas.

Non me quero olvidar da miña estadía en París, foron uns meses onde pasaron moitas cousas (boas e malas) e coincidimos persoas nun mesmo punto vital. Agradecer a toda a xente do Colexio de España (Cité Universitaire) que me fixo tan sinxelo estar lonxe da casa, foi un pracer compartir residencia. Moitas grazas aos meus galegos de confianza Fredo, Martín e Ramonda, por integrarme tan ben e con tanto gusto e, sobre todo, por levarme a tomar licor café. Tamén tiven a gran sorte de formar unha pequena familia, sin os cales a miña estadía tería sido moito máis aburrida. Non foron poucas as sesións de cine, de turismo por París e os viaxes por todo o norte francés, sen dúbida algunha percorrer a costa normanda nun Lexus entregado por *Frozono* é dos meus maiores momentos vitais. Tampouco estivo nada mal escapar do mont Saint-Michel no Tesla dun descoñecido ao cal aproveito para darlle as grazas se algún día le isto. Grazas a Cris, Claudia, Jaime, Inés, Mario, Miki, Tania e Yoli por ser o mellor equipo do mundo, onde todo sae xenial e sempre estamos uns para tirar dos outros. Espero que poidamos seguir celebrando os nosos éxitos colectivos durante moitísimos anos máis.

Ao longo dos últimos anos o deporte foi unha vía de escape importante para sobrelevar, en xeral, as adversidades. O meu templo, onde sempre puiden deixar todos os meus problemas na porta, foi o karate, sendo probablemente as horas máis esperadas dos distintos días da semana. Quero agradecer a todos os meus compañeiros tanto da USC como do club San Francisco Teo, por crear sempre un ambiente tan incrible. Por enriba de todo, infinitas grazas ao meu sensei Óscar Lafuente, por crear un espazo seguro que me devolveu ás artes marciais. O karate non só ten sorte de terte nas súas filas: enriquéceas, inspíraas e lévaas a outro nivel. Grazas tamén por todo o traballo organizando campionatos e polo agarimo, a confianza e o ben que nos coidas a todo o staff da Federación Galega de Karate. E estendo o agradecemento a todos os colegas cos que me xunto de cando en vez para montar tatamis durante toda unha fin de semana: quen me ía dicir que traballar podía ser tan divertido. A xente boa rodéase de xente boa... e vós sodes a proba.

Nas distintas etapas da miña vida, ademais de todo o que aprendín, tiven a inmensa sorte de rodearme dos mellores amigos que xamais puiden imaxinar. Grazas, Sabela e Marcos, por esas tardes infinitas de xogos de mesa, polos nosos plans de calidade que sempre saen redondos e por ser os mellores oídos e os mellores conselleiros. Fágoo curto: agardo toda a semana polo noso día para poñernos ao día.

A Alex, por todas esas tardes soleadas nun chiringuito intentando arranxar o mundo, mil grazas por estar aí, por apoiarme sempre e darme forza en cada decisión que fun tomando no meu día a día.

A Aitor[2], Laura, Manu e Brais por ser esa xentiña marabillosa que fai que encaixar nun entorno que de primeiras nos abruma a calquera sexa non só sinxelo, senón un auténtico pracer.

A Gamallo e a Fran téñolles que agradecer que sigan sendo os meus socios oficiais de verbena, festival e, en xeral, de calquera plan que implique boa música e un raio de sol. Agardo que sigamos montando festas coma nas épocas da carreira e do doutorado...

ata a xubilación e máis alá.

A Silvia e a Marta, grazas por aturar as miñas queixas constantes e, sobre todo, por eses campus-paseos de *coaching* que me puxeron (e me poñen) a vida en perspectiva.

E a César quero darlle unhas grazas maiúsculas por estar sempre aí, pase o que pase; polo seu apoio incondicional e por ser un amigo con todas as letras, deses que fan que todo sexa moito máis doado. Non cambies nunca: o mundo precisa moita máis xente coma ti. Que che quede gravado a lume: vales moito máis do que pensas e mereces que te valores como o auténtico crack que es. Eu, dende logo, estou fondamente agradecido de que os nosos camiños se cruzasen.

Por último, o meu gran pilar no día a día: a miña familia, os que me coñecen dende hai unha eternidade e dende o máis fondo de min. A primeira, a Roxy, esa gata lindísima que, dende que chegou ao mundo, non se separa de min e me acompaña en cada sesión de programación. E grazas tamén a Nito por ser como é: se hai algo máis complicado ca ser pai, é ser pai de alguén que non é o teu fillo.

Sempre digo que o mellor regalo que me fixo miña nai, foi poñerte no meu camiño. De pequeno non facía máis ca pedir unha irmá pequena, unha compañeira de xogos, de vida. E acórdome coma se fose hoxe do día no que souben que mamá estaba embarazada, foi un momento de infinita felicidade, estaba convencido de que ese era o momento máis feliz da miña vida. Pero estaba equivocado. O momento máis feliz da miña vida foi aquel día de novembro no que che vin por primeira vez. Eras tan pequena, tan bonita, tan túa... que o mundo enteiriño pareceu pararse só para mirarte. Desde aí todo foi rodado. Crecer contigo ao lado foi —e segue sendo— a mellor aventura. A miña infancia é un álbum cheo de momentos que son só nosos, códigos secretos que só nós entendemos. Non sei nin quero imaxinar unha vida sen terte ao meu carón.

Grazas por todo, Xeila. Pero, sobre todo, grazas por ser como es: por esa resiliencia brutal, por esa maneira tan adulta e madura de afrontar os problemas dende que eras ben pequena. Cada día aprendo algo novo de ti, e iso é porque tes unha forza e unha luz das que todas e todos podemos aprender. Estou infinitamente orgulloso da muller na que te estás convertendo. E ti sábelo: vou estar orgulloso de ti sempre, todos os días, en cada paso que des.

No momento de escribir estas palabras teño 26 anos, exactamente os mesmos que tiña miña nai cando me deu a luz. A día de hoxe paréceme impensable, coa miña idade, ter un fillo. E aínda máis selo todo á vez: pai e nai nun só corpo. Quizais pense así porque sei, con absoluta claridade, o traballo incansable que miña nai fai día tras día polos seus fillos; un traballo que case nunca se ve: non dá lugar a *papers*, nin a comunicacións en congresos, nin moito menos a citas. Esta tese podería levar perfectamente o seu nome, porque se hoxe a podo rematar é grazas a que ela loita cada día por min. Grazas por ser a miña compañeira máis fiel, por escoitar as miñas preocupacións e repetirme sempre: "*o que ti decidas estará ben*".

Para min xa es a persoa máis valente do mundo, unha desas heroínas que non saen nos cómics pero que sosteñen o mundo cos brazos espidos: polo teu esforzo inagotable, por non tirar nunca a toalla, por non escoller xamais o camiño fácil e, sobre todo, por

romper ese círculo de violencia do "teste que fastidiar porque eu tamén me fastidiei" que ti mesma tanto sufriches. Ti ensináchesme que todo está por escribir, que o pasado non nos define e que, mentres haxa vida, hai solución. Grazas por apoiarme sempre, por abrirme as ás cando quixen voar e por non porme nunca pedras no camiño.

Por certo, teño que dicir que superaches con matrícula a mítica "fase" das preguntas. A min sempre me encantou ser preguntón e ir descubrindo o mundo a base de curiosidade. Ao final, con toda esta historia, acabei facendo unha tese que, no fondo, non deixa de ser unha nova etapa na que sigo lanzando preguntas... coa diferencia de que agora tamén me toca atopar as respostas.

De todas as persoas do mundo, ti es quen máis mérito ten neste traballo. Sen ti, isto non existiría. Se ti tiras a toalla, nada pasa. Así de claro. As nosas vidas van en paralelo: os dous perdemos un pai con trece anos, un mes de febreiro. Pero hai unha diferencia brutal: eu tíñate a ti... e ti non tiñas a ninguén. E aínda así levantácheste, tiraches para adiante e empurráchesnos a todos. Grazas por converter aquel 2013 nun ano que, a pesar de todo o que nos caeu enriba, aínda se sente como ir no coche, ventanillas abaixo, música a tope e esa sensación de liberdade absoluta pegada á pel. Iso é cousa túa. Todo o mérito é teu.

Ogallá, se algún día teño fillos, sexa capaz de crialos tan ben como ti nos crias a nós. Porque o que ti fas non se aprende en libros nin se compra con cartos.

Non volvas permitir xamais sentirte máis pequena ca ninguén: conseguiches máis que moita xente chea de títulos, medallas e postureo. Ti si que es grande, e demóstralo cada día.

"O poder reside onde as persoas cren que reside. É un truco, unha sombra na parede. E unha persoa moi pequena pode proxectar unha sombra moi grande."

— Lord Varys, *"Game of Thrones"*.

24 de marzo do 2026

Additional acknowledgements

First and foremost, I would like to express my deepest gratitude for the privilege of having developed this thesis within the framework of a public education and research system. This doctoral degree is not an isolated achievement, but the culmination of an academic journey that has been entirely supported by public institutions. I am profoundly indebted to a system designed to provide equal opportunities and access to knowledge, which has allowed me to progress from my early studies to the highest level of academic research. Conducting this work at a public university and within a public research center, supported by public funding, reinforces my conviction that science must be preserved and nurtured as a common good for the benefit of society.

Specific financial support for this research has been provided by the following public grants and projects:

- This work has been supported by the *"Ministerio de Ciencia, Innovación y Universidades del Gobierno de España"* through the European Union NextGenerationEU recovery plan **PRTR-C17.I1**.

- This project is funded by the *"European Union"* through the EuroHPC programme under grant agreement **101194491**, and by the *"Ministerio de Ciencia, Innovación y Universidades del Gobierno de España"* (MICIU) and the *"Agencia Estatal de Investigación"* (AEI, 10.13039/501100011033), under the projects **PCI2025-163133**, **PCI2025-163180**, and **PCI2025-163229**.

- Financial support from the *"Ministerio de Economía, Comercio y Empresa del Gobierno de España"* under the grants **PID2019-104834GB-I00**, **PID2022-141623NB-I00**, **PID2022-137061OB-C22**.

- Financial support provided to CiTIUS (Research Centre on Intelligent Technologies) through the Galician Research Center accreditation 2019-2027 (grant **ED431G-2023/04**) funded by the *"Consellería de Cultura, Educación e Ordenación Universitaria"*.

- This research was co-funded by the *"European Union"* (European Regional Development Fund - ERDF).
- The author acknowledges the financial support from the *"Consellería de Educación, Cultura y Ordenación Universitaria de la Xunta de Galicia"* through the predoctoral grant **ED481A 2022/241**.

Contents

Resumo

A computación de altas prestacións (HPC, polas siglas en inglés) serviu historicamente como o motor principal para o avance científico, permitindo simulacións complexas que van dende a predición meteorolóxica ata o estudo de proteínas ou o descubrimento de fármacos. Durante décadas, o crecemento exponencial da potencia computacional sostívose sobre os axiomas empíricos da Lei de Moore e o escalado de Dennard. Non obstante, na última década, estes principios comezaron a amosar signos inequívocos de saturación. As limitacións físicas, termodinámicas e cuánticas fundamentais dos transistores de silicio obrigaron á industria a abandonar o escalado de frecuencia en favor do paralelismo masivo e a heteroxeneidade arquitectónica. En consecuencia, os supercomputadores contemporáneos de primeiro nivel xa non son máquinas homoxéneas, senón clústeres complexos que integran unidades de procesamento central (CPU) multinúcleo con aceleradores especializados como unidades de procesamento gráfico (GPU) ou Field-Programmable Gate Arrays (FPGAs).

Dentro desta diversificación de hardware, a computación cuántica (QC, polas siglas en inglés) emerxe non só como unha tecnoloxía disruptiva illada, senón como o seguinte paso lóxico na evolución dos aceleradores de High-Performance Computing (HPC). Ao explotar os principios da mecánica cuántica –especificamente a superposición e o entrelazamento– as novas unidades de procesamento cuántico (QPU) prometen resolver problemas que permanecen intratables para os sistemas clásicos, ofrecendo aceleracións exponenciais en dominios como a química cuántica, a optimización combinatoria e a criptografía. Non obstante, este avance computacional é meramente existente dentro do marco teórico-matemático, para a existencia de ordenadores cuánticos é necesaria unha implementación do marco teórico en dispositivos físicos. A materialización desta promesa enfróntase a barreiras de enxeñaría formidables.

Actualmente residimos nunha era "cuántica de escala intermedia ruidosa" (NISQ en inglés). Os dispositivos existentes, baseados en tecnoloxías como circuitos superconductores ou ións atrapados, están limitados a unhas poucas ducias ou centos de qubits. Escalar estes procesadores monolíticos cara aos millóns de qubits necesarios para a tolerancia a fallos presenta desafíos físicos extremos. Problemas como a saturación de frecuencias en chips superconductores, a diafonía (*crosstalk*) entre qubits adxacentes, a disipación de calor dentro dos refrixeradores de dilución e a complexidade do cableado

de control suxiren que a estratexia do "chip xigante único" enfróntase a un teito de escalabilidade difícil de superar a curto prazo.

Para superar estas limitacións físicas, a comunidade científica comezou a virar cara a un paradigma modular: a computación cuántica distribuida (DQC). Esta arquitectura propón a interconexión de múltiples Quantum Processing Units (QPUs) de tamaño moderado a través de canles cuánticas (enlaces ópticos) capaces de distribuír entrelazamento. Ao conectar en rede múltiples nodos cuánticos, faise posible construír un sistema lóxico unificado cunha capacidade computacional que escala linealmente co número de nodos, permitindo a execución de algoritmos que requiren un volume de qubits inalcanzable para calquera dispositivo monolítico individual.

Non obstante, aínda que a DQC ofrece unha vía clara para o escalado do hardware, introduce unha complexidade sen precedentes na capa de software. A diferenza dos clústeres clásicos, onde a información pode copiarse e transmitirse libremente, a comunicación entre nodos cuánticos está restrinxida polo "Teorema de Non-Clonado". A transmisión de información cuántica require protocolos sofisticados como a teletransportación cuántica (*Teledata*) e a execución remota de portas (*Telegate*), ambos os dous consumidores dun recurso funxible e custoso: os pares Einstein-Podolsky-Rosen (EPR).

Aquí radica o problema central que motiva esta tese doutoral: a existencia dunha profunda brecha tecnolóxica e semántica entre o hardware de rede cuántica e as ferramentas de desenvolvemento de software. Por unha banda, a investigación en física de redes logrou fitos significativos na xeración e purificación de entrelazamento na capa física. Por outra banda, a comunidade científica de algoritmos cuánticos continúa deseñando aplicacións asumindo unha abstracción idealizada de hardware monolítico e totalmente conectado. Existe unha ausencia dun "pegamento" intermedio: unha pila de software robusta capaz de xestionar a complexidade estocástica da rede cuántica e presentala ao programador dun xeito transparente e eficiente.

A ausencia desta pila de software obriga aos investigadores actuais a programar as redes cuánticas manualmente, xestionando explicitamente a xeración de pares EPR, as medicións de Bell e as correccións clásicas. Este enfoque de baixo nivel é propenso a erros, non escalable e dificulta significativamente a integración de recursos cuánticos distribuídos nos fluxos de traballo actuais dos centros de datos HPC.

En consecuencia, o obxectivo principal desta tese é deseñar, desenvolver e validar unha arquitectura de software completa e agnóstica do hardware que permita a programación, compilación e execución de algoritmos cuánticos distribuídos. Esta arquitectura ten como obxectivo integrar o paradigma cuántico distribuído dentro dos estándares probados de HPC, especificamente adaptando o modelo de Interface de Paso de Mensaxes (Message Passing Interface (MPI)) e as representacións intermedias modernas LLVM e Quantum Intermediate Representation (QIR), para solventar definitivamente a brecha entre a física da rede cuántica e a enxeñaría de software algorítmica.

A transición cara a DQC vese obstaculizada por unha desconexión fundamental

entre a capa física e a capa de aplicación, referida nesta tese como a "Brecha de Integración Cuántica-HPC". Ainda que os fundamentos teóricos das redes cuánticas –como o intercambio de entrelazamento e a purificación– están ben establecidos, a infraestrutura de software necesaria para utilizar estes recursos permanece nunha etapa incipiente.

Actualmente, o desenvolvemento de aplicacións cuánticas distribuídas depende en gran medida de modelos de programación imperativos de baixo nivel. Ferramentas como o SDK de NetQASM representan o estado da arte para controlar nodos de rede cuántica. Non obstante, estas ferramentas expoñen toda a complexidade do hardware ao desenvolvedor. Para implementar unha simple porta lóxica remota, un programador debe solicitar explicitamente a xeración de pares EPR, xestionar os resultados estocásticos das medicións de estados de Bell e sincronizar manualmente os sinais de control clásicos a través da rede. Este enfoque obriga ao deseñador de algoritmos a actuar como un enxeñeiro de redes, acoplando inextricablemente a lóxica da aplicación coa topoloxía específica e as limitacións físicas do hardware subxacente.

Este escenario reflicte os primeiros días da computación clásica, onde os programadores debían xestionar manualmente os rexistros de memoria e as interconexións físicas. No contexto da HPC, isto sería semellante a programar un supercomputador utilizando linguaxe ensamblador para sockets de rede en lugar de utilizar abstraccións de alto nivel como MPI. Esta falta de abstracción non só aumenta a taxa de erros, senón que tamén fai que o código non sexa portable; un algoritmo optimizado para unha topoloxía de rede específica (por exemplo, unha cadea lineal) debe reescribirse completamente para funcionar nunha arquitectura diferente (por exemplo, unha topoloxía en estrela). En consecuencia, a integración da DQC nos fluxos de traballo principais de supercomputación está estancada pola ausencia dunha pila de software estandarizada e agnóstica do hardware.

Para abordar este desafío, esta tese plantexa a seguinte hipótese central: *É posible definir unha pila de software agnóstica do hardware que integre recursos cuánticos distribuídos en ambientes HPC adoptando o paradigma "Un Programa, Múltiples Datos" (comúnmente coñecido como SPMD polas siglas en inglés), xestionando así a complexidade das redes cuánticas –especificamente a xeración de pares EPR e a teletransportación– de forma transparente para o usuario.*

A validación desta hipótese estrutúrase arredor de seis obxectivos estratéxicos:

- **O1 (Revisión do estado do arte):** Realizar unha caracterización rigorosa do estado da arte na compilación cuántica e arquitecturas distribuídas, identificando as limitacións específicas das Intermediate Representations (IRs) actuais.

- **O2 (Capa de Desenvolvemento):** Deseñar e implementar unha biblioteca de alto nivel (NetQMPI) que adapte o estándar clásico MPI ao dominio cuántico, permitindo o desenvolvemento de algoritmos distribuídos utilizando primitivas semánticas.

- **O3 (Capa de Compilación):** Definir unha IR estandarizada (NetQIR) baseada na infraestrutura Low-Level Virtual Machine (LLVM), capaz de describir operacións distribuídas e facilitar optimizacións a nivel de compilador.

- **O4 (Benchmarking):** Establecer un marco de avaliación rigoroso integrando simuladores de rede de alta fidelidade (SeQUeNCe, NetSquid) para validar a corrección funcional e o rendemento da pila proposta.

- **O5 (Integración):** Demostrar a interoperabilidade da pila con modelos de programación heteroxéneos, especificamente SYCL, permitindo a orquestración de cargas de traballo híbridas Central Processing Unit (CPU)-QPU.

- **O6 (Aplicación):** Validar a utilidade da pila a través da implementación e optimización dun algoritmo concreto, a Inverse Quantum Fourier Transform (iQFT) distribuída, introducindo estratexias novidosas para a redución da comunicación.

A eficacia de calquera sistema HPC baséase fundamentalmente na madurez da súa pila de compilación. Un compilador actúa como a ponte crítica que traduce as abstraccións lexibles por humanos en instrucións executables pola máquina. No dominio clásico, este proceso está desacoplado de forma notable en fases de análise (front-end) e síntese (back-end), mediadas por unha IR como LLVM IR. Este deseño modular permite que M linguaxes de programación se dirixan a N arquitecturas de hardware cunha complexidade de $M + N$.

No dominio cuántico, non obstante, o ecosistema sufriu historicamente de fragmentación. Os primeiros marcos de traballo dependían de compiladores monolíticos que traducían linguaxes específicas (e.g., Qiskit, Cirq) directamente a backends de hardware específicos, resultando nunha complexidade combinatoria de $M \times N$. Aínda que esforzos de estandarización como OpenQASM 2.0 e Quil proporcionaron unha interface común para dispositivos monolíticos, foron deseñados como secuencias de ensamblaxe lineais. Estes formatos describen un circuíto cuántico como unha lista estática de portas aplicadas a un rexistro local, un modelo que é fundamentalmente inadecuado para a DQC.

A computación cuántica distribuída introduce eventos asíncronos e un fluxo de control non determinista. Operacións como a xeración de entrelazamento son probabilísticas; poden ter éxito ou fallar, requirindo que o software de control agarde, tente de novo ou se ramifique baseándose nos resultados en tempo de execución. As IRs lineais tradicionais non poden expresar naturalmente estas construcións sen extensións engorrosas. Ademais, carecen do concepto de topoloxía de rede ou rexistros remotos.

A aparición da QIR, baseada no estándar LLVM, marcou un punto de inflexión. A QIR permite incrustar primitivas cuánticas dentro dun grafo de fluxo de control clásico robusto, herdando décadas de técnicas de optimización da literatura de compiladores clásicos. Non obstante, a especificación estándar de QIR foi deseñada pensando en

chips monolíticos. Carece das instrucións intrínsecas necesarias para describir operacións de rede, como a execución remota de portas ou o intercambio de entrelazamento. Esta análise do estado da arte leva a unha conclusión crítica: para que a DQC escale, a pila de software debe evolucionar máis aló do paradigma do "circuíto local". Require unha nova representación intermedia que estenda a modularidade da QIR con soporte explícito para a semántica distribuída.

Abordando as limitacións das IRs actuais identificadas na análise do estado da arte, a primeira gran contribución desta tese é a definición e especificación de NetQIR. Proponse como unha extensión formal da QIR estándar, que á súa vez se basea na omnipresente infraestrutura LLVM. A decisión estratéxica de construír sobre LLVM é fundamental: asegura que a pila de rede cuántica non sexa un silo illado, senón un compoñente integrado do ecosistema máis amplo de compiladores de código aberto, beneficiándose de ferramentas e pasos de optimización maduros.

A innovación fundamental de NetQIR radica na súa elevación das primitivas de rede a instrucións intrínsecas. Na QIR estándar, unha operación remota é opaca; o compilador só ve unha secuencia de portas locais e chamadas de medición. NetQIR introduce instrucións semánticas explícitas para operacións distribuídas. Ao formalizar estas operacións dentro da IR, NetQIR permite ao compilador realizar optimizacións conscientes da rede. Por exemplo, un paso do compilador pode agora analizar o grafo de dependencia dun algoritmo distribuído e identificar oportunidades para anticipar a obtención de entrelazamento. Se o compilador detecta que se require unha teletransportación na instrución N, pode elevar a solicitude de entrelazamento (a chamada `gen_ent`) á instrución $N - k$, enmascarando efectivamente a latencia da xeración do enlace cuántico tras a computación local. Esta técnica de "ocultación de latencia", estándar na HPC clásica pero novidosa na DQC, só é posible porque a IR expón explicitamente a semántica dos recursos de rede.

Ademais, NetQIR serve como unha fronteira agnóstica do hardware. Actúa como un contrato entre o software de alto nivel e o hardware de control de baixo nivel. O front-end (linguaxe de alto nivel) emite NetQIR sen preocuparse pola implementación física específica do enlace cuántico (e.g., centros nitróxeno-vacante vs. ións atrapados). O back-end sintetiza entón esta representación nos pulsos de control específicos ou instrucións de baixo nivel (como NetQASM) requiridas polo dispositivo obxectivo. Esta modularidade cumpre o Obxectivo O3, desacoplando efectivamente a lóxica de compilación das limitacións físicas da rede.

Aínda que NetQIR resolve o problema da representación da máquina, non aborda a brecha de usabilidade humana. Programar directamente nunha IR ou incluso en linguaxes imperativas de baixo nivel como NetQASM segue sendo prohibitivamente complexo para os deseñadores de algoritmos, que deben xestionar manualmente as complexidades dos ciclos de vida do entrelazamento e a sincronización clásica. Para salvar esta brecha e cumprir o Obxectivo O2, esta tese presenta NetQMPI, unha biblioteca en Python de alto nivel deseñada para levar o paradigma Single Program Multiple Data (SPMD) á DQC.

NetQMPI extrae a súa filosofía de deseño directamente do estándar MPI, estándar *de facto* da supercomputación clásica. A elección de MPI como modelo de referencia é estratéxica: proporciona unha sintaxe e un modelo mental que xa son familiares para a gran maioría dos desenvolvedores de HPC, rebaixando así a barreira de entrada para o dominio cuántico.

A arquitectura de NetQMPI actúa como unha capa de middleware que se sitúa entre a lóxica da aplicación e o controlador de rede. A súa innovación central é a abstracción de protocolos físicos en primitivas semánticas. Nun programa NetQMPI, o complexo protocolo "Teledata" (que implica xeración de entrelazamento, medición de Bell, transmisión clásica e corrección de Pauli) está encapsulado nunha única chamada de función bloqueante: `comm.qsend(qubit, dest_rank)`. De xeito similar, o receptor executa `comm.qrecv(source_rank)`.

Internamente, a biblioteca xestiona unha rigorosa máquina de estados. Cando se invoca `qsend`, NetQMPI:

1. Negocia automaticamente a creación dun par EPR co nodo de destino.

2. Realiza a medición local na base de Bell.

3. Serializa os resultados da medición.

4. Transmite estes bits clásicos a través do plano de control clásico.

No lado receptor, `qrecv` agarda pola mensaxe clásica e aplica automaticamente as correccións X/Z necesarias para reconstruír o estado cuántico. Esta automatización transforma a rede dun complexo experimento físico nun recurso computacional fiable. Crucialmente, NetQMPI habilita o modelo de execución SPMD. Un usuario escribe un único script onde o comportamento está determinado polo `rank` do nodo. Este script execútase simultaneamente en todos os nodos da simulación ou do banco de probas de hardware. A biblioteca xestiona sen problemas o intercalado do fluxo de control clásico (lóxica Python) e as instrucións cuánticas, despachando estas últimas ao backend NetQASM subxacente. Este deseño asegura que NetQMPI non sexa só un xoguete de simulador senón unha pila de software de sistema válida; o mesmo código que se executa nunha simulación de portátil pode despregarse nunha rede cuántica física, sempre que os nodos soporten o conxunto de instrucións NetQASM.

O desenvolvemento dunha pila de software para unha tecnoloxía incipiente como a DQC presenta un desafío único: a indispoñibilidade de hardware físico a gran escala para a validación. Aínda que existen bancos de probas a pequena escala, carecen do volume e a estabilidade necesarios para probas de estrés rigorosas dunha pila de compilación distribuída. En consecuencia, para cumprir o Obxectivo O4, esta tese estableceu un marco de avaliación comparativa (*benchmarking*) completo arraigado na simulación de alta fidelidade.

Esta fase experimental serviu un propósito: proporcionou unha liña base cuantitativa para os recursos necesarios para executar algoritmos distribuídos, datos que son vitais para o futuro co-deseño de hardware.

A computación cuántica distribuída non pode existir no baleiro; debe integrarse no ecosistema HPC existente. Os supercomputadores modernos son heteroxéneos, combinando CPUs multinúcleo con aceleradores como Graphics Processing Units (GPUs). O Obxectivo O5 desta tese abordou o desafío de integrar o fluxo de traballo cuántico neste ambiente heteroxéneo utilizando SYCL, o estándar aberto para a programación heteroxénea de fonte única.

A hipótese que impulsou esta sección foi que os recursos de DQC poderían tratarse abstractamente como só outro tipo de acelerador dentro do modelo SYCL. Para validar isto, desenvolveuse un backend SYCL personalizado. A diferenza de enfoques anteriores que simplemente envolvían chamadas cuánticas, esta implementación integrou un simulador cuántico –especificamente *Qiskit Aer*– directamente no ambiente de execución SYCL.

Este paso fundacional foi crucial para demostrar a Orquestración CPU-QPU. Ao habilitar un backend de simulación dentro de SYCL, fíxose posible escribir un único ficheiro fonte en C++ que:

1. Asigna datos na CPU anfitrioa.

2. Delega tarefas paralelas clásicas (e.g., pre-procesamento de matrices) a unha cola de GPU.

3. Delega tarefas cuánticas (e.g., execución de circuítos) á cola da QPU (simulada vía Aer).

4. Sincroniza os resultados a través de accesores de memoria unificada.

Esta integración confirmou empiricamente que o fluxo de traballo DQC é compatible cos estándares modernos de programación heteroxénea. Demostrou que a barreira entre código clásico e cuántico é artificial e pode eliminarse mediante un sistema de execución unificado. Aínda que o backend utilizado foi un simulador (Qiskit Aer), os patróns arquitectónicos establecidos –colas, búferes e grupos de comandos– son idénticos aos requiridos para hardware cuántico físico. Este logro abre o camiño para futuros sistemas onde un planificador asigna dinamicamente tarefas a CPUs, GPUs ou QPUs distribuídas baseándose na natureza da carga de traballo, xestionado enteiramente a través dunha interface de software unificada.

Para demostrar a utilidade práctica da pila de software proposta máis aló da simple tecnoloxía habilitadora, esta tese abordou un desafío algorítmico concreto: a optimización da iQFT Distribuída (Obxectivo O6). A selección da iQFT como caso de estudo é estratéxica; constitúe a columna vertebral computacional de subrutinas cuánticas fundamentais, como a Quantum Phase Estimation (QPE) e o algoritmo de Shor. Non

obstante, desde unha perspectiva de arquitectura distribuída, a iQFT representa o "peor escenario posible" debido aos seus rigorosos requisitos de conectividade. Na súa representación canónica, cada qubit debe interactuar con todos os demais qubits, o que implica unha topoloxía de conectividade todos-con-todos cuxa escalabilidade resulta deficiente ao estenderse a arquitecturas distribuídas baseadas en múltiples nodos interconectados en rede.

Nun escenario distribuído, unha implementación inxenua da iQFT incorrería nun custo de comunicación cuadrático ($\mathcal{O}(N^2)$), xa que cada porta de fase controlada non local require o consumo dun par EPR vía o protocolo Telegate. Esta sobrecarga ameaza con negar a vantaxe cuántica. Para mitigar isto, esta tese propuxo unha nova estratexia de optimización arraigada nas propiedades analíticas da transformada de Fourier.

A idea central é que o ángulo de rotación $\theta_k = \pi/2^k$ das portas de fase controlada decae exponencialmente coa distancia lóxica k entre qubits. Ao analizar a converxencia do erro de fase acumulado, definiuse un *Horizonte de Comunicación* ($d_{\max}$). Este parámetro establece unha fronteira física: se dous qubits están separados por unha distancia de rede maior que $d_{\max}$, a interacción pódase (apróximase como o operador identidade).

Utilizando as capacidades de abstracción da pila de software para modelar estas interaccións, os resultados teóricos demostraron que esta estratexia de podado altera drasticamente o escalado de recursos. Para un limiar de fidelidade fixo, o consumo de pares EPR transita dun crecemento cuadrático a un réxime de saturación ($\mathcal{O}(1)$) por qubit engadido. Este achado é crítico: proba que as limitacións físicas da rede (taxas finitas de xeración de entrelazamento) poden mitigarse a través dun deseño algorítmico intelixente e consciente da rede. A pila de software serve como o habilitador que permite ao compilador impor estes horizontes, localizando efectivamente un algoritmo global.

A investigación presentada nesta tese doutoral abordou un dos desafíos máis urxentes na folla de ruta cara á computación cuántica escalable: a integración de procesadores cuánticos distribuídos no ecosistema HPC. Comezando pola identificación dunha profunda brecha entre os protocolos físicos de rede e as ferramentas de desenvolvemento de software, este traballo definiu, implementou e validou con éxito unha pila de software completa e agnóstica do hardware.

A validez da hipótese central –que o paradigma SPMD pode abstraer efectivamente a complexidade das redes cuánticas– confirmouse a través da consecución dos obxectivos estratéxicos.

En conclusión, esta tese postula que a transición á DQC non é simplemente unha actualización de hardware, senón un cambio de paradigma necesario para o escalado continuo da potencia computacional. Ao proporcionar as ferramentas para tratar a rede cuántica non como un experimento de física, senón como un recurso computacional xestionable, este traballo senta a infraestrutura de software esencial requirida para

transformar dispositivos cuánticos illados nos supercomputadores cuánticos modulares e interconectados do futuro.

Resumen

La computación de altas prestaciones (HPC, por las siglas en inglés) ha servido históricamente como el motor principal para el avance científico, permitiendo simulaciones complejas que van desde la predicción meteorológica hasta el estudio de proteínas o el descubrimiento de fármacos. Durante décadas, el crecimiento exponencial de la potencia computacional se sostuvo sobre los axiomas empíricos de la Ley de Moore y el escalado de Dennard. No obstante, en la última década, estos principios han comenzado a mostrar signos inequívocos de saturación. Las limitaciones físicas, termodinámicas y cuánticas fundamentales de los transistores de silicio obligaron a la industria a abandonar el escalado de frecuencia en favor del paralelismo masivo y la heterogeneidad arquitectónica. En consecuencia, los supercomputadores contemporáneos de primer nivel ya no son máquinas homogéneas, sino clústeres complejos que integran unidades de procesamiento central (CPU) multinúcleo con aceleradores especializados como unidades de procesamiento gráfico (GPU) o FPGAs.

Dentro de esta diversificación de hardware, la computación cuántica (QC, por las siglas en inglés) emerge no solo como una tecnología disruptiva aislada, sino como el siguiente paso lógico en la evolución de los aceleradores de HPC. Al explotar los principios de la mecánica cuántica –específicamente la superposición y el entrelazamiento– las nuevas unidades de procesamiento cuántico (QPU) prometen resolver problemas que permanecen intratables para los sistemas clásicos, ofreciendo aceleraciones exponenciales en dominios como la química cuántica, la optimización combinatoria y la criptografía. No obstante, este avance computacional es meramente existente dentro del marco teórico-matemático; para la existencia de ordenadores cuánticos es necesaria una implementación del marco teórico en dispositivos físicos. La materialización de esta promesa se enfrenta a barreras de ingeniería formidables.

Actualmente residimos en una era "cuántica de escala intermedia ruidosa" (NISQ en inglés). Los dispositivos existentes, basados en tecnologías como circuitos superconductores o iones atrapados, están limitados a unas pocas docenas o cientos de qubits. Escalar estos procesadores monolíticos hacia los millones de qubits necesarios para la tolerancia a fallos presenta desafíos físicos extremos. Problemas como la saturación de frecuencias en chips superconductores, la diafonía (*crosstalk*) entre qubits adyacentes, la disipación de calor dentro de los refrigeradores de dilución y la complejidad del

cableado de control sugieren que la estrategia del "chip gigante único" se enfrenta a un techo de escalabilidad difícil de superar a corto plazo.

Para superar estas limitaciones físicas, la comunidad científica ha comenzado a virar hacia un paradigma modular: la computación cuántica distribuida (DQC). Esta arquitectura propone la interconexión de múltiples QPUs de tamaño moderado a través de canales cuánticos (enlaces ópticos) capaces de distribuir entrelazamiento. Al conectar en red múltiples nodos cuánticos, se hace posible construir un sistema lógico unificado con una capacidad computacional que escala linealmente con el número de nodos, permitiendo la ejecución de algoritmos que requieren un volumen de qubits inalcanzable para cualquier dispositivo monolítico individual.

No obstante, aunque la DQC ofrece una vía clara para el escalado del hardware, introduce una complejidad sin precedentes en la capa de software. A diferencia de los clústeres clásicos, donde la información puede copiarse y transmitirse libremente, la comunicación entre nodos cuánticos está restringida por el "Teorema de No Clonado". La transmisión de información cuántica requiere protocolos sofisticados como la teletransportación cuántica (*Teledata*) y la ejecución remota de puertas (*Telegate*), siendo ambos consumidores de un recurso fungible y costoso: los pares EPR.

Aquí radica el problema central que motiva esta tesis doctoral: la existencia de una profunda brecha tecnológica y semántica entre el hardware de red cuántica y las herramientas de desarrollo de software. Por un lado, la investigación en física de redes ha logrado hitos significativos en la generación y purificación de entrelazamiento en la capa física. Por otro lado, la comunidad científica de algoritmos cuánticos continúa diseñando aplicaciones asumiendo una abstracción idealizada de hardware monolítico y totalmente conectado. Existe una ausencia de un "pegamento" intermedio: una pila de software robusta capaz de gestionar la complejidad estocástica de la red cuántica y presentarla al programador de una manera transparente y eficiente.

La ausencia de esta pila de software obliga a los investigadores actuales a programar las redes cuánticas manualmente, gestionando explícitamente la generación de pares EPR, las mediciones de Bell y las correcciones clásicas. Este enfoque de bajo nivel es propenso a errores, no escalable y dificulta significativamente la integración de recursos cuánticos distribuidos en los flujos de trabajo actuales de los centros de datos HPC.

En consecuencia, el objetivo principal de esta tesis es diseñar, desarrollar y validar una arquitectura de software completa y agnóstica del hardware que permita la programación, compilación y ejecución de algoritmos cuánticos distribuidos. Esta arquitectura tiene como objetivo integrar el paradigma cuántico distribuido dentro de los estándares probados de HPC, específicamente adaptando el modelo de Interfaz de Paso de Mensajes (MPI) y las representaciones intermedias modernas LLVM y QIR, para solventar definitivamente la brecha entre la física de la red cuántica y la ingeniería de software algorítmica.

La transición hacia la DQC se ve obstaculizada por una desconexión fundamental entre la capa física y la capa de aplicación, referida en esta tesis como la "Brecha de Integración Cuántica-HPC". Aunque los fundamentos teóricos de las redes cuánticas

–como el intercambio de entrelazamento y la purificación– están bien establecidos, la infraestructura de software necesaria para utilizar estos recursos permanece en una etapa incipiente.

Actualmente, el desarrollo de aplicaciones cuánticas distribuidas depende en gran medida de modelos de programación imperativos de bajo nivel. Herramientas como el SDK de NetQASM representan el estado del arte para controlar nodos de red cuántica. No obstante, estas herramientas exponen toda la complejidad del hardware al desarrollador. Para implementar una simple puerta lógica remota, un programador debe solicitar explícitamente la generación de pares EPR, gestionar los resultados estocásticos de las mediciones de estados de Bell y sincronizar manualmente las señales de control clásicas a través de la red. Este enfoque obliga al diseñador de algoritmos a actuar como un ingeniero de redes, acoplando inextricablemente la lógica de la aplicación con la topología específica y las limitaciones físicas del hardware subyacente.

Este escenario refleja los primeros días de la computación clásica, donde los programadores debían gestionar manualmente los registros de memoria y las interconexiones físicas. En el contexto de la HPC, esto sería similar a programar un supercomputador utilizando lenguaje ensamblador para sockets de red en lugar de utilizar abstracciones de alto nivel como MPI. Esta falta de abstracción no solo aumenta la tasa de errores, sino que también hace que el código no sea portable; un algoritmo optimizado para una topología de red específica (por ejemplo, una cadena lineal) debe reescribirse completamente para funcionar en una arquitectura diferente (por ejemplo, una topología en estrella). En consecuencia, la integración de la DQC en los flujos de trabajo principales de supercomputación está estancada por la ausencia de una pila de software estandarizada y agnóstica del hardware.

Para abordar este desafío, esta tesis plantea la siguiente hipótesis central: *Es posible definir una pila de software agnóstica del hardware que integre recursos cuánticos distribuidos en entornos HPC adoptando el paradigma "Un Programa, Múltiples Datos" (comúnmente conocido como SPMD por las siglas en inglés), gestionando así la complejidad de las redes cuánticas –específicamente la generación de pares EPR y la teletransportación– de forma transparente para el usuario.*

La validación de esta hipótesis se estructura alrededor de seis objetivos estratégicos:

- **O1 (Revisión del estado del arte):** Realizar una caracterización rigurosa del estado del arte en la compilación cuántica y arquitecturas distribuidas, identificando las limitaciones específicas de las IRs actuales.
- **O2 (Capa de Desarrollo):** Diseñar e implementar una biblioteca de alto nivel (NetQMPI) que adapte el estándar clásico MPI al dominio cuántico, permitiendo el desarrollo de algoritmos distribuidos utilizando primitivas semánticas.
- **O3 (Capa de Compilación):** Definir una IR estandarizada (NetQIR) basada en la infraestructura LLVM, capaz de describir operaciones distribuidas y facilitar optimizaciones a nivel de compilador.

- **O4 (Benchmarking):** Establecer un marco de evaluación riguroso integrando simuladores de red de alta fidelidad (SeQUeNCe, NetSquid) para validar la corrección funcional y el rendimiento de la pila propuesta.
- **O5 (Integración):** Demostrar la interoperabilidad de la pila con modelos de programación heterogéneos, específicamente SYCL, permitiendo la orquestación de cargas de trabajo híbridas CPU-QPU.
- **O6 (Aplicación):** Validar la utilidad de la pila a través de la implementación y optimización de un algoritmo concreto, la iQFT distribuida, introduciendo estrategias novedosas para la reducción de la comunicación.

La eficacia de cualquier sistema HPC se basa fundamentalmente en la madurez de su pila de compilación. Un compilador actúa como el puente crítico que traduce las abstracciones legibles por humanos en instrucciones ejecutables por la máquina. En el dominio clásico, este proceso está desacoplado de forma notable en fases de análisis (front-end) y síntesis (back-end), mediadas por una IR como LLVM IR. Este diseño modular permite que M lenguajes de programación se dirijan a N arquitecturas de hardware con una complejidad de $M + N$.

En el dominio cuántico, no obstante, el ecosistema ha sufrido históricamente de fragmentación. Los primeros marcos de trabajo dependían de compiladores monolíticos que traducían lenguajes específicos (e.g., Qiskit, Cirq) directamente a backends de hardware específicos, resultando en una complejidad combinatoria de $M \times N$. Aunque esfuerzos de estandarización como OpenQASM 2.0 y Quil proporcionaron una interfaz común para dispositivos monolíticos, fueron diseñados como secuencias de ensamblaje lineales. Estos formatos describen un circuito cuántico como una lista estática de puertas aplicadas a un registro local, un modelo que es fundamentalmente inadecuado para la DQC.

La computación cuántica distribuida introduce eventos asíncronos y un flujo de control no determinista. Operaciones como la generación de entrelazamiento son probabilísticas; pueden tener éxito o fallar, requiriendo que el software de control espere, intente de nuevo o se ramifique basándose en los resultados en tiempo de ejecución. Las IRs lineales tradicionales no pueden expresar naturalmente estas construcciones sin extensiones engorrosas. Además, carecen del concepto de topología de red o registros remotos.

La aparición de la QIR, basada en el estándar LLVM, marcó un punto de inflexión. La QIR permite incrustar primitivas cuánticas dentro de un grafo de flujo de control clásico robusto, heredando décadas de técnicas de optimización de la literatura de compiladores clásicos. No obstante, la especificación estándar de QIR fue diseñada pensando en chips monolíticos. Carece de las instrucciones intrínsecas necesarias para describir operaciones de red, como la ejecución remota de puertas o el intercambio de entrelazamiento. Este análisis del estado del arte lleva a una conclusión crítica: para que la DQC escale, la pila de software debe evolucionar más allá del paradigma

del "circuito local". Requiere una nueva representación intermedia que extienda la modularidad de la QIR con soporte explícito para la semántica distribuida.

Abordando las limitaciones de las IRs actuales identificadas en el análisis del estado del arte, la primera gran contribución de esta tesis es la definición y especificación de NetQIR. Se propone como una extensión formal de la QIR estándar, que a su vez se basa en la omnipresente infraestructura LLVM. La decisión estratégica de construir sobre LLVM es fundamental: asegura que la pila de red cuántica no sea un silo aislado, sino un componente integrado del ecosistema más amplio de compiladores de código abierto, beneficiándose de herramientas y pasos de optimización maduros.

La innovación fundamental de NetQIR radica en su elevación de las primitivas de red a instrucciones intrínsecas. En la QIR estándar, una operación remota es opaca; el compilador solo ve una secuencia de puertas locales y llamadas de medición. NetQIR introduce instrucciones semánticas explícitas para operaciones distribuidas. Al formalizar estas operaciones dentro de la IR, NetQIR permite al compilador realizar optimizaciones conscientes de la red. Por ejemplo, un paso del compilador puede ahora analizar el grafo de dependencia de un algoritmo distribuido e identificar oportunidades para anticipar la obtención de entrelazamiento. Si el compilador detecta que se requiere una teletransportación en la instrucción N, puede elevar la solicitud de entrelazamiento (la llamada `gen_ent`) a la instrucción $N - k$, enmascarando efectivamente la latencia de la generación del enlace cuántico tras la computación local. Esta técnica de "ocultación de latencia", estándar en la HPC clásica pero novedosa en la DQC, solo es posible porque la IR expone explícitamente la semántica de los recursos de red.

Además, NetQIR sirve como una frontera agnóstica del hardware. Actúa como un contrato entre el software de alto nivel y el hardware de control de bajo nivel. El front-end (lenguaje de alto nivel) emite NetQIR sin preocuparse por la implementación física específica del enlace cuántico (e.g., centros nitrógeno-vacante vs. iones atrapados). El back-end sintetiza entonces esta representación en los pulsos de control específicos o instrucciones de bajo nivel (como NetQASM) requeridas por el dispositivo objetivo. Esta modularidad cumple el Objetivo O3, desacoplando efectivamente la lógica de compilación de las limitaciones físicas de la red.

Aunque NetQIR resuelve el problema de la representación de la máquina, no aborda la brecha de usabilidad humana. Programar directamente en una IR o incluso en lenguajes imperativos de bajo nivel como NetQASM sigue siendo prohibitivamente complejo para los diseñadores de algoritmos, que deben gestionar manualmente las complejidades de los ciclos de vida del entrelazamiento y la sincronización clásica. Para salvar esta brecha y cumplir el Objetivo O2, esta tesis presenta NetQMPI, una biblioteca en Python de alto nivel diseñada para llevar el paradigma SPMD a la DQC.

NetQMPI extrae su filosofía de diseño directamente del estándar MPI, estándar *de facto* de la supercomputación clásica. La elección de MPI como modelo de referencia es estratégica: proporciona una sintaxis y un modelo mental que ya son familiares para la gran mayoría de los desarrolladores de HPC, rebajando así la barrera de entrada para el dominio cuántico.

La arquitectura de NetQMPI actúa como una capa de middleware que se sitúa entre la lógica de la aplicación y el controlador de red. Su innovación central es la abstracción de protocolos físicos en primitivas semánticas. En un programa NetQMPI, el complejo protocolo "Teledata" (que implica generación de entrelazamiento, medición de Bell, transmisión clásica y corrección de Pauli) está encapsulado en una única llamada de función bloqueante: `comm.qsend(qubit, dest_rank)`. De manera similar, el receptor ejecuta `comm.qrecv(source_rank)`.

Internamente, la biblioteca gestiona una rigurosa máquina de estados. Cuando se invoca `qsend`, NetQMPI:

1. Negocia automáticamente la creación de un par EPR con el nodo de destino.
2. Realiza la medición local en la base de Bell.
3. Serializa los resultados de la medición.
4. Transmite estos bits clásicos a través del plano de control clásico.

En el lado receptor, `qrecv` espera por el mensaje clásico y aplica automáticamente las correcciones X/Z necesarias para reconstruir el estado cuántico. Esta automatización transforma la red de un complejo experimento físico en un recurso computacional fiable. Crucialmente, NetQMPI habilita el modelo de ejecución SPMD. Un usuario escribe un único script donde el comportamiento está determinado por el `rank` del nodo. Este script se ejecuta simultáneamente en todos los nodos de la simulación o del banco de pruebas de hardware. La biblioteca gestiona sin problemas el intercalado del flujo de control clásico (lógica Python) y las instrucciones cuánticas, despachando estas últimas al backend NetQASM subyacente. Este diseño asegura que NetQMPI no sea solo un juguete de simulador sino una pila de software de sistema válida; el mismo código que se ejecuta en una simulación de portátil puede desplegarse en una red cuántica física, siempre que los nodos soporten el conjunto de instrucciones NetQASM.

El desarrollo de una pila de software para una tecnología incipiente como la DQC presenta un desafío único: la indisponibilidad de hardware físico a gran escala para la validación. Aunque existen bancos de pruebas a pequeña escala, carecen del volumen y la estabilidad necesarios para pruebas de estrés rigurosas de una pila de compilación distribuida. En consecuencia, para cumplir el Objetivo O4, esta tesis estableció un marco de evaluación comparativa (*benchmarking*) completo arraigado en la simulación de alta fidelidad.

Esta fase experimental sirvió un propósito: proporcionó una línea base cuantitativa para los recursos necesarios para ejecutar algoritmos distribuidos, datos que son vitales para el futuro co-diseño de hardware.

La computación cuántica distribuida no puede existir en el vacío; debe integrarse en el ecosistema HPC existente. Los supercomputadores modernos son heterogéneos, combinando CPUs multinúcleo con aceleradores como GPUs. El Objetivo O5 de esta

tesis abordó el desafío de integrar el flujo de trabajo cuántico en este entorno heterogéneo utilizando SYCL, el estándar abierto para la programación heterogénea de fuente única.

La hipótesis que impulsó esta sección fue que los recursos de DQC podrían tratarse abstractamente como solo otro tipo de acelerador dentro del modelo SYCL. Para validar esto, se desarrolló un backend SYCL personalizado. A diferencia de enfoques anteriores que simplemente envolvían llamadas cuánticas, esta implementación integró un simulador cuántico –específicamente *Qiskit Aer*– directamente en el entorno de ejecución SYCL.

Este paso fundacional fue crucial para demostrar la Orquestación CPU-QPU. Al habilitar un backend de simulación dentro de SYCL, se hizo posible escribir un único fichero fuente en C++ que:

1. Asigna datos en la CPU anfitriona.

2. Delega tareas paralelas clásicas (e.g., pre-procesamiento de matrices) a una cola de GPU.

3. Delega tareas cuánticas (e.g., ejecución de circuitos) a la cola de la QPU (simulada vía Aer).

4. Sincroniza los resultados a través de accesores de memoria unificada.

Esta integración confirmó empíricamente que el flujo de trabajo DQC es compatible con los estándares modernos de programación heterogénea. Demostró que la barrera entre código clásico y cuántico es artificial y puede eliminarse mediante un sistema de ejecución unificado. Aunque el backend utilizado fue un simulador (Qiskit Aer), los patrones arquitectónicos establecidos –colas, búferes y grupos de comandos– son idénticos a los requeridos para hardware cuántico físico. Este logro abre el camino para futuros sistemas donde un planificador asigna dinámicamente tareas a CPUs, GPUs o QPUs distribuidas basándose en la naturaleza de la carga de trabajo, gestionado enteramente a través de una interfaz de software unificada.

Para demostrar la utilidad práctica de la pila de software propuesta más allá de la simple tecnología habilitadora, esta tesis abordó un desafío algorítmico concreto: la optimización de la iQFT Distribuida (Objetivo O6). La selección de la iQFT como caso de estudio es estratégica; constituye la columna vertebral computacional de subrutinas cuánticas fundamentales, como la QPE y el algoritmo de Shor. No obstante, desde una perspectiva de arquitectura distribuida, la iQFT representa el "peor escenario posible" debido a sus rigurosos requisitos de conectividad. En su representación canónica, cada qubit debe interactuar con todos los demás qubits, lo que implica una topología de conectividad todos-con-todos cuya escalabilidad resulta deficiente al extenderse a arquitecturas distribuidas basadas en múltiples nodos interconectados en red.

En un escenario distribuido, una implementación ingenua de la iQFT incurriría en un coste de comunicación cuadrático ($\mathcal{O}(N^2)$), ya que cada puerta de fase controlada no local requiere el consumo de un par EPR vía el protocolo Telegate. Esta sobrecarga amenaza con negar la ventaja cuántica. Para mitigar esto, esta tesis propuso una nueva estrategia de optimización arraigada en las propiedades analíticas de la transformada de Fourier.

La idea central es que el ángulo de rotación $\theta_k = \pi/2^k$ de las puertas de fase controlada decae exponencialmente con la distancia lógica k entre qubits. Al analizar la convergencia del error de fase acumulado, se definió un *Horizonte de Comunicación* ($d_{\max}$). Este parámetro establece una frontera física: si dos qubits están separados por una distancia de red mayor que $d_{\max}$, la interacción se poda (se aproxima como el operador identidad).

Utilizando las capacidades de abstracción de la pila de software para modelar estas interacciones, los resultados teóricos demostraron que esta estrategia de podado altera drásticamente el escalado de recursos. Para un umbral de fidelidad fijo, el consumo de pares EPR transita de un crecimiento cuadrático a un régimen de saturación ($\mathcal{O}(1)$) por qubit añadido. Este hallazgo es crítico: prueba que las limitaciones físicas de la red (tasas finitas de generación de entrelazamiento) pueden mitigarse a través de un diseño algorítmico inteligente y consciente de la red. La pila de software sirve como el habilitador que permite al compilador imponer estos horizontes, localizando efectivamente un algoritmo global.

La investigación presentada en esta tesis doctoral abordó uno de los desafíos más urgentes en la hoja de ruta hacia la computación cuántica escalable: la integración de procesadores cuánticos distribuidos en el ecosistema HPC. Comenzando por la identificación de una profunda brecha entre los protocolos físicos de red y las herramientas de desarrollo de software, este trabajo definió, implementó y validó con éxito una pila de software completa y agnóstica del hardware.

La validez de la hipótesis central –que el paradigma SPMD puede abstraer efectivamente la complejidad de las redes cuánticas– se confirmó a través de la consecución de los objetivos estratégicos.

En conclusión, esta tesis postula que la transición a la DQC no es simplemente una actualización de hardware, sino un cambio de paradigma necesario para el escalado continuo de la potencia computacional. Al proporcionar las herramientas para tratar la red cuántica no como un experimento de física, sino como un recurso computacional gestionable, este trabajo sienta la infraestructura de software esencial requerida para transformar dispositivos cuánticos aislados en los supercomputadores cuánticos modulares e interconectados del futuro.

Summary

High-Performance Computing (HPC) has historically served as the primary engine for scientific advancement, enabling complex simulations ranging from meteorological prediction to protein folding or drug discovery. For decades, the exponential growth in computational power was sustained by the empirical axioms of Moore's Law and Dennard scaling. However, in the last decade, these principles have begun to exhibit unambiguous signs of saturation. The fundamental physical, thermodynamic, and quantum limitations of silicon transistors have compelled the industry to abandon frequency scaling in favor of massive parallelism and architectural heterogeneity. Consequently, contemporary top-tier supercomputers are no longer homogeneous machines, but complex clusters that integrate multicore Central Processing Units (CPU) with specialized accelerators such as Graphics Processing Units (GPU) or FPGAs.

Within this landscape of hardware diversification, Quantum Computing (QC) emerges not merely as an isolated disruptive technology, but as the logical next step in the evolution of HPC accelerators. By exploiting the principles of quantum mechanics–specifically superposition and entanglement–the new Quantum Processing Units (QPU) promise to resolve problems that remain intractable for classical systems, offering exponential speedups in domains such as quantum chemistry, combinatorial optimization, and cryptography. However, this computational advancement currently exists merely within a theoretical-mathematical framework; for the existence of quantum computers, an implementation of this theoretical framework into physical devices is necessary. The materialization of this promise faces formidable engineering barriers.

We currently reside in the "Noisy Intermediate-Scale Quantum" (NISQ) era. Existing devices, based on technologies such as superconducting circuits or trapped ions, are limited to a few dozen or hundreds of qubits. Scaling these monolithic processors toward the millions of qubits required for fault tolerance presents extreme physical challenges. Issues such as frequency crowding in superconducting chips, crosstalk between adjacent qubits, heat dissipation within dilution refrigerators, and the complexity of control wiring suggest that the "single giant chip" strategy faces a scalability ceiling that is difficult to surmount in the near term.

To overcome these physical limitations, the scientific community has begun to shift toward a modular paradigm: Distributed Quantum Computing (DQC). This archi-

tecture proposes the interconnection of multiple moderate-sized QPUs via quantum channels (optical links) capable of distributing entanglement. By networking multiple quantum nodes, it becomes possible to construct a unified logical system with a computational capacity that scales linearly with the number of nodes, enabling the execution of algorithms that require a qubit volume unattainable by any single monolithic device.

However, while DQC offers a clear pathway for hardware scaling, it introduces unprecedented complexity at the software layer. Unlike classical clusters, where information can be freely copied and transmitted, communication between quantum nodes is constrained by the "No-Cloning Theorem". The transmission of quantum information necessitates sophisticated protocols such as quantum teleportation (*Teledata*) and remote gate execution (*Telegate*), both of which consume a fungible and expensive resource: EPR pairs.

Here lies the central problem motivating this doctoral thesis: the existence of a profound technological and semantic gap between the quantum network hardware and software development tools. On one hand, research in network physics has achieved significant milestones in the generation and purification of entanglement at the physical layer. On the other hand, the quantum algorithm community continues to design applications assuming an idealized abstraction of monolithic, fully connected hardware. There is an absence of an intermediate "glue": a robust software stack capable of managing the stochastic complexity of the quantum network and presenting it to the programmer in a transparent and efficient manner.

The absence of this software stack compels current researchers to program quantum networks manually, explicitly managing the generation of EPR pairs, Bell measurements, and classical corrections. This low-level approach is error-prone, unscalable, and significantly hinders the integration of distributed quantum resources into current HPC data center workflows.

Consequently, the primary objective of this thesis is to design, develop, and validate a complete, hardware-agnostic software architecture that enables the programming, compilation, and execution of distributed quantum algorithms. This architecture aims to integrate the distributed quantum paradigm within proven HPC standards, specifically by adapting the Message Passing Interface (MPI) model and modern intermediate representations (LLVM and QIR), to definitively bridge the gap between quantum network physics and algorithmic software engineering.

The transition towards DQC is hindered by a fundamental disconnect between the physical layer and the application layer, referred to in this thesis as the "Quantum-HPC Integration Gap". While the theoretical foundations of quantum networking–such as entanglement swapping and purification–are well-established, the software infrastructure required to utilize these resources remains in a nascent stage.

Currently, the development of distributed quantum applications relies heavily on low-level, imperative programming models. Tools such as the NetQASM SDK represent the state of the art for controlling quantum network nodes. However, these tools

expose the full complexity of the hardware to the developer. To implement a simple remote logic gate, a programmer must explicitly request the generation of EPR pairs, manage the stochastic outcomes of Bell state measurements, and manually synchronize classical control signals across the network. This approach forces the algorithm designer to act as a network engineer, inextricably coupling the application logic with the specific topology and physical constraints of the underlying hardware.

This scenario mirrors the early days of classical computing, where programmers were required to manually manage memory registers and physical interconnects. In the context of HPC, this would be akin to programming a supercomputer using assembly language for network sockets rather than utilizing high-level abstractions like MPI. This lack of abstraction not only increases the error rate but also renders code non-portable; an algorithm optimized for a specific network topology (e.g., a linear chain) must be completely rewritten to function on a different architecture (e.g., a star topology). Consequently, the integration of DQC into mainstream supercomputing workflows is stalled by the absence of a standardized, hardware-agnostic software stack.

To address this challenge, this thesis posits the following central hypothesis: *It is possible to define a hardware-agnostic software stack that integrates distributed quantum resources into HPC environments by adopting the "Single Program, Multiple Data" paradigm (commonly known as SPMD), thereby managing the complexity of quantum networks–specifically EPR pair generation and teleportation–transparently to the user.*

The validation of this hypothesis is structured around six strategic objectives:

- **O1 (State of the Art Review):** To conduct a rigorous characterization of the state of the art in quantum compilation and distributed architectures, identifying the specific limitations of current IRs.
- **O2 (Development Layer):** To design and implement a high-level library (NetQMPI) that adapts the classical MPI standard to the quantum domain, enabling the development of distributed algorithms using semantic primitives.
- **O3 (Compilation Layer):** To define a standardized IR (NetQIR) based on the LLVM infrastructure, capable of describing distributed operations and facilitating compiler-level optimizations.
- **O4 (Benchmarking):** To establish a rigorous evaluation framework integrating high-fidelity network simulators (SeQUeNCe, NetSquid) to validate the functional correctness and performance of the proposed stack.
- **O5 (Integration):** To demonstrate the interoperability of the stack with heterogeneous programming models, specifically SYCL, enabling the orchestration of hybrid CPU-QPU workloads.

- **O6 (Application):** To validate the utility of the stack through the implementation and optimization of a concrete algorithm, the Distributed iQFT, introducing novel strategies for communication reduction.

The efficacy of any HPC system is fundamentally predicated on the maturity of its compilation stack. A compiler acts as the critical bridge that translates human-readable abstractions into machine-executable instructions. In the classical domain, this process is famously decoupled into analysis (front-end) and synthesis (back-end) phases, mediated by an IR such as LLVM IR. This modular design allows M programming languages to target N hardware architectures with a complexity of $M + N$.

In the quantum domain, however, the ecosystem has historically suffered from fragmentation. Early frameworks relied on monolithic compilers that translated specific languages (e.g., Qiskit, Cirq) directly to specific hardware backends, resulting in a combinatorial complexity of $M \times N$. While standardization efforts like OpenQASM 2.0 and Quil provided a common interface for monolithic devices, they were designed as linear assembly sequences. These formats describe a quantum circuit as a static list of gates applied to a local register, a model that is fundamentally ill-suited for DQC.

Distributed quantum computing introduces asynchronous events and non deterministic control flow. Operations such as entanglement generation are probabilistic; they may succeed or fail, requiring the control software to wait, retry, or branch based on runtime outcomes. Traditional linear IRs cannot naturally express these constructs without cumbersome extensions. Furthermore, they lack the concept of network topology or remote registers.

The emergence of the QIR, based on the LLVM standard, marked a turning point. The QIR allows quantum primitives to be embedded within a robust, classical control flow graph, inheriting decades of optimization techniques from the classical compiler literature. However, the standard QIR specification was designed with monolithic chips in mind. It lacks the intrinsic instructions necessary to describe network operations, such as remote gate execution or entanglement swapping. This analysis of the state of the art leads to a critical conclusion: for DQC to scale, the software stack must evolve beyond the "local circuit" paradigm. It requires a new intermediate representation that extends the modularity of the QIR with explicit support for distributed semantics.

Addressing the limitations of current IRs identified in the state-of-the-art analysis, the first major contribution of this thesis is the definition and specification of NetQIR. It is proposed as a formal extension of the standard QIR, which in turn is based on the ubiquitous LLVM infrastructure. The strategic decision to build upon LLVM is pivotal: it ensures that the quantum network stack is not an isolated silo but rather an integrated component of the broader open-source compiler ecosystem, benefiting from mature optimization passes and tooling.

The fundamental innovation of NetQIR lies in its elevation of network primitives to intrinsic instructions. In standard QIR, a remote operation is opaque; the compiler

sees only a sequence of local gates and measurement calls. NetQIR introduces explicit semantic instructions for distributed operations. By formalizing these operations within the IR, NetQIR enables the compiler to perform network-aware optimizations. For instance, a compiler pass can now analyze the dependency graph of a distributed algorithm and identify opportunities to pre-fetch entanglement. If the compiler detects that a teleportation is required at instruction N, it can hoist the entanglement request (the `gen_ent` call) to instruction $N - k$, effectively masking the latency of the quantum link generation behind local computation. This "latency hiding" technique, standard in classical HPC but novel in DQC, is only possible because the IR explicitly exposes the network resource semantics.

Furthermore, NetQIR serves as a hardware-agnostic boundary. It acts as a contract between the high-level software and the low-level control hardware. The front-end (high-level language) emits NetQIR without worrying about the specific physical implementation of the quantum link (e.g., nitrogen-vacancy centers vs. trapped ions). The back-end then synthesizes this representation into the specific control pulses or low-level instructions (like NetQASM) required by the target device. This modularity fulfills Objective O3, effectively decoupling the compilation logic from the physical constraints of the network.

While NetQIR solves the problem of machine representation, it does not address the human usability gap. Programming directly in an IR or even in low-level imperative languages like NetQASM remains prohibitively complex for algorithm designers, who must manually handle the intricacies of entanglement lifecycles and classical synchronization. To bridge this gap and fulfill Objective O2, this thesis presents NetQMPI, a high-level Python library designed to bring the SPMD paradigm to DQC.

NetQMPI draws its design philosophy directly from the MPI standard, the *de facto* standard of classical supercomputing. The choice of MPI as a reference model is strategic: it provides a syntax and a mental model that are already familiar to the vast majority of HPC developers, thereby lowering the barrier to entry for the quantum domain.

The architecture of NetQMPI acts as a middleware layer that sits between the application logic and the network controller. Its core innovation is the abstraction of physical protocols into semantic primitives. In a NetQMPI program, the complex "Teledata" protocol (which involves entanglement generation, Bell measurement, classical transmission, and Pauli correction) is encapsulated in a single, blocking function call: `comm.qsend(qubit, dest_rank)`. Similarly, the receiver executes `comm.qrecv(source_rank)`.

Under the hood, the library manages a rigorous state machine. When `qsend` is invoked, NetQMPI:

1. Automatically negotiates the creation of an EPR pair with the destination node.

2. Performs the local Bell basis measurement.

3. Serializes the measurement outcomes.

4. Transmits these classical bits via the classical control plane.

On the receiving end, `qrecv` waits for the classical message and automatically applies the necessary X/Z corrections to reconstruct the quantum state. This automation transforms the network from a complex physical experiment into a reliable computational resource. Crucially, NetQMPI enables the SPMD execution model. A user writes a single script where behavior is determined by the node's `rank`. This script is executed simultaneously across all nodes in the simulation or hardware testbed. The library seamlessly handles the interleaving of classical control flow (Python logic) and quantum instructions, dispatching the latter to the underlying NetQASM backend. This design ensures that NetQMPI is not just a simulator toy but a valid system software stack; the same code running on a laptop simulation can be deployed to a physical quantum network, provided the nodes support the NetQASM instruction set.

The development of a software stack for a nascent technology such as DQC presents a unique challenge: the unavailability of large-scale physical hardware for validation. While small-scale testbeds exist, they lack the volume and stability required for rigorous stress testing of a distributed compilation stack. Consequently, to fulfill Objective O4, this thesis established a comprehensive benchmarking framework rooted in high-fidelity simulation.

This experimental phase served one purpose: it provided a quantitative baseline for the resources required to run distributed algorithms, data that is vital for future hardware co-design.

Distributed quantum computing cannot exist in a vacuum; it must be integrated into the existing HPC ecosystem. Modern supercomputers are heterogeneous, combining multicore CPUs with accelerators like GPUs. Objective O5 of this thesis addressed the challenge of integrating the quantum workflow into this heterogeneous environment using SYCL, the open standard for single-source heterogeneous programming.

The hypothesis driving this section was that DQC resources could be treated abstractly as just another type of accelerator within the SYCL model. To validate this, a custom SYCL backend was developed. Unlike previous approaches that merely wrapped quantum calls, this implementation integrated a quantum simulator–specifically *Qiskit Aer*–directly into the SYCL runtime environment.

This foundational step was crucial for demonstrating CPU-QPU Orchestration. By enabling a simulation backend within SYCL, it became possible to write a single C++ source file that:

1. Allocates data on the host CPU.

2. Offloads classical parallel tasks (e.g., matrix pre-processing) to a GPU queue.

3. Offloads quantum tasks (e.g., circuit execution) to the QPU queue (simulated via Aer).

4. Synchronizes the results via unified memory accessors.

This integration empirically confirmed that the DQC workflow is compatible with modern heterogeneous programming standards. It demonstrated that the barrier between classical and quantum code is artificial and can be removed by a unified runtime system. While the backend used was a simulator (Qiskit Aer), the architectural patterns established–queues, buffers, and command groups–are identical to those required for physical quantum hardware. This achievement paves the way for future systems where a scheduler dynamically assigns tasks to CPUs, GPUs, or distributed QPUs based on the nature of the workload, managed entirely through a unified software interface.

To demonstrate the practical utility of the proposed software stack beyond mere enabling technology, this thesis addressed a concrete algorithmic challenge: the optimization of the Distributed iQFT (Objective O6). The selection of the iQFT as a case study is strategic; it constitutes the computational backbone of pivotal quantum subroutines, such as QPE and Shor's algorithm. However, from a distributed architecture perspective, the iQFT represents the "worst-case scenario" due to its rigorous connectivity requirements. In its canonical formulation, each qubit is required to interact with every other qubit, resulting in an all-to-all connectivity topology whose scalability is severely constrained when extended across distributed or networked quantum nodes.

In a distributed setting, a naive implementation of the iQFT would incur a quadratic communication cost ($\mathcal{O}(N^2)$), as every non-local controlled-phase gate requires the consumption of an EPR pair via the Telegate protocol. This overhead threatens to negate the quantum advantage. To mitigate this, this thesis proposed a novel optimization strategy rooted in the analytic properties of the Fourier transform.

The core insight is that the rotation angle $\theta_k = \pi/2^k$ of the controlled-phase gates decays exponentially with the logical distance k between qubits. By analyzing the convergence of the accumulated phase error, a *Communication Horizon* ($d_{\max}$) was defined. This parameter establishes a physical boundary: if two qubits are separated by a network distance greater than $d_{\max}$, the interaction is pruned (approximated as the identity operator).

Using the abstraction capabilities of the software stack to model these interactions, theoretical results demonstrated that this pruning strategy drastically alters the resource scaling. For a fixed fidelity threshold, the consumption of EPR pairs transitions from a quadratic growth to a saturation regime ($\mathcal{O}(1)$) per added qubit. This finding is critical: it proves that the physical constraints of the network (finite entanglement generation rates) can be mitigated through intelligent, network-aware algorithmic design. The software stack serves as the enabler that allows the compiler to enforce these horizons, effectively localizing a global algorithm.

The research presented in this doctoral thesis has addressed one of the most pressing challenges in the roadmap toward scalable quantum computing: the integration of

distributed quantum processors into the HPC ecosystem. Starting from the identification of a profound gap between physical network protocols and software development tools, this work has successfully defined, implemented, and validated a comprehensive, hardware-agnostic software stack.

The validity of the central hypothesis–that the SPMD paradigm can effectively abstract the complexity of quantum networking–has been confirmed through the achievement of the strategic objectives.

In conclusion, this thesis posits that the transition to DQC is not merely a hardware upgrade, but a necessary paradigm shift for the continued scaling of computational power. By providing the tools to treat the quantum network not as a physics experiment, but as a manageable computational resource, this work lays the essential software infrastructure required to transform isolated quantum devices into the modular, interconnected quantum supercomputers of the future.

Contributions

The research outcomes derived from this doctoral thesis have been extensively disseminated and validated by the scientific community through a rigorous peer-review process. This work has resulted in a total of **11 scientific contributions** and **3 unpublished articles** currently submitted or in the pre-print stage pending peer review.

The academic output achieved during this period is summarized as follows:

- **3 Journal Articles** in high-impact indexed journals, including top-tier Q1, D1 (first decile) and C1 (first centile) rankings such as *Computer Science Review* and *Future Generation Computer Systems*.
- **1 Book Chapter** published in the Lecture Notes in Computer Science (LNCS) series, resulting from a peer-reviewed contribution to an international conference.
- **5 International Conference Contributions:** these works were presented at prestigious conferences venues within the High-Performance Computing domain, such as IEEE Cluster and Euro-Par (ranked in the CORE system), serving as forums for discussion and validation of the proposed methodologies.
- **2 National Conference Contributions:** presented at the "Jornadas de Paralelismo SARTECO" (Parallelism Conference SARTECO) to communicate science related to computer architecture in Spain.

The following sections detail the bibliographic references associated with these contributions, categorized by their publication status and venue type.

Results: published articles

Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing

- **Authors:** David Barral, **F. Javier Cardama**, Guillermo Díaz-Camacho, Daniel Faílde, Iago F. Llovo, Mariamo Mussa-Juane, Jorge Vázquez-Pérez, Juan Villasuso, César Piñeiro, Natalia Costas, Juan C. Pichel, Tomás F. Pena and Andrés Gómez.
- **Journal:** Computer Science Review, Elsevier.
- **Status:** published 2025 (Open Access).
- **DOI:** 10.1016/j.cosrev.2025.100747
- **JCR 2024:**
  - **Impact factor:** 12.7.
  - **Category:** Computer Science, Information Systems.
  - **Rank:** 3/258.
  - **Percentile:** 99,0.
  - **Quartile:** Q1, and first decile (D1), and first centile (C1).
- **SJR 2024:**
  - **Impact factor:** 3.276.
  - **Quartile:** Q1.
- **PhD candidate contribution:** Conceptualization of the work, preparation of the manuscript for sections 1, 4, 5, and 6, revision of the document.
- **Related thesis content in:** Chapter 5.

Review of intermediate representations for quantum computing

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, César Piñeiro, Juan C. Pichel, Tomás F. Pena and Andrés Gómez.
- **Journal:** The Journal of Supercomputing, Springer.
- **Status:** published 2025 (Open Access).
- **DOI:** 10.1007/s11227-024-06892-2
- **JCR 2024:**
  - **Impact factor:** 2.7.

  - **Category:** Computer Science, Theory and Methods.
  - **Rank:** 51/147.
  - **Percentile:** 65,6.
  - **Quartile:** Q2.
- **SJR 2024:**
  - **Impact factor:** 0.716.
  - **Quartile:** Q2.
- **PhD candidate contribution:** Conceptualization, research, revision and preparation of all the manuscript.
- **Related thesis content in:** Chapter 6.

NetQIR: an extension of QIR for Distributed Quantum Computing

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, César Piñeiro, Juan C. Pichel, Tomás F. Pena and Andrés Gómez.
- **Journal:** Future Generation Computer Systems, Elsevier.
- **Status:** published 2025 (Open Access).
- **DOI:** 10.1016/j.future.2025.107989
- **JCR 2024:**
  - **Impact factor:** 6.1.
  - **Category:** Computer Science, Theory and Methods.
  - **Rank:** 15/147.
  - **Percentile:** 90,1.
  - **Quartile:** Q1, and first decile (D1).
- **SJR 2024:**
  - **Impact factor:** 1.551.
  - **Quartile:** Q1.
- **PhD candidate contribution:** Conceptualization, research, design, development, revision and preparation of all the manuscript.
- **Related thesis content in:** Chapter 7.

Quantum Compilation Process: A Survey

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, Tomás F. Pena, Juan C. Pichel, and Andrés Gómez.
- **Journal:** Lecture Notes in Computer Science, Springer.
- **Status:** published 2025 (Open Access).
- **DOI:** 10.1007/978-3-031-90200-0_9
- **SJR 2024:**
  - **Impact factor:** 0.352.
  - **Quartile:** Q2.
- **PhD candidate contribution:** Conceptualization, research, revision and preparation of all the manuscript.
- **Related thesis content in:** Chapter 6.

Results: unpublished articles

NetQMPI: An MPI-Inspired library for programming Distributed Quantum Applications over Quantum Networks using NetQASM SDK

- **Authors: F. Javier Cardama** and Tomás F. Pena.
- **Expected Journal:** IEEE Access, IEEE.
- **Status:** submitted to IEEE Access on 22 December 2025.
- **Pre-print DOI:** 10.48550/arXiv.2512.11483
- **PhD candidate contribution:** Conceptualization, research, design, development, revision and preparation of all the manuscript.
- **Related thesis content in:** Chapter 8.

Benchmarking Distributed Quantum Computing Emulators: Scalability, Fidelity, and Architectural Trade-offs

- **Authors:** Guillermo Díaz-Camacho, Iago F. Llovo, **F. Javier Cardama**, Irais Bautista, Daniel Faílde, Mariamo Mussa Juane, Jorge Vázquez-Pérez, Natalia Costas, Tomás F. Pena and Andrés Gómez.
- **Expected Journal:** Future Generation Computer Systems, Elsevier.
- **Status:** initial version as preprint in Arxiv (2512.01807). Future submission to Future Generation Computer Systems, Elsevier.
- **Pre-print DOI:** 10.48550/arXiv.2512.01807
- **PhD candidate contribution:** Conceptualization, research, revision and preparation of all the manuscript.
- **Related thesis content in:** Chapter 10.

Communication-Efficient Distributed Inverse Quantum Fourier Transform

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, Tomás F. Pena and Andrés Gómez.
- **Expected Journal:** IEEE Transactions on Quantum Engineering, IEEE.

- **Status:** future submission to IEEE Transactions on Quantum Engineering, IEEE.
- **PhD candidate contribution:** Conceptualization, research, design, development, revision and preparation of all the manuscript.
- **Related thesis content in:** Chapter 11.

International conferences

NetQMPI: programming Distributed Quantum Applications over NetQASM using an MPI-like model

- **Authors: F. Javier Cardama** and Tomás F. Pena.
- **Conference:** ICE-10 Quantum Information in Spain.
- **Date and place:** 20-24 October 2025, Valencia, Spain.
- **CORE 2023:** unranked.
- **PhD candidate contribution:** Speaker, conceptualization, research, design, development, revision and preparation of all the work.
- **Related thesis content in:** Chapter 8.

NetQMPI: a practical MPI-inspired library for distributed quantum computing over NetQASM SDK

- **Authors: F. Javier Cardama** and Tomás F. Pena.
- **Conference:** IEEE International Conference on Cluster Computing (IEEE Cluster).
- **Date and place:** 3-8 September 2025, Edinburgh, Scotland.
- **Proceedings DOI:** 10.1109/CLUSTERWorkshops65972.2025.11164201
- **CORE 2023:** B.
- **Field of Research:** 4606.
- **PhD candidate contribution:** Speaker, conceptualization, research, design, development, revision and preparation of all the work.
- **Related thesis content in:** Chapter 8.

SYCL QPU: an LLVM-based QPU simulation framework built using DPC++

- **Authors:** Miguel Leal, **F. Javier Cardama** and Tomás F. Pena.
- **Conference:** IEEE International Conference on Cluster Computing (IEEE Cluster).

- **Date and place:** 3-8 September 2025, Edinburgh, Scotland.
- **Proceedings DOI:** 10.1109/CLUSTERWorkshops65972.2025.11164192
- **CORE 2023:** B.
- **Field of Research:** 4606.
- **PhD candidate contribution:** Speaker, conceptualization, research, revision and preparation of all the work.
- **Related thesis content in:** Chapter 9.

Quantum Process Compilation: a Survey

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, Tomás F. Pena, Juan C. Pichel and Andrés Gómez.
- **Conference:** International European Conference on Parallel and Distributed Computing (Euro-Par).
- **Date and place:** 25-28 August 2024, Madrid, Spain.
- **CORE 2023:** B.
- **Field of Research:** 4606.
- **PhD candidate contribution:** Speaker, conceptualization, research, revision and preparation of all the work.
- **Related thesis content in:** Chapter 6.

Review of Intermediate Representations for Quantum Computing

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, César Piñeiro, Juan C. Pichel, Tomás F. Pena, and Andrés Gómez.
- **Conference:** International Conference Computational and Mathematical Methods in Science and Engineering.
- **Date and place:** 2-8 July 2024, Rota, Cádiz, Spain.
- **CORE 2023:** unranked.
- **PhD candidate contribution:** Speaker, conceptualization, research, revision and preparation of all the work.
- **Related thesis content in:** Chapter 6.

National conferences

NetQIR: diseño de una representación intermedia para la computación cuántica distribuida

- **Authors: F. Javier Cardama**, Jorge Vázquez-Pérez, César Piñeiro, Juan C. Pichel, Tomás F. Pena, and Andrés Gómez.
- **Conference:** XXXV Jornadas de Paralelismo, Sociedad de Arquitectura y Tecnología de Computadores (SARTECO).
- **Date and place:** 25-27 June 2025, Sevilla, Spain.
- **PhD candidate contribution:** Speaker, conceptualization, research, design, development, revision and preparation of all the work.
- **Related thesis content in:** Chapter 7.

Optimización del proceso de compilación de códigos cuánticos a través de la herramienta Qat

- **Authors:** Daniela Cadilla Carballeda, **F. Javier Cardama**, Jorge Vázquez-Pérez, Tomás F. Pena, Andrés Gómez, Juan C. Pichel and César Piñeiro.
- **Conference:** XXXV Jornadas de Paralelismo, Sociedad de Arquitectura y Tecnología de Computadores (SARTECO).
- **Date and place:** 25-27 June 2025, Sevilla, Spain.
- **PhD candidate contribution:** Conceptualization, research, revision and preparation of all the work.
- **Related thesis content in:** Chapter 6.

Chapter 1

Introduction

A century after the formal consolidation of quantum mechanics as a physical theory [1–3], its mathematical foundations are increasingly emerging as a viable framework for computation [4, 5]. Unlike classical computing models, which are grounded in Boolean logic and deterministic state evolution, the quantum mathematical formalism exploits linear algebra, superposition, and entanglement to restructure the way information is represented and processed [6]. This shift is not merely conceptual: for specific problem classes, quantum algorithms can reduce computational complexity from exponential to polynomial, or from polynomial to sub-polynomial, compared to the best-known classical counterparts [7–9]. For decades, such advantages remained largely theoretical; however, recent advances in experimental physics and engineering have led to the development of programmable quantum hardware capable of instantiating this abstract model, thus transforming quantum computation from a purely mathematical curiosity into an emerging computing paradigm [10, 11].

The paradigm of Distributed Quantum Computing (DQC) inherently arises from the intersection of two distinct disciplines. To rigorously address this subject, it is requisite to first introduce the principles of distributed computing from a classical perspective, and subsequently, to establish the foundations of quantum information processing. Accordingly, this introduction serves as the thesis's starting point, addressing the research problem from these two complementary perspectives. First, it grounds the discussion in the context of classical high-performance and distributed computing (Section 1.1), revisiting fundamental concepts such as parallelism, system architecture, memory hierarchy, and the evolution from single-core processors to large-scale heterogeneous systems. Second, it approaches the problem from the field of quantum computing (Section 1.2), introducing the quantum computational model, its mathematical foundations, and its physical realization. Together, these sections establish the conceptual bridge required to reason about the integration of quantum computing within modern High-Performance Computing (HPC) environments, covering foundational topics ranging from the classical notion of memory locality and

communication costs to the postulates of quantum mechanics.

Building upon this technical background, Chapter 2 formally states the hypotheses and objectives that guide this doctoral research, clearly delimiting the scientific questions addressed throughout the thesis. These objectives are subsequently operationalized in Chapter 3, which outlines a general methodology.

Chapter 4 provides a global discussion of the work developed in this thesis, articulating a unified perspective that connects the individual contributions presented in the subsequent chapters. This chapter serves as a conceptual backbone, identifying common themes, open challenges, and design principles, and establishing a coherent narrative that links the diverse technical results into a single research thread.

Chapters 5 to 11 constitute the core of the thesis and address the stated objectives from complementary angles. Chapter 5 introduces the DQC paradigm, motivating the transition from isolated Quantum Processing Units (QPUs) to networked quantum systems integrated within HPC infrastructures. Chapter 6 analyzes the quantum compilation process, with particular emphasis on Intermediate Representation (IR) and their role in enabling portability, optimization, and interoperability across heterogeneous backends. Chapter 7 presents NetQIR, an extension of the Quantum Intermediate Representation (QIR) tailored to DQC. Chapter 8 introduces NetQMPI, an MPI-inspired programming model and library for the development of distributed quantum applications over quantum networks. Chapter 9 explores the simulation of QPUs within classical HPC systems through the *SYCL QPU* framework, leveraging modern heterogeneous programming models. Chapter 10 focuses on benchmarking methodologies for DQC emulators, analyzing scalability, fidelity, and architectural trade-offs. Finally, Chapter 11 applies the proposed abstractions and tools to a concrete case study, namely a communication-efficient distributed implementation of the inverse Quantum Fourier Transform (QFT), illustrating the practical implications of the thesis contributions.

1.1 On Classical Computing: From Single-Core to Distributed Systems

The evolution of computational capacity over the last five decades has closely followed the empirical observation formulated by Gordon Moore in 1965. Known as *Moore's Law*, it predicted that the number of transistors in a dense integrated circuit would double approximately every two years. For decades, this miniaturization provided computer engineers with a "free lunch" [12] regarding performance: with each new generation of processors, transistors became smaller, faster, and more energy-efficient.

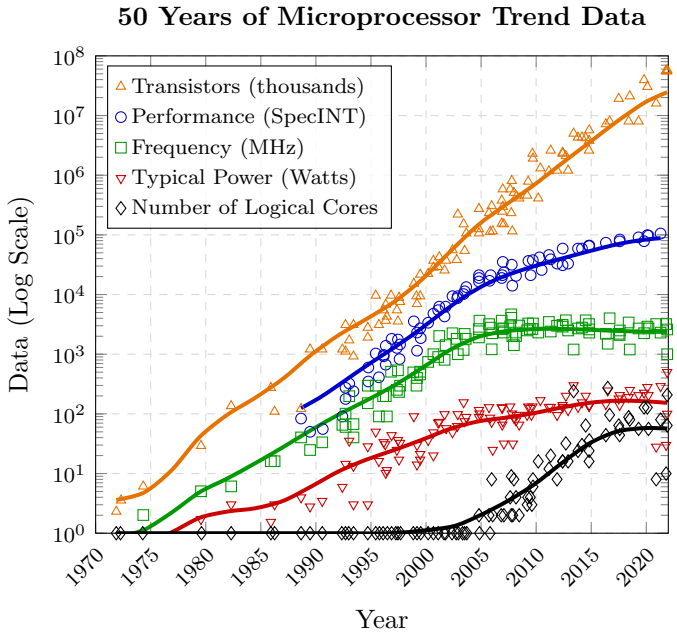


Figure 1.1: Evolution of key microprocessor metrics (1970–2021). The plot displays transistor count, single-thread performance, frequency, power, and logical cores. Adapted from [13, 14].

However, as illustrated in Figure 1.1, this historically sustained trend began to deteriorate in the mid-2000s. The plot, derived from historical microprocessor data [13, 14], shows a pronounced decoupling in clock frequency scaling, which exhibits a marked saturation around 2005. In addition, the growth rate of the transistor count is beginning to flatten, suggesting that the era of continuous device miniaturization is approaching its physical limits.

This phenomenon is driven by two fundamental physical barriers:

1. **The Power Wall:** Historically, *Dennard scaling* postulated that as transistors

got smaller, their power density (understood as the power per unit area of the chip) stayed constant [15]. This breakdown occurred when transistors became so small that leakage current became a dominant factor [16]. Increasing the clock frequency resulted in a cubic increase in power consumption, generating heat that the chip packaging could not dissipate [17, 18]. Consequently, the industry was forced to cap clock frequencies, which nowadays rarely exceed 4–5 GHz [19, 20].

2. **The Atomic Limit:** Modern fabrication processes are reaching absolute physical limits, currently working at scales of 5 nanometers (nm) or below.

   At these ultra-scaled dimensions, electron transport can no longer be described adequately within a purely classical framework. As the gate-oxide thickness is reduced, quantum-mechanical phenomena, such as *quantum tunneling*, become increasingly dominant [21]. Foundational studies have shown that when the thickness of the insulating barrier falls below approximately 2 nm, electrons can tunnel through the gate with a non-negligible probability [22]. The resulting leakage currents pose a substantial challenge to maintaining robust binary logic levels without incurring prohibitive power dissipation.

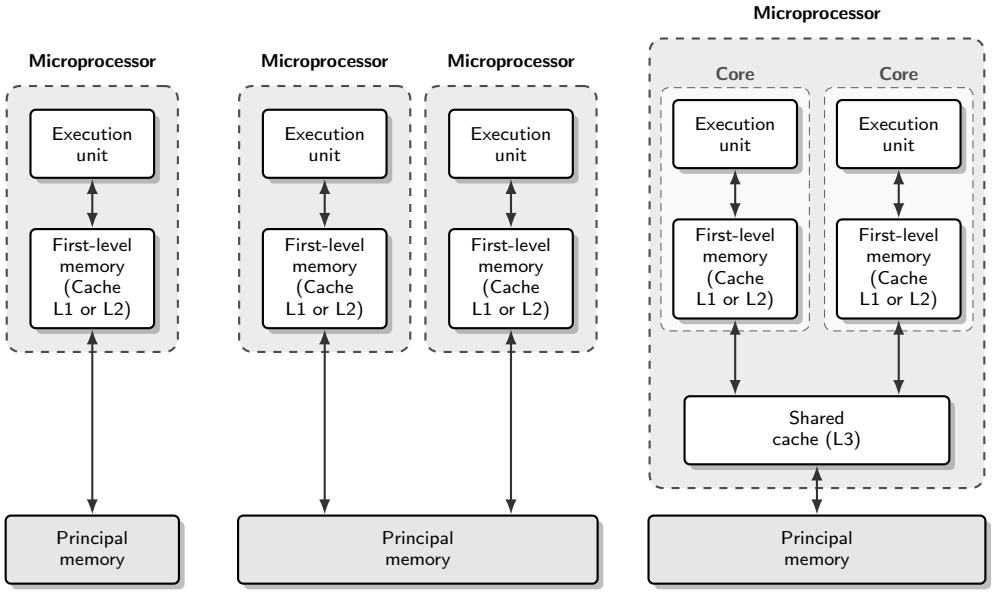
1.1.1 The paradigm shift: the multi-core era

Unable to increase performance by simply raising the frequency, the semiconductor industry made a pivotal shift in 2005: instead of making a single processor faster, they integrated multiple processors onto the same chip. This marked the beginning of the multi-core era, as shown by the black curve (diamond) in Figure 1.1.

To understand the implications of this shift for HPC, we must analyze the architectural evolution depicted in Figure 1.2.

Figure 1.2a illustrates the traditional design that dominated the first decades of computing. In a single-core processor, the hardware resources are centralized. The Central Processing Unit (CPU) contains a single Arithmetical Logical Unit (ALU) and a single control unit. Consequently, execution is strictly sequential at the hardware level. Even if the operating system provides the illusion of multitasking, the processor can only execute one instruction at a time [23]. If the ALU is busy calculating a floating-point operation, no other computation can proceed until it finishes. This architecture is limited by its single pipeline's physical speed. However, prior to the multicore era, performance was significantly enhanced by incorporating Instruction Level Parallelism (ILP). Architectures such as superscalar, vector, and Very Long Instruction Word (VLIW) introduced multiple execution pipelines, enabling the processor to issue and execute more than one instruction per clock cycle [24].

To overcome the limits of a single core, the initial approach was to duplicate the entire physical package, as shown in Figure 1.2b. In a multi-processor, the motherboard hosts two or more discrete CPUs. In this configuration, nothing is shared at



(a) Single-core. **(b)** Multi-processor. **(c)** Multi-core.

Figure 1.2: Architectural designs from single to multi-core. (a) Traditional single core architecture. (b) Multi-processor system, where discrete CPUs communicate via the system bus. (c) Multi-core processor, where cores are integrated on the same die.

the chip level. Each processor has its own independent cache hierarchy, control units, and interface to the main memory. While this doubles the theoretical computing power, communication between processors is inefficient. Data exchange must traverse the external system bus on the motherboard, which introduces significant latency and limits scalability [24]. Although specialized architectures, such as the IMS T800 Transputer [25] and the nCube system [26], incorporated direct inter-processor interconnects to mitigate this bottleneck, the predominant class of commercial multiprocessor systems relied on shared-memory communication implemented over a common system bus.

The modern standard, multi-core processors, is depicted in Figure 1.2c. Instead of duplicating the physical package, manufacturers integrate multiple execution units, now called cores, onto a single silicon die. In this design, each core retains its independence as an execution unit (possessing its own private L1 and L2 caches, ALUs, and registers) but shares the rest of the CPU's infrastructure. Crucially, cores often share a Last Level Cache (L3), allowing them to exchange data at on-chip speeds without accessing the slower main memory. This integration drastically reduces communication latency and improves energy efficiency [27, 28].

The shift from strictly sequential single-core architectures to multi-core designs constitutes a fundamental paradigm change in the execution model of software systems. Contemporary hardware platforms provide the capability to perform multiple computations concurrently; however, this potential is not utilized automatically. Instead, it requires the use of parallel programming techniques, in which algorithms must be explicitly decomposed into concurrent threads or processes to exploit the inherent hardware parallelism. The concepts of parallel programming, as well as the notions of processes and threads, will be elaborated in greater detail in the subsequent subsections.

1.1.2 The memory hierarchy and the memory wall

From an abstract, theoretical standpoint, an ideal computer memory system would possess unbounded capacity, zero access latency, and negligible cost. In practice, however, fundamental physical and economic constraints impose a pronounced trade-off: high-speed memory technologies tend to be expensive and often volatile, whereas low-cost memory technologies exhibit comparatively high access latencies. To approximate the performance of this unattainable ideal, contemporary computer architectures adopt a *memory hierarchy* [23, 24], in which multiple levels of memory with differing cost, capacity, and speed characteristics are organized to optimize overall system efficiency.

The different memory levels are organized according to their proximity to the CPU: faster, lower-capacity, and more costly per unit of storage are positioned closer to the processing unit, whereas slower, higher-capacity, and technologies with superior cost efficiency are situated at greater distances. The primary objective of this hierarchical organization is to present the processor with an effective memory subsystem that exhibits the latency characteristics of the highest level of the hierarchy while providing a capacity comparable to that of the lowest level. This arrangement exploits the *principle of locality*, encompassing both temporal and spatial locality, under the assumption that programs tend to access only a relatively small subset of their total address space during any given interval of execution.

The memory hierarchy is conventionally partitioned into the next principal levels:

- **Registers:** Residing directly within the processor core, registers constitute the fastest storage medium in the system, typically accessible within a single clock cycle. However, their capacity is severely constrained, limiting the volume of data they can hold simultaneously.
- **Cache Memory:** Implemented as small, high-speed memory banks on the processor die (commonly organized into L1, L2, and L3 levels), caches maintain copies of frequently accessed data and instructions. Their primary function is to reduce the frequency and latency of accesses to the main memory by exploiting locality of reference.

Memory technology	Typical access time	$ per GB in 2012
SRAM semiconductor mem.	0.5-2.5 ns	$500-$1000
DRAM semiconductor mem.	50-70 ns	$10-$20
Flash semiconductor memory	5 000-5 0000 ns	$0.75-$1.00
Magnetic disk	5 000 000-20 000 000 ns	$0.05-$0.10

Table 1.1: Representative characteristics of the principal technologies employed within the memory hierarchy. The data illustrate the inverse correlation between access latency and cost per gigabyte, thereby substantiating the rationale for a multi-level memory organization. Data obtained from [23].

- **Main Memory (RAM):** Serving as the primary working storage for executing programs, main memory offers substantially greater capacity than cache memories. Nonetheless, it is typically located off-chip from the processor, resulting in considerably higher access latency and lower bandwidth than on-die caches.

- **Secondary Storage (Non-Volatile):** This category includes persistent storage devices such as Solid State Drives (SSDs) and Hard Disk Drives (HDDs). These devices provide orders of magnitude greater storage capacity and ensure data persistence across power cycles, but exhibit access latencies and throughput characteristics that are significantly inferior to those of Random Access Memory (RAM).

The performance gap between these levels is primarily determined by the characteristics of the underlying physical memory technologies. As summarized in Table 1.1, the uppermost levels of the hierarchy employ Static Random Access Memory (SRAM), which provides access latencies sufficiently low to match the CPU's operating speed, but at the cost of high manufacturing complexity and relatively low storage density. The main memory subsystem is typically implemented using Dynamic Random Access Memory (DRAM), which offers substantially higher density and lower cost ($10–$20/GB), though with markedly increased access latency (on the order of 50–70 ns) attributable to the need for periodic capacitor refresh operations. At the lowest level, persistent storage technologies such as flash memory and magnetic disks achieve the minimal cost per unit capacity (<$1/GB), but incur access latencies that are several orders of magnitude greater (ranging from microseconds to milliseconds) [23].

Understanding this hierarchy is critical because processor performance has historically outpaced that of memory. This diverging trend, illustrated in Figure 1.3, has created a bottleneck known as the memory wall. Since 1980, CPU performance has improved exponentially (Figure 1.1), while memory latency has improved at a much slower rate.

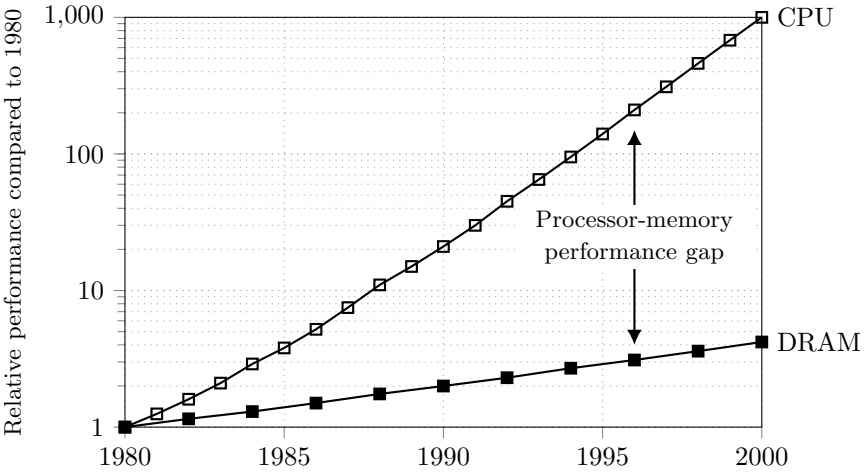


Figure 1.3: Processor–Memory Performance Gap. The figure depicts the growing disparity between the near-exponential growth in processor performance and the relatively modest reductions in memory latency observed since approximately 1980. This divergence gives rise to a fundamental system bottleneck commonly referred to as the *Memory Wall*. Adapted from [29].

1.1.3 Operating systems

Whereas the hardware architecture provides the underlying computational capabilities, it is the Operating System (OS) that orchestrates these resources and renders the system operational for end users and applications. Conceptually, the OS constitutes an intermediary software layer positioned between the physical hardware and user-level programs (see Figure 1.4), with two principal objectives: the efficient management and coordination of computational resources, and the provision of a hardware abstraction layer that conceals low-level implementation details and exposes a uniform higher-level interface [30].

First, as a resource manager, the OS is responsible for multiplexing finite hardware resources, such as CPU time, memory capacity, and Input/Output (I/O) bandwidth, across concurrently executing applications. Its primary objectives are to maximize overall system utilization and to ensure fairness, while resolving contention when multiple processes attempt to access the same resource. Second, the OS provides an abstraction layer that hides the complexity of the underlying hardware components [30]. Rather than requiring programmers to manipulate disk sectors or memory chips directly, the OS exposes a set of high-level, standardized interfaces (system calls) that facilitate safer and more efficient interaction with the system.

To achieve this, the OS defines three fundamental abstractions that map directly onto the primary hardware subsystems: the *process*, which encapsulates the execution units (typically one or more threads of control and their associated state); the *virtual*

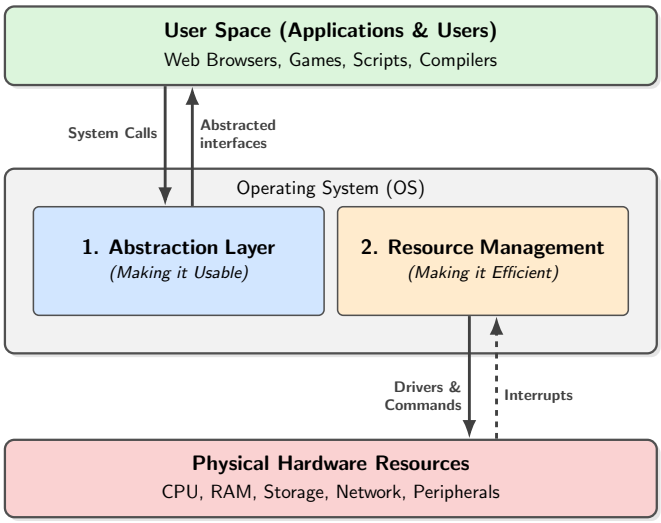


Figure 1.4: High-level overview of the Operating System (OS)'s role in a computer system. The OS acts as an intermediary abstraction layer between the underlying physical hardware and the user space (i.e., application software), and fulfills two principal responsibilities: (1) exposing a well-defined, high-level programming interface to system services via system calls, thereby providing an abstracted and consistent view of the hardware, and (2) orchestrating and optimizing the utilization of hardware resources through mechanisms such as device drivers, interrupt handling, and related resource management policies.

memory, which provides an abstract view of the main memory; and the *files or sockets*, which serve as the principal abstractions for I/O operations.

The most fundamental abstraction for execution in an operating system is the *process*. Formally, a process is defined as an instance of a computer program during its execution [31]. Whereas the program itself constitutes a passive sequence of instructions residing in persistent storage, the process represents the corresponding active computational entity that possesses a program counter, several processor registers, and an associated address space containing variables and other runtime data structures.

In contemporary computing systems, it is common for hundreds of processes to coexist simultaneously. Given that the number of physical processor cores is inherently limited, these processes must contend for available CPU resources. The OS scheduler mediates this contention and provides the illusion of parallel execution by performing rapid context switches, i.e., frequently reassigning the CPU from one process to another and allocating to each process a finite time quantum (time slice) for execution.

However, the execution unit within a process can be further subdivided. This leads to the distinction between *processes* and *threads* [30]:

- **Processes** are isolated execution units. Each process has its own private virtual address space, ensuring spatial isolation. As a result, the failure or abnormal termination of a single process does not directly compromise the execution of other processes. Interaction between processes, therefore, necessitates explicit and comparatively costly communication mechanisms, typically implemented via Inter-Process Communication (IPC). Processes *compete* with each other to obtain the execution unit.

- **Threads** are "lightweight" execution units that operate *within* the context of a single process. In contrast to processes, threads associated with the same process share a common virtual address space and system resources (such as open file descriptors). This architectural property enables very low-latency inter-thread communication and data exchange without requiring explicit OS mediation; however, it also entails that a memory fault in one thread can compromise the integrity of the entire process. Threads *collaborate* with each other.

Complementing the process is the abstraction of *Virtual Memory*. Just as the process creates the illusion that a program has exclusive use of the CPU, virtual memory creates the illusion that the process has a large, contiguous, and private block of main memory (RAM), regardless of the actual physical memory available [32]. The OS translates every "virtual" address generated by the program into a physical address in the hardware. This mechanism provides isolation and safety: a process cannot accidentally write into the memory of another process or the OS kernel, preventing system-wide instability.

Finally, the OS abstracts I/O devices through a uniform interface, typically modeling them as files or streams of bytes. Whether writing to a HDD, SSD, a terminal, or a network socket, the OS handles the specific low-level protocols (interrupts, drivers, and buffers), allowing the application to simply read or write data. While I/O operations are orders of magnitude slower than CPU or memory operations, this abstraction is essential for distributed computing, where data must be reliably transmitted across network interfaces.

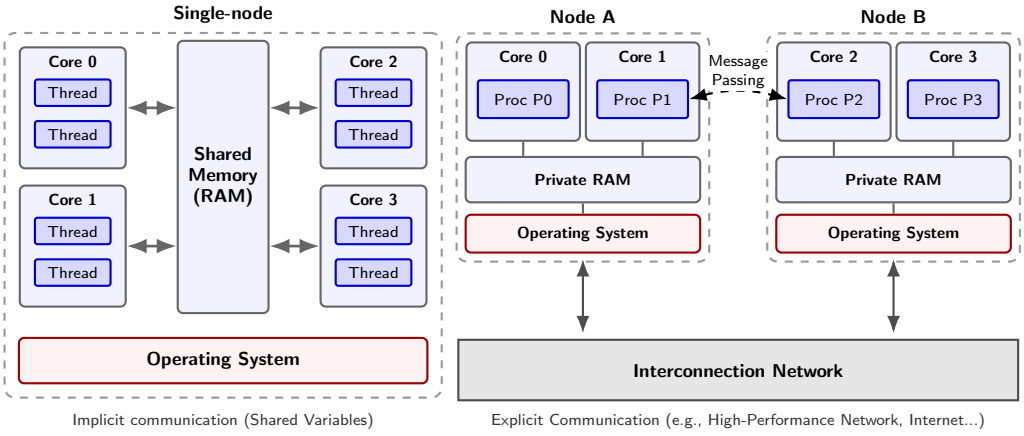
Table 1.2 presents an overview of these fundamental abstractions together with their associated hardware resources.

1.1.4 Parallel and distributed programming

The hardware and OS abstractions presented in the preceding sections constitute the foundational infrastructure for HPC. Nevertheless, the hardware's computational capabilities cannot be fully exploited without an appropriate software model to coordinate and manage concurrent operations. Depending on the characteristics of the underlying memory architecture (specifically, whether it is shared or distributed), two distinct programming paradigms exist, as depicted in Figure 1.5 [33].

Hardware Resource	OS Abstraction	Key Function & Benefit
Processor (CPU)	Process / Thread	**Virtualization of Execution.** Creates the illusion of exclusive CPU use via time-sharing. Processes ensure isolation, while threads enable lightweight concurrency within a shared memory space.
Main Memory (RAM)	Virtual Memory	**Isolation & Capacity.** Decouples logical addresses from physical addresses. Provides each process with a private, contiguous address space and prevents unauthorized access to other processes data.
I/O Devices	Files & Sockets	**Uniform Interface.** Abstracts the complexity of device drivers and protocols into a standardized stream of bytes (read/write), facilitating data persistence and network comm.

Table 1.2: Summary of the principal abstractions provided by the Operating System. The OS virtualizes and encapsulates physical hardware resources into higher-level logical entities that are more tractable to manage, provide stronger safety guarantees, and are mutually isolated.



(a) Intra-node parallelism (Shared Memory) **(b)** Inter-node parallelism (Distributed Memory)

Figure 1.5: Comparison between shared memory and distributed memory architectures.

In the shared-memory programming model, corresponding to the intra-node architecture depicted in Figure 1.5a, multiple processing units (cores) share a single, unified physical address space. In this environment, the natural unit of parallelism is the *thread*. Since all threads belonging to the same process can access the same variables in the main memory (RAM), inter-thread communication is *implicit* and can be realized with low latency (on the order of nanoseconds).

This performance advantage, however, entails a substantial architectural overhead in the form of *cache coherence*. To prevent a core from reading obsolete data while another core is updating it, the processor must implement sophisticated hardware coherence protocols (e.g., MESI) to maintain consistency across the private caches of all cores [34]. As the number of cores increases, the volume of coherence-related traffic grows nonlinearly and can eventually saturate the shared interconnect (e.g., the system bus). As a result, the scalability of shared-memory architectures is fundamentally constrained [24].

The *de facto* standard for programming shared-memory parallel systems is *OpenMP*, which employs compiler directives (`#pragma omp`) to abstract low-level thread management [35]. OpenMP adheres to the fork–join execution model: program execution starts with a single "master" thread, and additional threads are created (forked) only when this thread encounters an OpenMP parallel construct (a `#pragma`). Listing 1.1 exemplifies this behavior. The directive `#pragma omp parallel for` partitions the iterations of the loop across the available threads. The variable `total_sum` is, by default, a shared variable; however, to avoid race conditions[1] without explicit locking, the `reduction(+:total_sum)` clause instructs the compiler to generate a private copy of this variable for each thread and to combine these private values into the global shared variable upon completion of the parallel region.

[1] A race condition occurs when multiple threads concurrently access a shared resource and at least one access is a write operation. In the absence of appropriate synchronization mechanisms (e.g., locks or atomic operations), the final state of the system depends on the non-deterministic interleaving of thread executions, which can lead to incorrect program behavior.

```cpp
#include <omp.h>
#include <vector>

void parallel_sum(const std::vector<int>& data) {
    long total_sum = 0; // Shared variable target for reduction

    // Execution is sequential up to this point (Master thread).
    // The pragma forks a team of threads to execute the loop.
    #pragma omp parallel for reduction(+:total_sum)
    for (size_t i = 0; i < data.size(); ++i) {
        total_sum += data[i];
    }
    // Threads join back to the master thread here.
}
```

Listing 1.1: OpenMP implementation illustrating the Shared Memory model. The execution follows the fork-join pattern: a master thread spawns a team of threads at the directive, which concurrently access the shared `data` array. The `reduction` clause handles the race-free accumulation of results into a single variable.

To overcome the limited scalability inherent to a single compute node, HPC systems commonly adopt a distributed-memory architecture (Figure 1.5b). In such configurations, the system is composed of multiple autonomous nodes interconnected via an *interconnection network*. Each node maintains a private address space, and no hardware-enforced global memory coherence is provided across nodes. Consequently, remote data access must be explicitly performed via communication mechanisms, such as message passing or Remote Memory Access (RMA) operations. While these interactions incur higher latency than intra-node memory accesses, the distributed-memory model scales effectively by eliminating the need for a globally coherent cache hierarchy [36].

The de facto standard for the message passing paradigm is Message Passing Interface (MPI) [37]. Unlike the fork-join model of OpenMP, MPI programs typically follow the Single Program Multiple Data (SPMD) model. This means that if an MPI application is launched with N processes, a runtime system (e.g., `mpirun` or `mpiexec`) initiates N independent instances of the same executable. Each process owns a private address space and is identified by a unique integer identifier, known as the *rank*.

Listing 1.2 illustrates an example of explicit communication using MPI with the point-to-point directives. The `MPI_Comm_rank` function allows a process to determine its rank within a *communicator* (a defined group of processes, where `MPI_COMM_WORLD` encompasses all of them). Data exchange is performed using primitives like `MPI_Send` and `MPI_Recv`. The fundamental communication mechanisms in MPI are point-to-point operations between pairs of processes. MPI provides both blocking and non-blocking communication routines. For example, the blocking receive operation `MPI_Recv` suspends the calling process until a matching message has been received and the receive buffer can be safely accessed. Depending on the send mode employed, this operation may introduce synchronization between the participating processes.

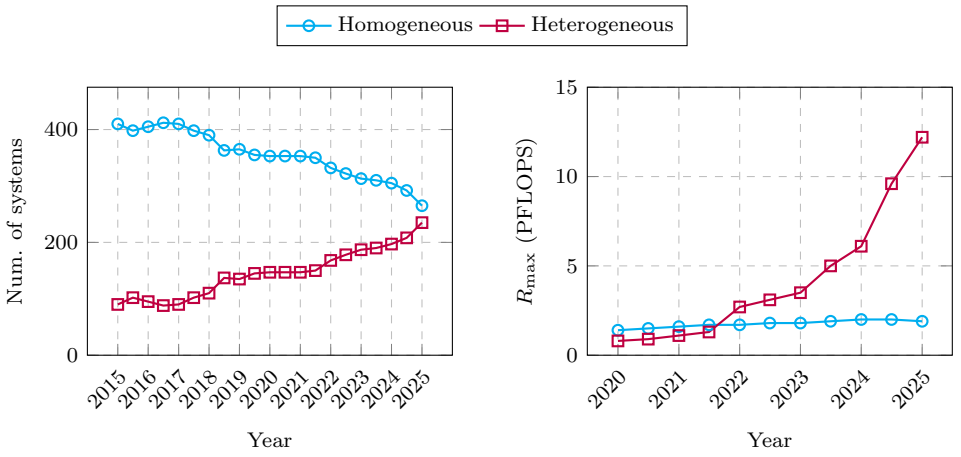
```cpp
#include <mpi.h>
#include <iostream>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    int rank;
    // Determine the unique ID (rank) of this process in the global group
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int data;

    if (rank == 0) {
        data = 42;
        // Blocking send: transfers 'data' to process with rank 1
        MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if (rank == 1) {
        // Blocking receive: waits until data arrives from rank 0
        MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    // Finalize the MPI environment
    MPI_Finalize();
}
```

Listing 1.2: MPI implementation illustrating the Distributed Memory model. Following the SPMD paradigm, strictly isolated processes execute the same code. Communication is performed explicitly via blocking network messages (`MPI_Send`/`MPI_Recv`), using the process `rank` to direct the control flow.

Finally, modern supercomputers typically combine both architectures. This approach, known as *hybrid programming*, uses MPI to manage coarse-grained communication across network nodes, while employing OpenMP within each node to efficiently exploit local shared-memory cores [38].

1.1.5 Heterogeneous devices

While multi-core CPUs and distributed-memory architectures provide a scalable substrate for general-purpose computation, they exhibit inherent limitations in both energy efficiency and achievable throughput for particular classes of workloads [12]. In contrast, Graphics Processing Units (GPUs) are fundamentally throughput-oriented devices, optimized for complex logic and massive parallelism rather than serial execution latency. The stagnation of frequency scaling, illustrated in Figure 1.1, has catalyzed a shift toward heterogeneous computing. Within this paradigm, a compute node is no longer composed exclusively of general-purpose processors; instead, it incorporates specialized hardware accelerators, most prominently GPUs and Field-Programmable Gate Arrays (FPGAs), explicitly engineered to offload and efficiently process massively data-parallel tasks [39].



(a) Number of systems.

(b) Performance $R_{\max}$.

Figure 1.6: Evolution of architectural trends in the Top500 list, contrasting conventional homogeneous computing systems with heterogeneous, accelerator-based platforms. The plots depict the increasing prevalence of heterogeneous architectures in terms of system count, as well as the widening performance differential, quantified by $R_{\max}$, between these two architectural paradigms. Data extracted from [40].

The consequences of this architectural transition are quantitatively observable in the historical progression of supercomputing. Figure 1.6 examines the adoption of heterogeneous systems within the Top500 list [41]. While homogeneous CPU-only systems continue to constitute a substantial fraction of the total system count, their performance evolution shows markedly divergent behavior. The aggregated computing capability ($R_{\max}$) of heterogeneous systems has increased substantially, surpassing that of homogeneous architectures. These data collectively indicate that achieving exascale performance critically depends on the specialized, large-scale parallelism afforded by hardware accelerators.

However, exploiting these architectures introduces significant software complexity, requiring appropriate programming models to manage heterogeneity. Historically, several paradigms have emerged to address this challenge. At the hardware-specific level, vendor-proprietary Application Programming Interfaces (APIs) such as Compute Unified Device Architecture (CUDA) (for NVIDIA GPUs) and Heterogeneous-compute Interface for Portability (HIP) (for AMD GPUs) offer low-level control and maximum performance but bind the application to specific hardware vendors [42]. To improve developer productivity and legacy code adaptation, directive-based models like OpenMP (specifically its target offloading features) and OpenACC allow for the annotation of code regions to be executed on accelerators. Furthermore, task-based runtime systems such as StarPU [43] or OmpSs [44] provide dynamic scheduling across heterogeneous

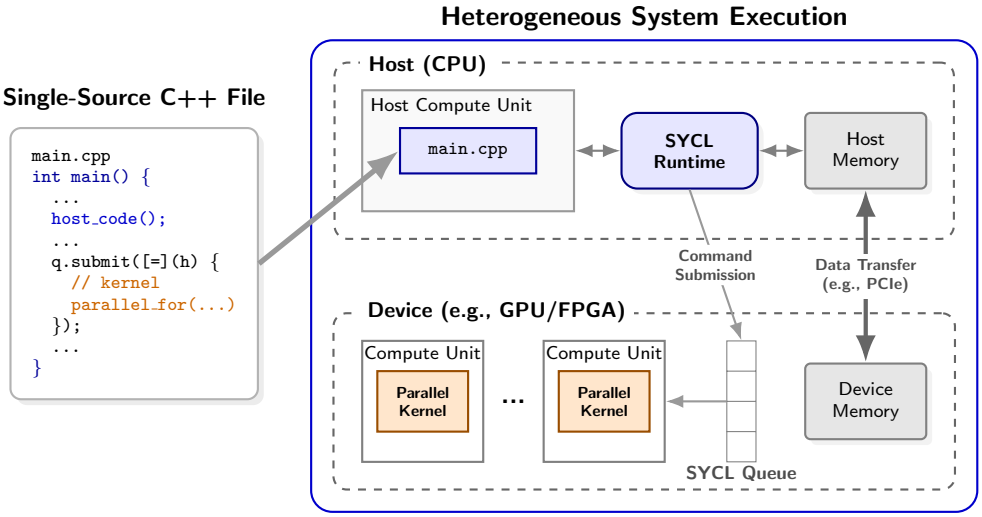


Figure 1.7: SYCL heterogeneous execution model. The workflow relies on a single-source C++ codebase that defines both control and computation logic. The *Host* (CPU) orchestrates the execution by submitting command groups to an asynchronous *SYCL Queue*. The runtime system manages the scheduling and data transfer between the disjoint *Host and Device memory* spaces (e.g., via PCIe), allowing the *Device* (accelerator) to execute parallel kernels across its Compute Units.

resources, while high-level frameworks (e.g., TensorFlow, PyTorch) and performance portability libraries like Kokkos [45] abstract the underlying hardware entirely.

Despite the utility of these approaches, there is a growing need for portable, single-source programming models that avoid ecosystem fragmentation. While OpenCL pioneered standard cross-platform computing, modern C++–based approaches such as oneAPI and, specifically, SYCL, have gained prominence. SYCL has been developed as an open-standard abstraction layer based on modern C++, supporting a single-source programming paradigm in which host and device code are expressed within the same file. This design facilitates performance portability across a wide range of hardware platforms and vendors without sacrificing control [46, 47].

The SYCL execution model, illustrated in Figure 1.7, orchestrates the interaction between the host system (typically a CPU) and accelerator devices. Execution proceeds asynchronously: the host coordinates the global control flow and submits work, encapsulated in *command groups*, to a *SYCL Queue*. This queue serves as an abstract scheduling construct through which kernels are dispatched to the target device. The runtime system is responsible for resolving dependencies and orchestrating data transfers between host memory and disjoint device memory, thereby abstracting low-level hardware characteristics.

A central feature of the SYCL architecture is its modular *backend* mechanism, which decouples the high-level programming interface from the concrete hardware implementation. Consequently, it is technically feasible to design and integrate a dedicated backend targeting emerging architectures, such as QPUs. This extensibility indicates that QPUs may be incorporated as specialized co-processors within a unified heterogeneous computing environment. The validity and practical realization of this hypothesis will be systematically investigated throughout this thesis.

1.2 Quantum Computing: From Bits to Qubits

Every computing paradigm relies on two fundamental pillars: a theoretical mathematical model and a physical system that implements it [5]. In the realm of classical computing, this foundation is built upon the abstract concepts of the Turing Machine and Boolean Algebra [48, 49]. Theoretically, a Turing-complete machine could be constructed using mechanical levers, hydraulic pipes, or even falling dominoes [50]; however, the practical viability and efficiency of the model depend entirely on the capabilities of the underlying physical substrate. It was the invention of the transistor, and its subsequent miniaturization, that allowed this mathematical abstraction to scale into the ubiquitous digital technology of the modern era [51, 52].

Quantum computing does not represent a mere acceleration of this classical model, but rather a fundamental substitution of the underlying mathematical framework [53]. It replaces the deterministic rules of Boolean algebra with the principles of linear algebra and the postulates of quantum mechanics. Consequently, the fundamental unit of information shifts from the binary digit (bit) to the quantum bit or qubit [54, 55].

While a classical bit is physically confined to one of two discrete states, 0 or 1, a qubit is mathematically defined as a normalized vector $|\psi\rangle$ residing in a two-dimensional complex Hilbert space, called *space state*. Adopting the standard Dirac notation (bra-ket), the computational basis states are represented as the orthogonal vectors $|0\rangle$ and $|1\rangle$ [3, 6]:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{1.1}$$

Unlike a classical bit, a qubit can exist in a state of superposition, a linear combination of the basis states described by a state vector:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha|0\rangle + \beta|1\rangle, \tag{1.2}$$

where α and β are complex probability amplitudes satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. This implies that the qubit does not store a definite value, but rather encodes a probability distribution. Upon measurement, the quantum state collapses to $|0\rangle$ with probability $|\alpha|^2$ or to $|1\rangle$ with probability $|\beta|^2$.

Geometrically, this state can be visualized using the Bloch Sphere (Figure 1.8), where any pure qubit state corresponds to a point on the surface of a unit sphere. While a classical bit can only be at the "North Pole" (0) or the "South Pole" (1), a qubit can occupy any point on the surface, illustrating the immense density of information that can be encoded in the relative phases and amplitudes of the superposition.

This mathematical distinction leads to an exponential divergence in information capacity. A classical register of n bits can represent only one of the 2^n possible permutations at a time. In contrast, a quantum register of N entangled qubits exists

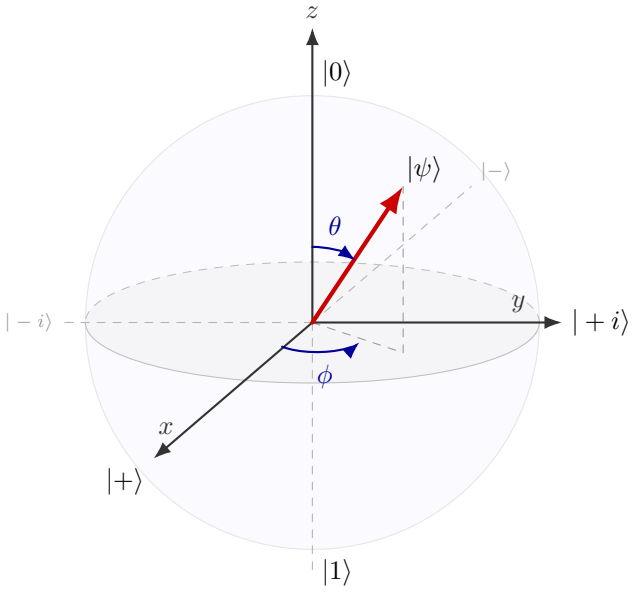


Figure 1.8: Geometric representation of a qubit: Bloch sphere. A quantum state $|\psi\rangle$ is visualized as a vector on the surface of a unit sphere, parameterized by the polar angle θ and the azimuthal angle ϕ. The z-axis poles correspond to the computational basis states $|0\rangle$ and $|1\rangle$. The equator represents maximal superposition states, where the intersections with the x-axis denote the Pauli-X eigenstates $(|+\rangle, |-\rangle)$ and the intersections with the y-axis denote the Pauli-Y eigenstates $(|+i\rangle, |-i\rangle)$.

in a superposition of all 2^n basis states simultaneously, each with its own complex amplitude [56]. This property allows quantum algorithms to manipulate an exponential amount of information with a linear amount of hardware resources, providing the potential for the computational speedups discussed in later sections.

1.2.1 The postulates of quantum mechanics

Once the qubit has been defined as the fundamental unit of information, it becomes necessary to specify the physical principles that determine its manipulation and dynamical evolution. These principles, formalized as the postulates of quantum mechanics, provide the rigorous mathematical framework that links the abstract linear-algebraic structure of the state space with physical observations [6, 57]. Within the context of quantum information processing, these postulates characterize the operational lifecycle of a quantum algorithm, encompassing state initialization, coherent gate application, and final measurement (readout).

Postulate 1: State Space

The state of an isolated physical system is fully characterized by a unit vector $|\psi\rangle$ belonging to a complex Hilbert space $\mathcal{H}$.

The condition that the state vector must be normalized, $\langle\psi|\psi\rangle = 1$, is not a mere convention but a necessary condition for the conservation of total probability [3]. This postulate asserts that the quantum state encapsulates all physically accessible information about the system.

Postulate 2: Time Evolution

The evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\psi(t)\rangle$ of the system at time t is related to the state $|\psi(0)\rangle$ at time $t = 0$ by a unitary time-evolution operator $U(t)$:

$$|\psi(t)\rangle = U(t)\,|\psi(0)\rangle\,. \tag{1.3}$$

The time evolution of a closed quantum system is deterministic and is governed by the time-dependent Schrödinger equation [2]:

$$i\hbar\frac{d|\psi(t)\rangle}{dt} = H|\psi(t)\rangle, \tag{1.4}$$

where $\hbar$ denotes the reduced Planck constant and H is the Hamiltonian operator, which encodes the total energy of the system. From the perspective of computer science and quantum information, it is often more convenient to represent the system's evolution in terms of discrete time steps. In this approach, each step (corresponding to the application of a single quantum gate) is typically treated under the approximation that the Hamiltonian remains constant over that interval. Under this assumption, the differential equation can be formally rearranged as:

$$\frac{d|\psi(t)\rangle}{dt} = -\frac{i}{\hbar}H|\psi(t)\rangle. \tag{1.5}$$

Integrating both sides from time 0 to t yields

$$\int_0^t d|\psi(\tau)\rangle = -\frac{i}{\hbar}H\int_0^t |\psi(\tau)\rangle\; d\tau \implies |\psi(t)\rangle = e^{-\frac{i}{\hbar}tH}. \tag{1.6}$$

Exponentiating this relation gives the explicit solution for the time evolution:

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle. \tag{1.7}$$

In this expression, the operator acting on the initial state is a unitary time-evolution operator[2]:

$$U = e^{-iHt/\hbar}, \tag{1.8}$$

so that $|\psi(t)\rangle = U|\psi(0)\rangle$. The unitarity of U guarantees that quantum time evolution is inherently reversible and norm-preserving [58]. This provides the physical justification for the quantum circuit model, which will be detailed in the subsequent section. Within this computational framework, quantum algorithms are implemented by decomposing a desired global unitary transformation into a sequence of elementary unitary operations, commonly called quantum gates. In contrast to classical logic gates (such as AND and OR), which are generally dissipative and information-losing, ideal quantum gates implement reversible transformations on the state vectors in Hilbert space [59].

Postulate 3: Quantum Measurement

Quantum measurements are described by a collection of measurement operators $\{M_m\}$, defined on the state space of the system, where the index m refers to the measurement outcomes.

Although the evolution of a closed system is deterministic, extracting information is intrinsically probabilistic. If the state of the system is $|\psi\rangle$ immediately before the measurement, then the probability that result m occurs is given by the *Born rule* [60]:

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle . \tag{1.9}$$

To ensure that the probabilities of all possible outcomes sum to one ($\sum_m p(m) = 1$), the measurement operators must satisfy the completeness equation:

$$\sum_m M_m^\dagger M_m = I. \tag{1.10}$$

A direct consequence of the measurement is the alteration of the quantum state. Immediately after the outcome m is observed, the system undergoes a non-unitary state reduction ("wave function collapse") to the normalized state:

$$|\psi'\rangle = \frac{M_m |\psi\rangle}{\sqrt{p(m)}}. \tag{1.11}$$

[2]A unitary operator U satisfies $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ denotes the adjoint (conjugate transpose) of U. This condition implies that U is an isometry with respect to the Hilbert space inner product, i.e., it preserves inner products and, consequently, the norms of state vectors. In quantum mechanics, this ensures conservation of total probability under time evolution and implies that unitary dynamics are strictly reversible.

In the standard computational basis $\{|0\rangle, |1\rangle\}$, measurements are typically projective (von Neumann measurements [61]), where the operators M_m are orthogonal projectors. This collapse signifies the transition from quantum superposition to classical definition. In algorithm design, this implies that observing intermediate states destroys the computation's coherence; therefore, measurements are typically deferred until the end of the execution.

Postulate 4: Composite Systems

The state space of a composite physical system is the tensor product of the state spaces of its component systems.

This postulate characterizes the scaling behaviour of quantum systems. Specifically, the state space of a composite physical system is given by the tensor product of the state spaces associated with its constituent subsystems. If system A is described by the Hilbert space $\mathcal{H}_A$ and system B by the Hilbert space $\mathcal{H}_B$, then the combined system is described by the Hilbert space

$$\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B.$$

This structural rule has far-reaching consequences for computational complexity. The addition of a single qubit to a quantum register doubles the dimension of the corresponding Hilbert space. Consequently, as it was previously mentioned, a system of n qubits is represented in a Hilbert space of dimension 2^n. This exponential growth in state-space dimension enables quantum computers to encode and process information that scales exponentially with the number of physical qubits, even though only a linear number of particles are employed.

Moreover, this postulate underlies the phenomenon of quantum entanglement: there exist states in $\mathcal{H}_{AB}$ that cannot be expressed in the separable form $|\psi\rangle_A \otimes |\phi\rangle_B$. Such non-factorizable states exhibit non-classical correlations [62], in which the state of one subsystem is intrinsically and inseparably correlated with the state of the other.

1.2.2 Quantum gates and circuit model

The abstract unitary evolution introduced in Postulate 2 of Section 1.2.1 constitutes the fundamental theoretical mechanism for state transformation in quantum mechanics. For the purpose of implementing practical quantum computations, however, this continuous time evolution must be decomposed into a discrete set of elementary operations [63]. Analogous to classical computation, which is organized around a finite set of Boolean logic gates (AND, OR, NOT) acting on bits, quantum computation in the circuit model is formulated in terms of quantum gates acting on qubits. Within this framework, a quantum algorithm is represented as a network of quantum wires and

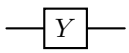
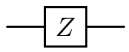
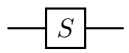
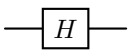
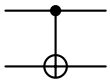
Gate Name	Symbol	Matrix	Effect on $\|\psi\rangle = \alpha\|0\rangle + \beta\|1\rangle$
Pauli-X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\beta\|0\rangle + \alpha\|1\rangle$
Pauli-Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$-i\beta\|0\rangle + i\alpha\|1\rangle$
Pauli-Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\alpha\|0\rangle - \beta\|1\rangle$
$\frac{\pi}{2}$ phase (S)		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\alpha\|0\rangle + i\beta\|1\rangle$
Hadamard		$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\frac{1}{\sqrt{2}}[\alpha(\|0\rangle + \|1\rangle) + \beta(\|0\rangle - \|1\rangle)] =$ $\alpha\|+\rangle + \beta\|-\rangle$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	On: $\alpha\|00\rangle + \beta\|01\rangle + \gamma\|10\rangle + \delta\|11\rangle$: $\alpha\|00\rangle + \beta\|01\rangle + \gamma\|11\rangle + \delta\|10\rangle$

Table 1.3: Fundamental Quantum Gates. Summary of the most common single and multi-qubit gates, showing their standard circuit symbol, unitary matrix representation, and effect on generic state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $\alpha, \beta \in \mathbb{C}$.

gates: the wires denote the qubits, conveying quantum information as a function of (logical) time, and the gates denote the corresponding unitary transformations applied to them. By convention, a quantum circuit is read from left to right, reflecting the temporal ordering of these operations.

Table 1.3 summarizes the fundamental single- and multi-qubit operators, specifying both their matrix representations and their action on the computational basis states [6].

Single-qubit gates can be interpreted geometrically as rotations of the state vector $|\psi\rangle$ on the Bloch sphere (Figure 1.8). In particular, the Pauli operators (X, Y, Z) implement rotations by an angle of π radians (180°) about the Cartesian axes x, y, and z, respectively. The S gate can be interpreted as the square-root operation of the Z gate, implementing a rotation by $\frac{\pi}{2}$ radians (90°) about the z-axis of the Bloch sphere. Analogously to the Z gate, it leaves the computational basis state $|0\rangle$ invariant, while multiplying the $|1\rangle$ component by a complex phase factor i.

- The Pauli-X gate acts as a "quantum bit-flip" (or NOT gate). By rotating the state about the x-axis, it swaps the computational basis states: $|0\rangle \leftrightarrow |1\rangle$.
- The Pauli-Y gate combines a bit-flip with a phase shift. It rotates the state about the y-axis, mapping $|0\rangle \to i|1\rangle$ and $|1\rangle \to -i|0\rangle$. Unlike the X gate, Y introduces a complex phase factor i, which is essential for operations requiring complex superposition manipulation.
- The Pauli-Z gate acts as a "phase-flip". It rotates the state about the z-axis, leaving $|0\rangle$ invariant while mapping $|1\rangle \to -|1\rangle$. This operation changes the relative phase of a superposition (e.g., $\alpha|0\rangle + \beta|1\rangle \to \alpha|0\rangle - \beta|1\rangle$). While this does not affect measurement probabilities in the computational basis, it modifies the quantum interference patterns, impacting measurements in other bases (such as the X-basis).
- The **S gate** (or $\frac{\pi}{2}$ phase gate) corresponds to a rotation of $\pi/2$ radians (90°) about the z-axis. Algebraically, it acts as the square root of the Pauli-Z gate ($S^2 = Z$). Like the Z gate, it leaves the basis state $|0\rangle$ invariant but maps $|1\rangle \mapsto i|1\rangle$.

While the Pauli operators can be viewed as π-rotations within the computational basis, the Hadamard gate (H) serves as a change-of-basis operator. It maps between the computational basis $\{|0\rangle, |1\rangle\}$ and the so-called Hadamard basis, given by the equal superposition states

$$|+\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right), \quad |-\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right). \tag{1.12}$$

In particular, H transforms $|0\rangle$ and $|1\rangle$ into $|+\rangle$ and $|-\rangle$, respectively, and thus is fundamental for creating and analyzing superposition states in quantum circuits. The Hadamard gate is widely employed because it generates quantum superpositions from computational basis states.

However, to fully exploit the computational capabilities implied by Postulate 4, it is necessary for qubits to interact so as to explore the exponentially large Hilbert space of multi-qubit states. This interaction is implemented via multi-qubit quantum gates, the most fundamental example of which is the Controlled-NOT (CNOT) gate [64]. As summarized in Table 1.3, the CNOT gate acts on two qubits: a *control* qubit and a *target* qubit. Its action in the computational basis is given by a simple rule: the target qubit is subjected to a bit-flip (Pauli-X operation) if and only if the control qubit is in the state $|1\rangle$.

Although this behavior resembles that of a classical XOR gate, its significance within the framework of quantum mechanics is substantially deeper. The controlled-NOT (CNOT) gate enables the state of one qubit (the control) to conditionally influence the state of another qubit (the target), and thus functions as a fundamental

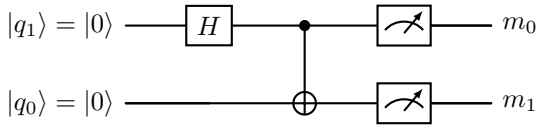


Figure 1.9: Generation of quantum entanglement: Bell-state preparation circuit. This quantum circuit prepares the maximally entangled Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$. Initially, a Hadamard gate is applied to qubit q_0, creating a superposition of the computational basis states. Subsequently, a controlled-NOT (CNOT) gate, with q_1 as the control qubit and the second qubit as the target, is applied to correlate their states, thereby generating entanglement between the two qubits.

primitive for the generation of quantum entanglement. When the control qubit is prepared in a coherent superposition of basis states, the action of the CNOT operation induces quantum correlations between the two qubits, typically resulting in a joint state that is non-separable and therefore cannot be expressed as a tensor product of individual single-qubit states.

This phenomenon is most clearly exemplified by the quantum circuit used to prepare a Bell state, as shown in Figure 1.9. The process begins with two qubits initialized in the computational basis ground state $|00\rangle$:

1. A Hadamard gate is applied to the first qubit (q_1), placing it in an equal superposition:
$$(H \otimes I)|00\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle = \frac{|00\rangle + |10\rangle}{\sqrt{2}} \tag{1.13}$$
At this stage, the system is still separable (a product state).

2. A CNOT gate is applied with q_1 as control and q_0 as target. Since the control is in a superposition of $|0\rangle$ and $|1\rangle$, the target is flipped only in the branch of the superposition where the control is $|1\rangle$:
$$\text{CNOT}\left(\frac{|00\rangle + |10\rangle}{\sqrt{2}}\right) = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{1.14}$$

The resulting state, denoted by $|\Phi^+\rangle$, is non-separable and therefore cannot be expressed as a tensor product of two independent single-qubit states, i.e., it does not admit a decomposition of the form $|\psi\rangle_A \otimes |\phi\rangle_B$. This state is one of the four Bell states, which constitute a complete set of maximally entangled two-qubit states forming an orthonormal basis of the corresponding two-qubit Hilbert space. These states exhibit perfect quantum correlations: a projective measurement performed on one qubit uniquely determines the post-measurement state, and thus the outcome statistics, of the other qubit [62, 65]. This fundamental property underlies several pivotal quantum information processing protocols, including quantum teleportation and superdense coding [66].

1.2.3 Quantum algorithms and complexity

The distinctive computational advantage of quantum computing arises from its fundamentally different computational model, based on quantum parallelism. This principle enables a quantum computer to evaluate a function $f(x)$ on a superposition of many different inputs x simultaneously, rather than processing each input sequentially as in classical computation [67].

To formalize this notion, consider a classical Boolean function

$$f : \{0,1\}^n \to \{0,1\},$$

which maps n-bit inputs to a single-bit output. Classical functions are generally not reversible, since the input x cannot be uniquely determined from the output $f(x)$. In other words, classical computation typically erases information, a limitation that quantum circuits must avoid to ensure unitary evolution.

To reconcile this irreversibility with the reversibility required in quantum theory, one exploits a well-known result from the theory of reversible computation: any classical circuit can be simulated with at most polynomial overhead by a reversible circuit, provided that a sufficient number of ancilla bits (or qubits in a quantum circuit) is available[3] [59, 68].

The standard procedure to implement a classical function f as a unitary transformation employs two quantum registers: an input register $|x\rangle$ and a target register $|y\rangle$. The function value is coherently computed and added modulo 2 (XOR operation) to the target register. This yields the transformation

$$|x\rangle|y\rangle \;\mapsto\; |x\rangle\,|y \oplus f(x)\rangle,$$

which is manifestly reversible, since applying the same operation twice returns the system to its original state due to the identity

$$y \oplus f(x) \oplus f(x) = y.$$

In this way, one defines the so-called oracle operator U_f associated with the function f as the unitary acting on the joint Hilbert space of the two registers according to

$$U_f : |x\rangle|y\rangle \;\longmapsto\; |x\rangle\,|y \oplus f(x)\rangle. \tag{1.15}$$

[3] An ancilla qubit is an auxiliary qubit initialized in a well-defined reference state, typically $|0\rangle$, and used to store intermediate computational results or to facilitate operations requiring an enlarged Hilbert space, such as embedding classically irreversible logic into a reversible quantum framework. These ancilla degrees of freedom are often required to be "uncomputed" (i.e., returned to their initial state) in order to eliminate residual entanglement with the data registers and thus preserve the intended logical structure of the computation.

If we prepare the input register in a uniform superposition of all possible $N = 2^n$ states (using Hadamard gates) and the target register in state $|0\rangle$, the linearity of quantum mechanics leads to a remarkable result. Applying the operator U_f to the joint state:

$$\begin{aligned} |\psi_{out}\rangle &= U_f \left(\left[\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \right] \otimes |0\rangle \right) \\ &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} U_f\big(|x\rangle|0\rangle\big) \quad \text{(by linearity)} \\ &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|0 \oplus f(x)\rangle \quad \text{(by Oracle definition)} \\ &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|f(x)\rangle \end{aligned} \tag{1.16}$$

In a single computational step, the circuit has generated a state containing the evaluations of $f(x)$ for all N possible inputs entangled with their corresponding output.

However, when examined from the perspective of information extraction, this apparent massive parallelism is largely illusory. In accordance with Postulate 3, a measurement of the output register collapses the superposition onto a single computational basis state $|k\rangle|f(k)\rangle$ with probability $1/2^n$ [69]. As a result, after measurement, we obtain only a single evaluation of the function, exactly as if f had been queried once by a classical algorithm. The large set of function values that is implicitly computed in superposition is, in this sense, fundamentally inaccessible to direct measurement.

To overcome this limitation and to exploit quantum parallelism in a meaningful way, quantum algorithms must be constructed not to enumerate individual values of $f(x)$, but rather to infer global properties of the function. This is accomplished by harnessing quantum interference. By applying additional unitary operations (most commonly Hadamard transformations) prior to measurement, quantum algorithms can arrange for the probability amplitudes corresponding to "incorrect" outcomes to interfere destructively, while the amplitudes associated with the "correct" outcome interfere constructively and are thereby amplified.

The paradigmatic illustration of this principle is provided by the Deutsch–Jozsa algorithm. It solves the decision problem of determining whether an oracle-implemented Boolean function is *constant* (returns the same output for all inputs) or *balanced* (returns 0 for exactly half of the inputs and 1 for the remaining half).

The execution of this procedure is schematically depicted in Figure 1.10. The protocol evolves according to the following steps:

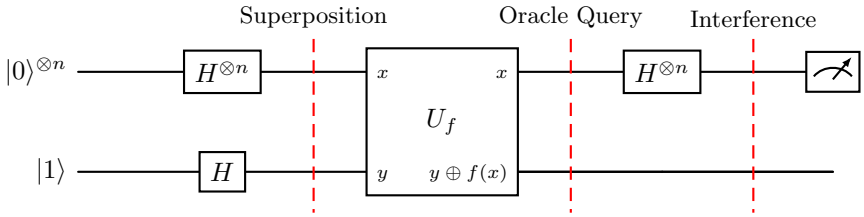


Figure 1.10: Circuit for the Deutsch-Jozsa Algorithm. The circuit determines if a function $f(x)$ is constant or balanced using a single query to the oracle U_f, exploiting quantum parallelism and interference.

1. **Initialization:** The data register is prepared in the computational basis state $|0\rangle^{\otimes n}$, while an ancilla qubit is initialized in the state $|1\rangle$.

2. **Superposition:** A layer of Hadamard gates is applied to all qubits. This operation transforms the data register into a uniform superposition over all basis states, and simultaneously prepares the ancilla to enable the "phase kickback" [70] effect[4].

3. **Oracle query:** The unitary operator U_f is then applied. Due to the superposition structure and the preparation of the ancilla, the function evaluation $f(x)$ is encoded not in the computational basis values but in the relative phases (i.e., sign inversions) of the quantum state.

4. **Interference:** A final layer of Hadamard gates is applied to the data register, inducing quantum interference among the amplitudes. If f is a constant function, all contributions interfere constructively exclusively in the state $|0\rangle^{\otimes n}$, while amplitudes corresponding to all other basis states destructively interfere.

Consequently, a single measurement of the data register yields the solution: if the outcome corresponds to the all-zero state ($|00\ldots0\rangle$), the function is constant; otherwise, it is balanced. So, the quantum circuit provides a deterministic solution using exactly one oracle query, independent of the input size. In contrast, a classical algorithm requires, in the worst case, up to $2^{n-1}+1$ oracle queries to determine whether the function is constant or balanced.

As demonstrated by the Deutsch–Jozsa algorithm [67], inherently quantum phenomena such as superposition and interference can yield reductions in query complexity that are impossible in the framework of classical computation.

[4]Phase kickback is a quantum phenomenon in which the eigenvalue associated with the action of an operator on a target qubit is transferred ("kicked back") to the relative phase of the control qubit. In this setting, the ancilla is prepared in the state $|-\rangle$, which is an eigenstate of the bit-flip operator X with eigenvalue -1 (since $X|-\rangle = -|-\rangle$). As a result, when the oracle evaluates $f(x)$, the state of the target qubit remains unchanged, while the factor $(-1)^{f(x)}$ is imprinted onto the phase of the control register $|x\rangle$, thereby enabling the subsequent interference effects.

Algorithm	**Problem Type**	**Classical**	**Quantum**	**Speedup**
Deutsch-Jozsa	Constant vs Balanced	$\mathcal{O}(2^n)$	$\mathcal{O}(1)$	Exponential
Grover	Unstructured Search	$\mathcal{O}(2^n)$	$\mathcal{O}(\sqrt{2^n})$	Quadratic
QFT	Discrete Fourier Transform	$\mathcal{O}(n2^n)$	$\mathcal{O}(n^2)$	Exponential
Shor	Integer Factorization	Super-polynomial	$\mathcal{O}((\log 2^n)^3)$	Exponential

Table 1.4: Comparison of Classical vs. Quantum Complexity. n represents the number of bits. The speedup highlights the theoretical advantage provided by quantum algorithms.

Moving beyond the primarily theoretical significance of oracle-based problems such as Deutsch–Jozsa, the field has produced quantum algorithms that exploit this computational advantage for tasks of more direct practical relevance. Two major algorithms, in particular, characterize this landscape, each providing a qualitatively different form of speedup over the best known classical algorithms. Table 1.4 summarizes these algorithms and their associated implications for computational complexity.

First, Grover's search algorithm addresses the canonical problem of locating a marked item in an unstructured database of N elements. A classical deterministic or randomized algorithm requires, on average, $N/2$ queries, yielding a time complexity that scales linearly as $\mathcal{O}(N)$. By contrast, Grover's algorithm exploits a procedure known as amplitude amplification to increase the probability amplitude of the target state, enabling the marked element to be found in $\mathcal{O}(\sqrt{N})$ oracle queries [8]. Although this constitutes a quadratic rather than an exponential speedup, the generality of the unstructured search model renders Grover's algorithm highly relevant for a broad class of optimization problems. Moreover, in the cryptographic domain, this result effectively reduces the security level of symmetric-key schemes–such as AES–by approximately one bit per two bits of key length, i.e., it "halves" the effective key strength against quantum adversaries [71].

Second, and of particular significance for contemporary cryptographic security, is Shor's factoring algorithm. This quantum algorithm addresses both the integer factorization problem and the discrete logarithm problem [7]. The security of widely deployed public-key cryptosystems [72], including RSA [73] and elliptic-curve-based schemes [74, 75], is predicated on the presumed intractability of these number-theoretic problems for classical computers. The fastest known classical factoring algorithms, such as the General Number Field Sieve (GNFS), run in super-polynomial time[5] to

[5]Super-polynomial time denotes a complexity class where the runtime grows faster than any polynomial function of the input size N, that is, faster than $\mathcal{O}(N^k)$ for every fixed constant k. Although such algorithms do not necessarily exhibit fully exponential complexity $\mathcal{O}(k^N)$, they are generally regarded as computationally intractable for large inputs.

factor large integers [76].

Shor's algorithm, however, circumvents these limitations by employing the QFT. The QFT serves as the quantum analogue of the classical Discrete Fourier Transform, performing a basis change that maps the computational basis states to the Fourier basis. This transformation enables the efficient extraction of the periodicity (frequency information) hidden within the phases of a quantum superposition, a task that effectively reduces the factorization problem to period finding. By leveraging this capability, Shor's algorithm solves these problems in polynomial time, specifically $\mathcal{O}((\log N)^3)$ for factoring an integer $N = 2^n$. This exponential asymptotic improvement over the best-known classical methods implies that a sufficiently large, fault-tolerant quantum computer would be capable of compromising the core public-key cryptographic primitives upon which the current Internet infrastructure relies.

While Shor's algorithm illustrates the theoretical power of future fault-tolerant systems, the realization of such error-corrected hardware remains a significant engineering challenge. Instead, we are currently in the Noisy Intermediate-Scale Quantum (NISQ) era [10]. This term, introduced by John Preskill, refers to the current generation of quantum processors, which include systems with tens to a few hundred qubits. These devices are characterized as "intermediate-scale" because their state spaces are too large to be tractably simulated on classical supercomputers, yet "noisy" because they operate without full-fledged quantum error correction. As a result, the achievable circuit depth (i.e., the number of sequential quantum operations) is fundamentally constrained by cumulative gate and decoherence errors. This limitation confines practical quantum algorithms either to those that exhibit intrinsic robustness to noise or to circuits sufficiently shallow to be executed before the quantum state undergoes significant decoherence.

To harness the computational power of these intermediate devices despite their limitations, the research focus has shifted towards hybrid quantum-classical algorithms, also known as Variational Quantum Algorithms (VQAs). VQAs employ a cooperative strategy based on a hybrid feedback loop, as illustrated in Figure 1.11. In this scheme, the QPU acts as a specialized co-processor to prepare a parameterized quantum state (defined by an *ansatz* $U(\vec{\theta})$) and estimate relevant observables. These measurement results are fed into a classical computer, which evaluates a specific cost function $C(\vec{\theta})$ and employs a classical optimizer (e.g., SPSA, Adam) to iteratively update the parameters $\vec{\theta}$, with the ultimate goal of minimizing the cost [77]. This approach keeps the quantum circuits relatively shallow, mitigating the impact of decoherence. Prominent examples include the Variational Quantum Eigensolver (VQE) for quantum chemistry simulations [78] and the Quantum Approximate Optimization Algorithm (QAOA) for combinatorial optimization problems [79].

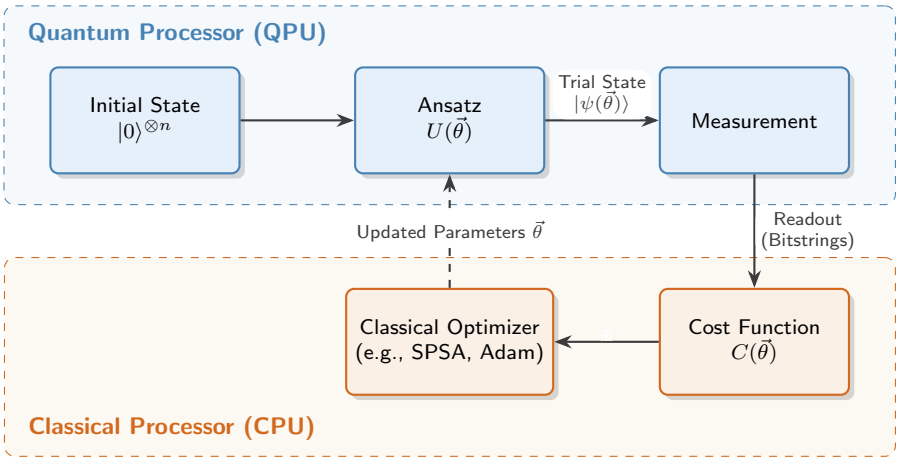


Figure 1.11: Schematic overview of a VQA. The process operates as a hybrid quantum-classical feedback loop. The QPU is responsible for preparing a trial state $|\psi(\vec{\theta})\rangle$ via a parameterized *ansatz* and performing measurements. The classical computer processes the measurement readout to evaluate a cost function $C(\vec{\theta})$ and subsequently uses a classical optimizer to determine updated parameters $\vec{\theta}$. These new parameters are fed back to the QPU for the next iteration, continuing until convergence.

1.2.4 Physical hardware in the NISQ era

As discussed in the introduction of Section 1.2 in the context of the Turing machine [48], a computational model remains a purely mathematical abstraction until it is instantiated in a physical system. Analogously to how the development of classical computation necessitated the invention of vacuum tubes and, subsequently, silicon-based transistors to initiate the digital revolution, the mathematical formalism of quantum mechanics–Hilbert spaces, unitary operators, and projective measurements–must be supported by an appropriate physical substrate in order to perform practically useful computational tasks. In the absence of a physical platform capable of reliably preparing, controlling, and measuring quantum states, the quantum circuit model presented in the preceding sections remains solely a theoretical construct [80].

The transition from mathematical abstraction to physical implementation in quantum information science begins with the identification of natural systems that realize the defining characteristics of a qubit. A key historical and conceptual foundation for such systems is provided by the Stern–Gerlach experiment, in 1922 [81], which empirically demonstrated that certain quantum observables, such as angular momentum, are not continuously distributed but instead assume discrete, quantized values. Figure 1.12 illustrates the experimental configuration together with its associated apparatus and measurement scheme, and serves to highlight the experiment's far-reaching conceptual and theoretical implications.

The spatial separation observed in the experiment enables a direct correspondence

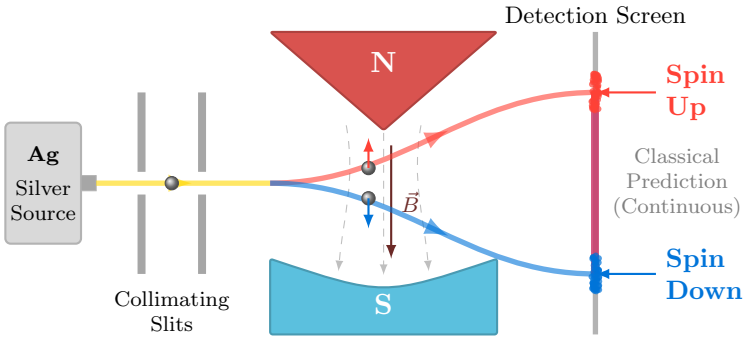


Figure 1.12: The Stern–Gerlach Experiment (2D view). A beam of silver atoms passes through an inhomogeneous magnetic field. While classical mechanics predicts a continuous distribution of impacts (gray zone), the observation of two discrete impact clusters provides direct evidence for the quantization of angular momentum (spin), establishing the physical basis for the qubit states $|0\rangle$ and $|1\rangle$.

between the abstract qubit basis states and physical observables. Specifically, the discrete upper trajectory is identified with the state $|0\rangle$ (spin-up), and the lower trajectory with $|1\rangle$ (spin-down). This quantization demonstrates that two-level quantum systems not only occur in nature but can also be clearly distinguished and manipulated, satisfying a foundational prerequisite for realizing a physical qubit.

However, although the Stern–Gerlach setup verifies the existence of controllable two-level quantum systems, the construction of a practical quantum computer necessitates extending such control to large-scale architectures comprising millions of interacting components. In 2000, David DiVincenzo formulated a widely adopted set of necessary conditions for the physical realization of a scalable circuit-based quantum computer, now referred to as the DiVincenzo Criteria [82, 83].

Simultaneously satisfying all five criteria constitutes a substantial engineering challenge. For example, physical implementations that are strongly isolated to enhance coherence (Criterion 3) are frequently difficult to address and manipulate for the purposes of gate operations and measurements (Criteria 4 and 5). This fundamental trade-off largely shapes the present landscape of quantum hardware.

At present, no single physical platform simultaneously meets all five DiVincenzo criteria to an optimal degree. Instead, the field is characterized by a diversity of hardware architectures, each competing by achieving a distinct balance among controllability, coherence, and scalability. These architectures are based in fundamentally distinct physical technologies, ranging from superconducting circuits to photonic systems. Representative examples include:

- **Superconducting Circuits (e.g., IBM, Google):** These platforms are based on lithographically defined LC oscillators fabricated on silicon or related sub-

The DiVincenzo Criteria

These five criteria constitute the principal benchmarks against which all candidate quantum hardware platforms are assessed:

1. **Scalability with well-characterized qubits:** The physical platform must support a scalable number of qubits (N). Since the dimension of the associated Hilbert space scales as 2^n, the architecture must accommodate the addition of qubits without incurring an exponential overhead in physical resources or noise.
2. **Initialization capability:** The system must enable high-fidelity initialization of the qubit register into a simple fiducial state, typically $|00\ldots0\rangle$. Reliable initialization is a fundamental prerequisite for implementing quantum error correction protocols.
3. **Long coherence times:** The hardware must preserve quantum coherence for a characteristic time substantially exceeding the duration of elementary gate operations ($T_{\text{coh}} \gg T_{\text{gate}}$), thereby allowing for the execution of sufficiently deep quantum circuits before decoherence dominates.
4. **A universal set of quantum gates:** The system must support the implementation of a universal gate set, i.e., a collection of unitary operations from which any desired unitary transformation U on the computational Hilbert space can be approximated to arbitrary accuracy.
5. **Qubit-specific measurement capability:** The architecture must permit projective or generalized measurements on individually addressed qubits, enabling the extraction of computational outcomes while minimally perturbing the quantum state of the remaining qubits.

strates and operated at millikelvin temperatures [84]. They currently constitute the leading architecture with respect to scalability (Criterion 1) due to their compatibility with established semiconductor fabrication and lithography processes. They also demonstrate strong performance in initialization (Criterion 2) and rapid gate control (Criterion 4) [11]. Nonetheless, their comparatively macroscopic physical dimensions render them susceptible to environmental noise, which substantially constrains their achievable *Coherence Times* (Criterion 3).

- **Trapped Ions (e.g., IonQ, Quantinuum):** In these systems, individual atomic ions are confined in ultra-high vacuum by means of static and time-varying electromagnetic fields [85]. They are widely regarded as the benchmark

for qubit fidelity, achieving near-ideal initialization (Criterion 2) and measurement (Criterion 5) because the qubits are encoded in well-defined, intrinsically identical atomic energy levels. Trapped-ion platforms exhibit the longest coherence times (Criterion 3) currently available in the field [86]. Their principal limitation concerns scalability (Criterion 1), since maintaining and controlling large ion registers in a single or networked trap architecture is experimentally demanding, and the typical gate operation times are slower than those of superconducting circuits.

- **Photonics (e.g., Xanadu, PsiQuantum):** This architecture encodes quantum information in photonic degrees of freedom, with photons propagating through integrated or free-space optical modes such as waveguides [87]. Because photons exhibit extremely weak interactions with their environment, they can achieve effective unbounded coherence (Criterion 3) over propagation distances relevant to computation and communication. Photonic platforms are highly promising for scalability (Criterion 1), as they can, in principle, function at or near room temperature and exploit the existing global fiber-optic infrastructure. However, the absence of strong intrinsic photon–photon interactions makes the deterministic implementation of universal gates (Criterion 4) technically challenging and typically reliant on probabilistic or measurement-induced nonlinearities [88].

- **Spin Qubits & NV Centers (e.g., Intel, QuTech):** These approaches employ the electronic (or, in some instances, nuclear) spin confined in semiconductor quantum dots or associated with point defects in crystals, such as nitrogen-vacancy (NV) centers in diamond [89, 90]. Owing to their nanometer-scale footprint, they offer exceptionally high qubit density and thus strong prospects for extreme scalability (Criterion 1). Coherence times (Criterion 3) in such systems are excellent, particularly in isotopically purified diamond hosting NV centers. The dominant challenge arises in control and measurement (Criteria 4 & 5): the need to route control lines and readout circuitry to densely packed qubit arrays without inducing significant crosstalk or decoherence remains a major engineering and systems-integration obstacle.

- **Topological quantum computing (e.g., Microsoft):** This approach relies on the existence of topological phases of matter whose quasiparticle excitations are non-Abelian anyons (most notably, Majorana zero modes), which obey non-Abelian braiding statistics. In this paradigm, quantum information is encoded nonlocally in collective many-body states that exhibit topological ground-state degeneracy. Quantum gates are implemented through braiding operations that exchange quasiparticles in space, producing unitary transformations determined solely by the topology of their trajectories. Topological platforms are particularly attractive with respect to coherence and fault tolerance (Criterion 3) [91], since

topological protection can provide intrinsic robustness against certain classes of local noise and decoherence mechanisms. They are also theoretically promising in terms of scalability (Criterion 1) [92]. However, the experimental realization of fully protected and controllable topological qubits is still at an early stage, and the unambiguous demonstration of non-Abelian statistics and scalable architectures remains an open challenge.

Despite these technological advances, contemporary quantum hardware has not yet achieved the level of robustness and scalability associated with a fully fault-tolerant Turing machine.

Chapter 2

Hypothesis and Objectives

The central premise of this doctoral research is driven by the scalability limitations of monolithic quantum processors and the subsequent necessity to shift towards Distributed Quantum Computing (DQC) architectures. However, realizing DQC in practical environments requires seamless integration with existing HPC ecosystems. Currently, a significant gap exists in the software stack required to bridge these two paradigms, particularly regarding communication protocols and heterogeneous programming models.

To address these challenges, this thesis is grounded in the following research hypotheses, which posit that established classical HPC paradigms, such as MPI and standard intermediate representations, can be effectively adapted to abstractions in the quantum domain:

H1. Feasibility of HPC-Quantum Integration: It is feasible to seamlessly integrate simulated distributed quantum computing into HPC infrastructures by leveraging heterogeneous programming frameworks.

H2. Adaptability of the MPI Paradigm: The MPI paradigm, commonly used in classical HPC, is adaptable to distributed quantum computing. Its implementation facilitates the management of quantum data exchange (qubits) and process synchronization, thereby mitigating the learning curve for HPC developers compared to low-level quantum hardware description languages.

H3. Advantages of Distributed Quantum Computing: It is possible to design distributed quantum algorithms that achieve a synergy between resource efficiency and the scalability of the global quantum state.

To validate the aforementioned hypotheses and advance the state of the art in hybrid HPC-quantum systems, a set of specific research objectives has been defined. These objectives range from the theoretical analysis of the current landscape to the

practical design, implementation, and benchmarking of a full software stack for distributed quantum computing:

O1. Conduct comprehensive state-of-the-art reviews on key topics in quantum computing:

a) *On the monolithic quantum compilation process:* To analyze the current software and hardware tools involved throughout the entire quantum circuit execution pipeline, ranging from high-level code abstraction down to the hardware-level layers. Fulfilled in the Chapter 6.

b) *On the use of quantum communication for DQC:* To examine current DQC techniques across each abstraction layer of the compilation process. This involves evaluating existing software tools, communication protocols, and optimization algorithms designed for data distribution. Fulfilled in the Chapter 5.

c) *On integrating quantum computing into HPC environments via Intermediate Representation (IR):* To investigate the current state of classical and quantum IRs used by compilers to facilitate system interoperability–a crucial aspect of distributed computing–and to define strategies for their integration within HPC ecosystems. Fulfilled in the Chapter 6.

O2. Develop a distributed quantum computing library inspired by the MPI standard, enabling developers to write distributed quantum applications using high-level primitives for the transfer of qubits and classical data. Fulfilled in the Chapter 8.

O3. Define an IR for distributed quantum systems that supports high-level quantum communication instructions and serves as a bridge between high-level distributed quantum programming languages and the underlying quantum network hardware. Fulfilled in the Chapter 7.

O4. Establish a benchmarking framework by creating a suite of tests to characterize and compare the performance of various distributed quantum computing emulators across all abstraction layers, from the physical layer up to the application layer. Fulfilled in the Chapter 10.

O5. Extend the support of standard heterogeneous programming environments (specifically SYCL) to enable the transparent integration and execution of quantum simulators within HPC workflows, a capability particularly relevant for variational quantum algorithms. Fulfilled in the Chapter 9.

O6. Apply distributed quantum computing paradigms to a specific algorithm, specifically by implementing and optimizing the communication overhead of a distributed version of the Inverse Quantum Fourier Transform (iQFT). Fulfilled in the Chapter 11.

Chapter 3

General Methodology

This doctoral research follows a design oriented and experimental methodology grounded in (i) systematic state-of-the-art analysis, (ii) formal modeling and abstraction design, (iii) software prototyping and implementation, and (iv) experimental validation and benchmarking. The methodological approach is aligned with the six research objectives (O1–O6) and was conducted in close collaboration with the quantum computing research group at CESGA and under the continuous supervision of the thesis advisors.

The research began with a comprehensive, structured review of the literature across key areas of quantum computing, with special emphasis on DQC architectures, quantum programming languages and IRs, and HPC integration of quantum workloads.

The review process combined systematic database searches, citation graph exploration, and technical analysis of open-source frameworks and research prototypes. The objective was twofold:

- To identify architectural gaps and abstraction mismatches in existing distributed quantum proposals.
- To extract design principles for scalable, hardware-agnostic distributed quantum software stacks.

This analysis directly informed the architectural decisions made in subsequent objectives.

The experimental validation of the proposals presented in this thesis, ranging from the performance characterization of quantum computing emulators to the execution of distributed algorithms, was conducted using two complementary high-performance computing infrastructures:

- **CiTIUS HPC Cluster:** For the validation of the distributed iQFT (Chapter 11), the High-Performance Computing cluster of the Research Centre in

Intelligent Technologies (CiTIUS)[1].

- **CESGA Quantum Infrastructure:** The general benchmarking of distributed quantum computing simulators (Chapter 10) was performed on the quantum computing infrastructure of the Galicia Supercomputing Center (CESGA)[2]. Specifically, the experiments were executed on high-memory compute nodes optimized for the *Qiskit AER* statevector simulator.

As it was previously mentioned, all objectives were achieved through continuous collaboration with the quantum computing research group at CESGA. The research process was iterative:

- Conceptual design → prototype implementation → experimental validation → refinement.
- Periodic technical reviews with thesis advisors.
- Internal evaluation cycles to ensure scientific rigor and architectural coherence.

Throughout the thesis, all software artifacts were version-controlled and experimental setups were documented and automated, with the results validated through repeated experimental runs. This ensured methodological rigor, reproducibility, and scientific validity of the contributions.

[1]CiTIUS HPC Infrastructure: `https://citius.gal/citius/our-spaces/it-infrastructures/`
[2]CESGA Quantum Infrastructure: `https://www.cesga.es/en/infrastructures/quantum/`

Chapter 4

Discussion: the Quantum-HPC Integration Gap

The convergence of High-Performance Computing (HPC) and Quantum Computing (QC) is widely recognized as the necessary trajectory for achieving practical quantum advantage. However, the integration of these two paradigms is currently hindered by a fundamental disconnect, referred to in this thesis as the Quantum-HPC Integration Gap. This gap arises primarily from the divergent focus of existing software tools. While the networking community has developed robust protocols for entanglement generation and fidelity management (the "Network Layer"), the computing community lacks the high-level abstractions required to program distributed algorithms without manually managing the underlying physics. In essence, current tools are designed to simulate quantum networks rather than to perform distributed quantum computing.

This chapter addresses this gap by deconstructing the software stack and identifying the critical missing links at the Development and Application layers. The discussion is structured to provide the theoretical context and architectural justification for the contributions presented in subsequent chapters, aligning with the specific objectives of this thesis.

To understand where to bridge this gap, one must first understand how quantum software is built. **Section 4.1** provides a comprehensive analysis of the Quantum Compilation Process. By dissecting the flow from high-level languages to physical pulses, we identify the specific stages where distributed primitives must be injected. This analysis allows us to comprehend the compilation flow and pinpoint optimization opportunities, thereby fulfilling objectives **O1a)** and **O1c)**.

In **Section 4.2**, we navigate the complex landscape of the Distributed Quantum Computing (DQC) paradigm. We present a state-of-the-art review and a taxonomy of distribution strategies, distinguishing between parallel execution, circuit cutting, and entanglement-assisted networked computing. This analysis justifies the thesis's focus on the communications between interconnected Quantum Processing Units (QPUs)

and identifies open research lines, thereby fulfilling objective **O1b)**.

Having defined the physical requirements, **Section 4.3** addresses the core of the integration gap: the Development Layer. Here, we propose a robust Distributed Quantum Software Stack designed to democratize access to DQC resources. We discuss the necessity of a standardized Intermediate Representation (IR), introducing NetQIR (an extension of the standard Quantum Intermediate Representation (QIR) and Low-Level Virtual Machine (LLVM)) to leverage established compilation infrastructures. Furthermore, we introduce NetQMPI, a high-level software library that abstracts the complexity of quantum network controllers (such as NetQASM). By enabling the programming of distributed applications using an Single Program Multiple Data (SPMD) paradigm similar to classical Message Passing Interface (MPI), we bridge the semantic gap between hardware protocols and algorithm design, fulfilling objectives **O3** and **O2**.

To validate these architectural proposals in the absence of large-scale physical hardware, **Section 4.4** focuses on the Emulation of distributed systems. We categorize the existing landscape of simulators and emulators, establishing a framework for benchmarking distributed backends. This ensures that the proposed tools can be rigorously evaluated, fulfilling objective **O4**.

Complementary to these distributed tools, we also address the integration of quantum resources into standard HPC environments. We propose extending the SYCL programming model to enable the transparent execution of quantum simulators within HPC workflows. This capability is particularly relevant for Variational Quantum Algorithms (VQAs), ensuring that the proposed stack aligns with established heterogeneous standards, fulfilling objective **O5**.

Finally, **Section 4.5** demonstrates the practical utility of this stack at the Application Layer. We present a concrete use case: a communication-efficient distributed optimization of the Inverse Quantum Fourier Transform (iQFT). As a critical subroutine for foundational algorithms like Quantum Phase Estimation (QPE) and Shor's algorithm, its efficient distribution is paramount. We demonstrate how algorithmic awareness allows for a reduction in entanglement resource consumption from quadratic complexity $\mathcal{O}(M^2)$ to a constant saturation $\mathcal{O}(1)$, fulfilling objective **O6**.

Through this structured discussion, this chapter establishes the roadmap for bridging the Quantum-HPC gap, moving from low-level compilation to high-level algorithmic efficiency.

4.1 Quantum Compilation Process

To effectively bridge the gap between HPC and QC, it is imperative to understand how quantum programs are constructed and executed. Programming a quantum computer is not merely about defining quantum gates; it involves a complex translation process from high-level abstractions, understandable by humans, into low-level ma-

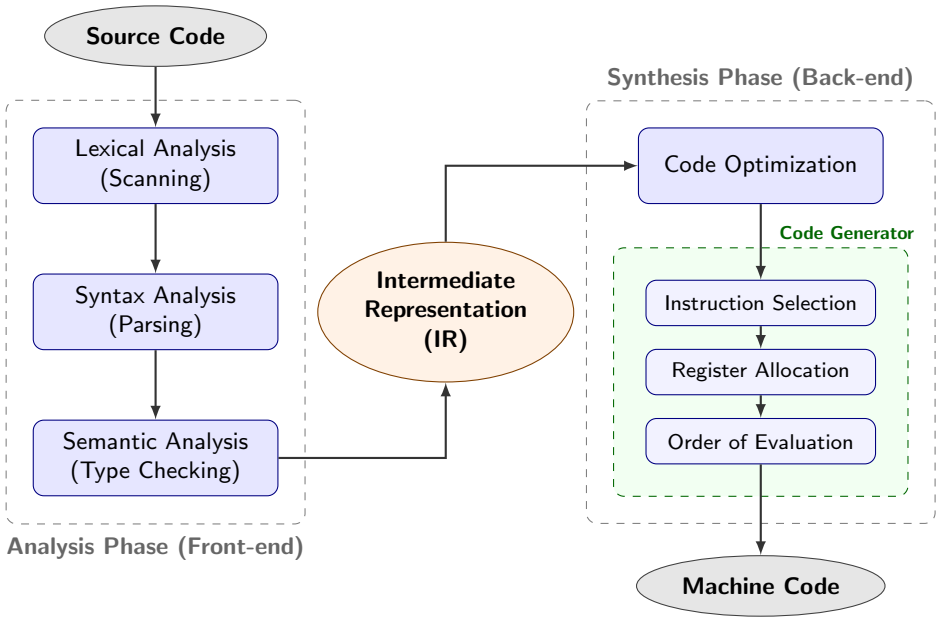


Figure 4.1: Standard Compilation Flow. The process is decoupled into an analysis phase, which validates the source code, and a synthesis phase. Notably, the *Code Generator* is further decomposed into instruction selection, register allocation, and evaluation ordering (steps that in the quantum context correspond to gate decomposition, qubit mapping, and scheduling).

chine instructions (pulses) that control the physical qubits. This translation layer is applied by the compiler[1] is the cornerstone of the quantum software stack and a primary candidate for optimization in hybrid HPC-QC environments [93].

4.1.1 Classical compilation process

Before addressing the specifics of quantum compilation, it is useful to revisit the principles of classical compilation, as the quantum workflow largely inherits this architecture. A compiler is fundamentally a translator that maps a source language to a target language while preserving semantics. As illustrated in the classical compiler design literature, this process is divided into two main phases: the *Analysis Phase* (front-end) and the *Synthesis Phase* (back-end), mediated by an IR [94].

[1] A compiler is a specialized software program that translates source code written in a high-level programming language (source language) into a lower-level language (target language), such as assembly or machine code, creating an executable program while preserving the original semantic meaning. Beyond mere translation, modern compilers perform critical analysis and synthesis tasks to optimize the code for the target architecture, minimizing resource usage and execution time.

The *Analysis Phase* decomposes the source code into its constituent parts (lexical analysis), imposes a grammatical structure (syntax analysis), and validates meaning and types (semantic analysis). The result is an IR: a hardware-agnostic data structure that facilitates optimization. The *Synthesis Phase* then transforms this IR into the final machine code, applying optimizations tailored to the target architecture to improve performance or resource usage.

4.1.2 Quantum compilation as a classical process

A crucial distinction in the current landscape is that quantum compilation is a classical process. The compilation of high-level quantum languages does not exploit quantum superposition or entanglement; it is performed entirely on classical processors using classical algorithms. Consequently, the efficiency of the quantum workflow depends heavily on the performance of classical compilation tools, a synergy that naturally aligns with HPC resources [95].

The quantum compilation pipeline mirrors the classical one depicted in Figure 4.1, adapting the stages to handle quantum primitives.

Analysis phase and software categories

In the analysis phase, the lexical, syntax, and semantic analyzers operate similarly to their classical counterparts but must recognize quantum-specific identifiers (e.g., qubits, registers) and operations (e.g., Hadamard, CNOT, measure). However, the way these identifiers are presented to the user varies significantly across the software stack. Based on the categorization presented in Chapter 6, the quantum software ecosystem can be classified into four main categories [95–97]:

- **Framework Libraries:** These are libraries embedded within established classical languages (typically Python or C++). They allow users to construct quantum circuits using host language objects. Examples include Qiskit and Cirq [98]. This is currently the most prevalent approach in the Noisy Intermediate-Scale Quantum (NISQ) era due to its flexibility.
- **Language Extensions:** These extend the syntax of an existing classical host language to support quantum primitives natively, often requiring a dedicated compiler or pre-processor. An example is the extension of C++ in frameworks like QCOR [99] or CUDA-Q, which facilitate hybrid kernel programming.
- **Standalone Languages:** These are new languages designed from the ground up for quantum computing, with their own syntax and type systems. Examples include Q# (Q Sharp) [100] and *Scaffold* [101]. These languages often provide stronger guarantees regarding quantum resource management during the semantic analysis.

- **Quantum Process Algebras:** These are formal modeling languages used primarily for theoretical verification and the study of quantum interaction patterns, such as Communicating Quantum Processes (CQP) [102]. While less common for direct programming, they are vital for verifying distributed quantum protocols.

4.1.3 Synthesis phase: Optimization and mapping

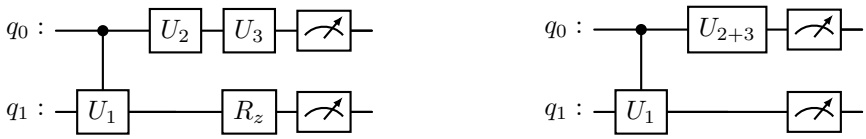
Once the code is analyzed and converted to an internal representation, the synthesis phase begins. As detailed in Figure 4.1, the classical "Code Generator" involves three critical sub-tasks: selecting instructions, allocating registers, and determining the evaluation order. In the context of quantum computing, these translate directly into the most computationally demanding challenges of the stack:

- **Instruction Selection → Gate Decomposition:** High-level abstract gates (e.g., multi-controlled rotation) must be decomposed into the specific native gate set supported by the target QPU (e.g., CNOT + single-qubit rotations) [103].
- **Register Allocation → Qubit Mapping:** Logical qubits must be mapped to physical qubits on the chip. Unlike classical registers, which have uniform access, physical qubits have restricted connectivity. The compiler must solve the routing problem, inserting SWAP gates to move quantum states to adjacent positions for interaction [104].
- **Evaluation Order → Scheduling:** The compiler must determine the precise timing of pulses, scheduling parallel operations to minimize circuit duration and reduce decoherence errors.

Circuit optimization

In contrast to classical optimization, which primarily targets improvements in runtime performance or memory consumption, quantum optimization places emphasis on minimizing circuit depth and the number of noise-prone gates [105]. Figure 4.2 depicts the substantial reduction in circuit complexity that can be attained through the application of two fundamental optimization techniques:

Circuit optimization is typically an iterative process performed at multiple stages of the compilation stack. However, the primary phase is *hardware-agnostic optimization*, which occurs prior to qubit mapping. In contrast to classical optimization, which primarily targets improvements in runtime performance or memory consumption, this quantum phase places emphasis on minimizing circuit depth and the number of noise-prone gates before addressing physical constraints [105]. Performing these simplifications early is crucial to prevent the compiler from solving the routing problem for redundant gates.



(a) Non-optimized circuit. **(b)** Optimized circuit.

Figure 4.2: Quantum Circuit Optimization. Comparison between (a) an unoptimized quantum circuit generated directly from a high-level algorithm, and (b) the optimized version of the same circuit. The reduction in gate count and depth in (b) is achieved by fusing consecutive unitary gates and removing redundant Z-rotations that precede the final Z-basis measurements.

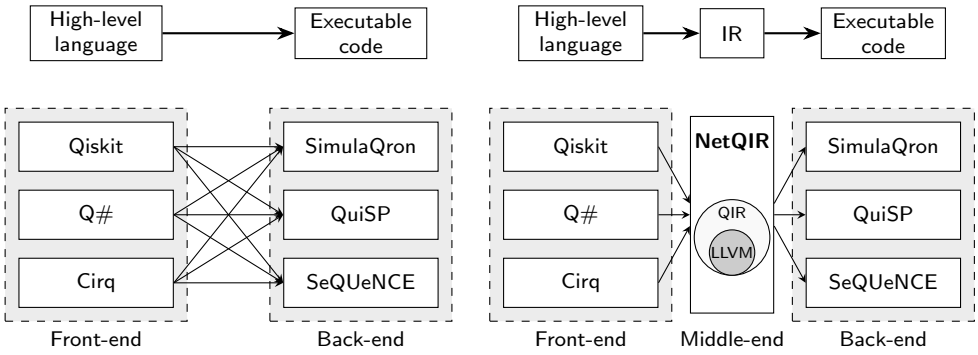
Figure 4.2 depicts the substantial reduction in circuit complexity that can be attained through the application of two fundamental optimization techniques:

- **Gate Fusion:** As illustrated in Figure 4.2, consecutive unitary gates acting on the same qubits can be multiplied into a single unitary matrix. This reduces the total gate count, though it may require decomposition later into the hardware's native gate set.
- **Z-Propagation and Removal:** Many quantum circuits typically end with measurements in the Z-basis. Since diagonal gates (like Pauli-Z, S, T) commute with the control of CNOTs and the target of other diagonal gates, they can often be commuted to the end of the circuit. If a Z-rotation is applied immediately prior to a measurement in the computational (Z) basis, it does not alter the corresponding outcome probabilities. Consequently, such a rotation can be omitted without loss of generality, thereby reducing the overall error budget [105].

Qubit mapping

Following the logical optimization, a critical component of the synthesis phase is the mapping of logical qubits to physical qubits. This process is fundamentally driven by the limited connectivity of current quantum processors. Unlike classical registers, which typically allow for uniform random access, physical qubits in architectures such as superconducting circuits or quantum dots are arranged in rigid topologies (e.g., Heavy-Hex, Sycamore) where only adjacent qubits can interact directly.

Consequently, if a two-qubit gate is requested between non-adjacent qubits, the compiler must resolve the routing problem by injecting SWAP gates to transport quantum states to neighboring sites. Minimizing the number of these SWAP operations is an NP-hard problem that directly dictates the fidelity of the execution, as each additional gate introduces significant noise and latency into the circuit [106, 107]. Once mapping is complete, a secondary pass of *hardware-aware optimization* is often performed to simplify any redundancies introduced by the inserted SWAP gates.



(a) Without IR. **(b)** With IR.

Figure 4.3: The role of IR in compiler modularity. (a) Without a unified IR, creating support for M languages across N hardware backends requires $M \times N$ specific compilers (monolithic approach). (b) By adopting a common IR, the ecosystem is decoupled: front-ends translate to the IR and back-ends synthesize from the IR, reducing the complexity to $M + N$ and facilitating the integration of new HPC-QC architectures.

4.1.4 Intermediate representations (IRs)

The efficacy of the synthesis phase described previously is fundamentally predicated on the capacity to represent the quantum program in a format that is simultaneously expressive for the front-end and explicit for the back-end. This format constitutes the IR. The IR functions as the middle-point of the compiler stack, serving as the critical interface that bridges the semantic gap between high-level programming abstractions and hardware-specific constraints [94].

During the nascent stages of quantum software development, the ecosystem was characterized by significant fragmentation. Compiler infrastructures were predominantly monolithic, translating specific languages directly into specific target simulators or devices. As illustrated in Figure 4.3a, this architecture results in a combinatorial complexity of $M \times N$, where M denotes the number of programming languages and N the number of hardware backends. Consequently, extending support to a new QPU necessitated the modification of every existing language front-end. The adoption of a standardized IR (Figure 4.3b) effectively mitigates this scalability challenge. By decoupling the software stack, language designers can target a common IR, while hardware engineers strictly synthesize machine code from that IR. This paradigm shift reduces integration complexity to linearly additive terms ($M+N$), adhering to a modular design principle that, while standard in classical HPC (e.g., via LLVM) [108], has only recently solidified within the quantum domain.

The landscape of quantum IRs has evolved substantially to address the requirements of hybrid algorithms. Early standards, such as OpenQASM or Quil, mirrored the functionality of classical assembly, describing circuits as linear sequences

of gates [109–111]. Although lightweight and portable, these representations provided limited support for classical control flow (loops, conditionals) and complex variable scoping, thereby limiting their utility for advanced variational algorithms or error-correction routines. Recent initiatives have introduced hierarchical approaches like MLIR (Multi-Level IR) [112], which enable code representation at varying levels of abstraction simultaneously, permitting optimization at the most appropriate semantic layer (e.g., optimizing loop structures prior to gate unrolling) [113].

Nevertheless, the prevailing industrial consensus favors the adoption of QIR, which leverages the LLVM infrastructure [114]. The transition towards LLVM-based representations is a strategic milestone for bridging the gap between QC and HPC. Given that standard HPC compilers (Clang/LLVM) rely on this infrastructure, embedding quantum primitives within the LLVM IR facilitates a unified compilation workflow. In this model, classical and quantum logic can be optimized jointly within a single dependency graph [99]. Furthermore, this approach allows the quantum stack to inherit mature classical optimization passes (e.g., dead code elimination, constant propagation) without duplicating development efforts, while seamlessly enabling interoperability between quantum kernels and existing HPC libraries written in C++ or Fortran. Building upon this extensible infrastructure, this thesis proposes NetQIR, a specialized IR that extends standard QIR with intrinsic operations designed to support DQC.

In summary, the standardization of IRs establishes a stable interface for constructing complex systems. This architectural stability is a prerequisite for the contributions presented in subsequent chapters.

4.2 Distributed Quantum Computing Paradigm

While the compilation techniques discussed in the previous section are essential for optimizing execution on single devices, the physical scalability of monolithic QPU remains a formidable barrier. To overcome the limitations of frequency crowding, crosstalk, and cooling power, the field is shifting towards a modular architecture: DQC [115, 116].

This paradigm proposes the interconnection of P distinct computing nodes (QPUs) to function as a single, large-scale quantum machine. However, the term "distribution" in quantum computing is polysemic. As illustrated in Figure 4.4, depending on the nature of the interconnection, classical or quantum, three distinct operational modes can be identified [117].

4.2.1 Taxonomy of distribution

Let us consider a scenario with P quantum processors available. The utilization strategy depends fundamentally on whether the target circuit fits within a single device and whether quantum communication channels exist between nodes.

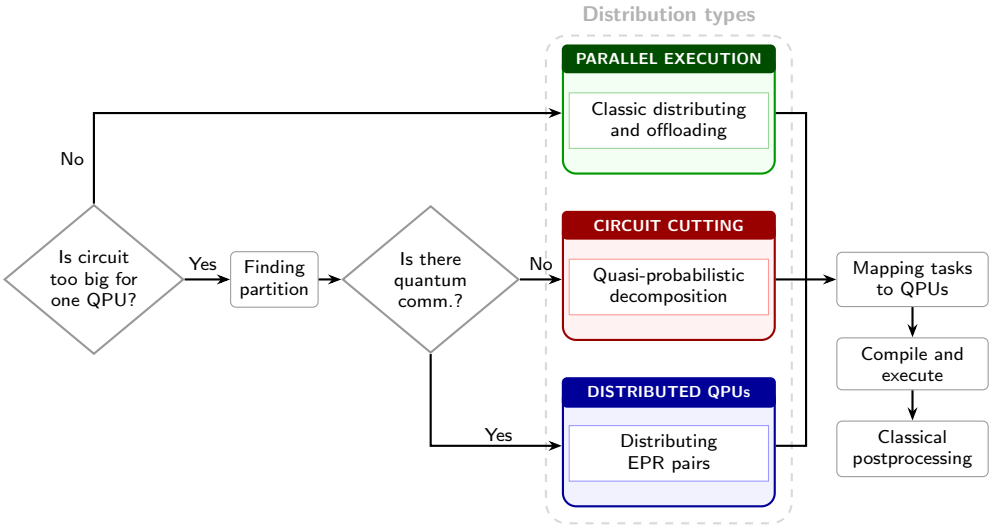


Figure 4.4: Taxonomy of DQC approaches. The strategy depends on the problem size and the availability of quantum links. (1) *Parallel Execution* utilizes unconnected QPUs to accelerate statistical sampling. (2) *Circuit Cutting* allows large circuits to run on smaller, classically connected devices via quasi-probabilistic decomposition. (3) *Distributed QPUs* relies on entanglement distribution to execute non-local operations, forming a true quantum network.

1. **Parallel Execution (Embarrassingly Parallel):** In variational algorithms or quantum machine learning, the goal is often to estimate an expectation value $\langle H \rangle$ by executing the same circuit S times (shots). If the circuit fits within the coherence time and qubit count of a single device, the most efficient strategy is to distribute the workload. By assigning S/P shots to each QPU, the total execution time is reduced by a factor of P. This requires only classical coordination.

2. **Circuit Cutting (Knitting):** When a circuit requires more qubits than available on a single node and no quantum link exists, techniques like Quasi-Probabilistic Decomposition can be employed. The large circuit is "cut" into sub-circuits that fit on individual devices. The results are reconstructed via classical post-processing [118, 119]. However, this comes at a steep cost: the number of shots required to reconstruct the result with the same precision scales exponentially ($\mathcal{O}(c^k)$) with the number of cuts k, where c is a constant overhead factor determined by the type of cut (e.g., wire vs. gate cut). This makes the approach efficient only for weak coupling.

3. **Distributed QPUs (Networked Quantum Computing):** This is the fo-

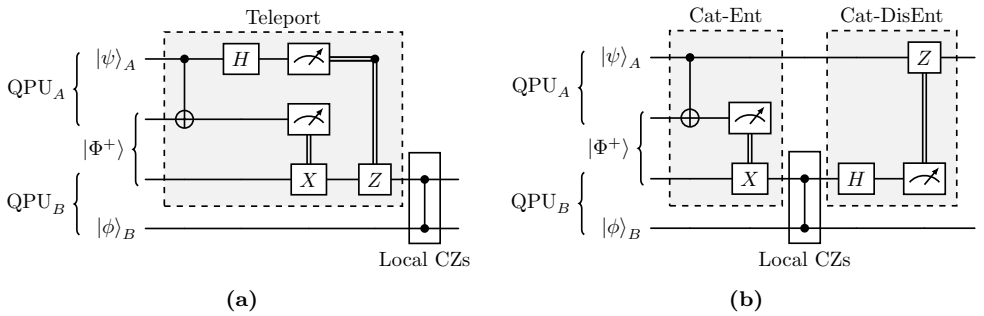


Figure 4.5: Circuit schematic of the *Teledata* and *Telegate* protocol. (a) The quantum state $|\psi_A\rangle$ is teleported from QPU$_A$ to QPU$_B$ utilizing a shared EPR pair. (b) This primitive executes a non-local controlled-Z gate between a control qubit $|\psi_A\rangle$ in QPU$_A$ and a target $|\phi_B\rangle$ in QPU$_B$ using a single shared EPR pair. The process involves an entanglement stage (Cat-Ent) to link the control to the channel, followed by the local interaction, and a disentanglement stage (Cat-DisEnt).

cus of this thesis. It involves QPUs connected by quantum channels that can share quantum information. This approach overcomes the exponential overhead of circuit cutting by utilizing quantum entanglement as a resource [120]. By consuming entangled pairs (Einstein-Podolsky-Rosen (EPR) pairs), nodes can execute operations across the network, effectively creating a "virtual" QPU with the aggregate capacity of all nodes.

4.2.2 Communication protocols

In the Distributed QPUs paradigm, information cannot be simply copied due to the No-Cloning Theorem [121]. Instead, communication relies on the consumption of EPR pairs (maximally entangled Bell states shared between nodes), typically denoted as:

$$|\Phi^+\rangle_{AB} = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B) \tag{4.1}$$

Here, $|\Phi^+\rangle_{AB}$ is an EPR pair involving a qubit in QPU A and another in QPU B.

Using these pairs as a resource, two fundamental primitives defined as "black boxes" for the compiler are established: *Teledata* [66] and *Telegate* [122] (Figure 4.5).

Teledata: State teleportation

The Teledata protocol physically relocates the quantum state of a qubit from a source node A to a destination node B. This approach is necessary when the logic requires the qubit to reside locally at B for subsequent operations. Mathematically, let $|\psi\rangle_A = \alpha|0\rangle_A + \beta|1\rangle_A$ be the state to be transmitted. The system begins with the joint state $|\psi\rangle_A \otimes |\Phi^+\rangle_{AB}$.

The protocol proceeds as follows:

1. **State Entanglement:** Node A interacts the qubit to be sent (q_A) with its local half of the EPR pair (e_A). This involves applying a CNOT gate with q_A as control and e_A as target, followed by a Hadamard gate on q_A. This operation changes the system's basis, correlating the amplitudes of the unknown state with the shared entanglement.

2. **Bell Measurement:** Node A performs a Bell-basis measurement on the qubit to be sent and its half of the entangled pair. This collapses the global state and yields two classical bits, $M_1, M_2 \in \{0, 1\}$.

3. **Feed-forward Correction:** These bits are transmitted via a classical channel to node B. At this stage, qubit from node B has collapsed into the state $|\phi\rangle_B$, which corresponds to the original state $|\psi\rangle$ subjected to a Pauli transformation determined by the measurement outcomes. To recover the original state $|\psi\rangle$, node B applies the corresponding inverse Pauli correction [66]:

$$|\psi\rangle_B = X^{M_2} Z^{M_1} |\phi\rangle_B \tag{4.2}$$

The primary advantage of Teledata is that once $|\psi\rangle$ is reconstructed at node B, it becomes local to that node, allowing an arbitrary number of subsequent gates to be executed without further communication penalties. However, if the algorithm later requires this qubit to act as a control for a target back in node A, it must be teleported back ("round-trip"), doubling the cost to 2 EPR pairs.

Telegate: Remote gate application

The Telegate protocol allows the execution of a non-local controlled unitary U between a control qubit q_A in node A and a target qubit q_B in node B without transporting the control qubit. Instead, it temporarily extends the control superposition across the network to mediate the interaction using a single EPR pair (e_A, e_B). As detailed in [122, 123], the protocol proceeds in three distinct phases:

1. **Cat-Entanglement (Cat-Ent):** The goal is to share the amplitudes of the control qubit into the communication channel. First, a local CNOT is applied between the control q_A and the local EPR half e_A, followed by a measurement on e_A. Depending on the outcome, a correction is applied to e_B. This results in a "cat-like" state where the amplitudes α, β are shared between the original control and the remote proxy e_B:

$$|\Psi_{Cat}\rangle = \alpha|0\rangle_A|0\rangle_{e_B} + \beta|1\rangle_A|1\rangle_{e_B} \tag{4.3}$$

2. **Local Interaction:** With the amplitudes shared, node B performs the intended controlled operation using the proxy e_B as the control and q_B as the target (CU_{e_B,q_B}). Since e_B is perfectly correlated with q_A, the logical state of the target evolves correctly according to the superposition of the control.

3. **Cat-Disentanglement (Cat-DisEnt):** Finally, the protocol decouples the channel to restore the control qubit. A Hadamard gate is applied to e_B followed by a measurement. The result is sent back to node A to apply a final Z-phase correction on q_A if necessary.

The defining advantage of Telegate is that it executes a non-local interaction consuming only one EPR pair. This is half the entanglement cost required to teleport a qubit, perform the gate locally, and teleport it back (Teledata round-trip) [124].

4.2.3 Abstraction layers and the development gap

Implementing these distributed protocols manually is error-prone and tedious. To manage this complexity, in this thesis, we envision the distributed quantum computing architecture as a hierarchical stack, as depicted in Figure 4.6, which decouples the physical implementation from the algorithmic logic [125]. This hierarchy can be categorized into four distinct layers:

1. **Physical Layer:** This is the hardware substrate. It comprises the QPUs, based on technologies such as superconducting circuits or trapped ions, and the physical communication channels.

2. **Network Layer (Entanglement Control):** This layer abstracts the probabilistic nature of the physical links. It manages the generation, purification, and swapping of entanglement. Its primary function is to provide high-fidelity EPR pairs on demand to the upper layers, effectively masking the noise and latency inherent in the quantum channels [126, 127].

3. **Development Layer (System Software):** This layer acts as the middleware between the hardware and the user. It encompasses the compiler, the optimizer, and the runtime environment. Its critical role is to translate abstract quantum operations into synchronized instructions for both the computation nodes (QPUs) and the network controllers. It must decide *when* to request entanglement and *how* to decompose non-local gates into Teledata or Telegate primitives.

4. **Application Layer:** The top-level environment where algorithms are defined using high-level languages (e.g., Qiskit, Q#) or domain-specific libraries. Ideally, code at this level should be agnostic to the underlying topology, focusing solely on the logic of the quantum algorithm (e.g., Quantum Fourier Transform (QFT), Shor's algorithm).

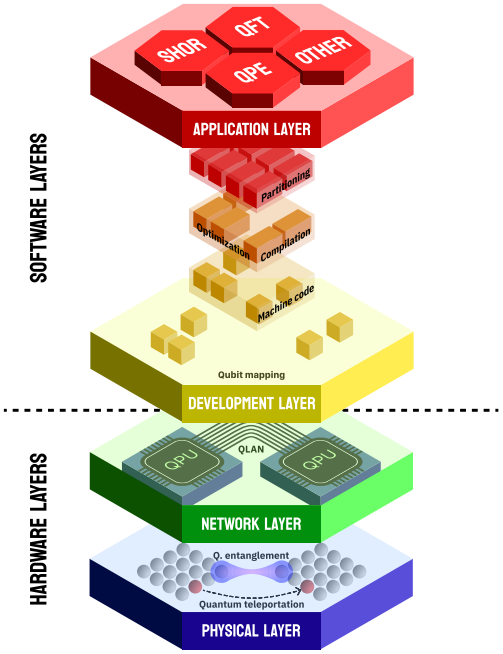


Figure 4.6: Full-Stack Architecture for Distributed Quantum Computing. The stack is organized into Hardware Layers (Physical and Network), handling entanglement generation, and Software Layers (Development and Application). A significant gap exists at the Development Layer, where current tools lack automated mechanisms to map high-level algorithms to distributed primitives, forcing developers to manually manage network protocols.

Currently, programming a distributed quantum computer often requires "manual wiring" of the network protocols. Developers must explicitly invoke primitives such as `genEnt` or `sendEnt` (as seen in low-level proposals such as InQuIR [128]), thereby inextricably coupling the algorithm to the network topology. This lack of abstraction prevents the portability of quantum applications.

The objective of this thesis, specifically addressed in the following section, is to bridge this gap by defining a Distributed Quantum Software Stack. This stack aims to automate the compilation and mapping of high-level circuits onto distributed architectures, rendering the complexity of Teledata and Telegate transparent to the algorithm designer.

4.3 Distributed Quantum Software Stack

The theoretical framework of DQC establishes the physical requirements necessary for scaling quantum processors, such as entanglement distribution and teleportation.

However, the practical utilization of these resources imposes a significant challenge at the software level [126]. To render DQC accessible to algorithm designers, a robust software stack is required to abstract the underlying physical primitives and bridge the gap between high-level algorithm design and low-level hardware execution.

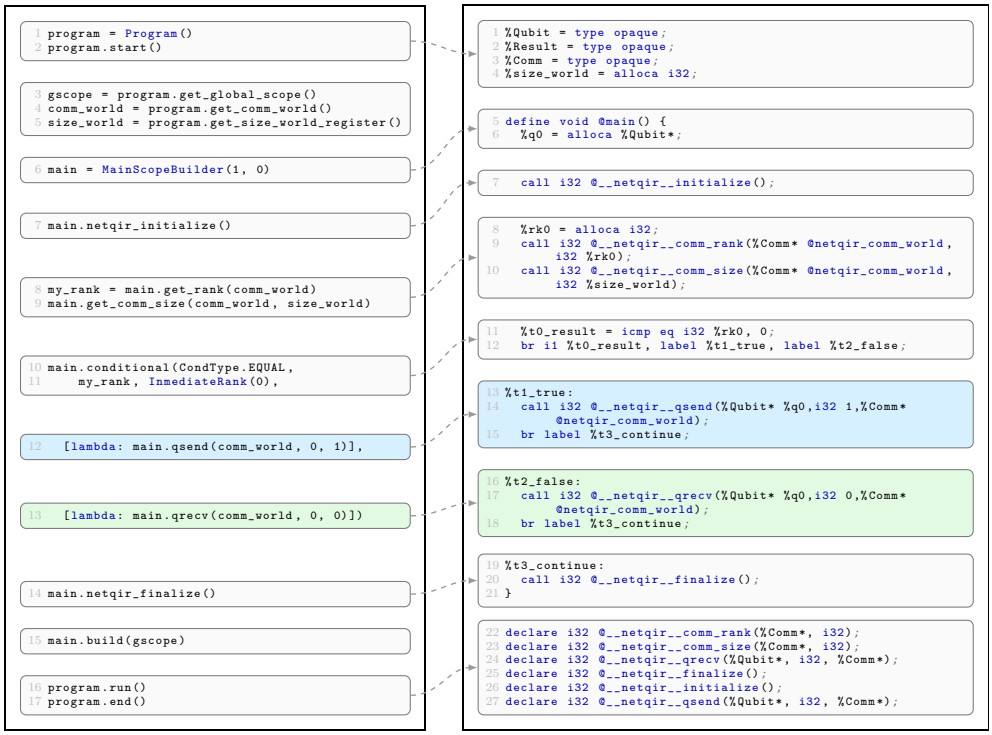
To address this challenge, this thesis identifies the definition of a standard IR as a foundational requirement. In this context, NetQIR (presented in Chapter 7) is introduced as an extension of the QIR specifically designed for DQC. NetQIR serves as the core infrastructure for the development layer, aiming to standardize the design of future distributed quantum programming languages. By adopting the proposed layered abstraction model, NetQIR decouples the development logic from the network implementation. It introduces high-level communication instructions—such as point-to-point and collective operations like the novel *expose* directive—which abstract the complexities of protocols like teledata or telegate. Furthermore, NetQIR leverages the LLVM infrastructure to ensure compatibility with classical HPC toolchains and enables the definition of logical topologies through communicators and groups, akin to classical distributed standards. To ensure a comprehensive development ecosystem, the NetQIR proposal is structured around three fundamental open-source components:

- NetQIR Specification[2]: This is the core formal definition that extends the standard QIR (and by extension, LLVM). It defines the necessary data structures (such as `%Comm` and `%Group`) and intrinsic functions to support distributed operations. It acts as a contract between high-level languages and hardware backends, ensuring that distributed instructions are represented in a hardware-agnostic manner.

- NetQIR SDK (PyNetQIR)[3]: To facilitate the adoption of the standard, a Python-based Software Development Kit (SDK) has been developed. As illustrated in Figure 4.7, this tool abstracts the complexity of writing raw LLVM IR syntax manually. By employing high-level Python scripts and builders, developers can automatically generate compliant NetQIR code, streamlining the compilation process. This component serves as a bridge for integrators targeting the NetQIR standard; a comprehensive description of its architecture and usage is provided in Chapter 7.

- NetQIR Grammar[4]: To enable the creation of parsers and code transformation tools, a formal grammar based on ANother Tool for Language Recognition (ANTLR) is provided. This component allows future developers to define structured listeners and visitors, facilitating the translation of NetQIR into machine code for specific architectures or simulators.

[2]NetQIR specification documentation: `https://netqir.github.io/netqir-spec/`
[3]PyNetQIR SDK GitHub repository: `https://github.com/netqir/netqir-sdk`
[4]NetQIR ANTLR grammar GitHub repository: `https://github.com/NetQIR/netqir-grammar/`



```
program = Program()
program.start()

gscope = program.get_global_scope()
comm_world = program.get_comm_world()
size_world = program.get_size_world_register()

main = MainScopeBuilder(1, 0)

main.netqir_initialize()

my_rank = main.get_rank(comm_world)
main.get_comm_size(comm_world, size_world)

main.conditional(CondType.EQUAL,
    my_rank, InmediateRank(0),

  [lambda: main.qsend(comm_world, 0, 1)],

  [lambda: main.qrecv(comm_world, 0, 0)])

main.netqir_finalize()

main.build(gscope)

program.run()
program.end()
```

(a) PyNetQIR Code

```
%Qubit = type opaque;
%Result = type opaque;
%Comm = type opaque;
%size_world = alloca i32;

define void @main() {
  %q0 = alloca %Qubit*;

  call i32 @__netqir__initialize();

  %rk0 = alloca i32;
  call i32 @__netqir__comm_rank(%Comm* @netqir_comm_world,
      i32 %rk0);
  call i32 @__netqir__comm_size(%Comm* @netqir_comm_world,
      i32 %size_world);

  %t0_result = icmp eq i32 %rk0, 0;
  br i1 %t0_result, label %t1_true, label %t2_false;

%t1_true:
  call i32 @__netqir__qsend(%Qubit* %q0,i32 1,%Comm*
      @netqir_comm_world);
  br label %t3_continue;

%t2_false:
  call i32 @__netqir__qrecv(%Qubit* %q0,i32 0,%Comm*
      @netqir_comm_world);
  br label %t3_continue;

%t3_continue:
  call i32 @__netqir__finalize();
}

declare i32 @__netqir__comm_rank(%Comm*, i32);
declare i32 @__netqir__comm_size(%Comm*, i32);
declare i32 @__netqir__qrecv(%Qubit*, i32, %Comm*);
declare i32 @__netqir__finalize();
declare i32 @__netqir__initialize();
declare i32 @__netqir__qsend(%Qubit*, i32, %Comm*);
```

(b) Generated NetQIR

Figure 4.7: Code generation using the NetQIR SDK. Comparison between (a) a high-level Python script utilizing the PyNetQIR library to define a distributed routine, and (b) the resulting NetQIR code. The SDK abstracts the verbose syntax of the IR, automating the generation of complex control flows and function declarations.

While NetQIR provides the necessary formal specification and IR for machine synthesis, it is inherently designed as a target for compilers rather than a direct interface for human programming. On the other hand, existing tools for programming quantum network controllers, such as the NetQASM SDK [129], often rely on low-level imperative designs. These tools tend to be network-oriented rather than computation-oriented, forcing developers to explicitly manage the intricacies of quantum sockets and entanglement generation. This creates a substantial barrier to the design of complex distributed algorithms.

To bridge this semantic gap, this thesis proposes a higher-level abstraction library: NetQMPI[5], presented in more detail in the Chapter 8.

[5]Code available: `https://github.com/netqir/netqmpi`

NetQMPI is proposed as a high-level Python library designed to bridge the development gap in distributed quantum computing. Drawing direct inspiration from the MPI, NetQMPI brings the familiarity of HPC workflows to the quantum domain standard. Fundamentally, this proposal adopts the SPMD paradigm, a cornerstone of classical HPC [130]. Unlike the client-server or role-based models (e.g., separate `alice.py` and `bob.py` scripts) often required by lower-level tools, the SPMD approach enables the execution of a single source code file across all nodes in the network. The execution flow diverges locally on each node based on its unique identifier, known as the *rank*.

A critical architectural decision in the design of NetQMPI is its implementation as a wrapper built directly on top of the NetQASM SDK. Rather than defining a new proprietary compilation target, NetQMPI generates standard NetQASM instructions. This ensures native compatibility with any backend that supports the NetQASM standard, including emerging physical DQC clusters and established or high-fidelity network simulators such as NetSquid [131] and SimulaQron [132]. Consequently, researchers can prototype algorithms using NetQMPI on a laptop and seamlessly deploy them to large-scale simulations or hardware testbeds without code modification.

Architecture and process management

The integration of the SPMD paradigm over the NetQASM infrastructure is mediated by a dedicated Middleware Layer, as illustrated in Figure 4.8. This layer is responsible for abstracting the underlying node initialization. When a NetQMPI program is launched, the middleware automatically injects the topology-specific context into the runtime, i.e., the size (S), which is the total number of quantum nodes (processes) involved in the computation, and the rank (R), a unique integer identifier $R \in \{0, 1, \ldots, S-1\}$ assigned to the local node.

Listing 4.1 demonstrates a minimal "Hello World" example. The user instantiates a communicator object, which provides access to the node's rank. Conditional logic based on this rank allows distinct behaviors (e.g., Node 0 prepares a state, Node 1 measures it) to be defined within a unified codebase.

```
from netqmpi.sdk.communicator import QMPICommunicator

def main(app_config=None, rank=0, size=1):
    # The rank is injected by NetQMPI
    # and used to initialize the local communicator
    COMM_WORLD = QMPICommunicator(rank, size, app_config)
    print(f"Hello, rank={rank} of {size} processes")

    # Run: $ netqmpi -n 3 script.py
```

Listing 4.1: An example of the rank and size variable injection with the global communicator creation.

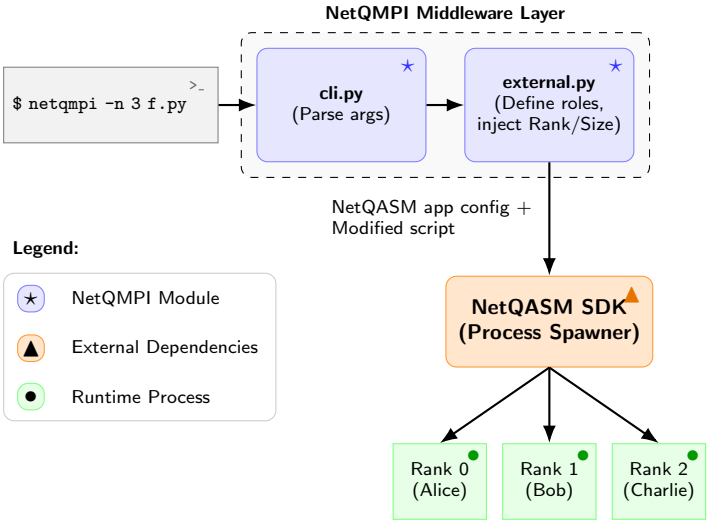


Figure 4.8: Execution flow of a NetQMPI application. The `external.py` module coordinates the multiprocess execution environment by injecting process rank information into the unified user-level script and delegating low-level process lifecycle management and orchestration responsibilities to the NetQASM SDK.

The advantages of this abstraction become evident when implementing quantum communication protocols. In the raw NetQASM SDK, the communication is modeled similarly to classical sockets, requiring explicit management of the entanglement lifecycle and classical message exchange. As shown in Listing 4.2 (Alice's side), the developer must manually request EPR pairs, perform local gates (CNOT, Hadamard), measure, and explicitly send the classical corrections. Furthermore, a complementary and distinct script would be required for the receiver (Bob) to listen for these corrections and apply the Pauli-X/Z corrections.

```
# Create a socket to send classical information
socket = Socket("Bob", "Alice", log_config=log_config)

# Create an EPR socket for entanglement generation
epr_socket = EPRSocket("Bob")

with NetQASMConnection(epr_sockets=[epr_socket]) as conn:
    # Create a qubit to teleport
    q = Qubit(conn)

    # Create EPR pairs
    epr = epr_socket.create_keep()[0]

    # Teleport
```

```
    q.cnot(epr)
    q.H()
    m1 = q.measure()
    m2 = epr.measure()

# Send the correction information
m1, m2 = int(m1), int(m2)

socket.send_structured(StructuredMessage("Corrections", (m1, m2)))
```

Listing 4.2: Low-level implementation of the Teledata protocol (Sender/Alice side) using the NetQASM SDK. The listing illustrates the imperative instructions required to manage the source node. Crucially, this code represents only half of the protocol; a complementary routine is strictly necessary on the receiver node (Bob) to asynchronously listen for the classical correction signals (m_1, m_2) and apply the corresponding Pauli operations to successfully reconstruct the quantum state.

In contrast, NetQMPI encapsulates the entire Teledata protocol within semantic primitives: `qsend` and `qrecv`. Listing 4.3 presents the implementation of the same logic. The complexity of entanglement generation, Bell measurements, and feed-forward corrections is handled transparently by the library. The code is concise, readable, and agnostic to the underlying hardware implementation, adopting a subtly adapted MPI syntax, specifically tailored to the requirements of quantum computing while remaining familiar and accessible to the HPC community.

```
def main(app_config=None, rank, size):
    # Initialize the communicator
    comm = QMPICommunicator(rank, size, app_config)

    if rank == 0:
        # Sender Logic (Alice)
        q = Qubit(comm.connection)
        q.H() # Prepare state |+>

        # Semantic send: "Send qubit q to Rank 1"
        # No EPR management required
        comm.qsend(q, 1)
        print("Rank 0: Qubit sent")

    elif rank == 1:
        # Receiver Logic (Bob)
        # Semantic receive: "Get qubit from Rank 0"
        q_received = comm.qrecv(0)

        # The qubit is ready for use
        m = q_received.measure()
        print(f"Rank 1: Qubit received, measure: {m}")
```

Listing 4.3: High-level implementation of qubit transfer using NetQMPI. The `qsend` and `qrecv` primitives automatically negotiate the Teledata protocol (entanglement consumption and corrections) transparently to the user.

4.4 Benchmarking of distributed quantum computing emulators

The progression of DQC is fundamentally contingent upon the availability and maturity of specialized software frameworks. Due to the limited accessibility and substantial deployment costs associated with physical quantum networks, researchers predominantly rely on simulation and emulation environments to validate distributed quantum protocols and to conduct performance analyses [133, 134]. Nonetheless, the current ecosystem of quantum simulation tools is highly heterogeneous and characterized by notable terminological inconsistencies. This section, therefore, introduces a formal conceptual framework and motivates the need for systematic and rigorous benchmarking of such tools, the methodology and results of which are presented in Chapter 10.

Unlike monolithic quantum computing, where the primary metric for a simulator is often the maximum number of qubits or the speed of unitary execution, DQC introduces a complex set of time-dependent variables. The fidelity of a distributed algorithm is not solely determined by gate errors but is intrinsically linked to network latencies, entanglement generation rates, and qubit decoherence while waiting for remote information.

4.4.1 Simulator vs. emulator

Within the domain of quantum technologies, the terms "simulator" and "emulator" are often used interchangeably, although they actually refer to conceptually distinct methodologies for reproducing quantum behavior.

- **Simulator:** Focuses on the *mathematical outcome*. It calculates the evolution of the quantum state vector or density matrix according to the laws of quantum mechanics. It answers the question: "What is the result of this circuit?"
- **Emulator:** Focuses on the *system behavior*. It mimics the target hardware's internal architecture, timing, and control flow in real-time or near real-time. It answers the question: "How does the system behave during execution?"

Table 4.1 classifies these concepts based on whether they target the physical machine (Quantum Computer) or the abstract logic (Quantum Circuit).

4.4.2 Parallel simulation vs. emulation of DQC

A prevalent source of confusion in the literature is the distinction between using parallel resources to simulate quantum mechanics versus emulating a distributed quantum architecture.

	Quantum Computer	Quantum Circuit
Simulator	*Possible.* Simulates the Hamiltonian dynamics and physical pulses of qubits (e.g., solving the Schrödinger equation for a Transmon qubit).	*Possible.* Simulates the logical execution of gates to predict state probabilities (e.g., Qiskit Aer `statevector_simulator`).
Emulator	*Possible.* Mimics the full control stack, including FPGA timing, latencies, and noise injection (e.g., QEMU for control electronics).	*Conceptually Redundant.* Emulating the abstract logic of a "Quantum Circuit" inherently implies introducing hardware constraints (timing, topology). Therefore, this distinction collapses: to emulate a circuit is effectively to emulate the Quantum Computer itself.

Table 4.1: Distinction between Simulator and Emulator. Simulators focus on mathematical correctness, while emulators focus on architectural behavior and temporal constraints.

1. **Parallel Quantum Simulation:** This refers to the use of classical HPC techniques (e.g., MPI, OpenMP) to simulate large monolithic quantum states [135]. Since the memory required to store a state vector scales exponentially ($\mathcal{O}(2^n)$), simulating 40+ qubits requires distributing the state vector across the Random Access Memory (RAM) of multiple classical nodes [136]. Here, the interconnection network is merely a classical utility employed to manage the memory distributed across the cluster.

2. **Emulation of DQC:** This refers to mimicking the protocols, timing, and behaviors of physically distributed quantum processors. The goal is not necessarily to compute the mathematical outcome of a massive state, but to model the *interactions* (e.g., teleportation fidelity, entanglement generation rates, EPR exchange latency) between distinct quantum nodes [131, 137]. Here, the quantum network itself is the primary object of study.

Table 4.2 categorizes existing software tools according to these dimensions, illustrating that a tool can be monolithic yet simulate DQC (e.g., NetSquid single-threaded) or parallel yet simulate a single QC (e.g., mpiQulacs).

The experimental evaluation and benchmarking of these specific architectures, focusing on the performance trade-offs between monolithic and distributed emulation backends, are extensively discussed in Chapter 10.

	Quantum Computing (QC)	Distributed Quantum Computing (DQC)
Monolithic	**Examples:** Qiskit Aer, Cirq. Runs on a single classical node. Simulates a local circuit (e.g., up to 30 qubits).	**Examples:** NetQMPI. Runs on a single node but can models multiple QPUs and network delays using discrete-event simulation.
Parallel	**Examples:** mpiQulacs. Uses MPI to distribute a single massive state vector across a supercomputer.	**Examples:** None currently known in the literature.

Table 4.2: Classification of Emulation and Simulation Approaches. The distinction lies in the target system: simulating a single large quantum state using parallel hardware (Parallel QC) versus simulating the interaction between networked quantum devices (DQC).

4.5 Applications of Distributed Quantum Computing

To conclude the discussion on the challenges and potential of the DQC paradigm, this section introduces a concrete algorithmic optimization proposed in this thesis. The algorithm selected for this analysis is the Inverse Quantum Fourier Transform (iQFT). The iQFT constitutes a fundamental primitive in quantum information theory, serving as the computational backbone for pivotal algorithms such as QPE and Shor's algorithm for integer factorization [7, 138].

In the context of the abstraction layer model proposed in previous sections, the iQFT represents a worst-case scenario for distributed architectures due to its high connectivity requirements [124]. As illustrated in Figure 4.9, this thesis proposes a distributed optimization of the iQFT that leverages the mathematical properties of the transformation to mitigate the communication overhead.

For a more rigorous mathematical treatment and experimental validation, this process is detailed in Chapter 11.

4.5.1 Mathematical formulation and interaction schemes

Mathematically, the iQFT maps state from the Fourier basis back to the computational basis state $|x\rangle$ (where $x = x_1 x_2 \dots x_n$ in binary representation). Figure 4.9 shows the circuit implementation for the iQFT, which is constructed from a sequence of Hadamard gates and controlled-phase rotation gates, denoted as $CP(\theta_k)$. The phase shift θ_k applied between two qubits depends on the index difference. Specifically, for a control qubit $i^{\bullet}$ and a target qubit $i^{\square}$, the rotation angle is defined as:

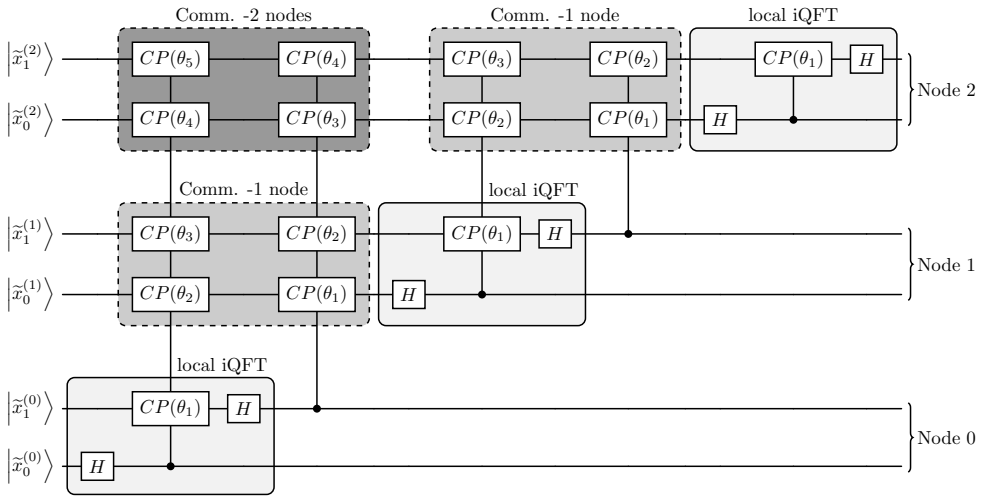


Figure 4.9: Distributed decomposition of a 6-qubit iQFT across 3 nodes ($p = 0, 1, 2$). The circuit is restructured into Local iQFT blocks (light gray), which require no external communication, and Communication Blocks (darker gray), where node p interacts with qubits from previous nodes $p' < p$. The phase angles follow $\theta_k = \pi/2^k$.

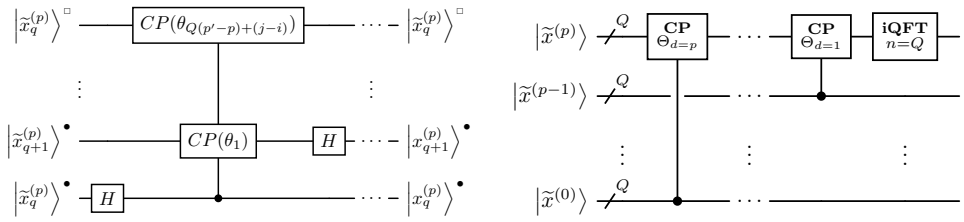
$$\theta(i^{\bullet}, i^{\square}) = -\frac{\pi}{2^{i^{\square} - i^{\bullet}}}. \tag{4.4}$$

When this monolithic circuit is embedded in a distributed architecture comprising multiple QPU nodes, the resulting two-qubit interactions naturally bifurcate into intra-node (local) and inter-node (non-local) operations. To explicitly formalize the mapping of the global quantum register onto the distributed topology, we define a quantum network consisting of P nodes, identified by the index $p \in \{0, 1, \ldots, P-1\}$. Assuming a homogeneous partition, each node hosts exactly Q local qubits, yielding a total logical register size of $n = P \cdot Q$.

To address individual qubits within this architecture, we adopt the notation $|x_q^{(p)}\rangle$. In this convention, the superscript p denotes the index of the host quantum node, whereas the subscript $q \in \{0, 1, \ldots, Q-1\}$ designates the local index of that qubit within the register of node p. This structural distinction is essential, as it enables the deterministic mapping of a global logical qubit index to a concrete physical coordinate (q, p) using the index function

$$I(q, p) = p \cdot Q + q. \tag{4.5}$$

Furthermore, to unequivocally distinguish the directionality of non-local entanglement and quantum operations, we introduce specific visual modifiers to denote functional roles. At the network level, we apply $p^{\bullet}$ and $p^{\square}$ to specify nodes acting



(a) Interaction between qubits. **(b)** Interaction between nodes.

Figure 4.10: Distributed iQFT generic interaction scheme. (a) Interaction between qubits. Target qubit $q^{\square}$ in node $p^{\square}$ and a control qubit $q^{\bullet}$ in a distant node $p^{\square}$. The rotation angle is determined by the global index difference, formalized as θ_k where $k = Q(p^{\square} - p^{\bullet}) + (q^{\square} - q^{\bullet})$. (b) Interaction between the previous nodes for an arbitraty node p. The circuit illustrates the sequential application of communication blocks $\mathbf{\Theta_d}$, starting from the most distant dependency (Node 0, $d = p$) up to the nearest (Node $p-1$, $d = 1$). The execution of this sequence assumes that all previous nodes $0, \ldots, p-1$ have already initiated their corresponding operations to act as controls. The process concludes with a local iQFT on the Q qubits of node p.

as control and target, respectively. Similarly, at the qubit level, $q^{\bullet}$ and $q^{\square}$ explicitly designate a control and target qubit. For notational brevity when describing complex distributed interactions, we establish the shorthand $|x_i^{(p)}\rangle^{\bullet} \equiv |x_{i^{\bullet}}^{(p^{\bullet})}\rangle$, which concisely indicates that both the physical qubit and its host node serve the control role. The same structural equivalence applies symmetrically to the target notation, where $|x_i^{(p)}\rangle^{\square} \equiv |x_{i^{\square}}^{(p^{\square})}\rangle$.

Distributed interaction blocks

To systematically analyze the distributed execution flow, two generic interaction schemes are identified and depicted in Figure 4.10. These schemes abstract the complex dependency graph of the distributed iQFT into fundamental building blocks:

- Figure 4.10a, interaction between two generic qubits: Represents the interaction between a control qubit $q^{\bullet}$ located in node $p^{\bullet}$ and a target qubit $q^{\square}$ in node $p^{\square}$. This requires a standard telegate or teledata protocol.
- Figure 4.10b, interaction between the previous nodes for an arbitrary node p: As shown in the macro-architecture of Figure 4.9, the algorithm is structured into communication blocks where a set of qubits in the current node must interact with qubits from a previous node.

Formally, the interaction between two distributed quantum nodes can be characterized by a collection of controlled-phase gates specified by a set of angles denoted by Θ. Adopting the notation introduced in this section, let Θ_d represent the subset of phases required to implement the interaction between a control node $p^{\bullet}$ and a target

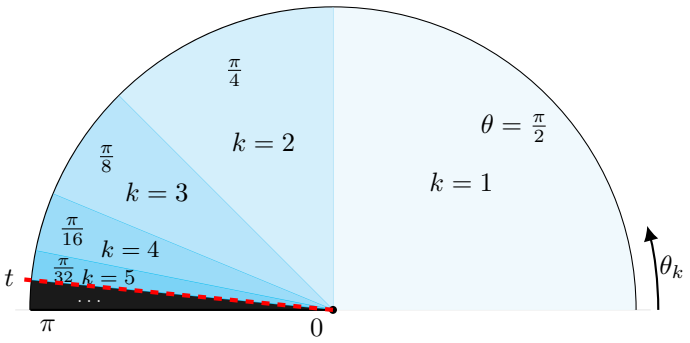


Figure 4.11: Geometric representation of the accumulated phase in the iQFT showing the rapid decay of rotation angles $\theta_k = \pi/2^k$. The explicit sectors for $k = 1$ to $k = 5$ are shown as an example. The threshold cutoff t (red dashed line) indicates the truncation point; interactions beyond this limit (black sector) contribute a negligible phase shift bounded by $\varepsilon\pi$.

node $p^{\square}$. These nodes are separated by a logical network distance d, which is strictly defined as the offset between their node indices, such that $p^{\square} = p^{\bullet} + d$. Under the assumption of a homogeneous architecture, in which each node contains Q qubits, this set is defined as:

$$\Theta_d = \{\theta\left(I\left(q^{\bullet}, p^{\bullet}\right), I\left(q^{\square}, p^{\square}\right)\right) \mid p^{\square} = p^{\bullet} + d\}\,, \tag{4.6}$$

where I is the function defined in Eq. 4.5, Q denotes the number of qubits per node and d represents the integer distance between the interacting nodes. The indices $q^{\bullet}$ and $q^{\square}$ iterate over the local qubits of the respective nodes.

4.5.2 Gate pruning and convergence analysis

The central optimization strategy relies on the analytic properties of the controlled-phase gates, $CP(\theta)$. The action of this gate on the computational basis states is diagonal, defined as $CP(\theta) = \mathrm{diag}(1, 1, 1, e^{i\theta})$. Consequently, the interaction introduces a relative phase shift whose magnitude is solely determined by the parameter θ.

Consider the sequence of rotation angles defined by the geometric progression $S_k = \pi/2^k$. This behavior is visually depicted in Figure 4.11, which plots the magnitude of the rotation angle θ_k as a function of the logical distance k. The graph clearly illustrates the sharp exponential decay, highlighting how rapidly the phase shift contributions diminish. For sufficiently large distances, the rotation angle becomes infinitesimal, rendering the gate operation practically indistinguishable from the identity operator [139].

To demonstrate the convergence of the total phase, we analyze the limit of the partial sums. Let Σ_N be the sum of the first N terms:

$$\Sigma_N = \sum_{k=1}^{N} \frac{\pi}{2^k} = \pi \sum_{k=1}^{N} \left(\frac{1}{2}\right)^k \tag{4.7}$$

Using the closed-form expression for the partial sum of a geometric series $\sum_{k=1}^{N} r^k = r(1-r^N)/(1-r)$ with ratio $r = 1/2$, we obtain:

$$\Sigma_N = \pi \left[\frac{1/2(1-(1/2)^N)}{1-1/2}\right] = \pi \left(1 - \frac{1}{2^N}\right) \tag{4.8}$$

Taking the limit as the system size scales arbitrarily ($N \to \infty$), the term $1/2^N$ vanishes, yielding the convergent result $\sum_N = \pi$.

This observation allows for the definition of an approximation strategy based on a phase fraction, denoted as ε. If the rotation angle $\theta(i^\bullet, i^\square)$ falls below this fraction, the gate is effectively approximated as the identity operator I, and the corresponding communication is suppressed:

$$CP(\theta(i^\bullet, i^\square)) \approx I \quad \text{if} \quad \theta(i^\bullet, i^\square) < \varepsilon\pi \tag{4.9}$$

Pruning threshold and communication horizon

To systematically apply the approximation, we first define the pruning threshold, denoted as t. This integer parameter represents the maximum logical difference between each pair of qubits (k) for which the phase rotation is considered significant. Given a phase fraction ε, t is determined by the ceiling function:

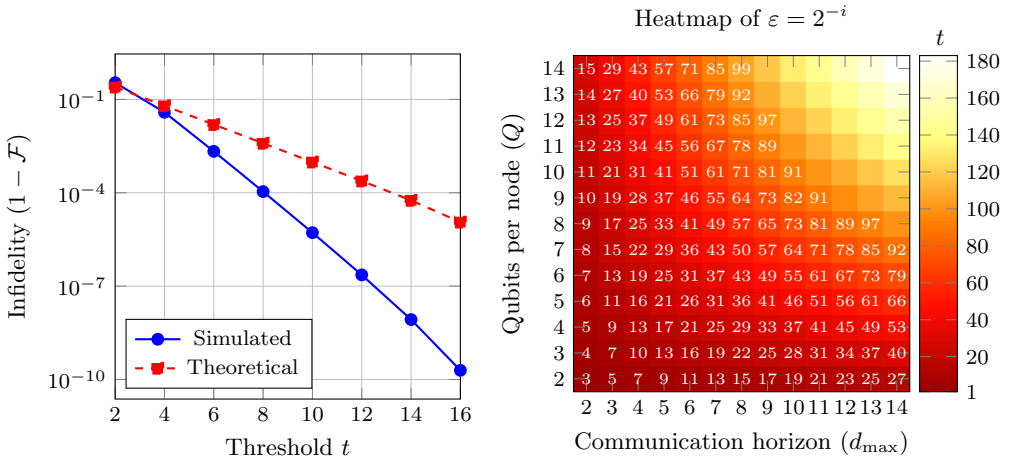
$$t = \lceil -\log_2(\varepsilon) \rceil \tag{4.10}$$

Consequently, any gate $CP(\theta_k)$ with $k > t$ implies a rotation angle smaller than the tolerance, justifying its removal from the circuit.

This logical constraint naturally defines a physical **communication horizon**, denoted as $d_{\max}$. This parameter represents the maximum rank distance between two nodes $p^\bullet$ and $p^\square$ such that quantum communication is maintained. Incorporating the local register capacity Q (qubits per node), $d_{\max}$ is derived from the threshold t as:

$$d_{\max} = \left\lfloor \frac{t-1}{Q} \right\rfloor + 1 \tag{4.11}$$

Any interaction that requires a logical distance spanning more than $d_{\max}$ nodes is pruned. This transformation effectively localizes the algorithm, reducing the network topology requirements from a fully connected graph (all-to-all) to a nearest-neighbor configuration restricted by the radius $d_{\max}$.



(a) Fidelity **(b)** $d_{\max}$

Figure 4.12: Fidelity and accuracy analysis of the distributed pruning strategy. (a) Evolution of the algorithmic infidelity $(1 - \mathcal{F})$ as a function of the pruning threshold t for a fixed local register size of $Q = 18$, illustrating the trade-off between aggressive gate reduction and state purity. (b) Heatmap characterizing the theoretical error bound relative to the communication horizon $d_{\max}$ and the local register size Q. The integer values in the cells represent the magnitude of error suppression i (corresponding to an error bound $\varepsilon \approx 2^{-i}$), demonstrating that linear increases in the communication horizon yield exponential improvements in algorithmic accuracy.

4.5.3 Performance and efficiency analysis

The viability of this approximation is validated through the analysis of algorithmic fidelity and resource efficiency, as detailed in Figures 4.12 and 4.13.

Fidelity and infidelity

The error introduced by the approximation is quantified using the *Infidelity* metric, defined as $\mathcal{I} = 1 - \mathcal{F}$, where $\mathcal{F}$ is the fidelity between the ideal state $|\psi_{\text{ideal}}\rangle$ and the approximate state $|\psi_{\text{real}}\rangle$.

$$\mathcal{F} = |\langle\psi_{\text{ideal}}|\psi_{\text{real}}\rangle|^2 \tag{4.12}$$

As illustrated in Figure 4.12a, the results indicate that limiting the number of interacting nodes (imposing a finite $d_{\max}$) is completely viable. The system maintains a low infidelity even for moderate pruning thresholds. This demonstrates that the significant overhead of long-range entanglement distribution yields diminishing returns in terms of algorithmic accuracy.

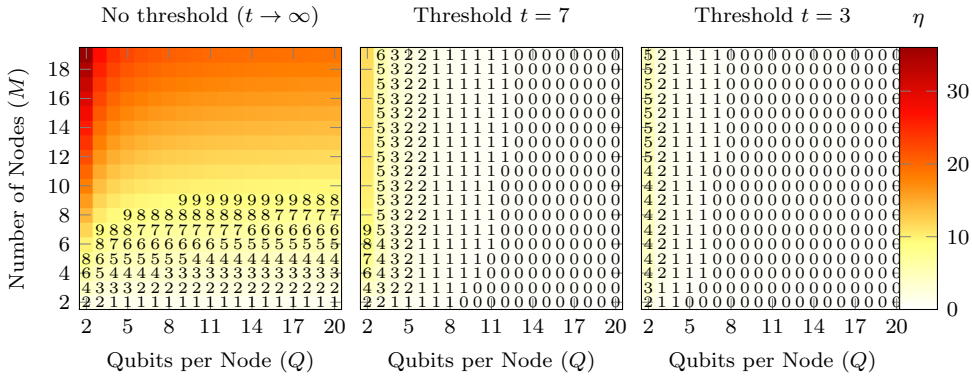


Figure 4.13: Evolution of the Coupling Ratio η, defined as the ratio between remote and local controlled-phase gates ($\eta = N_{\text{non-local}}/N_{\text{local}}$), as a function of the pruning threshold. Note that higher values indicate greater reliance on inter-node communication, which is the primary bottleneck in distributed architectures. The plot demonstrates how the proposed pruning strategy successfully minimizes this ratio ($\eta \to 0$), effectively decoupling the nodes and shifting the computational load towards efficient local processing.

Coupling ratio and resource consumption

To quantify the impact on network resources, we introduce the Coupling Ratio (η), defined as the ratio of non-local gates to the local number of gates at a given node.

$$\eta = \frac{N_{\text{non-local}}}{N_{\text{local}}} \tag{4.13}$$

In a naive distributed implementation, the generation of EPR pairs scales quadratically, $\mathcal{O}(N^2)$. However, as evidenced in Figure 4.13, the application of the horizon $d_{\max}$ drastically alters this scaling. The coupling ratio transitions from a growth trend to a practically constant value ($\mathcal{O}(1)$) as the system size increases.

A low coupling ratio implies a significantly reduced demand for inter-node communication. Since each non-local gate requires the consumption of EPR pairs (via Teledata or Telegate), reducing η directly translates to a lower overhead in entanglement generation and classical synchronization.

The analysis presented in this case study leads to a critical conclusion for the future of the field: DQC offers not only a pathway to scale the number of available qubits but also potential for algorithmic efficiency. If the software stack can intelligently identify and prune negligible communications, as proposed in the optimization of the iQFT, the distributed overhead can be minimized. The full demonstrations and experiments are presented in Chapter 11.

Chapter 5

Distributed Quantum Computing, from single QPU to High Performance Quantum Computing

The core content of this chapter is from the research presented in the following peer-reviewed article:

> D. Barral, F. J. Cardama, G. Díaz-Camacho, D. Faílde, I. F. Llovo, M. Mussa-Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing", *Computer Science Review*, vol. 57, p. 100 747, 2025, ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2025.100747
>
> - Impact Summary: Article published in a JCR 2024 first quartile (Q1) journal.

This publication specifically addresses and fulfills Objective O1 b) of this thesis. For a comprehensive overview of the publication's quality indicators (Impact Factor and Quartile), the reader is referred to the Results: published articles from Contributions section.

Computer Science Review 57 (2025) 100747

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing

David Barral [a], F. Javier Cardama [b], Guillermo Díaz-Camacho [a], Daniel Faílde [a], Iago F. Llovo [a], Mariamo Mussa-Juane [a], Jorge Vázquez-Pérez [a,b], Juan Villasuso [a], César Piñeiro [c], Natalia Costas [a], Juan C. Pichel [b,c], Tomás F. Pena [b,c,*], Andrés Gómez [a]

[a] Galicia Supercomputing Center (CESGA), Avda. de Vigo S/N, Santiago de Compostela, 15705, Spain
[b] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain
[c] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain

ARTICLE INFO

Keywords:
Distributed quantum computing
High-performance computing
Teleportation
Quantum networks
Distributed quantum compilers
Circuit knitting
Distributed quantum applications

ABSTRACT

The emerging field of quantum computing has shown it might change how we process information by using the unique principles of quantum mechanics. As researchers continue to push the boundaries of quantum technologies to unprecedented levels, distributed quantum computing raises as an obvious path to explore with the aim of boosting the computational power of current quantum systems. This paper presents a comprehensive survey of the current state of the art in the distributed quantum computing field, exploring its foundational principles, landscape of achievements, challenges, and promising directions for further research. From quantum communication protocols to entanglement-based distributed algorithms, each aspect contributes to the mosaic of distributed quantum computing, making it an attractive approach to address the limitations of classical computing. Our objective is to offer a comprehensive review that serves both experts in the field and researchers or enthusiasts in quantum computing looking for a starting point to explore the area of distributed quantum computing.

Contents

* Correspondence to: Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain.
E-mail address: tf.pena@usc.es (T.F. Pena).

https://doi.org/10.1016/j.cosrev.2025.100747
Received 10 April 2024; Received in revised form 24 February 2025; Accepted 9 March 2025
Available online 24 March 2025

1. Introduction

In pursuing superior computational abilities, quantum computing has emerged as a promising frontier with huge potential. While individual quantum systems have shown impressive capabilities, distributed quantum computing introduces a new approach that could vastly increase computational power. This study aims to explore in depth the current landscape of DQC, also known in certain literature as modular quantum computation, from physical devices and interconnection networks to distributed algorithms. In this review, we will analyze the different solutions proposed and the challenges posed by this rapidly advancing field.

As we examine distributed quantum systems more closely, it becomes clear that collaborative and interconnected quantum processors are essential for overcoming the constraints faced by standalone systems. Problems of both fundamental origin – decoherence, dissipation, and crosstalk – and practical origin – processor topology, cabling, connectors, and control electronics – hinder the fabrication of ultra-large Quantum Processing Units (QPUs) [1]. It is thus foreseeable in the short term that quantum computers will not scale in a local device with a large number of qubits in a single quantum processor. A distributed infrastructure with several quantum processors that contain a limited number of qubits could overcome this difficulty. In fact, there is a clear consensus among leading academic and industry stakeholders that the practical realization of large-scale quantum processors should adopt a distributed approach based on clusters of small, modular quantum chips within a network infrastructure, with classical and/or quantum communications [2–4]. QPUs are intended to be seamlessly integrated into a classical High-Perfomance Computing (HPC) infrastructure, alongside CPUs, GPUs, and other hardware accelerators [5–9]. This integration allows for their utilization in collaboration within a shared development environment, leading to what is already called quantum-centric supercomputing centers [10].

As an example of this trend, IBM recently unveiled Quantum System Two [11], a modular architecture that will serve as the basis for building their new quantum-centric HPC infrastructures. The model unveiled features three IBM Quantum Heron processors, each with 133 fixed-frequency qubits and tunable couplers. According to IBM, Heron yields a 3-5x improvement in performance with respect to the previous 127-qubit Eagle processor, virtually eliminating crosstalk.

However, the interest in DQC is not new. We have to go back to the end of the 20th century to find the first works that analyzed the possibility of using non-local effects to perform distributed computing [12,13]. This interest grew after Cirac et al.'s work, where it was shown that DQC is superior to classical computing for the phase estimation problem even under non-ideal conditions [14]. Shortly after, Eisert et al. [15] and Collins et al. [16] took a step forward, introducing resource-optimized protocols for non-local quantum gates necessary to move from specific problems like phase estimation to universal quantum computing. At the same time, DiVincenzo [17] included, in his famous criteria for a quantum computer, two additional not-so-well-known items related to DQC and the interconnection of QPUs: the ability to interconnect stationary and flying qubits and to transmit flying qubits between specified locations faithfully.

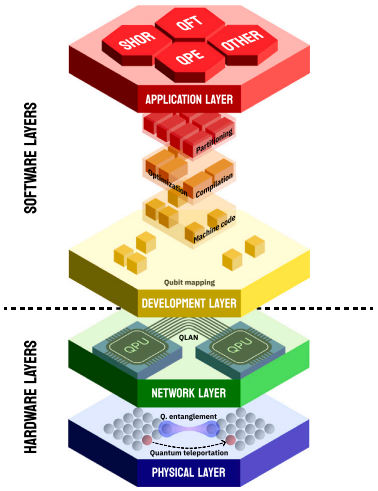


Fig. 1. Layered model for distributed quantum computing.

After the first theoretical studies on the feasibility of DQC, a series of proposals for experimental realizations began to appear gradually [18–21]. At the same time, several interesting developments regarding DQC algorithms were made, such as the distributed versions of the Grover and Shor algorithms [22,23]. The first taxonomy of DQC systems was proposed by Yepez [24] in the early 2000s, where two types of systems were described: those with entanglement between nodes, called type-I, and those with only inter-node classical communication, called type-II. Jozsa and Linden later demonstrated that a type-II quantum computer cannot achieve exponential speedup when the computation requires entanglement across the full set of qubits [25].

Considering these initial works as a starting point, this review extensively examines the current advancements in the field of DQC, extending and updating previous surveys on this subject. The previous[1] survey by Caleffi et al. [26] provides a comprehensive overview of DQC archetypes. It covers a range of configurations, from single QPU with multi-core execution to single datacenter with multiple

[1] During the review process of this article, we were notified that an updated version of the Caleffi et al. survey available in arXiv was published in the Computer Networks journal.

QPUs and even multi-farm networks, also comparing them to their classical counterparts. Additionally, it offers a detailed section on simulation tools for DQC development, organized into three distinct layers: hardware-oriented, protocol-oriented, and application-oriented.

Our review is specifically focused on the integration of HPC with DQC, and we excluded multi-farm networks due to the substantial latency introduced by geographically separated centers. Instead, we concentrate on multi-core and multi-QPU configurations, drawing parallels with classical HPC paradigms. For example, we explore techniques such as circuit cutting and embarrassingly parallel distribution of circuits, whenever fully realized quantum networks may not yet be available. Additionally, we emphasize the critical importance of the physical layer in DQC systems.

To facilitate the readers' understanding, this survey is structured according to a layered model, as depicted in Fig. 1, similar to the full-stack architecture presented by [27], the abstract model in [28], or the DQC simulator tools structure in [26].

The two lower layers in Fig. 1 encompass the hardware developments needed to implement a distributed quantum system and would be equivalent to the three lower layers of the classical OSI model. So, the physical layer refers to the mechanisms that allow two physically separated QPUs to be connected, while the network layer defines how to establish communication between multiple QPUs. Directly above this layer, we discuss advances in development tools that allow applications to be distributed and executed on a distributed quantum system, including partitioning, compilation, optimization, and mapping algorithms. Finally, in the uppermost layer, we address distributed algorithms. It is important to note that these layers are interdependent, with each layer influencing those immediately preceding and succeeding. For instance, the development of a compiler is influenced by the underlying hardware and provides support for different partitioning techniques in the application layer.

Following this structure, the review is organized as follows. Section 2 describes the available quantum mechanical tools to transmit quantum information. In Section 3, we present proposals oriented to creating networks interconnecting multiple QPUs. Next, Section 4 discusses solutions that allow applications to run in distributed environments, including partitioning, distribution, compilation, and mapping techniques. Section 5 presents different proposals for applications running in these environments. We will end the paper with a summary of the current state of the art and open lines in the field.

2. Physical layer for distributed quantum computing

DQC aims at performing arbitrary computational tasks between unknown quantum states at the distant nodes of a quantum network. These networks, identically to their classical counterparts, coordinate and distribute information across devices. However, quantum networks have multiple features and limitations that make these tasks difficult, primarily arising from the *no-cloning* theorem: arbitrary quantum states cannot be *perfectly* copied; therefore, quantum information cannot be replicated and broadcast [29]. Fortunately, the properties of quantum systems can be exploited in a way that allows us to circumvent this impediment and reliably transmit quantum information or control quantum systems remotely. This section will briefly describe which quantum mechanical tools are available for this purpose.

First and foremost, the physical resource that enables performing non-local computation is *entanglement*, a unique correlation of joint quantum systems stronger than any classical counterpart but very fragile, hard to create and to maintain long. Entanglement lies at the heart of quantum communications, facilitating the distribution of quantum states encoding quantum information through a protocol known as *quantum teleportation* or *teledata*. Multiple teleportation variants exist, which are designed to either transmit data in one direction – *quantum teleportation* or *teledata* – but also bi-directional communication – *entanglement swapping* – and gate operation at a distance – *gate teleportation* or *telegate*. Furthermore, the basic two-node teleportation can be extended to multi-party distribution networks composed of many nodes. Some parties may either help the rest of the network in the quantum communication protocol – *assisted teleportation* –, or the quantum information may be imperfectly broadcast from one sender to the rest – *quantum telecloning*.

In the following sections, we will introduce these protocols in detail.

2.1. Quantum entanglement

Entanglement is the property of a quantum system that illustrates the impossibility of describing a composed system in terms of just its individual components due to nonclassical correlations of certain degree(s) of freedom of the subsystems [30]. Typical examples of these degrees of freedom are the position and momentum of free particles, the polarization of light, energy levels of trapped ions, or transverse atomic spins. These degrees of freedom are related to observables that present a discrete and finite spectrum or a continuous and infinite one. Hence, the terms discrete variable (DV) and continuos variable (CV). This review focuses on DV because it is the most common in quantum computing.

Archetypical examples of DV entangled quantum states are the pure states

$$|\Phi^{\pm}\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle_A|0\rangle_B \pm |1\rangle_A|1\rangle_B\right),$$
$$|\Psi^{\pm}\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle_A|1\rangle_B \pm |1\rangle_A|0\rangle_B\right), \tag{1}$$

dubbed Bell states or Einstein, Podolski and Rosen (EPR) pairs, where two parties – Alice and Bob – share two qubits A and B encoded in a dichotomic degree of freedom as polarization, spin, or any other two-level quantum variable [31]. A perfect non-local correlation arises as Alice's measurement outcome determines Bob's measurement outcome. This property allows us to build an intuition of how Bell states are a natural choice for quantum communication: if a quantum gate, whose matrix representation is symmetric, is applied to one of the qubits of the Bell state $|\Phi^{+}\rangle$, it is the same as if the gate was applied to the other qubit. The gate somewhat 'slides' between qubits through the entanglement, like beads on a string [32].

These entangled states are the basis of a large number of quantum information protocols, one of which is quantum teleportation, which we introduce in the following section.

2.2. Quantum teleportation or teledata

Quantum teleportation, one of the more remarkable quantum information protocols, was introduced thirty years ago in a landmark paper [33]. This quantum protocol enables the reconstruction of an unknown quantum state of a given physical system at a different location without actually transmitting the system. Quantum teleportation requires two key ingredients: Quantum entanglement and classical communication between the locations (which excludes superluminal communication).

Quantum teleportation plays a pivotal role in the development of quantum technologies [34]. It overcomes some of the limitations of quantum communications and quantum computing using the non-local transfer of unknown information. Quantum teleportation networks [35], entanglement swapping [36], and quantum repeaters [37] enable the distribution of entanglement over long distances [38], while quantum gate teleportation [39] and measurement-based quantum computing [40] are examples of techniques that distribute local gate operations among physically disconnected parties [41]. Entanglement swapping and gate teleportation will be discussed further in the next section.

Proof-of-principle demonstrations of quantum teleportation were successfully achieved using diverse physical substrates as photonic

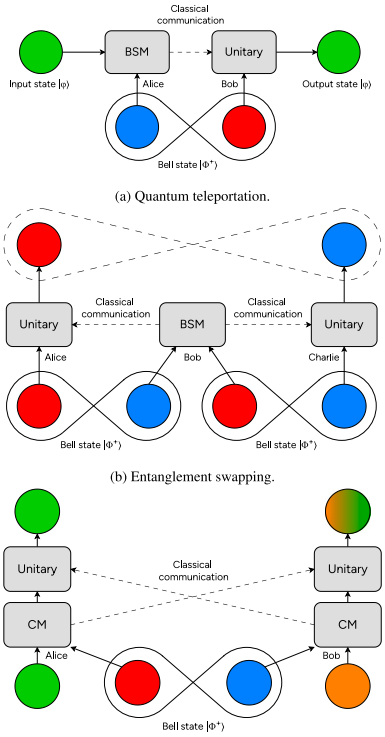


Fig. 2. Sketch of quantum communication protocols: (a) Quantum-state teleportation (teledata), (b) entanglement swapping, and (c) quantum-gate teleportation (telegate). BSM: Bell-state measurement. CM: controlled operation and projective measurement.

qubits [42], optical modes [43], atomic ensembles [44], nuclear magnetic resonance [45], trapped atoms [46,47], and solid-state systems [48]. Over the last years, the focus has moved to teleporting more complex states – larger number of degrees of freedoms or higher dimension qubits [49,50] – and to real-world applications in quantum communications and computation [38,51,52].

In the teledata protocol, Alice and Bob share an entangled Bell state as that given by Eq. (1) [42], see Figs. 2(a) and 3(a) in physical and circuit representations, respectively. A third party provides Alice with a qubit C to be teleported to Bob. Importantly, the quantum state of qubit C –represented by ρ– is unknown to both Alice and Bob unlike in remote state preparation [54]. Alice then performs a Bell-state measurement (BSM), which randomly projects with equal probability her qubits A and C into one of the four Bell states $|\Phi^{\pm}\rangle$ or $|\Psi^{\pm}\rangle$. As a result, Bob's qubit B is simultaneously projected onto the state $T^{\dagger}\rho T$, where $T \in \{I, X, Z, ZX\}$ is an elementary or a combination of Pauli operators. As the last step, Alice informs Bob of the BSM outcome through the classical channel using two classical bits – feed-forward – and Bob applies the suitable gate T to his qubit to recover the unknown state ρ of qubit C at his location.

Regarding the figures of merit of quantum teleportation, there are mainly two:

1. The *BSM efficiency* or Alice's success probability for distinguishing a complete basis of entangled states – like the four Bell states. This varies for different information encodings: for instance, for a simple realization of Bell-state measurement using DV photonic qubits, the Bell efficiency is 50% at maximum [55].
2. The *teleportation fidelity* $F \in [0, 1]$ between the input state ρ and Bob's output state averaged over all Alice's measurement results and input states. The benchmark for the teleportation fidelity is surpassing the fidelity for state transfer without quantum resources, using, for instance, just classical correlations, i.e., $F > F_{\text{class}}$, where $F_{\text{class}} = 2/3$ for DV [56].

Table 1
Some milestones in quantum teleportation in terms of Bell efficiency, fidelity, distance of teleportation, and quantum memory. QED: quantum electrodynamics.

Quantum technol.	Bell eff.	Fidel.	Max. dist.	Memory
Polarization [38]	25%	0.80	1400 km	NA
Integrated opt. [51]	25%	0.894	10 m	NA
Superconduct. [41]	100%	0.79	chip	1 ms
Cavity QED [64,65]	100%	0.833	60 m	–
Ion Trap [66]	100%	0.845	chip	–
Rare-earth [67]	50%	0.86	1 km	17.5 μs

Table 1 shows examples of recent milestones in quantum teleportation in different technologies. More details on the state of the art can be found in [57,58].

Quantum teleportation has seamlessly made the leap from laboratory conditions to real-world implementation in urban environments, showcasing its adaptability and robust functionality. Teleportation networks allow for the reliable transfer of quantum information between a number of distant nodes, even in the presence of non-ideal features such as noise and loss. Recent advances include demonstrations of two-node teleportation over a metropolitan network [59,60], links between nanophotonic memories and ion traps in an urban network [61,62], and multinode entanglement over a metropolitan network with a cloud of Rubidium atoms in a ring cavity acting as a quantum memory [63]. More on quantum networks will be delved in Section 3.

2.3. Variants of quantum teleportation

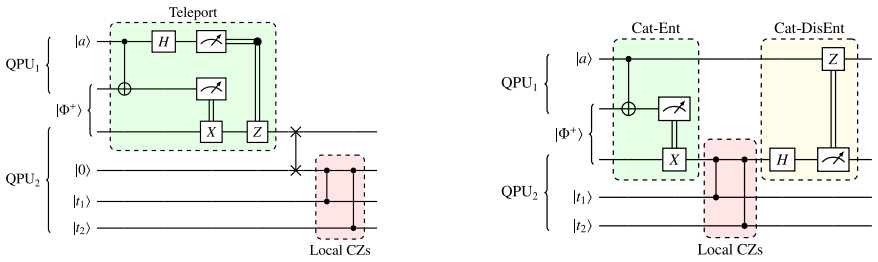
Quantum teleportation is a primitive of quantum information science and has a number of variants essential for DQC. In the following, we review the most important three: entanglement swapping, quantum gate teleportation – telegate – and multipartite teleportation.

2.3.1. Entanglement swapping

Entanglement swapping is a variant of quantum teleportation that enables remote correlations by the transfer of quantum entanglement between distant end-users that do not directly share a quantum resource. In this case, Bob shares two entangled states, one with Alice and the other with Charlie, as shown in Fig. 2(b). Bob acts as a relay between them, performing Bell measurements and broadcasting the outcomes by a classical channel to them, who apply the suitable gates to their qubits. As a result, Alice and Charlie now share an entangled state conditioned on the result of Bob's measurement [36]. This protocol, together with entanglement distillation[2] [68], enables the distribution of entanglement over large distances, being the basis of quantum repeaters [37]. Related to entanglement swapping are fusion gates [69,70], where projective measurements probabilistically *fuse* small entangled states in order to produce large entangled states – cluster states – useful for measurement-based quantum computing [40].

The first demonstration of entanglement swapping was carried out by Pan et al. using polarization-entangled photons [71]. Swapping

[2] Entanglement distillation, aka entanglement purification, involves converting N copies of any entangled state ρ into a certain quantity of nearly pure Bell pairs, solely through local operations and classical communication.



(a) Teledata. (b) Telegate.

Fig. 3. Examples of teledata and telegate circuits for the application of CZs gates over $|t_1\rangle$ and $|t_2\rangle$ with the remote state $|a\rangle$ as control. (a) The state $|a\rangle$ in QPU_1 is teleported to the first qubit of QPU_2 (b) Cat-entangler and cat-disentangler primitives [53] are used to implement the remote control.

has been recently applied to connect two spatially-separated solid-state quantum memories by telecom links [67], and to entangle non-neighboring Nitrogen Vacancy (NV) qubits in a multinode teleportation network [72].

2.3.2. Quantum gate teleportation or telegate

In gate-based quantum computing, a sequence of unitary operations (usually single- and two-qubit) are applied on a set of qubits. However, sometimes there is no direct interaction between qubits on which we want to apply a two-qubit gate [20]. Quantum gate teleportation, also known as telegate, reduces the topological requirements by substituting two-qubit gates with other cost-effective resources: auxiliary entangled states, local measurements, and single-qubit operations [39]. Typically, Alice and Bob want to perform a non-local operation on unknown control and target states using a shared Bell state as a quantum channel. To this end, both perform locally controlled operations and projective measurements (CM) on their half Bell state and control/target states. After this step, partial quantum information is transferred between the two parties conditioned to the measurement outcomes. Cross communication of the results through a classical channel enables Alice and Bob to perform suitable corrections to the control and target states. This procedure results in a controlled gate operation on two non-interacting input states – see Figs. 2(c) and 3(b) for physical and circuit representations, respectively. The first experimental demonstration of quantum gate teleportation was a remote CNOT operation carried out through photon entanglement and linear optical manipulations [73]. Recent advances in remote operations comprise superconducting qubits, trapped ions, and quantum electrodynamics cavity nodes [41,64,66].

When applied to multipartite entangled states with a given topology, suitable measurement on a given network node teleport unitary-transformed-state to other nodes. This is the basis of measurement-based quantum computing [40].

2.3.3. Multipartite teleportation

Multipartite entangled states as the Greenberger-Horne-Zeilinger (GHZ) state enable a natural extension of quantum teleportation to more than two parties [74]. These N-party protocols for multipartite teleportation enable two variants: assisted and unassisted teleportation – commonly referred to as quantum telecloning. In the first case, *assisted teleportation*, Alice *helps* the communication between Bob and Charlie by performing a tailored measurement and broadcasting the result to them, thus improving the entanglement between them [35]. In the second case, *quantum telecloning*, Charlie teleports to Alice and Bob simultaneously, hence with a teleportation fidelity, limited by the no-cloning theorem, given by $F = (MN + M + N)/(MN + 2M)$, for N senders and M receivers of qubits [75].

Examples of assisted teleportation are open-destination teleportation [76] and, more recently, shared-quantum-secret teleportation [77]. Quantum telecloning was, in turn, demonstrated in DV by means of partial teleportation [78]. Cloning of entanglement [79] and copy distribution [80] are recent examples of this variant of teleportation.

2.4. Quantum devices for entanglement distribution

In the search for maximum performance and demonstrating quantum advantage for distributed, scalable quantum computing systems, modular architectures featuring specialized, single-purpose hardware are currently under development [81]. The quantum devices that are part of these architectures, apart from QPUs, can be categorized in one of the following categories: *quantum transducers*, *quantum memories*, *quantum repeaters*, and *entanglement routers and switches*. This section will describe the aforementioned devices in detail and discuss the current research advances in each technology.

This section will detail the aforementioned devices in detail and discuss the current research advances in each technology.

2.4.1. Quantum transducers

The communication between *local* qubits of systems where the quantum operations take place (e.g., QPUs, memories or repeaters) requires the conversion, or *transduction*, of their states to a different system used for delivery of quantum states in the form of *flying* qubits, which have the requirements of being highly mobile and well coupled to the specific local platform. Multiple flying qubit systems have been proposed, such as short-distance electronic states in semiconductor devices [82], direct delivery of nuclei with long-lived nuclear-spin qubit encoding [83] and, more commonly, single photons, given their naturally mobile nature and their low coupling with the environment.

In classical communications, the high-rate transfer of current technologies is only possible due to the high bandwidth, and low attenuation and latency provided by light in fiber optics, enabling the underwater connection of continents at tens of thousands of kilometers [84]. The current state-of-the-art telecommunication systems also implement multiplexing, i.e., encoding information at multiple wavelengths through the same fiber [85]. As DQC requires deterministically distributing entanglement, the requirements for flying qubits are, primarily, good coupling to the particular local quantum system, either by direct emission or interaction, and that the fidelity of the resulting entangled states is maximal. Quantized states of light are also the most natural information carrier choice for the distribution of quantum states at a distance, and extensive research has focused on the accurate manipulation of photonic states using linear and nonlinear optical devices [86,87].

The most widely studied way of generating entanglement between remote systems is via entanglement swapping between two independently entangled flying qubit-matter qubit systems, i.e., generating entanglement between photons and local qubits (trapped ions, neutral atoms, or NV centers), then performing BSM on the photons of each pair. Hence, their joint wavefunction collapses in the same non-separable state, and the matter systems become entangled. To this purpose, heralded entanglement of photons emitted after de-excitation from prepared excited states has been shown in trapped-ion qubits [88–

90], neutral atoms [91] and diamond NV-center qubits [92–96]. After the subsequent BSM, fidelities to Bell states of up to 88% at 230 m have been demonstrated in trapped-ions [97], and even above 60% between NV-centers separated by a 25 km metropolitan fiber link [98]. Deterministic qubit state transfer between different NV-center nodes has also been achieved [72]. One promising proposal is the coupling of ion- or Rydberg atom-chains in optical cavities [99], which has been shown capable of providing any-to-any entanglement for large systems with over 500 qubits in trapped ions by using two atomic species, one of which acting as a communication qubit and another as memory qubit. Quantum dots are also promising due to their tunable emission wavelengths in the infrared range, yet some challenges remain such as extending the qubit lifetime [100,101]. Rare-earth doped crystals, commonly Eu^{3+}- or Pr^{3+}-doped Y_2O_3 crystals also have emission near the 1550 nm band with sharp linewidths and long coherence times [102]. Spin-photon coupling in the microwave range has also been demonstrated in Si double quantum dot spin qubits by coupling the charge dipole of a trapped electron to the electric field component of a cavity photon stored in a superconducting resonator [103]. The promising advantages of these links is that multiple quantum dot qubits can use the same resonator, as their coupling can be switched on and off in nanoseconds, and the resonators' dimensions are large compared to the double quantum dots.

Finding mechanisms to link superconducting chips together to overcome their scaling needs is a current technological challenge. One of the most interesting experiences has been the deterministic transmission of excitations between superconducting QPUs using cryogenic microwave waveguides [104,105]. Applying modulated microwave pulses, an effective coupling between energy levels in the transmon qubits and their respective transfer resonators can be achieved, transferring an excitation from node A to its transfer resonator, which then emits a single microwave photon towards node B where it is absorbed, exciting its qubit. Cryogenic, lattice-based quantum networks have been proposed based on this method of connecting superconducting chips [106], demonstrating high fidelity, in excess of 85% when correcting for readout errors. However, some serious drawbacks remain, such as the high cost of cryogenic equipment, complexity and low modularity of these systems.

Correlated photon sources such as spontaneous parametric down-conversion (SPDC) or quantum dots can also be utilized to achieve the initial photon–matter qubit entanglement. SPDC sources consist of a non-linear crystal pumped by a strong laser beam generating pairs of maximally entangled photons with some probability, which can then be frequency-filtered and used to interact with the physical qubits. Using this technique, distant solid-state quantum memories have been entangled at distances of 1 km and fidelities of up to 86% using two photons at 606 nm and 1550 nm [107]; the former is stored in the collective excitation of Pr^{3+} in a doped crystal using the Atomic Frequency Comb (AFC) protocol, while the latter is sent to a BSM analyzer and corrected, resulting in entanglement. Hyperentanglement, where more than one degree of freedom can simultaneously be maximally entangled (e.g., polarization and direction of two photons) has also been demonstrated using this type of sources [87,108]. Alternative quantum dot-based sources have very attractive properties for this purpose, such as being triggered on-demand and energy-tunable [109–111], and reaching fidelities over 90% [112,113].

A less widely studied possibility is making single photons interact *in-flight* with two separated quantum systems. Using ancillary doubly reflected single photons followed by a measurement of the photon and a conditional rotation of the target qubit, heralded deterministic teledata [65] and telegate entanglement of remote qubits [64] have been demonstrated. Achieving fidelities up to 90% and 85% respectively at 60 m, this technique could enable deterministic, short-distance, low-latency DQC. Its advantages are two-fold: teleportation is performed without the need for preformed Bell pairs, so it can be done just-in-time; and losing the photon (i.e., not being able to measure it) does not lead to a mixed state in the matter qubits, so it can be tried again. While it is a new and promising avenue for the distribution of quantum states, further research is required to bring its quantum efficiency and fidelity closer to unity.

Moreover, interconnecting quantum systems may require coupling platforms that operate at different photon frequencies. For this purpose, techniques are being developed to implement frequency conversion of single photons on demand, maintaining certain properties (such as polarization) intact, which would enable the transcoding of qubits between platforms. One such technique is heralded up-conversion from infrared to visible light, which has been achieved through sum frequency generation in nonlinear crystals [114,115]. More recently, Murakami et al. [116] have demonstrated frequency conversion from visible to infrared using pairs of non-degenerate photons generated by SPDC, and Weaver et al. [117] have shown frequency bidirectional transduction from microwave to infrared light using transduction assisted by a resonant mechanical mode. However, the quantum efficiency of these techniques is currently low and significant efforts are underway to push it towards unity. In addition to the aforementioned frequency conversion techniques, recent work by Sahu et al. [118] has demonstrated deterministic entanglement between the quadratures of propagating microwave and optical photons in cryogenic waveguides, a first step towards interconnecting superconducting qubits with long-range communication systems and memories.

In summary, there are multiple competing techniques which allow distant QPU to generate entanglement in virtually all matter qubit technologies. However, much research is still required to push both fidelity and efficiency towards unity. Frequency conversion is a promising technique which may allow future interfacing of different systems, enabling heterogeneous DQC.

2.4.2. Quantum memories

To fully take advantage of the entanglement distribution and distillation protocols for both short and long distance quantum communication, it is paramount that the coherence time of the communication qubits is longer than the protocol itself, surviving multiple rounds of qubit exchange and entanglement purification. These long-lived qubits, organized as large registries, are known as quantum memories or quantum Random Access Memorys (qRAMs).

The simplest quantum memories are photonic memories, in which photons are stored and then retrieved after a given time. Multiple approaches exist, such as using free space optical loops triggered by heralding [119] or fiber delay lines [120] and cavities with tunable Q-factor [121,122]. Stimulated photon-echo is a more advanced technique based on the absorption and delayed reemission of single photons with the same quantum state after an ensemble of atoms is rephased [123–125], which has been demonstrated e.g., using slow light by electromagnetically-induced transparency (EIT) [126], controlled reversible inhomogeneous broadening (CRIB) [127] and atomic frequency combs (AFC) in rare-earth doped crystals [102,128,129]. All-photonic systems (i.e., photonic quantum computing) can already take advantage of photonic memories, as they do not require transduction [130,131].

However, both the difficulty of retrieving single photons with high fidelity as well as the low scalability of photonic-based memories have pushed forward extensive research on multiple alternative quantum memory technologies, demonstrating high-fidelity single-qubit gates in excess of the threshold needed for quantum error correction [132,133]. Notable examples are trapped-ion and -neutral atom qubits, which use the hyperfine structure of atomic ensembles of ions [134], or neutral alkali or alkaline earth single atoms in optical tweezers [135–137] to encode the quantum states, which can be individually addressed by microwave pulses [138]. Quantum memories based on diamond NV-centers have also been demonstrated (see [139] and references therein). Some of these technologies have demonstrated long coherence times, of up to 10 min in single trapped-ion qubits [140] and up to six hours

in cryogenically cooled Eu^{3+}-doped yttrium orthosilicate nuclear spin qubits [83]. More recently, Barnes et al. [137] have demonstrated an individually addressable 21-qubit register of highly coherent and independent qubits with coherence times of about 40 s using nuclear spin qubits in optical tweezers, opening the gate to intermediate-scale quantum memories.

2.4.3. Quantum repeaters

As we previously discussed, light is the most natural long-distance carrier of quantum states. However, the absorption of light imposes intrinsic physical limits on the distance at which single photons can travel. In long-distance fiber communications, absorption is mainly produced by the fiber, with an attenuation coefficient in the range ∼0.14–0.4 dB/km in low loss telecom fibers [141,142]. Furthermore, even in the short-distance communication range of a datacenter, the rate at which photons are lost is nontrivial: the typical loss per SC connector is ∼0.25 dB [143], so the shortest possible connection between two nodes accounts for ∼0.5 dB of attenuation, i.e., ∼11% of the photons are lost. Hence, if frequent quantum communication is required for a distributed algorithm, the error probability quickly increases as $e = 1-10^{n\cdot dB/10}$ after n exchanges, limiting the scalability and reliability of the calculation.

It is important to understand that any improvements in the connector losses and fiber attenuation cannot and will not solve the problem of exponential decay with n. Given that standard telecommunications erbium-doped fiber amplifiers (EDFA) cannot be used to amplify arbitrary quantum states due to the no-cloning theorem, *quantum repeaters* are essential to the implementation of entanglement distribution and teleportation which enable deterministic transmission of quantum states and remote quantum operations between nodes [144,145]. An early solution to the problem of implementing a quantum repeater was proposed by Briegel et al. [37], which consisted of first entangling noisy and imperfect qubits and then creating a high-fidelity entangled pair through entanglement distillation. Recent proposals have extended the idea of entanglement distillation to qudits (i.e., d-state systems) [146], multiple simultaneously entangled degrees of freedom (hyperentanglement) [147,148], and logical qubits [114,149]. Van Leent et al. [150] have demonstrated single-atom entanglement over a 33 km telecom fiber using quantum repeaters, proving that long distance entanglement is already a technical possibility. Recent work has also shown that Er^{3+} inclusions in calcium tungstate greatly diminish optical spectral diffusion [151], a requirement to generate indistinguishable single photons needed for optical repeaters, as this ion is well coupled by its telecom band optical transition.

2.4.4. Entanglement routers and switches

As previously explained, the execution of general quantum algorithms in multiple qubit-limited QPUs requires entanglement to be generated on demand between pairs of arbitrary qubits [152]. For this reason, recent research has focused on implementing teleportation protocols between non-neighboring nodes. The simplest way to obtain arbitrary entanglement with interconnected QPUs is pre-establishing shared entanglement, as discussed in Section 2.4.1, in a *one-to-one* fashion between specific communication qubits in different nodes. In these *one-to-one* schemes, not every pair of QPUs ought to be physically connected, reducing the complexity of implementation for small integrated systems.

However, this apparent simplicity faces a major scalability challenge, resulting in substantial qubit swap and distillation overhead in complex, strongly entangled algorithms [14]. While compilation optimizations can reduce swap operations, more general and modular quantum networks will need *entanglement routers* and *switches* to distribute entanglement between arbitrary qubits, akin to classical counterparts [153–155].

For quick reference, classical routers are capable of finding optimal routes in a complex network and understand the Internet Protocol (IP), while switches only recognize which physical addresses are routed through their connections to redirect traffic. The current absence of a quantum IP standard makes the distinction of the quantum counterparts difficult, so authors have been using these terms interchangeably. Moreover, the quantum hardware required is essentially the same and any differences would arise from the higher-level classical network management. Following this description, any two QPUs in the network can be connected through either one or multiple switches and/or routers in a Quantum Local Area Network (QLAN), or through an efficient routing path that connects multiple routers (which may require repeaters to maintain entanglement) and lead to a Quantum Wide Area Network (QWAN) [106,156]. The interconnection of quantum networks could eventually lead to a worldwide Quantum Internet [157,158]. However, this escapes the scope of this review [156,159,160].

Entanglement switches and routers can then be thought of as single-purpose QPUs: their sole objective is establishing entanglement among compute nodes through entanglement swapping, for which implement all the quantum technology required, such as quantum registries, entanglement sources and means to perform BSM, as well as all the hardware required for networking logic and classical communications [160]. Moreover, these devices may also be built on different quantum platforms than the proper QPUs, e.g., not requiring the implementation of a complete set of quantum gates but only those required for the swapping protocol and instead requiring registries of qubits with very high fidelity and coherence times longer than the entanglement distillation protocol, or access to quantum memories that fulfill these two requirements. Some proposals suggest networks based on single atoms trapped and coupled to optical resonators as memory qubits, which have long coherence times and good photon coupling (see [161] and references therein).

3. Networks for distributed quantum computing

The scientific literature on quantum networks is really extensive and a dedicated review would be needed to properly address all aspects regarding architecture, entanglement creation and distribution, network orchestration, network software stack and protocols. Therefore, our intention in this section is to provide sufficient information to give the reader an outlook of some relevant progress on this subject. Although considerable amounts of research has been oriented towards communication systems for the Quantum Internet, much of it can be applied to DQC. However, in DQC, the focus should be on short/datacenter distance limits.

Quantum networks (QNs) enable the execution of distributed operations among two or more qubits that may be very close to each other or separated by long distances. The mechanisms used for communication could be based on the transmission of the quantum states, or on the creation, distribution and consumption of entanglement. The entanglement resources provided by these QNs can be used both in DQC and in other applications of quantum technologies e.g., sensing or encryption. A comprehensive review on entanglement networks covering the fundamental mechanics, the enabling technologies, the network architecture and elements and research challenges associated can be found in [162].

Entanglement networks allow the execution of the previously discussed swap, teleport or telegate mechanisms. Besides these well known network mechanisms, Miguel-Ramiro et al. propose the inclusion of full quantum functionalities that increase the parallelism of operations using superposition of tasks with quantum control [163]. Examples of this are the execution of superposed tasks such as superposed measurement/non-measurement, superposed paths, superposed teleporting/non-teleporting or superposed merging/non-merging of graph states.

Classical network architectures and protocols cannot be directly extrapolated to quantum networks for entanglement distribution due to their particularities compared to the transmission of classical bits, such as:

- The duration of entanglement and the lifetime of the qubits due to decoherence.
- The probabilistic nature of some quantum mechanisms.
- The need for mechanisms to improve fidelity, such as distillation.
- The possibility of joining entanglement links not only through sequential operations but also through operations carried out in parallel on the various links.
- The different entangled resources – bipartite, multipartite by means of GHZ, W, cluster states, etc.
- The need for both quantum and classical channels to achieve the desired functionality.
- The possible use of quantum networks not only for the transmission of quantum information but also for the distribution of entanglement between distant points, which can be used as a resource by itself.
- The resource reservation strategy, if needed.

Software development requires the definition of a set of protocols that satisfies the different communication requirements from the most basic physical level communication to the application level communication, usually defined as a stack of network software layers. The survey [164] summarizes the main works on network protocol stacks, compared to the classical OSI or TCP/IP, and provides a comparison of the different stacks. Also noteworthy is the publication of the Internet Research Task Force (IRTF) Architectural Principles for a Quantum Internet [165], where the general guidelines for the design of quantum networks are presented. Regarding hardware architecture, much work has focused on tackling the delivery of entanglement in different technologies for the development of quantum repeaters. However, this is also to applicable to the development of quantum network devices for DQC: all optical and matter qubits architectures, based on discrete variable, based on continuous variable, based on bipartite entanglement or multipartite. In what follows, a list of relevant work related to DQC about quantum network hardware architecture and software stacks is presented:

- [166] presents a CV all-photonic switch for entanglement creation among end-nodes that uses Gottesman–Kitaev–Preskill qubit encoding and Steane codes error correction.
- [167] proposes an architecture for a CV continuous variable quantum switch where end nodes share entanglement links error corrected by means of Noiseless Linear Amplification (NLA) [168].
- Dür et al. [169] propose an architecture and network stack for quantum networks based on multipartite entanglement (GHZ graph states) allowing the generation of graph states of any type among clients.
- Van Meter et al. [170,171] propose a Quantum Recursive Network Architecture (QRNA) describing the layers of network communications that tackle entanglement distribution end to end. They introduce a recursive layer architecture in which swapping and purification functions are repeated to build *end-to-end* entanglement paths from a sequence of links, being entanglement performed at link level. Physical and link layer are in charge of entanglement establishment at link level (point-to-point), while *Remote State Composition* and *Error Management* layers are recursive and are continuously repeated performing swapping and purification from entangled links until the system is able to build an end-to-end entangled path.
- Li et al. [172] and Dahlberg et al. [173] propose protocol architectures for quantum networks based on bipartite entanglement where the mission of physical and link layers is the establishment of reliable entanglement, the network layer's goal is the establishment of long distance entanglement, and the transport layer copes with the qubits reliable/deterministic qubits transmission.
- Pirker and Dür [169] propose an architecture and network stack for quantum networks based on multipartite entanglement (GHZ graph states) allowing the generation of graph states of any type among clients. This architecture is composed of four layers: physical, connectivity, link, and network. The main difference to the traditional OSI layer architecture relies on the introduction of the connectivity layer, which is responsible for allowing *point-to-point* or *point-to-multipoint* connectivity, as well as error correction and establishment of long-distance links. The link layer allows the creation of graph states in the network that clients will subsequently use for the creation of end-to-end graph states.

In relation to the protocols necessary for the QNs they are classified in layers of the Open Systems Interconnection (OSI) model, in an analogue way to the classical counterparts. Several implementations of some of the functionalities of each communication layer have been proposed:

- Layer 1 (Physical): A. Dahlberg et al. define a protocol for physical entanglement generation based initially on NV center platforms [173]. A. S. Cacciapuoti et al. compare the addressing needs in quantum networks and classical networks (for instance, entanglement to a destination might require to perform entanglement to an intermediate node instead of the destination), and also the implications in the superposition of paths for both the addressing and routing design [174], while J. Miguel-Ramiro et al. assign each network device an identification register and an activation register which depend on the target node [163]. When a Toffoli operation on both registers is successful the node takes an active role in the operations.
- Layer 2 (Link): J. Illiano et al. propose an entanglement access control mechanism (analogue to the ethernet MAC mechanism) to grant access to a subset of the nodes to the entangled state (contention resource) [175]. The mechanism is based on quantum communications using multipartite entanglement Dicke states and preserves the anonymity of the granted nodes. R. Hanson and S. Wehner define a link layer protocol for robust entanglement generation sensitive to specific application needs (create and keep or create and measure, number of entangled links, atomicity of the links creation, fidelity, and other relevant parameters for the link creation) [173].
- Layer 3 (Routing): M. Caleffi designs a routing protocol and metric for quantum networks considering the key parameters for entanglement generation and the needed optimization to determine the optimum path between two points in the network [176].
- Layer 4 (Transport): Yu et al. propose a protocol for the reliable transmission of quantum information [177]. The protocol is based on the three way handshake of the classical counterpart TCP [178] and on a recursive quantum secret sharing method ((2,3) threshold scheme [179]) to achieve the transmission of the quantum data reliably, where the message to be transmitted is encoded in segments, being able to recover the message when only 2 out of 3 segments are available. If one of the segments is lost, one of the remaining segments will be reencoded and the method repeated.
- Layer 5 (Application)[3]: T. Satoh, R. van Meter et al. include the design of quantum sockets in analogy to the classical communications sockets, allowing the applications to access the services and having similar functions (creation and destruction of the socket, connection to the socket; reading from the socket, writing to the socket and configuring the socket) [171,181].

Regarding the control plane, [182] propose a control architecture for entanglement generation in quantum networks that moderates the requests for entanglement resources with the goal of fair distribution of the service to the network end nodes. [183] proposes a protocol to manage entanglement requests and resources (memories) management.

[3] A quantum operating system for quantum network applications has been implemented and experimentally tested running a client–server application on NV nodes in [180].

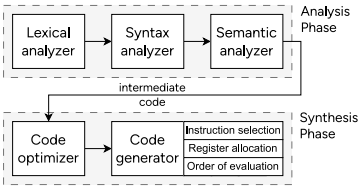


Fig. 4. Sequential phases of classic compiler process: analysis and synthesis stages.

4. Development layer

In the realm of classical computing, compilation serves two primary purposes: translating complex programming constructs into machine-specific executable instructions and optimizing machine resources to produce efficient code. Typically, this process follows a common scheme, as illustrated in Fig. 4, which consists of two main phases: analysis and synthesis. The analysis phase is responsible for conducting the code's lexical, syntactic, and semantic analysis to ensure correctness. Once validated, the code is translated into an Intermediate Representation (IR), which simplifies the implementation of optimizations in the synthesis phase.

Regarding quantum compilation, the scheme followed is usually the same as in the classical world. This is mostly because quantum compilation turns out to be a fully classical task, leaving the quantum workload just for the execution part. This leads to the situation where many quantum development software tools are actually built on top of classical languages, allowing the analysis phase to be integrated into an existing implementation.

Adding distribution to this task does not alter the compilation scheme; it remains largely the same with some additional steps and restrictions. To fully picture the differences and intricacies of compiling a distributed program, this section will be divided into two parts: Section 4.1 will elucidate the various methods by which a quantum process – usually referred to as a quantum circuit – can be distributed, while Section 4.2 will delve into how the compilation process is executed considering the distributed nature of the task.

4.1. Types of distribution

Distributed computing makes it possible to organize the computation of a problem in different Processing Units (PUs), which are connected through an interconnection network. The advantages of this model are evident: reducing the execution time by leveraging multiple PUs computing in parallel or, for large problems that do not fit within a single node, partitioning them to enable their solution. The time reduction comes with its own set of disadvantages, notably the increased difficulty in adapting algorithms and codes to a distributed approach. This is due to the significant overhead caused by communications and synchronizations, which must be carefully considered and managed [184].

Therefore, the complexity of developing a code increases when it is distributed. This complexity especially impacts the compiler design. In the analysis phase, new communication directives need to be developed, while in the synthesis phase, various network architectures must be considered to optimize data transmission and reception [185].

Certainly, the network's communication mechanisms and the resources the quantum task requires dictate the applicable distribution model, as depicted in Fig. 5. Three distinct categories of quantum distribution emerge: *circuit distribution*, *circuit cutting*, and *embarrassingly parallel*. It is clear, looking at Fig. 5, that all categories converge in compiling, executing, measuring, and post-processing information. Each of these distribution methods involves specific strategies for partitioning

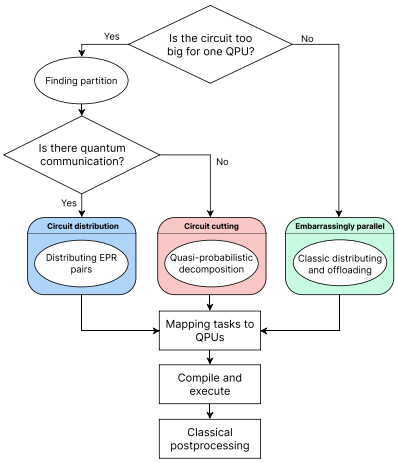


Fig. 5. Types of quantum distribution and their stages simplified.

and managing the quantum circuit across different QPUs. For instance, Fig. 6 illustrates how Bell pairs can be created in the three main distribution categories. The following sections will dissect each of these categories to fully understand how the quantum distribution works in each case.

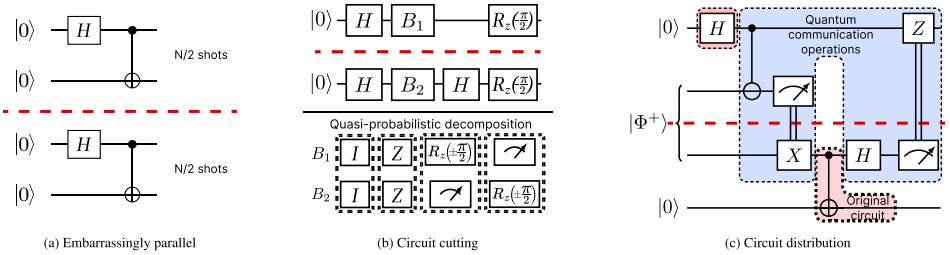
4.1.1. Circuit distribution

Circuit distribution, as shown in Fig. 5, involves three main phases: first, finding an optimal or near-optimal partition; second, distributing the partition among the available QPUs, and third, mapping this partition to each QPU. However, partitioning the circuit presents the most significant challenge and will be the primary focus of our efforts in this section. The other aspects are common to all the distribution types and will be further explained in the compilation Section 4.2.

First, for partitioning, the quantum circuit is mapped onto a graph that shows interconnections between elements. Thus, quantum circuit partitioning turns into a graph partitioning problem: given an undirected graph $G = (V, E)$ with a vertex set V and an edge set E, the aim is to partition V into two or more subsets regarding a cost function, like the number of edge cuts generated by the partition.

Graphs assume that the interaction between vertices is by pairs. However, even the most trivial phenomenon implies more than two vertices interacting concurrently. It is necessary to broaden the graph concept to gather these multilateral connections. The so-called *hypergraphs* [186] generalize the graphs to more complex situations. In short, while a graph can establish connections by pairs, a hypergraph is an object that connects more than two vertices or pins through elements called hyperedges or nets, as shown in Fig. 7. Thus, a hypergraph $H = (V, E)$ is an ensemble of pins V and nets E among those pins, and a net $e \in E$ is a subset of more than two pins.

Hence, hypergraph partitioning generalizes graph partitioning. More precisely, a k-way hypergraph partitioning groups the pins of a hypergraph into k blocks minimizing an objective function so that few nets connect pins from different blocks. The exchangeable objective functions are the cut-net and the connectivity metrics. The cut-net metric generates independent blocks of vertex sets by minimizing the nets belonging to several blocks, whereas the connectivity metric weights each net e with a factor $\lambda_e - 1$ to diminish the λ_e blocks connected by a net. The cut-net objective function sums over the nets among blocks and the connectivity metric over the λ_e blocks connected



(a) Embarrassingly parallel (b) Circuit cutting (c) Circuit distribution

Fig. 6. Example of the creation of a Bell pair for each type of distribution studied. In all types a partition of the tasks or the circuit (dotted line) is specified.

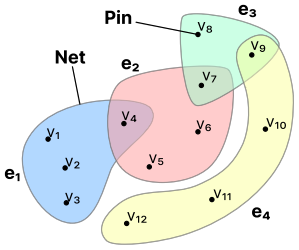


Fig. 7. Example of a hypergraph with twelve pins v_i and four nets e_j. Net e_1 has a size of four as it ensembles four pins, and pin v_4 has a degree of 2 as it belongs to two nets.

by a net. Nevertheless, both are analogue to the edge-cut problem in graph partitioning.

Underneath the goal of minimizing the cut-net and connectivity metrics lies an important consideration: while a valid partition may suffice for DQC, it may not necessarily be an optimal partition. For instance, in the circuits responsible for teledata and telegate operations – as illustrated in Fig. 3 –, these operations add up to four layers of depth to the circuit to enable operations among qubits in different QPUs. Consequently, this introduces latency to the quantum circuit, especially considering the additional synchronization required for intermediate measurements contained in both protocols between both QPUs. This latency represents a significant bottleneck in circuit distribution. Therefore, all circuit partitioning methods aim to minimize the utilization of teledata or telegate protocols. This aspect will be crucial in the circuit distribution techniques discussed in this section and beyond, summarized in Fig. 8.

Zomorodi et al. [187] introduced a first approach aiming to reduce communication between partitions, considering only two QPUs. They use the Kernighan-Lin (KL) [188] algorithm, a heuristic algorithm for graph partitioning, to divide the graph vertices into two subsets to reduce the edges across the subsets to minimize communication between the two partitions. After that, they apply a custom algorithm that aims to reduce the number of teleportations applied.

The work of Martínez and Heunen [189] was one of the most significant contributions in the field, serving as a foundational reference in many of the articles discussed here. Their method involves two key phases: a pre-processing phase, which groups equivalent gates, and a second phase, where hypergraph partitioning is performed using Karlsruhe Hypergraph Partitioning (KaHyPar) [190,191], a multilevel hypergraph partitioning framework that enhances cut-net and connectivity metrics. They evaluated their algorithm using five quantum algorithms known for their quantum speedup, such as Quantum Fourier Transform (QFT). A criticism of this work is that it did not consider optimizations such as moving gates back and forth to bring them closer together nor explore the entire search space of different partitioning options for executing global gates, which limits their ability to produce optimal solutions.

These limitations were pointed out in the work of Houshmand et al. [192], who improved on the work of Zomorodi et al. by exchanging the algorithm responsible for reducing teleportations – which had exponential cost – for a genetic one, which allowed them to significantly reduce the execution time. This work, like the Zomorodi et al. one, only considers a two QPU scheme, reason why Daei et al. [193] enhanced it by effectively mapping a quantum circuit into an appropriate number of distributed components. Moreover, Nikahd et al. [194] also go beyond, categorizing the binary gates into distinct "levels", followed by determining the optimal partitioning of qubits for each level through the solution of an integer linear program.

The work by Martínez and Heunen [189], on the other hand, was extended with an entanglement-efficient protocol [195] derived from [15] and with, among other things, a hypergraph approach to arbitrary network topologies [196]. In the first case, authors pack multiple non-local controlled unitary gates locally with one maximally entangled pair through a distributing and embedding pipeline. In the second, the authors also search for efficient entanglement within the network by reusing already available connections. In fact, this work led to many different articles employing hypergraph partitioning with KaHyPar.

Following the KaHyPar line, Sundaram et al. contribution concerns communication timing, non-local operations availability (teledata and/or telegate), and partitioning. First, engaging KaHyPar, Sundaram et al. [197] present a two-step heuristic for the distribution of quantum circuits: dividing the given qubits among the computers in the network with KaHyPar and scheduling communication operations, called migrations – equivalent to cat-entanglement operations [53]. They present a polynomial-time solution for the second step in a specific setting and a $\mathbb{O}(\log n)$-approximate solution in the general setting. Second, they amplify the available remote protocol for communications between QPUs, [198]: while Daei et al. [193] use teledata and, on the contrary, Martinez and Heunen [189] and Sundaram et al. [197] use telegate, there is upgrading in Sundaram et al. [198] applying both. For the telegate protocol, they consider a method similar to the initial two-step heuristic work, [197]. However, to partition the given qubits among QPUs, they use a Tabu-search-based heuristic regarding the heterogeneity of the network and storage limits. For the general DQC problem, they employ two heuristics: *Sequence*, a greedy approach, and *Split*, similar to the previous one, but with an iterative approach. Both contemplate the telegate solution as a subroutine. Lastly, Sundaram et al. take a step further in a recent work [199] by designing two different protocols to reduce the number of teleportations needed to perform the distributed task. The first method, termed *Local-Best*, tries to minimize the teleportation of qubits by selecting them only when necessary, with the choice of teleportation influenced by gates in the near future. The algorithm consists of two steps:

1. Find an initial assignment of qubits to computers to minimize the number of resulting non-local binary gates.

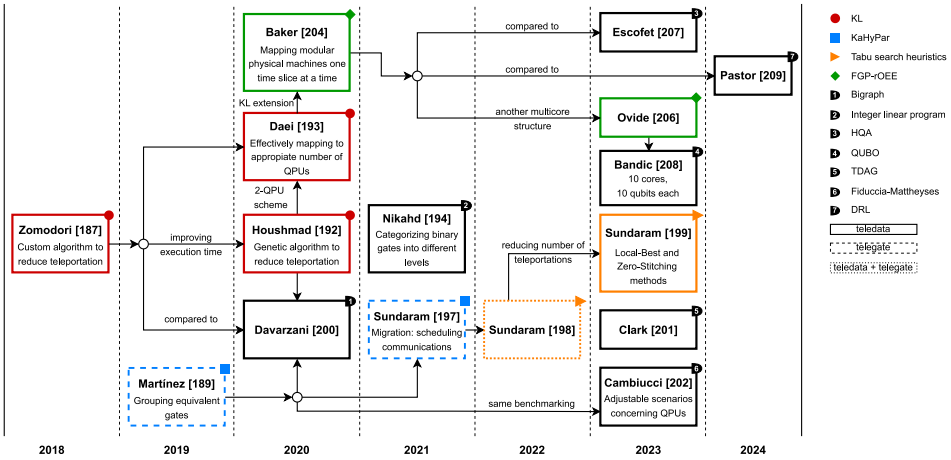


Fig. 8. Scheme of contributions to quantum circuit partitioning.

2. For each non-local binary gate G, select the teleportations to execute G locally based on the "near future" in order to minimize the total number of teleportations.

The second method to shorten the number of teleportations, *Zero-Stitching*, comprises also two steps:

1. Identify "zero-cost" subcircuits: These are contiguous subcircuits that can be executed without any teleportations.
2. Divide the given circuit into zero-cost subcircuits and "stitching" them together using teleportations.

There are also approaches employing bipartite graphs instead of hypergraphs. Davarzani et al. [200] introduce an algorithm for distributing quantum circuits to optimize the number of teleportations between qubits that consists of two steps: first, the quantum circuit is converted to a bipartite graph (bigraph), and, second, the bigraph is partitioned into K parts employing a dynamic programming approach. Finally, they compare their results with the ones yielded by works previously analyzed [187,189,192] and they claimed that the experiments gave better or equal results for benchmark circuits.

In another approach, proposed by Clark et al. [201], a different model than hypergraph is employed. They introduce the Tree-based Directed Acyclic Graph (TDAG) partitioning for quantum circuits, a novel method that views circuits as a series of binary trees and selects the tree containing the most gates for partitioning.

Besides minimizing the communication between partitions, Cambiucci et al. suggest [202] adjustable scenarios to the capabilities and constraints of the processing units involved in the distribution are considered. In this work, instead of the KL from the original hypergraphic approach, authors implement a variation of the Fiduccia–Mattheyses algorithm [203], which is a faster approximation algorithm for min-cut partitioning with a computational time that grows linearly with the network size. They use the same circuits as [189] for benchmarking.

A field-changing approach was the work developed by Baker et al. [204]. While still based on graph partitioning, this method seeks to avoid reaching a single static assignment for an entire circuit by employing near-optimal graph partitioning techniques. It leverages the inherent clustering of the DQC paradigm and the statically-known control flow of quantum programs to develop tractable partitioning heuristics. These heuristics map quantum circuits to modular physical machines one time slice at a time. Specifically, optimized mappings are created for each time slice, considering the cost to move data from the previous time slice and utilizing a tunable lookahead scheme to reduce the cost of moving to future time slices. To achieve this, a customized version of the Overall Extreme Exchange (OEE) algorithm [205] – considered a natural extension of the KL algorithm – referred to as relaxed-OEE (rOEE), is employed. Because the primary approach to map the circuit to the hardware is Fine Grained Partitioning (FGP), this method is usually referred to as FGP-rOEE. This method was further analyzed by Ovide et al. examining it under another multi-core architecture but maintaining the all-to-all qubit and cores connectivity [206]. Moreover, a Hungarian Qubit Assignment (HQA) method for partitioning is developed by Escofet et al. which also describes the assignment of qubits to cores between timeslices, and it is compared to the FGP-rOEE method [207].

A recent approach that has elevated the work of Baker et al. is the technique presented by Bandic et al. which relays on Quadratic Unconstrained Binary Optimization (QUBO) to partition the circuit at each time slice [208]. Their method's primary strengths are rooted in the formulation of the QUBO itself. This structure enables the decoupling of the problem definition from the solver as well as surpassing the limitations of look-ahead approaches utilized in the Baker et al. solution. It is worth noting that, in this approach, two different multi-core architecture layouts composed of 10 cores with a capacity of 10 qubits each were tested, in contrast with the non-realistic all-to-all connectivity assumed by the previous approaches.

Last but not least, one of the most novel algorithms is a circuit partitioning method that employs Deep Reinforcement Learning (DRL) [209]. Once again, the FGP-rOEE is employed as a baseline to compare the results and as an inspiration due to its time-sliced graph partitioning. This work has considered three approaches: Proximal Policy Optimization (PPO), Soft Mask, and Hard Mask. The first one, the PPO, is a widely used algorithm within the DRL scheme, while the remaining two, Soft and Hard Mask, are a variant of the former PPO algorithm that introduces a masking mechanism. The Soft Mask approach adds a simple mask, which disables useless operations – such as swapping identical qubits, swapping two qubits situated on the same machine, or advancing to the subsequent time slice without establishing a valid assignment for the current one – whereas Hard Mask implements a *direct-swap* heuristic in top of the Soft Mask which solely evaluates the relocation of misplaced qubits to the respective core they need to interact with.

Now that we have explored the state-of-the-art in the circuit partitioning problem, we can understand why it poses such a significant

challenge. Finding the optimal partition directly impacts performance and is a critical aspect in the later stages of compilation, where the boundaries between software and hardware become narrow. Specifically, this problem is closely related to the qubit mapping and circuit optimization stages of the distributed quantum compiler, which will be defined and explained in Section 4.2.3 as part of the compilers's synthesis phase.

4.1.2. Circuit cutting

As detailed in Section 3, on the road to fully functional DQC, one needs quantum communication in the form of a quantum network between the devices. In the absence of such of networks, there are several alternative techniques to simulate, or at least approximate, this entanglement using a classical network. In this context, circuit cutting has been suggested as a solution to partitioning a wide circuit requiring many qubits into smaller, non-entangled subcircuits. These subcircuits can then be executed (emulated) sequentially on a limited-qubit (memory) device or in parallel across multiple devices. There are several different strategies for circuit cutting, such as gate-cutting and wire-cutting (shown in Fig. 9), which will produce different subcircuits. The output of the original circuit is recovered using a combination of the results of the subcircuits, with some cost in accuracy that is overcome by increasing the number of circuit executions. This extra cost is often called sampling overhead, and it is known to grow exponentially with the number of cuts.

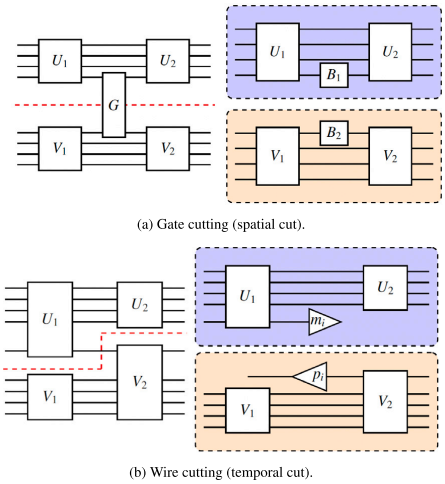
Quasi-probabilistic decomposition of quantum channels. Most circuit-cutting algorithms rely on the quasi-probabilistic simulation (QPS) of a quantum circuit, which uses the quasi-probabilistic decomposition (QPD) of the *quantum channel* of the circuit. A quantum channel $\mathcal{E}$, or *quantum operation*, is a trace-preserving, completely positive linear map between density operators. Quantum channels are typically represented through the operator-sum representation, also known as Kraus decomposition. In this representation, a channel $\mathcal{E}$ acts on a state described by a density matrix ρ as a sum of k terms $\mathcal{E}(\rho) = \sum_{j=1}^{k} E_j \rho E_j^\dagger$, where E_i are (Kraus) operators on the Hilbert space of ρ.

This representation is not unique, i.e., one has the freedom to choose the operators E_i of the representation and still get the same channel $\mathcal{E}$. In particular, one can choose the operators to be quantum gates that are *local* in separate sets of qubits. Consider the n-qubit bipartite system $\rho = \rho^{(1)} \otimes \rho^{(2)}$ with Hilbert space $\mathcal{H} = \mathcal{H}^{(1)} \otimes \mathcal{H}^{(2)}$, where $\mathcal{H}^{(1)}$ and $\mathcal{H}^{(2)}$ are the space of the two unconnected sets of qubits $\rho^{(1)}$ and $\rho^{(2)}$. Now consider a quantum circuit C consisting of products of arbitrary quantum gates, some of them multi-qubit gates acting on both $\mathcal{H}^{(1)}$ and $\mathcal{H}^{(2)}$ simultaneously. Our hardware may not be able to execute those non-local gates, but one can always find a decomposition such that

$$\begin{aligned}\mathcal{E}(\rho) &= \sum_i^m q_i \left(V_i^{(1)} \otimes V_i^{(2)}\right)\left(\rho^{(1)} \otimes \rho^{(2)}\right)\left(V_i^{(1)\dagger} \otimes V_i^{(2)\dagger}\right) \\ &= \sum_i^m q_i \left(V_i^{(1)} \rho^{(1)} V_i^{(1)\dagger}\right) \otimes \left(V_i^{(2)} \rho^{(2)} V_i^{(2)\dagger}\right) \\ &= \sum_i^m q_i \, \mathcal{E}_i^{(1)}\left(\rho^{(1)}\right) \otimes \mathcal{E}_i^{(2)}\left(\rho^{(2)}\right) \, ,\end{aligned}$$

with coefficients $q_i \in \mathbb{R}$ with $\sum_{i=1}^m q_i = 1$, and $V_i^{(1)}$ and $V_i^{(2)}$ are operations acting locally in $\mathcal{H}^{(1)}$ and $\mathcal{H}^{(2)}$ respectively, that our hardware can physically execute. The choice of q_i and the set of $V_i^{(1)}$ and $V_i^{(2)}$ is not unique, and it is known as a QPD of the quantum channel [210].

The q_i can be either positive or negative, which is why they are called quasi-probabilities. The larger the number of negative coefficients in the decomposition, the larger the 1-norm $\kappa = \sum_{i=0}^m |q_i|$ of the QPD becomes. Crucially, this κ quantity is related to the cost of executing the circuit C that has non-local gates, using only local operations [211,212]. Negative probabilities in the simulation of quantum circuits were already known to be related to the "quantumness" of quantum circuit. Thus they could be used as a resource to classically simulate quantum processes by separating the "hard" and the "easy" parts of the circuit [213,214], and also for performing error mitigation through a quasi-probabilistic decomposition of an ideal circuit from noisy ones [215,216].



(a) Gate cutting (spatial cut).

(b) Wire cutting (temporal cut).

Fig. 9. Two schemes for cutting a quantum circuit: gate-cutting (or spatial cut) as shown by Mitarai and Fujii [223] and wire-cutting (or temporal cut) by Peng et al. [224]. Both can be shown to be equivalent to simulating teleportation [225], wire-cutting being analogous to teledata and gate-cutting to telegate.

In practice, to calculate the expected value of an observable, we sample the outcome of the circuit measured in the appropriate basis for some number of shots N_s. We want N_s to be large enough so as to have some desired degree of accuracy ϵ. When using QPS to simulate circuits, the variance of the result increases with κ^2, and we have to compensate for increasing N_s proportionally. This effect is known as sampling overhead. This overhead is multiplicative, increasing exponentially with the number of cut gates N_c. Given a large enough number of shots, the outcome of the original circuit is recovered with arbitrary precision. However, noise sources will still introduce a bias in the computation independent of the QPS, as noise is a separate quantum channel evolving the state ρ. Quasi-probabilistic methods can also aid in error mitigation, which as mentioned above has some practical overlap with circuit cutting. Furthermore, there is experimental evidence that QPS can reduce the effect of noise sources by employing smaller circuits [217,218]. Another issue appearing when sampling a QPS appears when reconstructing the evolved ρ from the partitions. Due to finite sampling error, finding a distribution with negative terms is possible. To solve this, post-processing can be used to find the "most likely" output state [219,220], although this is not necessary for calculating expected values of observables.

Finding an efficient QPD of a general circuit C, i.e., a QPD with a small κ, is difficult. If the circuit is known to produce a state with a particular bi-partite structure, one can turn to similar techniques to execute the parts locally, such as Entanglement Forging [221,222]. However, the main direction that has been followed in the literature for circuit cutting was to perform only the QPD of specific regions of the circuit with sparse correlations, targeting non-local gates or wires.

Circuit cutting techniques: gate-cutting and wire-cutting. One preliminary work, which was later labeled as circuit cutting (and in particular, wire-cutting), was the *cluster simulation scheme* by Peng et al. [224], which decomposes the corresponding tensor network of a given quantum

circuit into smaller clusters. Inter-cluster communication is then simulated classically. The authors apply these techniques for Hamiltonian simulation using the Variational Quantum Eigensolver (VQE) [226], and suggest using this hybrid variational ansatz for future modular architectures. Later, Mitarai and Fujii [223] introduce the idea of *virtual two-qubit gates*, where the action of the virtual gate is substituted with local operations. This way they only apply QPS for the non-local gates we want to get rid of. Given that most QPUs can only execute single- and two-qubit gates, it is more convenient to find an efficient QPD of the particular two-qubit gate and simulate them with local single-qubit gates. The total overhead of the QPS then scales as $\mathbb{O}(\kappa^{2N_c})$ with N_c being the number of virtual gates. Mitarai and Fujii also provide an efficient QPD for a two-qubit gate with $\kappa = 3$ at most, from which most common two-qubit gates such as $CNOT$, CZ, $RZZ(\theta)$, etc., can be derived. Fig. 9 compares the two methods, which can also be used simultaneously in the same circuit.

The main drawback of circuit cutting is the exponential overhead. This overhead has been proven to be strictly exponential [227], so it cannot be reduced to a polynomial increase and it proves a big challenge for scaling to large problems. Still, minimizing κ is an active research topic. In [225,228], the minimal sampling overhead to cut wires and two-qubit gates is derived analytically. Brenner et al. [225] show that cutting an identity gate that transported the state of the qubit before and after the cut (a wire cut) is equivalent to a teleportation protocol. As shown in Section 2.2, one needs a prepared Bell state and two bits of classical communication to teleport one qubit of data. Gate-cutting of a Bell pair between two qubits ($\kappa = 3$) is already more efficient than cutting a wire ($\kappa = 4$), although it requires ancilla qubits.

Piveteau et al. [228] suggests that this overhead can be reduced when jointly cutting multiple gates or wires, using classical communication between partitions. This is because the joint QPD of a larger unitary of N_c Bell states (also called *Bell State factory*) required for teleporting N_c gates has a lower overhead than individually cutting N_c Bell states. This overhead now scales better $\kappa = (2^{N_c+1} - 1)$, albeit using local operations and classical communication (LOCC) and ancilla qubits as requirements (one per partition and cut). While Piveteau et al. did not give the explicit QPD of this *Bell State factory*, it was later provided in [229] for $N_c = 2$ and $N_c = 3$.

Lowe et al. [230] reduced the ancilla qubit requirements for large-scale Quantum Approximate Optimization Algorithm (QAOA) simulations by combining wire-cutting and random measurement bases (inspired by classical shadow tomography [231]) and subsequent studies improved bounds for multiple wire-cuts with LOCC [232,233]. For gate-cutting [234] improved on Piveteau et al. result, finding an optimal decomposition of an arbitrary two-qubit rotation gate and reducing the ancilla requirements for cutting multiple parallel gates. Soon after, [235] achieved a similar result for clustered Hamiltonian simulation, and [236] did the same for general two-qubit unitaries.

Reducing sampling overhead can also be achieved by cutting larger unitaries. For instance, cutting a SWAP gate using QPS results in a lower overhead ($\kappa = 7$) than decomposing it into three CNOT gates and cutting each individually ($\kappa = 3^3$). This, of course, is the idea behind cutting the Bell State factory in [228] but can also be extended to higher dimension operators like Toffoli gates [236], multi-controlled CZ gates [237], and even the QFT [238]. Furthermore, in the case of Variational Quantum Algorithm (VQA) one can choose variational ansatzes designed with reduced entanglement between parts, so they are easier to partition. This can be in the form of clustered ansatzes for VQE [239,240], or more general ansatzes where the amount of entanglement is tuned so that the overhead of the QPD is always kept below a tolerance value [241].

Other strategies reduce the number of subcircuits in decompositions to lower sampling overhead. Note that, while related in their exponential scaling, the number of subcircuits in a QPD (its 0-norm) is not the same as the sampling overhead (its 1-norm). Reducing subcircuits can aid scheduling and post-processing without increasing κ. Nagai et al. realize this by introducing pre- or post-selection methods for quantum channels [242], while Chen et al. use approximate methods that directly neglect some of the elements [243,244].

Efforts to minimize quantum communication between machines focus on smart qubit assignment. A solution that minimizes the sampling overhead also minimizes the number of Bell pairs in a DQC protocol, and thus, the same compiling tools could be used for both techniques. Combining gate- and wire-cutting finds better partitions [245], which is crucial for DQC, not only for circuit cutting, as already detailed in Section 4.1.1. Some Software Development Kits (SDKs), such as Qiskit or Pennylane, incorporate these techniques in their compilation routines. Moreover, several tools such as CutQC [246], ScaleQC [247] or SuperSim [248] perform the whole circuit cutting pipeline, finding cuts, executing the subcircuit, and reconstructing the state. There is also, as we will delve in Section 4.2, a compiler named Qurzon [249] which performs all the aforementioned techniques – in fact, it uses CutQC in combination with other tools.

Herzog et al. [250] illustrated the practical application of these methods by cutting a QAOA ansatz for a combinatorial optimization problem. Their approach combined the strategies in [224,232] to reduce ancilla and classical communication requirements, while utilizing classical graph shrinking techniques to lower the overall overhead. However, the authors noted that the same classical techniques could potentially solve the problem faster through purely classical computation. Similarly, IBM's recent work [229] demonstrated the execution of a 142-qubit graph state across two 127-qubit QPUs. By implementing Piveteau et al.'s concept of a Bell State factory, they utilized real-time classical communication and parametric circuits to reduce compilation time, showcasing the practical application of circuit cutting in large-scale quantum computations.

Despite these advancements, circuit cutting occupies a challenging position in the quantum computing landscape. It is more suited for problems with sparse entanglement, which are often more easily tackled using classical methods. At the very least, circuit cutting can be useful for early DQC applications, serving as a transitional strategy until robust quantum communication networks are fully realized. This technique could thus provide a crucial stepping stone in the evolution of quantum computing infrastructure.

4.1.3. Embarrassingly parallel

The term *embarrassingly parallel* was coined within the HPC domain to describe applications that are inherently amenable to parallelization without significant effort. Notable examples include bag-of-tasks workloads – jobs devoid of dependencies that can be executed in any sequence – and parameter sweep applications, which involve numerous parallel executions with varying parameter configurations.

Similarly, in the context of quantum computing, the term *embarrassingly parallel* refers to the scenario where a problem can be divided into multiple smaller computations that can be executed independently without the need for direct communication among them. The simplest example of this in the quantum case is the *distribution of shots*, where a quantum algorithm or kernel needs to be executed multiple times without any structural changes – except for the modification needed to map the circuit to the different QPUs. Despite the quantum nature of the tasks involved, this method essentially involves classical parallelism.

A different approach comes from a distribution of the circuits needed to reconstruct the expectation value of a given observable or to support the optimization protocol. This allows several possibilities:

- *Distribution of terms in an observable.* The distribution of the expectation value terms $\langle O_i \rangle$ of a given observable $\langle O \rangle = \sum \langle O_i \rangle$ is a case of embarrassingly parallelization. An intuitive example is the VQE [226], where the function to minimize is the energy, i.e., the expectation value of a Hamiltonian $\langle H \rangle$. Depending on the specific problem, Hamiltonians can be commonly expressed using fermionic operators in second quantization formalism, as in the case

of many systems in condensed matter/chemistry, bosonic operators, or directly in Pauli operators, as in spin Hamiltonians that apply to different problems in physics, route optimization, protein folding [251], and scheduling, among others. In all cases, except the last one, the Hamiltonian has to be mapped to qubit instructions via some encoding techniques [252,253]. After that, it appears as a weighted sum of tensor products of Pauli operators, most commonly known as Pauli strings. Initially, each Pauli string can be individually sent to different QPUs. However, the scaling in the number of Pauli strings for complex problems makes this procedure inefficient. A common practice is to form groups of Pauli strings that will share the same quantum circuit to construct their expectation value. These groups are made of commuting Pauli operators that are determined using some classical routine. The simplest strategy is *qubit-wise commutativity*, where each of the commuting groups built can be measured using a single quantum circuit without difficulties [252]. An alternative is *general commutativity*, which is more efficient in reducing the number of commuting groups but entails the non-trivial task of finding the appropriate unitaries for the joint measurement of the groups [252,254].

- *Gradient and Hessian's distribution*. Just like the preparation of a parameterized trial wave function $|\psi(\theta)\rangle$ to our problem, first and second partial derivatives of the state $|\psi(\theta)\rangle$ can be analyzed with a quantum computer [255–257]. In many cases, the quantum circuits that arise from the partial derivatives can be expressed as a linear combination of circuits that use the same structure of the original circuit to prepare $|\psi(\theta)\rangle$, with a shift in their parameters, which is known as parameter shift rule [258].
- *Distribution in a gradient-free optimization*. That is a particular case of distribution that sources from the usage of gradient-free optimizers such as evolutionary ones. These optimizers overcome the need to compute gradients at the cost of using several individuals/particles that interact in a certain way to modify their parameters or generate other candidates. That is the case, for example, of Differential Evolution and the Particle Swarm Optimization algorithms [251,259,260]. Each individual is a different set of parameters that can be executed in parallel using the same quantum circuit structure. One of the possible benefits of the previously mentioned optimizers is that they can mitigate problems in the optimization landscape [259,261]. However, this would come at the cost of increasing drastically the number of circuit executions.
- *Distribution of data*. As in the case of classical Machine Learning, another possibility is to distribute the data or the model during the training. For example, [262] proposes a tool for distributing training of Quantum Machine Learning models that can also be used for VQEs. A federated approach has also been proposed [263].

There are some packages that permit the distribution of these kinds of jobs among several QPUs [262,264], based on a master-worker architecture. These packages must cope with additional issues not seen in classical Machine Learning distributed learning, such as the different architectures of the QPUs (different gate sets, different topology, or different timing for execution), the noise of each single QPU and the possible drift of these errors with the time, for counting some of the current challenges. Additionally, these techniques can also be used when circuit cutting is applied.

Another paradigm that can be considered in this context is *multi-programming* of quantum computers. The segmentation of a QPU, better known as multi-programming in quantum computing, can maximize the hardware throughput – the number of used qubits divided by the total number of qubits – and reduce the runtime. The pioneering work for multi-programming by Das et al. [265] advocated for its use to enhance the utilization and throughput of Noisy Intermediate-Scale Quantum (NISQ) computers, wherein the qubits are employed to execute multiple workloads concurrently. Other works introduce enhancements like selecting the appropriate number of circuits to execute, qubit mapping, device benchmarking, crosstalk[4] characterization, or even vulnerability analysis [266–273].

Another paradigm that may be interesting to delve into is *quantum offloading*. As mentioned in the introduction, QPUs is intended to be seamlessly integrated into classical HPC infrastructures, working along other hardware accelerators. This way of distributing the workload allows concurrent computations of classical and quantum tasks, letting CPUs proceed with calculations while QPUs accelerate specific processes in which the so-called *quantum advantage* takes part.

A profound quantum offloading analysis diverges from this work's main scope, but some relevant works can be outlined. For instance, the eXtreme-scale Accelerator programming framework (XACC) is a system-level software infrastructure for quantum-classical computing that promotes a service-oriented architecture to expose interfaces for core quantum programming, compilation, and execution tasks [8]. Strongly related is QCOR, a language extension specification of C++ that enables single-source quantum-classical programming and that employs XACC as a base [9]. Another work leveraged the OpenMP API to target quantum devices, which provides an easy-to-use and efficient interface for HPC applications to utilize quantum computing resources [274]. Similar to this were the efforts made to add QPUs to the OpenCL ecosystem of execution [7]. Even the NVIDIA company has developed the CUDA Quantum Platform for hybrid quantum–classical computation [275], enabling the aforementioned integration and programming of QPUs along with other accelerators.

4.2. Compilation

After resolving the distribution challenge, it is essential to explore the compilation process thoroughly. We will adhere to a structure akin to the classical approach, which involves an analysis phase, an intermediate representation referred to as Quantum Intermediate Representation (QIR), and a synthesis phase. This framework will aid in comprehending the compilation process for DQC and underscore the disparities between classical and quantum computing in terms of compilation.

4.2.1. Analysis phase

The analysis phase in the distributed and monolithic quantum compilation is quite similar, with the additional challenge in the distributed case of limited literature and software development compared to the monolithic counterpart. In the monolithic scenario, the underutilization of standalone languages is not because they do not exist; rather, options like Scaffold [276], Q# [277], isQ [278], Q|SI⟩ [279], among others, are available. However, they are less favored due to the need for users to understand and adapt to these languages. In contrast, libraries like Qiskit [280], Cirq [281] and Qulacs [282], built on well-known classical languages such as Python (Qiskit and Cirq) and C++ (Qulacs), are more widely adopted. This situation is even more pronounced in the distributed case because there is a shortage of standalone languages specifically designed for distributed purposes. Consequently, the previously mentioned quantum monolithic libraries are often repurposed to simulate the distributed structure.

This is the case for Quantum MPI (QMPI) [283], which represents an extension of the Message Passing Interface (MPI) protocol for distributed quantum systems. We refer to this as a formal approach due to the absence of a usable library that allows for actual or simulated DQC. However, a reference implementation for QMPI has recently been published [284], although none of the code is available for use, neither in open source nor as a binary, to the best of our knowledge.

The aim of QMPI is, obviously, to add quantum functionalities to an already widely used specification such as MPI. For this purpose, it

[4] Crosstalk is an unwanted coupling between qubits. It is one of the noise sources in NISQ devices and can condition the hardware throughput.

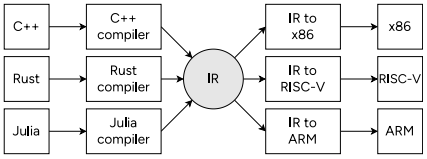


Fig. 10. The significance of intermediate representation in the compilation process - Facilitating decoupling between high-level and machine code.

defines two types of nodes: classical and quantum. The only difference between them is that classical nodes cannot be the target of quantum directives, whereas quantum nodes can manage both quantum and classical calls. The core of this difference lies in the inherent distinction between classical datatypes and quantum datatypes – bits and qubits – along with the inclusion of EPR pairs, a crucial element for the development of quantum communication protocols, as shown in Section 2. Other than that, although MPI is much more advanced than QMPI, as expected, the communication modes supported by the latter are the same: point-to-point communication and collective operations. Moreover, they define a simple performance model called SENDQ. It is worth mentioning that, contrary to almost all literature on DQC, they anticipate a relatively low logical clock speed for quantum computers due to the overhead introduced by the quantum error correction. Consequently, they do not expect classical communication to significantly affect performance, choosing to ignore classical communication in the SENDQ model. This approach contrasts significantly with all the circuit distribution methodologies discussed in Section 4.1.1, where the focus was primarily on minimizing the number of teledata and telegates, considered the main bottleneck of quantum distribution. The SENDQ model is closely associated with the NISQ era and may not be sustainable when transitioning to the fault-tolerant era.

Anyway, as it is explained in Wakizaka [285], there is a need to develop a proper quantum programming language that takes consideration of a distributed structure and extracts profit from that structure via advanced distributed computational techniques, just as it happens in classical computation.

4.2.2. Distributed quantum intermediate representation

The compilation process is complex; therefore, Intermediate Representations (IR) were introduced to establish a break in the compiler in order to obtain modularity and decoupling [286]. An IR allows to intermediate between the front-end and the back-end, improving the efficiency of compiler development and allowing abstract optimizations to the target machine. Fig. 10 shows the use of IRs as a break in the compilation process to facilitate compiler development so that programs are implemented for abstract machine code such as an IR.

An important feature of IRs is that they have to be able to represent the operations of different high-level languages to be implemented in different machine codes. Therefore, with the evolution of quantum computing, it is necessary to extend classical IRs (or create new ones) to include quantum instructions. This process has been evolving in recent years, where the number of quantum IRs has grown considerably [287–292].

For DQC, specialized IR are needed to allow the use of classical and quantum communication instructions between different PUs. This objective is what InQuIR [293], an IR specialized in DQC, aims to solve.

To exemplify the operation of this IR, we use the circuit shown in Fig. 3(b), which implements a CNOT remote gate between two separate nodes, but connected through a Bell pair $|\Phi^+\rangle$. Fig. 11(a) shows the OpenQASM code to implement this, which does not consider communication directives. The compilation of OpenQASM to InQuIR produces the code shown in Fig. 11(b) for node 0 and Fig. 11(c) for

```
1 OPENQASM 2.0;
2 qreg q[2];
3 h q[0];
4 cx q[0],q[1];
```

(a) OpenQASM 2.0 code for the creation of an EPR pair.

```
1  0 {
2    world = open[0,1];
3    q0 = init();
4    _cq0 = genEnt[1](10);
5    CX q0 _cq0;
6    _m0 = measure _cq0;
7    free _cq0;
8    send[1](world, l1:_m0);
9    recv(world, l1_2:_m1);
10   Z[_m1] q0;
11 }
```

(b) InQuIR code for node 0 (qubit 0).

```
1  1 {
2    world = open[0,1];
3    q1 = init();
4    _cq1 = genEnt[0](10);
5    CX _cq1 q1;
6    H _cq1;
7    _m2 = measure _cq1;
8    free _cq1;
9    send[0](world, l1_2:_m2);
10   recv(world, l1:_m3);
11   X[_m3] q1;
12 }
```

(c) InQuIR code for node 1 (qubit 1).

Fig. 11. InQuIR representation of the creation of an EPR pair using remote gates.

node 1. InQuIR automatically adds the necessary directives to do the remote operation using the telegate technique.

The IR code extends the basic quantum operations to a distributed setting, where quantum communication and entanglement generation across different nodes (0 and 1) are involved. Lines 2 to 4 in both Figs. 11(b) and 11(c) correspond to the initialization of the communication channel between both nodes, the initialization of the local qubits, and the generation of the EPR pair, respectively. Lines 5–6 in 11(b) and 5–7 in 11(c) correspond to the gates and measurements. The measurement results are transferred between the two nodes by `send/recv` operations and used in the conditional gates.

4.2.3. Synthesis phase

In classical compilation, this corresponds to the lowest level of abstraction. At this stage, low-level, less human-readable languages—analogous to classical assembly languages—are utilized within the compilation chain. Although it is challenging to map each quantum compilation stage to distinct levels of abstraction, a parallel with classical assembly can be established through the use of Quantum Assembly Language (QASM). There are a lot of different versions, such as OpenQASM [294], cQASM [295], eQASM [296] and f-QASM [297]. But, to the best of our knowledge, only NetQASM [298] takes into account an underlying distributed structure.

In [298], Dahlberg et al. introduced an abstract model featuring a Quantum Network Processing Unit (QNPU) for end-nodes in a QN. NetQASM is proposed as an Instruction Set Architecture (ISA) designed to execute arbitrary programs on end nodes equipped with the QNPU. So, NetQASM can be seen as a low-level, assembly-like language tailored for the quantum segments of quantum network program code. It specifies the interaction between the QNPU and executes QN code, a functionality not available in other QASM languages. The language is designed to be extensible, with a core set of instructions for classical control and memory operations and a set of quantum-specific

instructions grouped into "flavors". A "vanilla" flavor is introduced for universal, platform-independent quantum gates, enabling platform-independent quantum network program descriptions, with the possibility of developing platform-specific flavors for optimized quantum operations on specific hardware.

It is also worth mentioning the work of Ying and Feng [299]. They developed an algebraic language for formally specifying quantum circuits in DQC that aims to represent circuits conveniently and compactly, akin to how Boolean expressions are used for classical circuits.

Building on the classical analogy, this stage involves optimizing the code and adapting it to the target machine. The compiler performs operations such as register allocation, branch optimization, loop unrolling, and other well-known optimization techniques. Similarly, quantum compilation employs analogous optimization methods. However, unlike in classical compilation, these techniques are not always applied directly to QASM. Instead, they can be applied to the higher-level languages considered in this work. This distinction underscores the current lack of abstraction in quantum computing.

To maintain consistency with classical methodologies, the remainder of this section will elaborate on the three primary components of the synthesis phase: *optimization*, *qubit mapping*, and *verification*. First, *optimization* and *qubit mapping* will be discussed, as they are fundamental aspects of quantum compilation, particularly in the current NISQ era. Finally, the *verification* stage will be examined. Although *verification* differs in nature from the preceding two components, it serves as a crucial feature in quantum programming by providing an alternative to classical debugging techniques, ans so it will be explained at the end of the section as an important side aspect of the quantum compilation.

Optimization. The optimization phase in monolithic quantum computing encompasses a broad range of techniques aimed at minimizing various metrics, such as the number of 2-qubit gates, the circuit depth, etc. In DQC, we encounter similar optimization challenges as in the monolithic case, but with the added complexity of distributing or cutting the circuits. On the contrary, if the distribution technique performed is embarrassingly parallel, the optimization phase is, naturally, equivalent to the monolithic one, excepting the case of multi-programming where optimizations are subtle and tend to be related with crosstalk and fidelity [267,273].

Delving into circuit distribution, we have discussed in Section 4.1.1 the circuit distribution methods and efforts made to partition the circuit optimally before performing local mapping. In essence, optimization in this case mirrors that of the monolithic case but with the additional consideration of the partitioning problem, which is intricately linked to qubit mapping. Indeed, the close relationship between qubit mapping and circuit optimization is not surprising, even in the monolithic case. It is logical because an efficient mapping of qubits directly impacts circuit performance, much like how effective register management optimizes classical computing tasks. However, although we are only adding one more constraint with the circuit distribution, it is of vital importance since the teleport and telegate costs are significantly higher than those of local 2-qubit gates. As previously discussed in Section 4.1.1, this serves as justification for why circuit partitioning methods consistently aim to minimize the utilization of these remote protocols. Qiu and Chen [300] realized an interesting analysis of this topic, where the quantum cost figure of merit is employed. The quantum cost of a circuit is calculated by summing the cost of each gate present in the circuit. Any gate can be broken down into several basic gates, each with a unit cost, irrespective of their internal complexity. Using this definition of cost, they showed the expensiveness of quantum teleportation and dense coding. However, circling back to the main topic, while we have extensively covered and will further discuss partitioning in the qubit mapping section, we have deliberately chosen not to get deeply into the intricate domain of monolithic quantum optimizations, as it exceeds the scope of this work.

Regarding circuit cutting, optimizations aim at reducing the sampling overhead, or the number of subcircuits. Although both quantities are related in that both increase exponentially with the number of shots, in general, they do not need to scale the same way. The most important of the two is the sampling overhead. Still, a reduction of the number of subcircuits (without an increase in the sampling overhead) can also help in the scheduling and post-processing part of the computation. Some works reduce the sampling overhead by including LOCC, either when jointly cutting several gates [301], or in smart prepare-and-measure protocols in wire-cutting [230,232,233]. Other works attempt to cut larger unitaries [237] or constrain the overhead using parameterized gates [241]. Regarding the number of subcircuits, they can be reduced using pre- or post-selection methods [242], and some of them can be neglected in approximated methods without incurring in large errors [243,244].

Qubit mapping. When it comes to classic computing, register allocation is about finding the best way to use the limited number of registers available to store variables [302]. In the field of quantum computing, qubit mapping can be compared to register allocation in classical computing. This process involves finding an optimal mapping of logical qubits to physical qubits in a quantum device, taking into account the device's connectivity and other constraints. It is important to note the growth in complexity of this process as it moves from classical to quantum compilation. In the realm of quantum compilation, it is not only the use of the qubit's value that must be evaluated – meaning if it is thought to be a communication qubit or a computing qubit. Other factors, such as the error associated with the specific qubit and its interconnection with the remaining qubits, assume significance in the decision-making process. Qubit mapping is an NP-hard problem [303]. Therefore, exact algorithms are only computable for a reduced number of qubits, making it necessary to use techniques that are able to obtain an optimal solution even if it is not the best one. Additionally, the quantum mapping process can be separated into three processes:

- *Gate decomposition:* Refers to the stage in which gates composing the circuit are transformed into a series of native gates implementable in the actual quantum processor. This is one of the aforementioned device's constraints that have been taken into account.
- *Quantum allocation:* Refers to the process of physically assigning specific logical qubits in a quantum processor. For a correct qubit allocation, in most cases, it is necessary to add additional SWAP gates to move the qubit information [304].
- *Quantum routing:* Refers to the task of finding efficient paths for communication between qubits in a quantum processor. This is important when mapping gates of two logic qubits that are not interconnected to maximize efficiency [305,306]. For a thorough analysis of the qubit routing problem, one can check the review on the subject by Barnes [307].

Regarding DQC, it is essential to distinguish between distribution methods that require partitioning and those that do not. In the former case, where partitioning is necessary, the qubit mapping problem aligns with the classical problem. Still, it includes the additional challenge of optimizing circuit partitioning to minimize communication, as detailed in Section 4.1.1, where we already mentioned how linked those methods are with this stage of compilation.

Nevertheless, a few works that have not been mentioned in that section are of interest. The first one is the work of Mao et al. [308], who named the problem as qubit allocation problem for distributed quantum computing (QA-DQC), proved the NP-hardness of it and proposed two algorithms to deal with it: a heuristic local search algorithm and a multistage hybrid simulated annealing (MHSA) algorithm. In the latter, they combine the local search algorithm and a simulated annealing meta-heuristic algorithm, along with extensive simulations to evaluate it. The second work was also carried out by Mao et al. [309] that

proposed a probability-aware qubit-to-processor mapping model, incorporating communication overhead between processor pairs determined through probabilistic analyses based on link entanglement generation rates. Additionally, they introduced a multi-flow routing protocol to enhance overall entanglement rates. Subsequently, they employed a multistage hybrid simulated annealing algorithm, which is reminiscent of the previous one, to minimize total communication overhead. As we have already mentioned, extensive simulations are conducted to demonstrate the effectiveness of these solutions across various system settings. The third work of interest in this line was the one developed by Nakai [310], which deeply developed the qubit allocation problem for DQC along with a formal definition of the problem as an optimization problem similar to how we have defined the partitioning one. Finally, the last work was developed by Chen et al. [311], where they focused on the step following the circuit partitioning, i.e., the qubit routing stage. Specifically, they focused on investigating the influence of the quantum state transmission direction during the execution of global gates on the number of transmissions and subsequent routing. It utilizes a heuristic algorithm, called Genetic Algorithm for Global Gate Direction Optimization (GAGDO), to ascertain the optimal transmission direction for all global gates in the circuit, with the goal of minimizing the overall cost of the executable circuit generated in the distributed architecture model.

Also, two works have been developed to characterize the inter-core qubit traffic in which some benchmarks arise in order to analyze mapping performance [312,313]. They employed the OpenQL compiler [314], which is not a distributed compiler *per se* but allows the embedding of a modified version of the Qmap mapper [315]. In particular, for this case, they extended it to the multi-core case employing the proposal by Baker et al. [204], i.e., the FGP-rOEE algorithm, already explained in Section 4.1.1.

In the cases of embarrassingly parallel distribution that do not require partitioning, the qubit mapping problem mirrors that of the monolithic case, with the added complexity of needing to perform mapping for each QPU. This complexity arises from the potential differences in architectures among the QPUs contained in the distributed scheme. There is just one case in the embarrassingly parallel scenario where qubit mapping differs from the monolithic case: the multi-programming scenario. This paradigm of quantum execution, which involves segmenting the QPU, imposes a series of constraints on the qubit mapping problem. One of the first approaches was the already mentioned work by Das et al. [265]. Three techniques were developed in this work:

1. Fair and Reliable Partitioning (FRP) algorithms, developed to partition qubit resources into multiple groups fairly while avoiding qubits or links with excessively high error rates.
2. Delayed Instruction Scheduling (DIS) policy, devised to mitigate interference from measurement operations of one program on the gate operations of co-running programs.
3. Adaptive Multi-Programming (AMP) design, proposed to monitor reliability impact at runtime and revert the system to isolated execution mode if the impact is high.

Different techniques were developed under the QuCloud framework by Liu and Dou [267]. In this work, they also developed three approaches. First, they utilized community detection techniques to partition physical qubits among concurrent quantum programs, mitigating resource waste. They even proposed a new technology based on these techniques called Community Detection Assistant Partitioning (CDAP). Second, they designed the X-SWAP scheme, which enables inter-program SWAPs and gives priority to SWAPs linked with critical gates to minimize SWAP overheads. Finally, they introduced a compilation task scheduler that prioritizes the compilation and execution of concurrent quantum programs based on estimated fidelity for optimal performance.

This was further extended in a subsequent work by the same authors under the QuCloud+ framework [273], in which they tried to take into consideration the crosstalk effect on real-world applications.

Verification. The verification of quantum programs is a significant side aspect of quantum compiling. Unlike in the classical world, where developers rely on debuggers to identify and fix errors, debugging quantum programs is inherently difficult due to the destructive nature of measurement. Once a quantum state is measured, it collapses irreversibly, making it impossible to observe the state at different time steps without altering it. Therefore, the verification of quantum programs becomes crucial for ensuring the correct functionality of a quantum circuit. It is essential to incorporate this verification step as a phase in the synthesis stage of compilation. This ensures that the circuit is checked immediately before execution and after optimizations have been applied to confirm that those optimizations have not altered the functionality of the quantum circuit. In the monolithic realm, several approaches have been made combining optimization and verification in what is usually referred to as *verified optimization* [291,316,317].

One way of verifying quantum programs is using quantum process algebras, which are derivations of the classical process algebras. Process algebras, also known as process calculi, are mathematically rigorous languages with well-defined semantics that allow the description and verification of properties of concurrent communicating systems, including, in this case, quantum systems.

There are some examples of these types of formal methods. For instance, Extended Quantum Process Algebra (eQPAlg) [318], which extends Quantum Process Algebra (QPAlg) [319]. More specifically, QPAlg provides a homogeneous style for formal descriptions of concurrent and distributed computations, encompassing both quantum and classical components. As authors claim, QPAlg introduces quantum variables, operations on these variables – unitary operators and measurement observables – as well as different forms of communication involving the quantum realm. The operational semantics ensure that these quantum objects, operations, and communications adhere to the postulates of quantum mechanics. Regarding eQPAlg, it extends the previous formal specification to accommodate the concept of formally specifying the quantum teleportation protocol, which has been shown in this work to be a key part of the quantum distribution model. The relationship between quantum process algebras and the algebraic language defined in the aforementioned work by Ying and Feng [299] can be compared to that between classical process algebras and Boolean algebra. In broad terms, quantum process algebras are well-suited for high-level formal specification of DQC, while the language Ying and Feng paper is mainly intended to describe low-level circuit implementation.

Regarding the verification of distributed quantum programs, the work of Feng et al. [320] introduced a distributed programming language designed for formalizing and verifying distributed quantum systems. They presented a Hoare-style logic[5] that is both sound and complete, aiding in the analysis and verification of quantum programs, including quantum teleportation and CNOT gates. Talking specifically about distributed quantum protocols, Wang's work [322] profoundly delves into the verification of several distributed quantum protocols, such as the BB84 protocol [323].

4.2.4. Available compilers

Not many full-stack tools or compilers are designed considering a distributed quantum scheme as a base. In fact, to the best of our knowledge, there is no compiler for DQC available for use, just conceptual designs and prototypes. These conceptual quantum compilers can be classified depending on which type of distribution they use from the ones described in Section 4.2, i.e., usual circuit distribution, circuit

[5] Hoare logic is indeed a formal system equipped with a set of logical rules used for rigorous reasoning about the correctness of computer programs [321].

Table 2
Summary of available compilers for DQC, including their authors, reference, descriptions, main focus, and categorization by distribution type.

Category	Tool/Compiler	Authors	Reference	Available	Main focus	Description
Circuit distribution	Distributed Quantum Compiler	Ferrari et al.	[324]	✓	Circuit depth minimization with specific partitioning strategies	Designed to minimize circuit depth using strategies based on data-qubit swapping and entanglement swapping. Compared against works like Martínez and Heunen.
Circuit distribution	Modular Quantum Compilation Framework for DQC	Ferrari et al.	[325]	✓	Comprehensive optimization considering network, hardware, and specific algorithms	A modular framework considering network and device constraints. Includes qubit assignment with METIS, EPR pair minimization algorithms, and optimized local routing.
Circuit distribution	Cuomo's compiler	Cuomo et al.	[326]	✗	Optimization of distributed architectures for dynamic networks	Models the compilation problem using Integer Linear Programming and time-expanded network representations. Optimized for dynamic network problems and quasi-parallelism.
Circuit cutting	Qurzon (with CutQC)	Chatterje et al.	[249]	✓	t|*ket*⟩ for optimal qubit routing. Reconstructs original circuit results	Employs CutQC for cutting circuits into optimal subcircuits without quantum communications. Schedules execution using a greedy algorithm.
Embarrassingly parallel	palloq	Ohkura et al.	[327]	✓	High-fidelity layout synthesis for multi- programming scenarios	Manages multi-programming with layout synthesis based on noise adaptative layouts. Introduces a crosstalk detection protocol and integrates randomized benchmarking for multi-circuit allocation.
Combining techniques	Quantum Divide and Conquer Algorithm (QDCA)	Tomesh et al.	[328]	✗	Hybrid variational approaches combining cutting and distribution	Combines circuit cutting and distribution for hybrid variational applications. Uses graph partitioning techniques like METIS and KL for mapping large combinatorial optimization problems to distributed architectures.

cutting, and embarrassing parallelism. Table 2 provides a summary of the available compilers, detailing their authors, reference, descriptions, main focus, and categorization by distribution type, as discussed below.

Compilers for circuit distribution. Ferrari et al. [324] designed a distributed quantum compiler that focuses on the minimization of the depth of the circuit and, for this matter, two different techniques are tested: the *data-qubit-swapping-based* strategy and the *entanglement-swapping-based* strategy. They compared the performance of the partitioning – and, hence, of the distribution – of these two strategies with the already analyzed work by Martinez and Heunen [189]. Also, Ferrari et al. [325] designed a versatile modular quantum compilation framework for DQC, which considers both network and device constraints and characteristics. For qubit assignment, they employed METIS's multilevel k-way partitioning. Moreover, for gate scheduling, they implemented an algorithm to minimize the consumed EPR pairs and a local routing algorithm that scans the circuit and, for every gate that involves qubits not directly connected on their specific QPU, it computes the shortest sequence of necessary SWAP gates. The experimental evaluation of a quantum compiler based on this framework was demonstrated, using circuits of interest such as VQE, QFT, and graph state preparation, characterized by varying widths – ranging from 0 up to 600 qubits.

Cuomo et al. [326] modeled the compilation problem using an Integer Linear Programming formulation inspired by the extensive theory on dynamic network problems. They defined the problem as a generalization of the quickest multi-commodity flow, enabling optimization using techniques from the literature, such as a time-expanded representation of the distributed architecture. This approach, which also incorporates quasi-parallelism[6] allows for more efficient circuit operation and broader solution exploration. The work is modular, enabling adaptation to circuits with varying degrees of operation commutativity and leveraging existing network flow literature. The study aims to refine compiler efficiency and performance through an in-depth analysis of quantum circuits and focus on normal forms. Testing on square and hexagonal lattice topologies, showed that square lattices offer superior performance, attributed to their favorable edges-to-nodes ratio, indicating promising avenues for future quantum computing advancements.

Compilers for circuit cutting. As for now, the only quantum compiler considering the circuit-cutting strategy, as was explained in Section 4.1.2 is Qurzon [249]. For the first part of the compilation, an algorithm responsible for cutting the circuit into optimal parts is employed, called CutQC [246]. After the circuit is cut into several pieces, a scheduling algorithm is responsible for the execution of each of the pieces in the available quantum devices. This problem is nothing more than a classic problem of scheduling jobs, well known in the HPC environment. In this case, a greedy algorithm is employed, at least in the theoretical development of the compiler (since to obtain the results, they applied a so-called "naive" algorithm, which is not specified). For the optimal qubit routing, they reach out for the work of t|ket⟩ [329]. Then, a distributed parallel execution is performed over the whole group of subcircuits employing the different devices, and once the results are

[6] The authors define quasi-parallelism as a relaxed version of parallelism based on grouping logically sequenced gates within the same time step.

obtained, the CutQC work is again used to reconstruct the result of the original circuit using every result obtained in each subcircuit.

Compilers for embarrassing parallelism. Despite the absence of compilers specifically designed for embarrassingly parallel tasks in quantum computing, the inherent parallelizable nature of these tasks – primarily the distribution of shots across multiple QPUs – means that any quantum compiler or framework could be easily modified to support this mode of distribution. This adaptability is due to the fact that the distribution of computational tasks among different processors is a well-established practice in the field of HPC. Consequently, leveraging existing classical job distribution techniques allows for the straightforward parallel execution of quantum computations on multiple QPUs, highlighting a seamless integration of classical parallelism principles within quantum computing frameworks.

Nevertheless, an appreciation of the multi-programming case has to be made. Even though the already presented QuCloud and QuCloud+ [267,273] are considered mapping mechanisms, they possess a compilation task scheduler and could be naturally extended to be able to perform as compilers with a multi-programming approach. This is precisely the scope of *palloq*, presented by Ohkura et al. [327], which includes a layout synthesis for multiple quantum circuits and a job scheduler to manage efficient and high fidelity quantum multi-programming. This compiler takes multiple quantum circuits, written in OpenQASM, and the device's local gate error information as input. Their layout synthesis employs a heuristic based on noise-adaptive layout, where the device's calibration data is analyzed to search for improved allocation using a greedy approach. Additionally, they propose a software-based crosstalk detection protocol utilizing a novel combination of randomized benchmarking methods to characterize the hardware's suitability for multi-programming.

Compilers combining types of distributions. The work from Tomesh et al. [328] combines aspects of circuit distribution with the circuit-cutting technique [328]. This work introduced an algorithm called Quantum Divide and Conquer Algorithm (QDCA), a hybrid variational approach aimed at mapping large combinatorial optimization problems onto distributed quantum architectures. The QDCA specification contains several key elements: the partition of the input combinatorial optimization problem into multiple subproblems, the construction of the variational quantum circuit, and the execution of it on distributed quantum computers using quantum circuit cutting techniques. The partition of the input is where the classical techniques of graph partitioning employed for circuit distribution take place, in this case, KL and METIS. Even though it is not circuit distribution *per se*, it employs the graph partitioning techniques used in this kind of distribution to perform circuit cutting, which narrows the boundaries between these two approaches. This work presents quantum circuit cutting as a compilation tool within a hybrid, variational application. With this approach, they claimed to achieve approximate solutions to Maximum Independent Set (MIS) problems.[7]

5. Application layer

This section explores proposed quantum applications that leverage some of the methods previously outlined. Any quantum application executing at least one quantum circuit requiring multiple shots is inherently parallelizable, as the required shots can be distributed across available QPUs or the circuit can be partitioned using telegates or teledata. However, it is important to note that such parallelization does not necessarily guarantee enhanced performance; in fact, it could lead to a significant degradation.

[7] The MIS problem is a classic NP-Complete combinatorial optimization challenge defined on a graph $G = (V, E)$. Its objective is to identify the largest feasible independent set within G, where an independent set, denoted as $S \subset V$, consists of nodes that are not adjacent to each other.

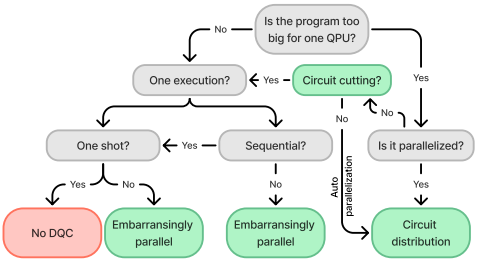


Fig. 12. Decision workflow for executing a quantum application in parallel.

Similar to classical HPC computing, the motivations for executing an application in parallel may include insufficient resources on a single QPU (e.g., a lack of qubits), stringent time constraints (where the time-to-solution fails to meet the requirements of the intended use case), distributed input data across various storage locations (it is easier to move the computation close to the data than the reverse), concerns over data security or confidentiality, among other considerations. The decision to parallelize can be made either by the user or delegated to an automatic scheduler.

For instance, focusing solely on spatial constraints (i.e., the number of qubits) and the required shots to achieve an acceptable result, Fig. 12 illustrates a decision workflow for selecting the appropriate execution method for an application. If the circuit demands more qubits than are available on a single QPU and the program is parallelized, only the required number of QPUs needs to be selected to execute it in parallel (e.g., circuit distribution, as outlined in Section 4.1). Conversely, if the application is not parallelized, a preliminary step could involve assessing the feasibility of circuit cutting — dividing the problem into multiple smaller circuits that can be executed independently without quantum communication, which implies effectively transforming the task into an embarrassingly parallel process. Should circuit cutting prove insufficiently efficient, an autoparallelization mechanism may reconfigure the circuit into a genuinely parallel program.

Conversely, if the basic circuit fits within the available QPU capacity, other options exist to use multiple QPUs in parallel to accelerate execution. For instance, if the problem involves running several circuits that can be executed independently, such as during the optimization of a variational quantum circuit, these instances can be distributed across the available QPUs. Additionally, the same circuit instance can be split among several QPUs, each handling only a fraction of the required shots. However, if these circuits are interdependent, where the execution of one depends on the results of another, only shot parallelization is feasible.

The complexity of the decision workflow of Fig. 12 increases if time constraints are included. In such cases, even if the program fits within a single QPU, parallelization may become necessary to meet the time requirements. However, selecting the number of QPUs must be approached cautiously, as parallel execution inherently introduces overhead that must be accounted for.

In the remainder of this section, we will present some selected examples of distributed quantum applications considering the division of Section 4.1 to show the possibilities of DQC. Specifically, we will discuss applications based on circuit distribution, those leveraging circuit cutting, and others that can be formulated as embarrassingly parallel. Other classifications of DQC applications also exist. For example, [330] recently analyzed the applications of distributed quantum computing, categorizing them into two main types: *resource DQC*, which addresses scenarios where a single device lacks sufficient resources, and *data DQC*, where data is distributed and QPUs can work collaboratively to get the result. Within this framework, the authors review various applications and discuss the challenges of implementing them on current

hardware. Readers are encouraged to consult this article for further examples of applications leveraging DQC.

5.1. Circuit-distribution based applications

As mentioned in the introduction of the paper, one of the first distributed algorithms was proposed by Grover [12]. In this work, he used a circuit distribution with quantum communications to estimate the mean of N numbers between −1 and 1 under ideal conditions. Later, Gupta et al. [22] presented a distributed version of the Grover search algorithm using quantum communications. Initially, the algorithm was shown using only two QPUs, where an additional qubit was needed in each QPU to handle the quantum communications using an EPR pair. The complexity analysis showed that the classical Grover requirements for operations are maintained in this distributed version since the increase in the number of operations due to the distribution scales with the number of qubits as in the original algorithm, but the number of classical communications per iteration is not increased. The paper did not show if the algorithm can scale to more than two QPUs.

One of the key quantum algorithms that present an exponential scaling is the Shor algorithm. The main drawback of this algorithm is the high number of qubits that are needed for a correct execution. Due to this requirement, it is a perfect candidate to use the circuit distribution technique. In [23], a first proposal to use several QPU was made. Firstly, they showed that the QFT could be executed in parallel, substituting each controlled operation with a remote-controlled one, and that the modular exponentiation could be parallelized using a set of QPUs. Although a communication complexity of $\mathbb{O}((log_2 N)^2)$ is needed, being N the number of bits of the number to factorize, and the total number of qubits is increased, the size of each QPU is drastically reduced.

More recently, Gidney et al. [331] analyzed the hardware resources for factoring large numbers, using the Ekerå and Håstad algorithm [332] instead of the Shor one. Applying several optimizations and considering the current methods for making logical qubits, they asserted that a number of 2048 bits can be factorized in 8 h with 20 million noisy qubits (if the operations work in the range of nanoseconds). However, due to the capabilities of the implemented additions needed to factorize the number, the qubits can be reduced to 11 million for each QPU when 2 are used and to 4 million for 8 QPUs. They require a quantum network with a low (but efficient) bandwidth of 150 qb/s. Later, Xiao et al. [333] presented a parallel algorithm that reduces the number of needed qubits, dividing the algorithm between several QPUs, each calculating one subset of the bits. Although it uses several QPUs, it is sequential because to guarantee that the correct state is used on each step, it is teletransported between them at the end of each step.

More well-known quantum algorithms have been parallelized. For example, Neumann et al. [334] studied the Quantum Phase Estimation algorithm using a remote-controlled operation. They compared two possible approaches. The first one is called standard (or automatic), where each controlled operation in the standard QFT is replaced by a remote-controlled operation. This case needs n^2 entangled pairs to execute. The second approach uses the iterative nature of the QFT, aggregating all controlled operations by a single qubit in a unique transport operation. In this case, the number of transport operations is reduced to n. They used a simulator for the experiments, introducing different noise levels to create entangled pairs. The results obtained are similar for both approaches, given the last systematically better results. This experiment showed that automatic partitioning of the problems must take care of possible optimizations and multiple usage of a single pair. One important point is that they studied only the effect of imperfect entangling in the needed pairs without considering other errors such as the measurement, controlled operations between the pairs, and the QPU qubits, etc.

Also, Van Meter et al. [335] studied some of the possible arithmetic operations using teledata and telegate methods in different distributed topologies. They found that, for these problems, the teledata outperforms the telegate method and that a linear architecture is the best choice. In [336], Tan et al. described a parallel algorithm for Simon's problem that still keeps the exponential scaling compared to the classical algorithm.

Recently, Li et al. [337] presented a family of distributed quantum algorithms for the classical Deutsch-Jozsa problem. These algorithms are based on a set of computers with remote communications. However, in the current description, the nature is still sequential, without a clear path to reduce the global depth and time. Finally, Shi et al. [284] made a first proof of concept of using QMPI for the Quantum Phase Estimation and Trotter time evolution, but without including real quantum communications.

5.2. Circuit cutting and other hybrid applications

As described in Section 4.1.2, algorithms based on circuit-cutting only need classical communications to calculate the final solution. Automatic cutting of a circuit (in space or time) is feasible when the number of control operations to cut is limited. However, it is also possible to use non-automatic clever designs to divide a single problem (usually executed using a single quantum circuit) in the execution of several independent quantum programs that later are combined classically to find the right solution.

As already mentioned in the introduction, the paper from Yepez [24] was one of the first proposals to analyze this parallel computation in a hybrid scheme. He considered the case of a system composed of quantum nodes but exclusively connected by a classical network. He named this architecture type-II quantum architecture to differentiate it from the monolithic quantum processors (or type-I), which maintain the global phase coherence. His proposal suggested that some problems need only short spatial and time entanglement, as some kinds of molecules. So they are tractable in parallel quantum computers, unlike other algorithms that need long and spatially large entanglement. For solving those problems, there are three assumptions: first, that the wave function is separable, i.e., can be expressed as a tensor product of subwave functions, each of them residing in one QPU; second, that we can apply a projection operator simultaneously on each qubit of each QPU; and, third, that this projection can be applied after each time step. Yepez proposes a quantum computer composed of many small QPUs arranged in a regular periodic lattice, where local operations are applied to the local qubits simultaneously across the lattice. He applies this proposal to solve problems with lattice gases. For small QPUs, the problems could be tractable using modern Tensor Networks techniques.

In [338,339], Zhou et al. presented distributed quantum algorithms for the Bernstein-Vazirani classical problem and the Grover search, respectively. They divide the binary functions used in the algorithms into a set of subfunctions that can be executed in parallel, getting the final result by composing the different binary parts. In the case of Grover's search, the algorithm only works when a single solution exists, while the extension is still open to multiple solutions. Similarly, Avron et al. [340] studied Deutsch-Jozsa's, Simon's, and Grover's on a distributed environment, finding that, for these algorithms, there are still advantages when comparing with the classical solutions, being the advantage reduced when compared with the fault-tolerant versions. But since these distributed algorithms require shallow circuits, they may be a short-term solution in today's NISQ era.

Several parallel versions of VQAs also use circuit-cutting techniques. For example, [224] used a circuit-cutting based VQE to calculate the ground state of BeH_2. Eddins et al. [221] presented another kind of methodology. They use the Schmidt decomposition to divide a chemical problem of $2N$ qubits in several circuits that need only N qubits, applying VQE to those and joining the results to calculate the final value of the observable. Fujii et al. [341] proposed another method to divide the problem into smaller cases that are combined hierarchically to find the final solution. The technique can be applied when the problem has

some structure that aggregates the entanglement in clusters that can be linked later at a higher level. They applied the technique to a kagome lattice, using several layers of aggregation. This technique could also be used in a hybrid scheme, where part of the calculation is done by QPUs at the first steps, and later, the system is solved by a classical computer using tensor networks.

The usage of these divide-and-conquer techniques can also be applied to combinatorial optimization, where a larger problem can be solved using several computers [328,342], and to Quantum Machine Learning (QML). Marshall et al. [343] examine it for the case of classification. They found that automatic circuit cutting could avoid executing all the subcircuits because some of them do not contribute significantly to the final result and proposed a small change in the process that permits the achievement of results close to the classical Neural Networks.

5.3. Embarrassingly parallel applications

The cutting techniques presented in the previous section convert a complex problem into an example of an embarrassingly parallel application, where each smaller circuit can be executed in parallel, later combining the results classically. Other examples of these kinds of applications are [344,345], which studied the use of partial diffusion operator [346] for Grover's search algorithm. The use of this technique does not reduce the number of required qubits but presents some advantages because each circuit is smaller in depth (and, consequently, needs less time to execute in parallel), and the angles of rotations are bigger, reducing the errors in current quantum devices.

Other quantum algorithms, such as the Phase Estimation for a single phase, can be executed using this formalism [347] because it is possible to split the algorithm into several smaller circuits and combine the results classically at the end. Other classical quantum algorithms, such as the Amplitude Estimation, require large resources that can be approximated by distributing several smaller tasks and post-processing classically their results [348].

In order to get the maximum profit from the available distributed infrastructure or, in the short term, to permit the calculation of VQAs, a combination of the aforementioned techniques can be applied. For instance, DiAdamo et al. [349] proposed placing some circuits needed for calculating the expectation value on available QPUs, using the remaining free qubits to create a distributed version of the Ansatz. Alternatively, the Ansatz could be split using the circuit cutting technique.

6. Final remarks and open challenges

Distributed quantum computing emerges as a clear pathway to enhance the computational capabilities of current quantum systems. In this work, we have presented a comprehensive survey of this field's current state of the art. Using a four-layered model – physical, network, development, and application –, we have guided readers to explore its foundational principles, achievements, challenges, and promising directions for further research. Next, we conclude this work by summarizing some of the most important open challenges in the DQC field:

- **Quantum Teleportation**: It is the most fundamental mechanism required at the physical layer for distributed algorithms in DQC applications. Two types of teleportation protocols are essential: gate teleportation (telegate) and qubit state teleportation (teledata). Telegate enables the remote execution of quantum gates on entangled qubits, allowing quantum information to be manipulated without direct physical interaction. Teledata allows an unknown quantum state processed at one network node to be transmitted to a remote location.

  Open challenges: Enhancing the fidelity of these protocols is an active area of research, as high fidelity is critical for ensuring quantum-computational accuracy in future distributed quantum computers.

- **Quantum Networks**: To achieve interconnected, datacenter-scale QPUs, quantum networks must enable entanglement distribution between any two nodes in the network. Current scalable proposals suggest using quantum networking devices such as repeaters, switches, and routers. These devices support the pre-establishment of entangled qubits through transduction to flying qubits and successive entanglement distribution to end nodes, where computation occurs.

  Open challenges: Such devices must include registers of qubits and implement a limited quantum operation instruction set to execute entanglement distillation, swapping, and teleportation protocols. These advancements are essential to unlock true deterministic DQC architectures. Alternative approaches based on teledata operation with single flying qubits instead of EPR pairs could simplify network architectures. However, further research is required to match the fidelity and efficiency of current entanglement-based protocols. On the other hand, from a practical and market-oriented perspective, current quantum networking solutions are costly and lack the required performance, fidelity, and robustness. Higher-level aspects remain in the early stages of research, including developing networking protocols, scalable connectivity architectures, and robust systems. Auxiliary protocols for synchronization, resource management for entanglement distribution, network service definition, error correction, and qubit encoding must still be developed to achieve fault-tolerant, highly available, and performant quantum networks suitable for distributed quantum computing.
- **Circuit Cutting**: In the current noisy and limited QPUs scenario, circuit cutting can be a useful tool for solving large problems with small quantum computers by distributing parts of the circuit between them without requiring a fully realized quantum network.

  Open challenges: The cost associated with this technique scales exponentially with the amount of cut entanglement between the parts, and, for general quantum circuits, entanglement may have a very complex structure that is unknown beforehand. Some improvements have been proposed, which could avoid the execution of a large fraction of the subcircuits, thereby reducing the computing requirement. Nonetheless, there are criticisms about the overall utility of these techniques. Moreover, dividing circuits and executing them on different QPUs requires a better understanding of the effect of different noise profiles on each QPU. Additionally, when different architectures are employed, the execution times must be carefully managed.
- **Compilers**: Using agnostic compilers to find the best partitions for a general algorithm is similar to auto-parallelism in classical computing, which scales poorly. Designing problems that are easier to partition, such as well-designed ansatzes for variational quantum algorithms or problems tailored for modular architectures, may be more effective. In addition to automatic circuit-breaking tools, experienced programmers can develop methods for dividing and parallelizing algorithms. Tools like QMPI or frameworks for distributing programs are also necessary.

  Open challenges: Research is needed to improve agnostic compilers, develop more efficient partitioning methods, and create tools that enable programmers to parallelize quantum computations across different quantum processors efficiently.
- **Applications**: Embarrassingly parallel applications or those based on circuit knitting are the most widely used solutions in the current NISQ era.

  Open challenges: Further research is needed to develop high-level parallel programming models for distributed quantum computing that efficiently use future quantum networks.

It can be concluded from this work that distributed quantum computing offers a promising way to overcome the limitations of current quantum systems by connecting and scaling quantum processors.

While significant challenges remain – such as improving teleportation fidelity, developing scalable networks and optimizing compilers – advances in these areas will facilitate the path towards robust and fault-tolerant quantum computing, unlocking unprecedented computational capabilities.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve language and readability. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Tomas F. Pena reports financial support and article publishing charges were provided by European Union. Tomas F. Pena reports financial support was provided by Goverment of Spain MINECO. Tomas F. Pena reports financial support was provided by Government of Galicia. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by MICINN through the European Union NextGenerationEU recovery plan (PRTR-C17.I1), the Galician Regional Government through "Planes Complementarios de I+D+I con las Comunidades Autónomas" in Quantum Communication, MINECO (grants PID2019-104834GB-I00, PID2022-141623NB-I00 and PID2022-137061OB- C22), Consellería de Cultura, Educación e Ordenación Universitaria (accreditations ED431C 2022/16 and ED431G-2019/04), and the European Regional Development Fund (ERDF).

Data availability

No data was used for the research described in the article.

References

[1] R. Van Meter, W.J. Munro, K. Nemoto, Architecture of a quantum multicomputer implementing shor's algorithm, in: Theory of Quantum Computation, Communication, and Cryptography, Springer, 2008, pp. 105–114, http://dx.doi.org/10.1007/978-3-540-89304-2_10.

[2] QuEra, Benefits of modular quantum computing for business, 2023, URL https://www.quera.com/blog-posts/benefits-of-modular-quantum-computing-for-business. (Accessed 21 January 2025).

[3] Quantum news, Distributed quantum computing: Scaling quantum power with multiple processors and noise simulation, 2024, URL https://quantumzeitgeist.com/distributed-quantum-computing-scaling-quantum-power-with-multiple-processors-and-noise-simulation. (Accessed 21 January 2025).

[4] IonQ, IonQ achieves critical first step towards developing future quantum networks, 2024, URL https://ionq.com/news/ionq-achieves-critical-first-step-towards-developing-future-quantum-networks. (Accessed 21 January 2025).

[5] N. Saurabh, S. Jha, A. Luckow, A conceptual architecture for a Quantum-HPC middleware, in: Int. Conf. on Quantum Software, QSW, IEEE, 2023, pp. 116–127, http://dx.doi.org/10.1109/QSW59989.2023.00023.

[6] K. Wintersperger, H. Safi, W. Mauerer, QPU-system co-design for quantum HPC accelerators, in: Architecture of Computing Systems, ARCS 2022, in: Lecture Notes in Computer Science, vol. 13642, Springer, 2022, pp. 100–114, http://dx.doi.org/10.1007/978-3-031-21867-5_7.

[7] J. Vázquez-Pérez, C. Piñeiro, J.C. Pichel, T.F. Pena, A. Gómez, QPU integration in OpenCL for heterogeneous programming, J. Supercomput. (2024) 1–22, http://dx.doi.org/10.1007/s11227-023-05879-9.

[8] A.J. McCaskey, D.I. Lyakh, E.F. Dumitrescu, S.S. Powers, T.S. Humble, XACC: A system-level software infrastructure for heterogeneous quantum-classical computing, 2019, http://dx.doi.org/10.48550/arXiv.1911.02452, arxiv preprint.

[9] A.J. McCaskey, T. Nguyen, A. Santana, D. Claudino, T. Kharazi, H. Finkel, Extending C++ for heterogeneous quantum-classical computing, ACM Trans. Quantum Comput. 2 (2) (2021) 1–36, http://dx.doi.org/10.1145/3462670.

[10] J. Gambetta, Quantum-centric supercomputing: The next wave of computing, 2022, URL https://www.ibm.com/quantum/blog/next-wave-quantum-centric-supercomputing. (Accessed 21 January 2025).

[11] J. Gambetta, The hardware and software for the era of quantum utility is here, 2023, URL https://research.ibm.com/blog/quantum-roadmap-2033. (Accessed 21 January 2025).

[12] L.K. Grover, Quantum telecomputation, 1997, http://dx.doi.org/10.48550/arXiv.quant-ph/9704012, arXiv.

[13] R. Cleve, H. Buhrman, Substituting quantum entanglement for communication, Phys. Rev. A 56 (2) (1997) 1201–1204, http://dx.doi.org/10.1103/PhysRevA.56.1201.

[14] J.I. Cirac, A.K. Ekert, S.F. Huelga, C. Macchiavello, Distributed quantum computation over noisy channels, Phys. Rev. A 59 (1999) 4249–4254, http://dx.doi.org/10.1103/PhysRevA.59.4249.

[15] J. Eisert, K. Jacobs, P. Papadopoulos, M.B. Plenio, Optimal local implementation of nonlocal quantum gates, Phys. Rev. A 62 (5) (2000) http://dx.doi.org/10.1103/PhysRevA.62.052317.

[16] D. Collins, N. Linden, S. Popescu, Nonlocal content of quantum operations, Phys. Rev. A - At. Mol. Opt. Phys. 64 (2001) 7, http://dx.doi.org/10.1103/PhysRevA.64.032302.

[17] D.P. DiVincenzo, The physical implementation of quantum computation, Fortschr. Phys. 48 (2000) 771–783, http://dx.doi.org/10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E.

[18] Y.L. Lim, A. Beige, L.C. Kwek, Repeat-until-success linear optics distributed quantum computing, Phys. Rev. Lett. 95 (2005) http://dx.doi.org/10.1103/PhysRevLett.95.030505.

[19] A. Serafini, S. Mancini, S. Bose, Distributed quantum computation via optical fibres, Phys. Rev. Lett. 96 (2006) http://dx.doi.org/10.1103/PhysRevLett.96.010503.

[20] L. Jiang, J.M. Taylor, A.S. Sørensen, M.D. Lukin, Distributed quantum computation based on small quantum registers, Phys. Rev. A - At. Mol. Opt. Phys. 76 (6) (2007) http://dx.doi.org/10.1103/PhysRevA.76.062323.

[21] D.K.L. Oi, S.J. Devitt, L.C.L. Hollenberg, Scalable error correction in distributed ion trap computers, Phys. Rev. A 74 (2006) http://dx.doi.org/10.1103/physreva.74.052313.

[22] M. Gupta, A. Pathak, A scheme for distributed quantum search through simultaneous state transfer mechanism, Ann. Phys., Lpz. 16 (12) (2007) 791–797, http://dx.doi.org/10.1002/andp.200710265.

[23] A. Yimsiriwattana, S.J. Lomonaco Jr., Distributed quantum computing: A distributed shor algorithm, Proc. SPIE 5436 (2004) 360–372, http://dx.doi.org/10.1117/12.546504.

[24] J. Yepez, Type-II quantum computers, Internat. J. Modern Phys. C 12 (9) (2001) 1273–1284, http://dx.doi.org/10.1142/S0129183101002668.

[25] R. Jozsa, N. Linden, On the role of entanglement in quantum-computational speed-up, Proc. R. Soc. A: Math. Phys. Eng. Sci. 459 (2003) 2011–2032, http://dx.doi.org/10.1098/rspa.2002.1097.

[26] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, A.S. Cacciapuoti, Distributed quantum computing: a survey, Comput. Netw. 254 (2024) 110672, http://dx.doi.org/10.1016/j.comnet.2024.110672.

[27] S. Rodrigo, S. Abadal, E. Alarcon, M. Bandic, H.V. Someren, C.G. Almudever, On double full-stack communication-enabled architectures for multicore quantum computers, IEEE Micro 41 (5) (2021) 48–56, http://dx.doi.org/10.1109/MM.2021.3092706.

[28] D. Cuomo, M. Caleffi, A.S. Cacciapuoti, Towards a distributed quantum computing ecosystem, IET Quantum Commun. 1 (1) (2020) 3–8, http://dx.doi.org/10.1049/iet-qtc.2020.0002.

[29] W.K. Wootters, W.H. Zurek, A single quantum cannot be cloned, Nature 299 (5886) (1982) 802–803, http://dx.doi.org/10.1038/299802a0.

[30] R. Horodecki, P. Horodecki, M. Horodecki, K. Horodecki, Quantum entanglement, Rev. Modern Phys. 81 (2009) 865–942, http://dx.doi.org/10.1103/RevModPhys.81.865.

[31] C.C. Gerry, P.L. Knight, Introductory Quantum Optics, Cambridge Universiy Press, 2005, http://dx.doi.org/10.1017/CBO9780511791239.

[32] P. Andres-Martinez, Towards Distributed Quantum Algorithms (Master's thesis), School of Informatics, University of Edinburgh, 2018, URL https://project-archive.inf.ed.ac.uk/msc/20183076/msc_proj.pdf.

[33] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, W.K. Wootters, Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels, Phys. Rev. Lett. 70 (13) (1993) 1895–1899, http://dx.doi.org/10.1103/PhysRevLett.70.1895.

[34] A. Acín, I. Bloch, H. Buhrman, T. Calarco, C. Eichler, J. Eisert, D. Esteve, N. Gisin, S.J. Glaser, F. Jelezko, S. Kuhr, M. Lewenstein, M.F. Riedel, P.O. Schmidt, R. Thew, A. Wallraff, I. Walmsley, F.K. Wilhelm, The quantum technologies roadmap: A European community view, New J. Phys. 20 (2018) http://dx.doi.org/10.1088/1367-2630/aad1ea.

[35] A. Karlsson, M. Bourennane, Quantum teleportation using three-particle entanglement, Phys. Rev. A 58 (1998) 4394–4400, http://dx.doi.org/10.1103/PhysRevA.58.4394.

[36] M. Żukowski, A. Zeilinger, M.A. Horne, A.K. Ekert, "Event-ready-detectors" bell experiment via entanglement swapping, Phys. Rev. Lett. 71 (26) (1993) 4287–4290, http://dx.doi.org/10.1103/PhysRevLett.71.4287.

[37] H.-J. Briegel, W. Dür, J.I. Cirac, P. Zoller, Quantum repeaters: The role of imperfect local operations in quantum communication, Phys. Rev. Lett. 81 (1998) 5932–5935, http://dx.doi.org/10.1103/PhysRevLett.81.5932.

[38] J.-G. Ren, P. Xu, H.-L. Yong, L. Zhang, S.-K. Liao, J. Yin, W.-Y. Liu, W.-Q. Cai, M. Yang, L. Li, K.-X. Yang, X. Han, Y.-Q. Yao, J. Li, H.-Y. Wu, S. Wan, L. Liu, D.-Q. Liu, Y.-W. Kuang, Z.-P. He, P. Shang, C. Guo, R.-H. Zheng, K. Tian, Z.-C. Zhu, N.-L. Liu, C.-Y. Lu, R. Shu, Y.-A. Chen, C.-Z. Peng, J.-Y. Wang, J.-W. Pan, Ground-to-satellite quantum teleportation, Nature 549 (7670) (2017) 70–73, http://dx.doi.org/10.1038/nature23675.

[39] D. Gottesman, I.L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, Nature 402 (6760) (1999) 390–393, http://dx.doi.org/10.1038/46503.

[40] R. Raussendorf, H.J. Briegel, A one-way quantum computer, Phys. Rev. Lett. 86 (2001) 5188–5191, http://dx.doi.org/10.1103/PhysRevLett.86.5188.

[41] K.S. Chou, J.Z. Blumoff, C.S. Wang, P.C. Reinhold, C.J. Axline, Y.Y. Gao, L. Frunzio, M.H. Devoret, L. Jiang, R.J. Schoelkopf, Deterministic teleportation of a quantum gate between two logical qubits, Nature 561 (7723) (2018) 368–373, http://dx.doi.org/10.1038/s41586-018-0470-y.

[42] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, A. Zeilinger, Experimental quantum teleportation, Nature 390 (6660) (1997) 575–579, http://dx.doi.org/10.1038/37539.

[43] A. Furusawa, J.L. Sørensen, S.L. Braunstein, C.A. Fuchs, H.J. Kimble, E.S. Polzik, Unconditional quantum teleportation, Science 282 (5389) (1998) 706–709, http://dx.doi.org/10.1126/science.282.5389.706.

[44] J.F. Sherson, H. Krauter, R.K. Olsson, B. Julsgaard, K. Hammerer, I. Cirac, E.S. Polzik, Quantum teleportation between light and matter, Nature 443 (2006) 557–560, http://dx.doi.org/10.1038/nature05136.

[45] M. Nielsen, E. Knill, R. Laflamme, Complete quantum teleportation using nuclear magnetic resonance, Nature 396 (1998) 52–55, http://dx.doi.org/10.1038/23891.

[46] M. Riebe, H. Häffner, C.F. Roos, W. Hänsel, J. Benheim, G.P. Lancaster, T.W. Körber, C. Becher, F. Schmidt-Kaler, D.F. James, R. Blatt, Deterministic quantum teleportation with atoms, Nature 429 (2004) 734–737, http://dx.doi.org/10.1038/nature02570.

[47] M.D. Barrett, J. Chiaverini, T. Schaetz, J. Britton, W.M. Itano, J.D. Jost, E. Knill, C. Langer, D. Leibfried, R. Ozeri, D.J. Wineland, Deterministic quantum teleportation of atomic qubits, Nature 429 (2004) 737–739, http://dx.doi.org/10.1038/nature02608.

[48] L. Steffen, Y. Salathe, M. Oppliger, P. Kurpiers, M. Baur, C. Lang, C. Eichler, G. Puebla-Hellmann, A. Fedorov, A. Wallraff, Deterministic quantum teleportation with feed-forward in a solid state system, Nature 500 (2013) 319–322, http://dx.doi.org/10.1038/nature12422.

[49] X.L. Wang, X.D. Cai, Z.E. Su, M.C. Chen, D. Wu, L. Li, N.L. Liu, C.Y. Lu, J.W. Pan, Quantum teleportation of multiple degrees of freedom of a single photon, Nature 518 (2015) 516–519, http://dx.doi.org/10.1038/nature14246.

[50] X.M. Hu, C. Zhang, B.H. Liu, Y. Cai, X.J. Ye, Y. Guo, W.B. Xing, C.X. Huang, Y.F. Huang, C.F. Li, G.C. Guo, Experimental high-dimensional quantum teleportation, Phys. Rev. Lett. 125 (2020) http://dx.doi.org/10.1103/PhysRevLett.125.230501.

[51] D. Llewellyn, Y. Ding, I.I. Faruque, S. Paesani, D. Bacco, R. Santagati, Y.J. Qian, Y. Li, Y.F. Xiao, M. Huber, M. Malik, G.F. Sinclair, X. Zhou, K. Rottwitt, J.L. O'Brien, J.G. Rarity, Q. Gong, L.K. Oxenlowe, J. Wang, M.G. Thompson, Chip-to-chip quantum teleportation and multi-photon entanglement in silicon, Nat. Phys. 16 (2020) 148–153, http://dx.doi.org/10.1038/s41567-019-0727-x.

[52] J.C. Hoke, M. Ippoliti, E. Rosenberg, D. Abanin, R. Acharya, T.I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, J.C. Bardin, A. Bengtsson, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, B.B. Buckley, D.A. Buell, T. Burger, B. Burkett, N. Bushnell, Z. Chen, B. Chiaro, D. Chik, J. Cogan, R. Collins, P. Conner, W. Courtney, A.L. Crook, B. Curtin, A.G. Dau, D.M. Debroy, A. Del Toro Barba, S. Demura, A. Di Paolo, I.K. Drozdov, A. Dunsworth, D. Eppens, C. Erickson, E. Farhi, R. Fatemi, V.S. Ferreira, L.F. Burgos, E. Forati, A.G. Fowler, B. Foxen, W. Giang, C. Gidney, D. Gilboa, M. Giustina, R. Gosula, J.A. Gross, S. Habegger, M.C. Hamilton, M. Hansen, M.P. Harrigan, S.D. Harrington, P. Heu, M.R. Hoffmann, S. Hong, T. Huang, A. Huff, W.J. Huggins, S.V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, P. Juhas, D. Kafri, K. Kechedzhi, T. Khattar, M. Khezri, M. Kieferová, S. Kim, A. Kitaev, P.V. Klimov, A.R. Klots, A.N. Korotkov, F. Kostritsa, J.M. Kreikebaum, D. Landhuis, P. Laptev, K.-M. Lau, L. Laws, J. Lee, K.W. Lee, Y.D. Lensky, B.J. Lester, A.T. Lill, W. Liu, A. Locharla, O. Martin, J.R. McClean, M. McEwen, K.C. Miao, A. Mieszala, S. Montazeri, A. Morvan, R. Movassagh, W. Mruczkiewicz, M. Neeley, C. Neill, A. Nersisyan, M. Newman, J.H. Ng, A. Nguyen, M. Nguyen, M.Y. Niu, T.E. O'Brien, S. Omonije, A. Opremcak, A. Petukhov, R. Potter, L.P. Pryadko, C. Quintana, C. Rocque, N.C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K.J. Satzinger, H.F. Schurkus, C. Schuster, M.J. Shearn, A. Shorter, N. Shutty, V. Shvarts, J. Skruzny, W.C. Smith, R. Somma, G. Sterling, D. Strain, M. Szalay, A. Torres, G. Vidal, B. Villalonga, C.V. Heidweiller, T. White, B.W.K. Woo, C. Xing, Z.J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, Y. Zhang, N. Zhu, N. Zobrist, H. Neven, R. Babbush, D. Bacon, S. Boixo, J. Hilton, E. Lucero, A. Megrant, J. Kelly, Y. Chen, V. Smelyanskiy, X. Mi, V. Khemani, P. Roushan, G.Q. AI, Collaborators, Measurement-induced entanglement and teleportation on a noisy quantum processor, Nature 622 (7983) (2023) 481–486, http://dx.doi.org/10.1038/s41586-023-06505-7.

[53] A. Yimsiriwattana, S.J. Lomonaco Jr., Generalized GHZ states and distributed quantum computing, 2004, http://dx.doi.org/10.48550/arXiv.quant-ph/0402148, arXiv.

[54] C.H. Bennett, D.P. DiVincenzo, P.W. Shor, J.A. Smolin, B.M. Terhal, W.K. Wootters, Remote state preparation, Phys. Rev. Lett. 87 (2001) http://dx.doi.org/10.1103/PhysRevLett.87.077902.

[55] H. Weinfurter, Experimental bell-state analysis, Europhys. Lett. 25 (1994) 559–564, http://dx.doi.org/10.1209/0295-5075/25/8/001.

[56] S. Massar, S. Popescu, Optimal extraction of information from finite quantum ensembles, Phys. Rev. Lett. 74 (1995) 1259–1263, http://dx.doi.org/10.1103/PhysRevLett.74.1259.

[57] S. Pirandola, J. Eisert, C. Weedbrook, A. Furusawa, S.L. Braunstein, Advances in quantum teleportation, Nat. Photonics 9 (2015) 641–652, http://dx.doi.org/10.1038/nphoton.2015.154.

[58] X.M. Hu, Y. Guo, B.H. Liu, C.F. Li, G.C. Guo, Progress in quantum teleportation, Nat. Rev. Phys. 5 (2023) 339–353, http://dx.doi.org/10.1038/s42254-023-00588-x.

[59] Q.C. Sun, Y.L. Mao, S.J. Chen, W. Zhang, Y.F. Jiang, Y.B. Zhang, W.J. Zhang, S. Miki, T. Yamashita, H. Terai, X. Jiang, T.Y. Chen, L.X. You, X.F. Chen, Z. Wang, J.Y. Fan, Q. Zhang, J.W. Pan, Quantum teleportation with independent sources and prior entanglement distribution over a network, Nat. Photonics 10 (2016) 671–675, http://dx.doi.org/10.1038/nphoton.2016.179.

[60] R. Valivarthi, M.G. Puigibert, Q. Zhou, G.H. Aguilar, V.B. Verma, F. Marsili, M.D. Shaw, S.W. Nam, D. Oblak, W. Tittel, Quantum teleportation across a metropolitan fibre network, Nat. Photonics 10 (2016) 676–680, http://dx.doi.org/10.1038/nphoton.2016.180.

[61] C.M. Knaut, A. Suleymanzade, Y.-C. Wei, D.R. Assumpcao, P.-J. Stas, Y.Q. Huan, B. Machielse, E.N. Knall, M. Sutula, G. Baranes, N. Sinclair, C. De-Eknamkul, D.S. Levonian, M.K. Bhaskar, H. Park, M. Lončar, M.D. Lukin, Entanglement of nanophotonic quantum memory nodes in a telecommunication network, 2023, http://dx.doi.org/10.48550/arXiv.2310.01316, arXiv eprint.

[62] V. Krutyanskiy, M. Galli, V. Krcmarsky, S. Baier, D.A. Fioretto, Y. Pu, A. Mazloom, P. Sekatski, M. Canteri, M. Teller, J. Schupp, J. Bate, M. Meraner, N. Sangouard, B.P. Lanyon, T.E. Northup, Entanglement of trapped-ion qubits separated by 230 meters, Phys. Rev. Lett. 130 (2023) http://dx.doi.org/10.1103/PhysRevLett.130.050803.

[63] J.-L. Liu, X.-Y. Luo, Y. Yu, C.-Y. Wang, B. Wang, Y. Hu, J. Li, M.-Y. Zheng, B. Yao, Z. Yan, D. Teng, J.-W. Jiang, X.-B. Liu, X.-P. Xie, J. Zhang, Q.-H. Mao, X. Jiang, Q. Zhang, X.-H. Bao, J.-W. Pan, A multinode quantum network over a metropolitan area, 2023, http://dx.doi.org/10.48550/arXiv.2309.00221, arXiv eprint.

[64] S. Daiss, S. Langenfeld, S. Welte, E. Distante, P. Thomas, L. Hartung, O. Morin, G. Rempe, A quantum-logic gate between distant quantum-network modules, Science 371 (6529) (2021) 614–617, http://dx.doi.org/10.1126/science.abe3150.

[65] S. Langenfeld, S. Welte, L. Hartung, S. Daiss, P. Thomas, O. Morin, E. Distante, G. Rempe, Quantum teleportation between remote qubit memories with only a single photon as a resource, Phys. Rev. Lett. 126 (2021) 130502, http://dx.doi.org/10.1103/PhysRevLett.126.130502.

[66] Y. Wan, D. Kienzler, S. Erickson, K. Mayer, T. Tan, J. Wu, H. Vasconcelos, S. Glancy, E. Knill, D. Wineland, A. Wilson, D. Leibfried, Quantum gate teleportation between separated qubits in a trapped-ion processor, Science 364 (2019) 875–878, http://dx.doi.org/10.1126/science.aaw9415.

[67] D. Lago-Rivera, S. Grandi, J. Rakonjac, A. Seri, H. de Riedmatten, Telecom-heralded entanglement between multimode solid-state quantum memories, Nature 594 (2021) 37–40, http://dx.doi.org/10.1038/s41586-021-03481-8.

[68] C.H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J.A. Smolin, W.K. Wootters, Purification of noisy entanglement and faithful teleportation via noisy channels, Phys. Rev. Lett. 76 (1996) 722–725, http://dx.doi.org/10.1103/PhysRevLett.76.722.

[69] D.E. Browne, T. Rudolph, Resource-efficient linear optical quantum computation, Phys. Rev. Lett. 95 (2005) http://dx.doi.org/10.1103/PhysRevLett.95.010501.

[70] S. Bartolucci, P. Birchall, H. Bombín, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant, F. Pastawski, T. Rudolph, C. Sparrow, Fusion-based quantum computation, Nat. Commun. 14 (2023) http://dx.doi.org/10.1038/s41467-023-36493-1.

[71] J.-W. Pan, D. Bouwmeester, H. Weinfurter, A. Zeilinger, Experimental entanglement swapping: Entangling photons that never interacted, Phys. Rev. Lett. 80 (1998) 3892–3894, http://dx.doi.org/10.1103/PhysRevLett.80.3891.

[72] S.L.N. Hermans, M. Pompili, H.K.C. Beukers, S. Baier, J. Borregaard, R. Hanson, Qubit teleportation between non-neighbouring nodes in a quantum network, Nature 605 (7911) (2022) 663–668, http://dx.doi.org/10.1038/s41586-022-04697-y.

[73] Y.F. Huang, X.F. Ren, Y.S. Zhang, L.M. Duan, G.C. Guo, Experimental teleportation of a quantum controlled-NOT gate, Phys. Rev. Lett. 93 (2004) http://dx.doi.org/10.1103/PhysRevLett.93.240501.

[74] J.-W. Pan, D. Bouwmeester, M. Daniel, H. Weinfurter, A. Zeilinger, Experimental test of quantum nonlocality in three-photon Greenberger-Horne-Zeilinger entanglement, Nature 403 (2000) 515–519, http://dx.doi.org/10.1038/35000514.

[75] M. Murao, D. Jonathan, M.B. Plenio, V. Vedral, Quantum telecloning and multiparticle entanglement, Phys. Rev. A 59 (1999) 156–161, http://dx.doi.org/10.1103/PhysRevA.59.156.

[76] Z. Zhao, Y.-A. Chen, A.-N. Zhang, T. Yang, H.J. Briegel, J.-W. Pan, Experimental demonstration of five-photon entanglement and open-destination teleportation, Nature 430 (6995) (2004) 54–58, http://dx.doi.org/10.1038/nature02643.

[77] S.M. Lee, S.W. Lee, H. Jeong, H.S. Park, Quantum teleportation of shared quantum secret, Phys. Rev. Lett. 124 (2020) http://dx.doi.org/10.1103/PhysRevLett.124.060501.

[78] Z. Zhao, A.N. Zhang, X.Q. Zhou, Y.A. Chen, C.Y. Lu, A. Karlsson, J.W. Pan, Experimental realization of optimal asymmetric cloning and telecloning via partial teleportation, Phys. Rev. Lett. 95 (2005) http://dx.doi.org/10.1103/PhysRevLett.95.030502.

[79] L.C. Peng, D. Wu, H.S. Zhong, Y.H. Luo, Y. Li, Y. Hu, X. Jiang, M.C. Chen, L. Li, N.L. Liu, K. Nemoto, W.J. Munro, B.C. Sanders, C.Y. Lu, J.W. Pan, Cloning of quantum entanglement, Phys. Rev. Lett. 125 (2020) http://dx.doi.org/10.1103/PhysRevLett.125.210502.

[80] Q. Wang, Y. Wang, X. Sun, Y. Tian, W. Li, L. Tian, X. Yu, J. Zhang, Y. Zheng, Controllable continuous variable quantum state distributor, Opt. Lett. 46 (2021) 1844, http://dx.doi.org/10.1364/ol.419261.

[81] D. Awschalom, K.K. Berggren, et al., Development of quantum interconnects (QuICs) for next-generation information technologies, PRX Quantum 2 (2021) http://dx.doi.org/10.1103/PRXQuantum.2.017002.

[82] H. Edlbauer, J. Wang, T. Crozes, P. Perrier, S. Ouacel, C. Geffroy, G. Georgiou, E. Chatzikyriakou, A. Lacerda-Santos, X. Waintal, D.C. Glattli, P. Roulleau, J. Nath, M. Kataoka, J. Splettstoesser, M. Acciai, M.C. da Silva Figueira, K. Öztas, A. Trellakis, T. Grange, O.M. Yevtushenko, S. Birner, C. Bäuerle, Semiconductor-based electron flying qubits: Review on recent progress accelerated by numerical modelling, EPJ Quantum Technol. 9 (1) (2022) 21, http://dx.doi.org/10.1140/epjqt/s40507-022-00139-w.

[83] M. Zhong, M.P. Hedges, R.L. Ahlefeldt, J.G. Bartholomew, S.E. Beavan, S.M. Wittig, J.J. Longdell, M.J. Sellars, Optically addressable nuclear spins in a solid with a six-hour coherence time, Nature 517 (7533) (2015) 177–180, http://dx.doi.org/10.1038/nature14025.

[84] P. Hurst, A. Miller, Trends in undersea fiber optic systems, in: OCEANS 2000 MTS/IEEE Conference and Exhibition. Conference Proceedings (Cat. No.00CH37158), Vol. 1, 2000, pp. 479–488, http://dx.doi.org/10.1109/OCEANS.2000.881303, vol.1.

[85] K. Grobe, M. Eiselt, Wavelength Division Multiplexing: A Practical Engineering Guide, in: Wiley Series in Pure and Applied Optics, Wiley, 2013, http://dx.doi.org/10.1002/9781118755068.

[86] R.W. Munn, C.N. Ironside, Principles and Applications of Nonlinear Optical Materials, Springer, 1993, http://dx.doi.org/10.1007/978-94-011-2158-3.

[87] M. Barbieri, C. Cinelli, P. Mataloni, F. De Martini, Polarization-momentum hyperentangled states: Realization and characterization, Phys. Rev. A 72 (2005) 052110, http://dx.doi.org/10.1103/PhysRevA.72.052110.

[88] D. Hucul, I.V. Inlek, G. Vittorini, C. Crocker, S. Debnath, S.M. Clark, C. Monroe, Modular entanglement of atomic qubits using photons and phonons, Nat. Phys. 11 (1) (2015) 37–42, http://dx.doi.org/10.1038/nphys3150.

[89] D.L. Moehring, P. Maunz, S. Olmschenk, K.C. Younge, D.N. Matsukevich, L.-M. Duan, C. Monroe, Entanglement of single-atom quantum bits at a distance, Nature 449 (7158) (2007) 68–71, http://dx.doi.org/10.1038/nature06118.

[90] P. Maunz, D.L. Moehring, S. Olmschenk, K.C. Younge, D.N. Matsukevich, C. Monroe, Quantum interference of photon pairs from two remote trapped atomic ions, Nat. Phys. 3 (8) (2007) 538–541, http://dx.doi.org/10.1038/nphys644.

[91] J. Hofmann, M. Krug, N. Ortegel, L. Gérard, M. Weber, W. Rosenfeld, H. Weinfurter, Heralded entanglement between widely separated atoms, Science 337 (6090) (2012) 72–75, http://dx.doi.org/10.1126/science.1221856.

[92] E. Togan, Y. Chu, A.S. Trifonov, L. Jiang, J. Maze, L. Childress, M.V.G. Dutt, A.S. Sørensen, P.R. Hemmer, A.S. Zibrov, M.D. Lukin, Quantum entanglement between an optical photon and a solid-state spin qubit, Nature 466 (7307) (2010) 730–734, http://dx.doi.org/10.1038/nature09256.

[93] S. Castelletto, A. Edmonds, T. Gaebel, J. Rabeau, Production of multiple diamond-based single-photon sources, IEEE J. Sel. Top. Quantum Electron. 18 (6) (2012) 1792–1798, http://dx.doi.org/10.1109/JSTQE.2012.2199283.

[94] N. Kalb, A.A. Reiserer, P.C. Humphreys, J.J.W. Bakermans, S.J. Kamerling, N.H. Nickerson, S.C. Benjamin, D.J. Twitchen, M. Markham, R. Hanson, Entanglement distillation between solid-state quantum network nodes, Science 356 (6341) (2017) 928–932, http://dx.doi.org/10.1126/science.aan0070.

[95] P.C. Humphreys, N. Kalb, J.P.J. Morits, R.N. Schouten, R.F.L. Vermeulen, D.J. Twitchen, M. Markham, R. Hanson, Deterministic delivery of remote entanglement on a quantum network, Nature 558 (7709) (2018) 268–273, http://dx.doi.org/10.1038/s41586-018-0200-5.

[96] C.T. Nguyen, D.D. Sukachev, M.K. Bhaskar, B. Machielse, D.S. Levonian, E.N. Knall, P. Stroganov, R. Riedinger, H. Park, M. Lončar, M.D. Lukin, Quantum network nodes based on diamond qubits with an efficient nanophotonic interface, Phys. Rev. Lett. 123 (2019) http://dx.doi.org/10.1103/PhysRevLett.123.183602.

[97] V. Krutyanskiy, M. Galli, V. Krcmarsky, S. Baier, D.A. Fioretto, Y. Pu, A. Mazloom, P. Sekatski, M. Canteri, M. Teller, J. Schupp, J. Bate, M. Meraner, N. Sangouard, B.P. Lanyon, T.E. Northup, Entanglement of trapped-ion qubits separated by 230 meters, Phys. Rev. Lett. 130 (2023) 050803, http://dx.doi.org/10.1103/PhysRevLett.130.050803.

[98] A.J. Stolk, K.L. van der Enden, M.-C. Slater, I. te Raa-Derckx, P. Botma, J. van Rantwijk, J.J.B. Biemond, R.A.J. Hagen, R.W. Herfst, W.D. Koek, A.J.H. Meskers, R. Vollmer, E.J. van Zwet, M. Markham, A.M. Edmonds, J.F. Geus, F. Elsen, B. Jungbluth, C. Haefner, C. Tresp, J. Stuhler, S. Ritter, R. Hanson, Metropolitan-scale heralded entanglement of solid-state qubits, Sci. Adv. 10 (44) (2024) eadp6442, http://dx.doi.org/10.1126/sciadv.adp6442.

[99] J. Ramette, J. Sinclair, Z. Vendeiro, A. Rudelis, M. Cetina, V. Vuletić, Any-to-any connected cavity-mediated architecture for quantum computing with trapped ions or rydberg arrays, PRX Quantum 3 (2022) http://dx.doi.org/10.1103/PRXQuantum.3.010344.

[100] Y. Arakawa, M.J. Holmes, Progress in quantum-dot single photon sources for quantum information technologies: A broad spectrum overview, Appl. Phys. Rev. 7 (2) (2020) http://dx.doi.org/10.1063/5.0010193.

[101] S. Castelletto, A. Boretti, Perspective on solid-state single-photon sources in the infrared for quantum technology, Adv. Quantum Technol. 6 (10) (2023) 2300145, http://dx.doi.org/10.1002/qute.202300145.

[102] T. Zhong, P. Goldner, Emerging rare-earth doped material platforms for quantum nanophotonics, Nanophotonics 8 (11) (2019) 2003–2015, http://dx.doi.org/10.1515/nanoph-2019-0185.

[103] N. Samkharadze, G. Zheng, N. Kalhor, D. Brousse, A. Sammak, U.C. Mendes, A. Blais, G. Scappucci, L.M.K. Vandersypen, Strong spin-photon coupling in silicon, Science 359 (6380) (2018) 1123–1127, http://dx.doi.org/10.1126/science.aar4054.

[104] P. Kurpiers, P. Magnard, T. Walter, B. Royer, M. Pechal, J. Heinsoo, Y. Salathé, A. Akin, S. Storz, J.-C. Besse, S. Gasparinetti, A. Blais, A. Wallraff, Deterministic quantum state transfer and remote entanglement using microwave photons, Nature 558 (7709) (2018) 264–267, http://dx.doi.org/10.1038/s41586-018-0195-y.

[105] P. Magnard, S. Storz, P. Kurpiers, J. Schär, F. Marxer, J. Lütolf, T. Walter, J.-C. Besse, M. Gabureac, K. Reuer, A. Akin, B. Royer, A. Blais, A. Wallraff, Microwave quantum link between superconducting circuits housed in spatially separated cryogenic systems, Phys. Rev. Lett. 125 (2020) 260502, http://dx.doi.org/10.1103/PhysRevLett.125.260502.

[106] M. Renger, S. Gandorfer, W. Yam, F. Fesquet, M. Handschuh, K.E. Honasoge, F. Kronowetter, Y. Nojiri, M. Partanen, M. Pfeiffer, H. van der Vliet, A.J. Matthews, J. Govenius, R.N. Jabdaraghi, M. Prunnila, A. Marx, F. Deppe, R. Gross, K.G. Fedorov, Cryogenic microwave link for quantum local area networks, 2023, http://dx.doi.org/10.48550/arXiv.2308.12398, arXiv eprint.

[107] D. Lago-Rivera, J.V. Rakonjac, S. Grandi, H.d. Riedmatten, Long distance multiplexed quantum teleportation from a telecom photon to a solid-state qubit, Nat. Commun. 14 (1) (2023) 1889, http://dx.doi.org/10.1038/s41467-023-37518-5.

[108] P. Zhao, M.-Y. Yang, S. Zhu, L. Zhou, W. Zhong, M.-M. Du, Y.-B. Sheng, Generation of hyperentangled state encoded in three degrees of freedom, Sci. China Phys. Mech. Astron. 66 (10) (2023) http://dx.doi.org/10.1007/s11433-023-2164-7.

[109] W. Li, L. Zhang, H. Tan, Y. Lu, S.-K. Liao, J. Huang, H. Li, Z. Wang, H.-K. Mao, B. Yan, Q. Li, Y. Liu, Q. Zhang, C.-Z. Peng, L. You, F. Xu, J.-W. Pan, High-rate quantum key distribution exceeding 110 Mb s–1, Nat. Photonics 17 (5) (2023) 416–421, http://dx.doi.org/10.1038/s41566-023-01166-4.

[110] J. Zhang, E. Zallo, B. Höfer, Y. Chen, R. Keil, M. Zopf, S. Böttner, F. Ding, O.G. Schmidt, Electric-field-induced energy tuning of on-demand entangled-photon emission from self-assembled quantum dots, Nano Lett. 17 (1) (2017) 501–507, http://dx.doi.org/10.1021/acs.nanolett.6b04539.

[111] W. Ou, X. Wang, W. Wei, T. Jin, Y. Zhu, T. Wang, J. Zhang, X. Ou, J. Zhang, Strain tuning self-assembled quantum dots for energy-tunable entangled-photon sources using a photolithographically fabricated microelectromechanical system, ACS Photonics 9 (10) (2022) 3421–3428, http://dx.doi.org/10.1021/acsphotonics.2c01033.

[112] C. Hopfmann, W. Nie, N.L. Sharma, C. Weigelt, F. Ding, O.G. Schmidt, Maximally entangled and gigahertz-clocked on-demand photon pair source, Phys. Rev. B 103 (2021) 075413, http://dx.doi.org/10.1103/PhysRevB.103.075413.

[113] P. Aumann, M. Prilmüller, F. Kappe, L. Ostermann, D. Dalacu, P.J. Poole, H. Ritsch, W. Lechner, G. Weihs, Demonstration and modeling of time-bin entangled photons from a quantum dot in a nanowire, AIP Adv. 12 (5) (2022) 055115, http://dx.doi.org/10.1063/5.0081874.

[114] L. Zhou, Y.-B. Sheng, Purification of logic-qubit entanglement, Sci. Rep. 6 (1) (2016) http://dx.doi.org/10.1038/srep28813.

[115] F. Kaiser, P. Vergyris, A. Martin, D. Aktas, M.P.D. Micheli, O. Alibart, S. Tanzilli, Quantum optical frequency up-conversion for polarisation entangled qubits: Towards interconnected quantum information devices, Opt. Express 27 (18) (2019) 25603–25610, http://dx.doi.org/10.1364/OE.27.025603.

[116] S. Murakami, R. Fujimoto, T. Kobayashi, R. Ikuta, A. Inoue, T. Umeki, S. Miki, F. China, H. Terai, R. Kasahara, T. Mukai, N. Imoto, T. Yamamoto, Quantum frequency conversion using 4-port fiber-pigtailed PPLN module, Opt. Express 31 (18) (2023) 29271–29279, http://dx.doi.org/10.1364/OE.494313.

[117] M.J. Weaver, P. Duivestein, A.C. Bernasconi, S. Scharmer, M. Lemang, T.C.v. Thiel, F. Hijazi, B. Hensen, S. Gröblacher, R. Stockill, An integrated microwave-to-optics interface for scalable quantum computing, Nature Nanotechnology 19 (2) (2023) 166–172, http://dx.doi.org/10.1038/s41565-023-01515-y.

[118] R. Sahu, L. Qiu, W. Hease, G. Arnold, Y. Minoguchi, P. Rabl, J.M. Fink, Entangling microwaves with light, Science 380 (6646) (2023) 718–721, http://dx.doi.org/10.1126/science.adg3812.

[119] T.B. Pittman, B.C. Jacobs, J.D. Franson, Single photons on pseudodemand from stored parametric down-conversion, Phys. Rev. A 66 (2002) 042303, http://dx.doi.org/10.1103/PhysRevA.66.042303.

[120] O. Landry, J.A.W. van Houwelingen, A. Beveratos, H. Zbinden, N. Gisin, Quantum teleportation over the Swisscom telecommunication network, J. Opt. Soc. Am. B 24 (2) (2007) 398–403, http://dx.doi.org/10.1364/JOSAB.24.000398.

[121] P.M. Leung, T.C. Ralph, Quantum memory scheme based on optical fibers and cavities, Phys. Rev. A 74 (2006) http://dx.doi.org/10.1103/PhysRevA.74.022311.

[122] T. Tanabe, M. Notomi, E. Kuramochi, A. Shinya, H. Taniyama, Trapping and delaying photons for one nanosecond in an ultrasmall high-Q photonic-crystal nanocavity, Nat. Photonics 1 (1) (2007) 49–52, http://dx.doi.org/10.1038/nphoton.2006.51.

[123] S.A. Moiseev, B.S. Ham, Photon-echo quantum memory with efficient multipulse readings, Phys. Rev. A 70 (2004) http://dx.doi.org/10.1103/PhysRevA.70.063809.

[124] A.I. Lvovsky, B.C. Sanders, W. Tittel, Optical quantum memory, Nat. Photonics 3 (12) (2009) 706–714, http://dx.doi.org/10.1038/nphoton.2009.231.

[125] M. Guo, S. Liu, W. Sun, M. Ren, F. Wang, M. Zhong, Rare-earth quantum memories: The experimental status quo, Front. Phys. 18 (2) (2023) 21303, http://dx.doi.org/10.1007/s11467-022-1240-8.

[126] A.V. Gorshkov, A. André, M. Fleischhauer, A.S. Sørensen, M.D. Lukin, Universal approach to optimal photon storage in atomic media, Phys. Rev. Lett. 98 (2007) 123601, http://dx.doi.org/10.1103/PhysRevLett.98.123601.

[127] W. Tittel, M. Afzelius, T. Chaneliére, R. Cone, S. Kröll, S. Moiseev, M. Sellars, Photon-echo quantum memory in solid state systems, Laser Photonics Rev. 4 (2) (2010) 244–267, http://dx.doi.org/10.1002/lpor.200810056.

[128] M. Afzelius, C. Simon, H. de Riedmatten, N. Gisin, Multimode quantum memory based on atomic frequency combs, Phys. Rev. A 79 (2009) http://dx.doi.org/10.1103/PhysRevA.79.052329.

[129] A. Ortu, A. Holzäpfel, J. Etesse, M. Afzelius, Storage of photonic time-bin qubits for up to 20 ms in a rare-earth doped crystal, Npj Quantum Inf. 8 (1) (2022) 29, http://dx.doi.org/10.1038/s41534-022-00541-3.

[130] E. Knill, R. Laflamme, G.J. Milburn, A scheme for efficient quantum computation with linear optics, Nature 409 (6816) (2001) 46–52, http://dx.doi.org/10.1038/35051009.

[131] P. Kok, W.J. Munro, K. Nemoto, T.C. Ralph, J.P. Dowling, G.J. Milburn, Linear optical quantum computing with photonic qubits, Rev. Modern Phys. 79 (2007) 135–174, http://dx.doi.org/10.1103/RevModPhys.79.135.

[132] E. Knill, R. Laflamme, W.H. Zurek, Resilient quantum computation, Science 279 (5349) (1998) 342–345, http://dx.doi.org/10.1126/science.279.5349.342.

[133] A.Y. Kitaev, Fault-tolerant quantum computation by anyons, Ann. Physics 303 (1) (2003) 2–30, http://dx.doi.org/10.1016/S0003-4916(02)00018-0.

[134] T.P. Harty, D.T.C. Allcock, C.J. Ballance, L. Guidoni, H.A. Janacek, N.M. Linke, D.N. Stacey, D.M. Lucas, High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit, Phys. Rev. Lett. 113 (2014) http://dx.doi.org/10.1103/PhysRevLett.113.220501.

[135] N. Schlosser, G. Reymond, I. Protsenko, P. Grangier, Sub-poissonian loading of single atoms in a microscopic dipole trap, Nature 411 (6841) (2001) 1024–1027, http://dx.doi.org/10.1038/35082512.

[136] M.A. Norcia, A.W. Young, A.M. Kaufman, Microscopic control and detection of ultracold strontium in optical-tweezer arrays, Phys. Rev. X 8 (2018) 041054, http://dx.doi.org/10.1103/PhysRevX.8.041054.

[137] K. Barnes, P. Battaglino, B.J. Bloom, K. Cassella, R. Coxe, N. Crisosto, J.P. King, S.S. Kondov, K. Kotru, S.C. Larsen, J. Lauigan, B.J. Lester, M. McDonald, E. Megidish, S. Narayanaswami, C. Nishiguchi, R. Notermans, L.S. Peng, A. Ryou, T.-Y. Wu, M. Yarwood, Assembly and coherent control of a register of nuclear spin qubits, Nat. Commun. 13 (1) (2022) 2779, http://dx.doi.org/10.1038/s41467-022-29977-z.

[138] L. Isenhower, E. Urban, X.L. Zhang, A.T. Gill, T. Henage, T.A. Johnson, T.G. Walker, M. Saffman, Demonstration of a neutral atom controlled-NOT quantum gate, Phys. Rev. Lett. 104 (2010) http://dx.doi.org/10.1103/PhysRevLett.104.010503.

[139] M.W. Doherty, N.B. Manson, P. Delaney, F. Jelezko, J. Wrachtrup, L.C. Hollenberg, The nitrogen-vacancy colour centre in diamond, Phys. Rep. 528 (1) (2013) 1–45, http://dx.doi.org/10.1016/j.physrep.2013.02.001.

[140] Y. Wang, M. Um, J. Zhang, S. An, M. Lyu, J.-N. Zhang, L.-M. Duan, D. Yum, K. Kim, Single-qubit quantum memory exceeding ten-minute coherence time, Nat. Photonics 11 (10) (2017) 646–650, http://dx.doi.org/10.1038/s41566-017-0007-1.

[141] ITU-T, Transmission Media and Optical Systems Characteristics – Optical Fibre Cables, Standard, Telecomm. Standardization Sector of ITU, 2016.

[142] Y. Tamura, H. Sakuma, K. Morita, M. Suzuki, Y. Yamamoto, K. Shimada, Y. Honma, K. Sohma, T. Fujii, T. Hasegawa, The first 0.14-dB/km loss optical fiber and its impact on submarine transmission, J. Lightwave Technol. 36 (1) (2018) 44–49, http://dx.doi.org/10.1109/JLT.2018.2796647.

[143] E. Sugita, R. Nagase, K. Kanayama, T. Shintaku, SC-type single-mode optical fiber connectors, J. Lightwave Technol. 7 (11) (1989) 1689–1696, http://dx.doi.org/10.1109/50.45890.

[144] Y. Yamamoto, The quantum optical repeater, Science 263 (5152) (1994) 1394–1395, http://dx.doi.org/10.1126/science.263.5152.1394.

[145] M. Takeoka, S. Guha, M.M. Wilde, Fundamental rate-loss tradeoff for optical quantum key distribution, Nat. Commun. 5 (1) (2014) 5235, http://dx.doi.org/10.1038/ncomms6235.

[146] J. Miguel-Ramiro, W. Dür, Efficient entanglement purification protocols for d-level systems, Phys. Rev. A 98 (2018) http://dx.doi.org/10.1103/PhysRevA.98.042309.

[147] X.-M. Hu, C.-X. Huang, Y.-B. Sheng, L. Zhou, B.-H. Liu, Y. Guo, C. Zhang, W.-B. Xing, Y.-F. Huang, C.-F. Li, G.-C. Guo, Long-distance entanglement purification for quantum communication, Phys. Rev. Lett. 126 (2021) http://dx.doi.org/10.1103/PhysRevLett.126.010503.

[148] F.-F. Du, G. Fan, Y.-M. Wu, B.-C. Ren, Faithful and efficient hyperentanglement purification for spatial-polarization-time-bin photon system, Chin. Phys. B 32 (6) (2023) http://dx.doi.org/10.1088/1674-1056/aca395.

[149] C.-C. Luo, L. Zhou, W. Zhong, Y.-B. Sheng, Purification for hybrid logical qubit entanglement, Quantum Inf. Process. 21 (8) (2022) 300, http://dx.doi.org/10.1007/s11128-022-03646-y.

[150] T. van Leent, M. Bock, F. Fertig, R. Garthoff, S. Eppelt, Y. Zhou, P. Malik, M. Seubert, T. Bauer, W. Rosenfeld, W. Zhang, C. Becher, H. Weinfurter, Entangling single atoms over 33 km telecom fibre, Nature 607 (7917) (2022) 69–73, http://dx.doi.org/10.1038/s41586-022-04764-4.

[151] S. Ourari, Ł. Dusanowski, S.P. Horvath, M.T. Uysal, C.M. Phenicie, P. Stevenson, M. Raha, S. Chen, R.J. Cava, N.P. de Leon, J.D. Thompson, Indistinguishable telecom band photons from a single Er ion in the solid state, Nature 620 (7976) (2023) 977–981, http://dx.doi.org/10.1038/s41586-023-06281-4.

[152] J. Laurat, On-demand entanglement could lead to scalable quantum networks, Nature 558 (7709) (2018) 192–193, http://dx.doi.org/10.1038/d41586-018-05336-1.

[153] Q. Zhu, Y. Zhao, H. Xu, L. Huang, C. Qiao, Integrating all-optical switching and entangled photon source placement for entanglement routing, in: IEEE/ACM 31st Int. Symp. on Quality of Service, IWQoS, 2023, pp. 1–10, http://dx.doi.org/10.1109/IWQoS57198.2023.10188776.

[154] A.S. Cacciapuoti, M. Viscardi, J. Illiano, M. Caleffi, Entanglement distribution in the quantum internet: Knowing when to stop!, 2023, http://dx.doi.org/10.48550/arXiv.2307.05123, arxiv eprint.

[155] Y. Zeng, J. Zhang, J. Liu, Z. Liu, Y. Yang, Entanglement routing design over quantum networks, IEEE/ACM Trans. Netw. 32 (1) (2023) 352–367, http://dx.doi.org/10.1109/TNET.2023.3282560.

[156] L. Gyongyosi, S. Imre, Advances in the quantum internet, Commun. ACM 65 (8) (2022) 52–63, http://dx.doi.org/10.1145/3524455.

[157] H.J. Kimble, The quantum internet, Nature 453 (7198) (2008) 1023–1030, http://dx.doi.org/10.1038/nature07127.

[158] S. Wehner, D. Elkouss, R. Hanson, Quantum internet: A vision for the road ahead, Science 362 (6412) (2018) eaam9288, http://dx.doi.org/10.1126/science.aam9288.

[159] A.S. Cacciapuoti, M. Caleffi, F. Tafuri, F.S. Cataliotti, S. Gherardini, G. Bianchi, Quantum internet: Networking challenges in distributed quantum computing, IEEE Netw. 34 (1) (2020) 137–143, http://dx.doi.org/10.1109/MNET.001.1900092.

[160] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, L. Jiang, P. Basu, D. Englund, S. Guha, Routing entanglement in the quantum internet, Npj Quantum Inf. 5 (1) (2019) 25, http://dx.doi.org/10.1038/s41534-019-0139-x.

[161] A. Reiserer, G. Rempe, Cavity-based quantum networks with single atoms and optical photons, Rev. Modern Phys. 87 (2015) 1379–1418, http://dx.doi.org/10.1103/RevModPhys.87.1379.

[162] Z. Li, K. Xue, J. Li, L. Chen, R. Li, Z. Wang, N. Yu, D.S. Wei, Q. Sun, J. Lu, Entanglement-assisted quantum networks: Mechanics, enabling technologies, challenges, and research directions, IEEE Commun. Surv. Tutor. 25 (4) (2023) 2133–2189, http://dx.doi.org/10.1109/COMST.2023.3294240.

[163] J. Miguel-Ramiro, A. Pirker, W. Dür, Genuine quantum networks with superposed tasks and addressing, Npj Quantum Inf. 7 (2021) http://dx.doi.org/10.1038/s41534-021-00472-5.

[164] J. Illiano, M. Caleffi, A. Manzalini, A.S. Cacciapuoti, Quantum internet protocol stack: A comprehensive survey, Comput. Netw. 213 (2022) 109092, http://dx.doi.org/10.1016/j.comnet.2022.109092.

[165] W. Kozlowski, S. Wehner, R.V. Meter, B. Rijsman, A.S. Cacciapuoti, M. Caleffi, S. Nagayama, RFC 9340: Architectural principles for a quantum internet, 2023, http://dx.doi.org/10.17487/RFC9340.

[166] M. Azari, P. Polakos, K.P. Seshadreesan, Quantum switches for Gottesman–Kitaev–Preskill qubit-based all-photonic quantum networks, 2024, http://dx.doi.org/10.48550/arXiv.2402.02721, arxiv preprint.

[167] I. Tillman, T. Vasantam, K.P. Seshadreesan, A continuous variable quantum switch, in: Proc. of Int. Conf. on Quantum Computing and Engineering, QCE, IEEE, 2022, pp. 365–371, http://dx.doi.org/10.1109/QCE53715.2022.00057.

[168] T.C. Ralph, A.P. Lund, Nondeterministic Noiseless Linear Amplification of Quantum Systems, Vol. 1110, 2009, pp. 155–160, http://dx.doi.org/10.1063/1.3131295.

[169] A. Pirker, W. Dür, A quantum network stack and protocols for reliable entanglement-based networks, New J. Phys. 21 (3) (2019) http://dx.doi.org/10.1088/1367-2630/ab05f7.

[170] R.V. Meter, J. Touch, Designing quantum repeater networks, IEEE Commun. Mag. 51 (8) (2013) 64–71, http://dx.doi.org/10.1109/MCOM.2013.6576340.

[171] R. Van Meter, R. Satoh, N. Benchasattabuse, K. Teramoto, T. Matsuo, M. Hajdušek, T. Satoh, S. Nagayama, S. Suzuki, A quantum internet architecture, in: 2022 IEEE Int. Conf. on Quantum Computing and Engineering, QCE, 2022, pp. 341–352, http://dx.doi.org/10.1109/QCE53715.2022.00055.

[172] Z. Li, K. Xue, J. Li, N. Yu, J. Liu, D.S.L. Wei, Q. Sun, J. Lu, Building a large-scale and wide-area quantum internet based on an OSI-alike model, China Commun. 18 (10) (2021) 1–14, http://dx.doi.org/10.23919/JCC.2021.10.001.

[173] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpundefineddek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, S. Wehner, A link layer protocol for quantum networks, in: Proceedings of the ACM Special Interest Group on Data Communication, ACM, New York, NY, USA, 2019, pp. 159–173, http://dx.doi.org/10.1145/3341302.3342070.

[174] A.S. Cacciapuoti, J. Illiano, M. Caleffi, Quantum internet addressing, IEEE Netw. 38 (1) (2024) 104–111, http://dx.doi.org/10.1109/MNET.2023.3328393.

[175] J. Illiano, M. Caleffi, M. Viscardi, A.S. Cacciapuoti, Quantum MAC: Genuine entanglement access control via many-body dicke states, IEEE Trans. Commun. 72 (4) (2024) 2090–2105, http://dx.doi.org/10.1109/TCOMM.2023.3344140.

[176] M. Caleffi, Optimal routing for quantum networks, IEEE Access 5 (2017) 22299–22312, http://dx.doi.org/10.1109/ACCESS.2017.2763325.

[177] N. Yu, C.-Y. Lai, L. Zhou, Protocols for packet quantum network intercommunication, IEEE Trans. Quantum Eng. 2 (2021) 1–9, http://dx.doi.org/10.1109/TQE.2021.3112594.

[178] Defense Advanced Research Projects Agency, Transmission control protocol, 1981, URL https://www.ietf.org/rfc/rfc793.txt. (Accessed 21 January 2025).

[179] R. Cleve, D. Gottesman, H.-K. Lo, How to share a quantum secret, Phys. Rev. Lett. 83 (1999) 648–651, http://dx.doi.org/10.1103/PhysRevLett.83.648.

[180] C.D. Donne, M. Iuliano, B. van der Vecht, G.M. Ferreira, H. Jirovská, T. van der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, P. Filippov, A.R.-P. Montblanch, J. Fischer, B. van Ommen, N. Demetriou, D. Leichtle, L. Music, H. Ollivier, I. te Raa, W. Kozlowski, T. Taminiau, P. Pawełczak, T. Northup, R. Hanson, S. Wehner, Design and demonstration of an operating system for executing applications on quantum network nodes, 2024, http://dx.doi.org/10.48550/arXiv.2407.18306.

[181] T. Satoh, R.V. Meter, Quantum sockets for the NISQ era quantum internet, 2018, URL https://www.ngc.is.ritsumei.ac.jp/~ger/static/AQIS18/OnlineBooklet/158.pdf. (Accessed 21 January 2025).

[182] S. Gauthier, G. Vardoyan, S. Wehner, A control architecture for entanglement generation switches in quantum networks, in: Proc. of the 1st Workshop on Quantum Networks and Distributed Quantum Computing, QuNet, ACM, 2023, pp. 38–44, http://dx.doi.org/10.1145/3610251.3610552.

[183] Z. Xiao, J. Li, K. Xue, Z. Li, N. Yu, Q. Sun, J. Lu, A connectionless entanglement distribution protocol design in quantum networks, IEEE Netw. 38 (1) (2024) 131–139, http://dx.doi.org/10.1109/MNET.2023.3321044.

[184] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, W. Zwaenepoel, TreadMarks: Shared memory computing on networks of workstations, Computer 29 (1996) 18–28, http://dx.doi.org/10.1109/2.485843.

[185] M. Van Steen, A.S. Tanenbaum, Distributed Systems, Maarten van Steen Leiden, The Netherlands, 2017, URL https://www.distributed-systems.net/index.php/books/ds3/.

[186] C. Berge, Hypergraphs: Combinatorics of Finite Sets, in: North-Holland Mathematical Library, Elsevier Science, 1984, URL https://shop.elsevier.com/books/hypergraphs/berge/978-0-444-87489-4.

[187] M. Zomorodi-Moghadam, M. Houshmand, M. Houshmand, Optimizing teleportation cost in distributed quantum circuits, Int. J. Theor. Phys. 57 (3) (2018) 848–861, http://dx.doi.org/10.1007/S10773-017-3618-X.

[188] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (2) (1970) 291–307, http://dx.doi.org/10.1002/j.1538-7305.1970.tb01770.x.

[189] P. Andrés-Martínez, C. Heunen, Automated distribution of quantum circuits via hypergraph partitioning, Phys. Rev. A 100 (3) (2019) 032308, http://dx.doi.org/10.1103/PhysRevA.100.032308.

[190] Y. Akhremtsev, T. Heuer, P. Sanders, S. Schlag, Engineering a direct k-way hypergraph partitioning algorithm, in: Proc. of ALENEX, SIAM, 2017, pp. 28–42, http://dx.doi.org/10.1137/1.9781611974768.3.

[191] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, P. Sanders, High-quality hypergraph partitioning, ACM J. Exp. Algorithmics 27 (2023) 1–39, http://dx.doi.org/10.1145/3529090.

[192] M. Houshmand, Z. Mohammadi, M. Zomorodi-Moghadam, M. Houshmand, An evolutionary approach to optimizing teleportation cost in distributed quantum computation, Int. J. Theor. Phys. 59 (4) (2020) 1315–1329, http://dx.doi.org/10.1007/S10773-020-04409-0.

[193] O. Daei, K. Navi, M. Zomorodi-Moghadam, Optimized quantum circuit partitioning, Internat. J. Theoret. Phys. 59 (12) (2020) 3804–3820, http://dx.doi.org/10.1007/s10773-020-04633-8.

[194] E. Nikahd, N. Mohammadzadeh, M. Sedighi, M.S. Zamani, Automated window-based partitioning of quantum circuits, Phys. Scr. 96 (3) (2021) 035102, http://dx.doi.org/10.1088/1402-4896/ABD57C.

[195] J.-Y. Wu, K. Matsui, T. Forrer, A. Soeda, P. Andrés-Martínez, D. Mills, L. Henaut, M. Murao, Entanglement-efficient bipartite-distributed quantum computing, Quantum 7 (2023) 1196, http://dx.doi.org/10.22331/q-2023-12-05-1196.

[196] P. Andrés-Martínez, T. Forrer, D. Mills, J.-Y. Wu, L. Henaut, K. Yamamoto, M. Murao, R. Duncan, Distributing circuits over heterogeneous, modular quantum computing network architectures, 2023, http://dx.doi.org/10.48550/arXiv.2305.14148, arxiv preprint.

[197] R. G. Sundaram, H. Gupta, C.R. Ramakrishnan, Efficient distribution of quantum circuits, in: 35th Int. Symp. on Distributed Computing, DISC, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 209, 2021, pp. 1–20, http://dx.doi.org/10.4230/LIPIcs.DISC.2021.41.

[198] R.G. Sundaram, H. Gupta, C.R. Ramakrishnan, Distribution of quantum circuits over general quantum networks, in: 2022 IEEE Int. Conf. on Quantum Computing and Engineering, QCE, 2022, pp. 415–425, http://dx.doi.org/10.1109/QCE53715.2022.00063.

[199] R.G. Sundaram, H. Gupta, Distributing quantum circuits using teleportations, in: 2023 IEEE Int. Conf. on Quantum Software, QSW, 2023, pp. 186–192, http://dx.doi.org/10.1109/QSW59989.2023.00030.

[200] Z. Davarzani, M. Zomorodi-Moghadam, M. Houshmand, M. Nouri-Baygi, A dynamic programming approach for distributing quantum circuits by bipartite graphs, Quantum Inf. Process. 19 (2020) http://dx.doi.org/10.1007/s11128-020-02871-7.

[201] J. Clark, T. Humble, H. Thapliyal, TDAG: Tree-based directed acyclic graph partitioning for quantum circuits, in: Proc. of the Great Lakes Symposium on VLSI, ACM, 2023, pp. 587–592, http://dx.doi.org/10.1145/3583781.3590234.

[202] W. Cambiucci, R. Silveira, W. Ruggiero, Hypergraphic partitioning of quantum circuits for distributed quantum computing, in: 2023 IEEE Int. Conf. on Quantum Computing and Engineering, QCE, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 268–269, http://dx.doi.org/10.1109/QCE57702.2023.10237.

[203] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Papers on Twenty-Five Years of Electronic Design Automation, 1982, pp. 175–181, http://dx.doi.org/10.1109/DAC.1982.1585498.

[204] J.M. Baker, C. Duckering, A. Hoover, F.T. Chong, Time-sliced quantum circuit partitioning for modular architectures, in: Proc. of the 17th ACM Int. Conf. on Computing Frontiers, ACM, 2020, pp. 98–107, http://dx.doi.org/10.1145/3387902.3392617.

[205] T. Park, C.Y. Lee, Algorithms for partitioning a graph, Comput. Ind. Eng. 28 (4) (1995) 899–909, http://dx.doi.org/10.1016/0360-8352(95)00003-J.

[206] A. Ovide, S. Rodrigo, M. Bandic, H. Van Someren, S. Feld, S. Abadal, E. Alarcon, C.G. Almudever, Mapping quantum algorithms to multi-core quantum computing architectures, in: IEEE Int. Symp. on Circuits and Systems, ISCAS, IEEE, 2023, pp. 1–5, http://dx.doi.org/10.1109/ISCAS46773.2023.10181589.

[207] P. Escofet, A. Ovide, C.G. Almudever, E. Alarcón, S. Abadal, Hungarian qubit assignment for optimized mapping of quantum circuits on multi-core architectures, IEEE Comput. Archit. Lett. (2023) http://dx.doi.org/10.1109/LCA.2023.3318857.

[208] M. Bandic, L. Prielinger, J. Nüßlein, A. Ovide, S. Rodrigo, S. Abadal, H. van Someren, G. Vardoyan, E. Alarcon, C.G. Almudever, et al., Mapping quantum circuits to modular architectures with QUBO, in: 2023 IEEE Int. Conf. on Quantum Computing and Engineering, QCE, Vol. 1, IEEE, 2023, pp. 790–801, http://dx.doi.org/10.1109/QCE57702.2023.00094.

[209] A. Pastor, P. Escofet, S.B. Rached, E. Alarcón, P. Barlet-Ros, S. Abadal, Circuit partitioning for multi-core quantum architectures with deep reinforcement learning, 2024, http://dx.doi.org/10.48550/arXiv.2401.17976, arxiv preprint.

[210] H. Pashayan, J.J. Wallman, S.D. Bartlett, Estimating outcome probabilities of quantum circuits using quasiprobabilities, Phys. Rev. Lett. 115 (7) (2015) http://dx.doi.org/10.1103/PhysRevLett.115.070501.

[211] K. Mitarai, K. Fujii, Overhead for simulating a non-local channel with local channels by quasiprobability sampling, Quantum 5 (2021) 388, http://dx.doi.org/10.22331/q-2021-01-28-388.

[212] C. Piveteau, D. Sutter, S. Woerner, Quasiprobability decompositions with reduced sampling overhead, Npj Quantum Inf. 8 (1) (2022) 1–9, http://dx.doi.org/10.1038/s41534-022-00517-3.

[213] H.F. Hofmann, How to simulate a universal quantum computer using negative probabilities, J. Phys. A 42 (27) (2009) 275304, http://dx.doi.org/10.1088/1751-8113/42/27/275304.

[214] S. Bravyi, G. Smith, J.A. Smolin, Trading classical and quantum computational resources, Phys. Rev. X 6 (2) (2016) http://dx.doi.org/10.1103/PhysRevX.6.021043.

[215] K. Temme, S. Bravyi, J.M. Gambetta, Error mitigation for short-depth quantum circuits, Phys. Rev. Lett. 119 (18) (2017) http://dx.doi.org/10.1103/PhysRevLett.119.180509.

[216] S. Endo, S.C. Benjamin, Y. Li, Practical quantum error mitigation for near-future applications, Phys. Rev. X 8 (3) (2018) http://dx.doi.org/10.1103/PhysRevX.8.031027.

[217] C. Ying, B. Cheng, Y. Zhao, H.-L. Huang, Y.-N. Zhang, M. Gong, Y. Wu, S. Wang, F. Liang, J. Lin, Y. Xu, H. Deng, H. Rong, C.-Z. Peng, M.-H. Yung, X. Zhu, J.-W. Pan, Experimental simulation of larger quantum circuits with fewer superconducting qubits, Phys. Rev. Lett. 130 (11) (2023) 110601, http://dx.doi.org/10.1103/PhysRevLett.130.110601.

[218] A.P. Singh, K. Mitarai, Y. Suzuki, K. Heya, Y. Tabuchi, K. Fujii, Y. Nakamura, Experimental demonstration of a high-fidelity virtual two-qubit gate, Phys. Rev. Res. 6 (2024) http://dx.doi.org/10.1103/PhysRevResearch.6.013235.

[219] J.A. Smolin, J.M. Gambetta, G. Smith, Efficient method for computing the maximum-likelihood quantum state from measurements with additive Gaussian noise, Phys. Rev. Lett. 108 (7) (2012) 070502, http://dx.doi.org/10.1103/PhysRevLett.108.070502.

[220] M.A. Perlin, Z.H. Saleem, M. Suchara, J.C. Osborn, Quantum circuit cutting with maximum-likelihood tomography, Npj Quantum Inf. 7 (64) (2021) 1–8, http://dx.doi.org/10.1038/s41534-021-00390-6.

[221] A. Eddins, M. Motta, T.P. Gujarati, S. Bravyi, A. Mezzacapo, C. Hadfield, S. Sheldon, Doubling the size of quantum simulators by entanglement forging, PRX Quantum 3 (1) (2022) 010309, http://dx.doi.org/10.1103/PRXQuantum.3.010309.

[222] D. Faílde, J. Santos-Suárez, D.A. Herrera-Martí, J. Mas, Hamiltonian forging of a thermofield double, 2023, http://dx.doi.org/10.48550/arXiv.2311.10566, arxiv preprint.

[223] K. Mitarai, K. Fujii, Constructing a virtual two-qubit gate by sampling single-qubit operations, New J. Phys. 23 (2) (2021) 023021, http://dx.doi.org/10.1088/1367-2630/abd7bc.

[224] T. Peng, A.W. Harrow, M. Ozols, X. Wu, Simulating large quantum circuits on a small quantum computer, Phys. Rev. Lett. 125 (15) (2020) http://dx.doi.org/10.1103/PhysRevLett.125.150504.

[225] L. Brenner, C. Piveteau, D. Sutter, Optimal wire cutting with classical communication, 2023, http://dx.doi.org/10.48550/arXiv.2302.03366, arxiv preprint.

[226] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P.J. Love, A. Aspuru-Guzik, J.L. O'Brien, A variational eigenvalue solver on a photonic quantum processor, Nat. Commun. 5 (1) (2014) 4213, http://dx.doi.org/10.1038/ncomms5213.

[227] S.C. Marshall, J. Tura, V. Dunjko, All this for one qubit? Bounds on local circuit cutting schemes, 2023, http://dx.doi.org/10.48550/arXiv.2303.13422, arxiv preprint, arXiv:2303.13422.

[228] C. Piveteau, D. Sutter, Circuit knitting with classical communication, IEEE Trans. Inform. Theory (2023) http://dx.doi.org/10.1109/TIT.2023.3310797, 1–1.

[229] A. Carrera Vazquez, C. Tornow, D. Ristè, S. Woerner, M. Takita, D.J. Egger, Combining quantum processors with real-time classical communication, Nature (2024) 1–5, http://dx.doi.org/10.1038/s41586-024-08178-2.

[230] A. Lowe, M. Medvidović, A. Hayes, L.J. O'Riordan, T.R. Bromley, J.M. Arrazola, N. Killoran, Fast quantum circuit cutting with randomized measurements, Quantum 7 (2023) 934, http://dx.doi.org/10.22331/q-2023-03-02-934.

[231] H.-Y. Huang, R. Kueng, J. Preskill, Predicting many properties of a quantum system from very few measurements, Nat. Phys. 16 (2020) 1050–1057, http://dx.doi.org/10.1038/s41567-020-0932-7.

[232] H. Harada, K. Wada, N. Yamamoto, Doubly optimal parallel wire cutting without ancilla qubits, PRX Quantum 5 (4) (2024) 040308, http://dx.doi.org/10.1103/PRXQuantum.5.040308.

[233] E. Pednault, An alternative approach to optimal wire cutting without ancilla qubits, 2023, http://dx.doi.org/10.48550/arXiv.2303.08287, arxiv preprint.

[234] C. Ufrecht, L.S. Herzog, D.D. Scherer, M. Periyasamy, S. Rietsch, A. Plinge, C. Mutschler, Optimal joint cutting of two-qubit rotation gates, Phys. Rev. A 109 (5) (2024) 052440, http://dx.doi.org/10.1103/PhysRevA.109.052440.

[235] A.W. Harrow, A. Lowe, Optimal quantum circuit cuts with application to clustered Hamiltonian simulation, 2024, http://dx.doi.org/10.48550/arXiv.2403.01018, arXiv.

[236] L. Schmitt, C. Piveteau, D. Sutter, Cutting circuits with multiple two-qubit unitaries, 2023, http://dx.doi.org/10.48550/arXiv.2312.11638, arxiv preprint.

[237] C. Ufrecht, M. Periyasamy, S. Rietsch, D.D. Scherer, A. Plinge, C. Mutschler, Cutting multi-control quantum gates with ZX calculus, Quantum 7 (2023) 1147, http://dx.doi.org/10.22331/q-2023-10-23-1147.

[238] J. Chen, E.M. Stoudenmire, S.R. White, Quantum Fourier transform has small entanglement, PRX Quantum 4 (4) (2023) http://dx.doi.org/10.1103/PRXQuantum.4.040318.

[239] Y. Zhang, L. Cincio, C.F.A. Negre, P. Czarnik, P.J. Coles, P.M. Anisimov, S.M. Mniszewski, S. Tretiak, P.A. Dub, Variational quantum eigensolver with reduced circuit complexity, Npj Quantum Inf. 8 (96) (2022) 1–10, http://dx.doi.org/10.1038/s41534-022-00599-z.

[240] S. Zhang, Z. Qin, Y. Zhou, R. Li, C. Du, Z. Xiao, Single entanglement connection architecture between multi-layer HEA for distributed VQE, 2023, http://dx.doi.org/10.48550/arXiv.2307.12323, arxiv preprint.

[241] G. Gentinetta, F. Metz, G. Carleo, Overhead-constrained circuit knitting for variational quantum dynamics, Quantum 8 (2024) 1296, http://dx.doi.org/10.22331/q-2024-03-21-1296.

[242] R. Nagai, S. Kanno, Y. Sato, N. Yamamoto, Quantum channel decomposition with preselection and postselection, Phys. Rev. A 108 (2) (2023) http://dx.doi.org/10.1103/PhysRevA.108.022615.

[243] D. Chen, B. Baheri, V. Chaudhary, Q. Guan, N. Xie, S. Xu, Approximate quantum circuit cutting, 2022, http://dx.doi.org/10.48550/arXiv.2212.01270, arxiv preprint.

[244] D.T. Chen, E.H. Hansen, X. Li, V. Kulkarni, V. Chaudhary, B. Ren, Q. Guan, S. Kuppannagari, J. Liu, S. Xu, Efficient quantum circuit cutting by neglecting basis elements, in: IEEE IPDPS Workshops, IPDPSW, 2023, pp. 517–523, http://dx.doi.org/10.1109/IPDPSW59300.2023.00091.

[245] S. Brandhofer, I. Polian, K. Krsulich, Optimal partitioning of quantum circuits using gate cuts and wire cuts, IEEE Trans. Quantum Eng. 5 (2024) 1–10, http://dx.doi.org/10.1109/TQE.2023.3347106.

[246] W. Tang, T. Tomesh, M. Suchara, J. Larson, M. Martonosi, CutQC: Using small quantum computers for large quantum circuit evaluations, in: Proc. of the 26th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems, ACM, 2021, pp. 473–486, http://dx.doi.org/10.1145/3445814.3446758.

[247] W. Tang, M. Martonosi, ScaleQC: A scalable framework for hybrid computation on quantum and classical processors, 2022, http://dx.doi.org/10.48550/arXiv.2207.00933, arxiv preprint.

[248] K.N. Smith, M.A. Perlin, P. Gokhale, P. Frederick, D. Owusu-Antwi, R. Rines, V. Omole, F. Chong, Clifford-based circuit cutting for quantum simulation, in: Proc. of the 50th Int. Symp. on Computer Architecture, ACM, 2023, pp. 1–13, http://dx.doi.org/10.1145/3579371.3589352.

[249] T. Chatterjee, A. Das, S.I. Mohtashim, A. Saha, A. Chakrabarti, Qurzon: A prototype for a divide and conquer-based quantum compiler for distributed quantum systems, SN Comput. Sci. 3 (4) (2022) http://dx.doi.org/10.1007/s42979-022-01207-9.

[250] L.S. Herzog, F. Wagner, C. Ufrecht, L. Palackal, A. Plinge, C. Mutschler, D.D. Scherer, Improving Quantum and Classical Decomposition Methods for Vehicle Routing, 2024, http://dx.doi.org/10.48550/arXiv.2404.05551, arXiv.

[251] A. Robert, P.K. Barkoutsos, S. Woerner, I. Tavernelli, Resource-efficient quantum algorithm for protein folding, Npj Quantum Inf. 7 (1) (2021) 38, http://dx.doi.org/10.1038/s41534-021-00368-4.

[252] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G.H. Booth, J. Tennyson, The variational quantum eigensolver: A review of methods and best practices, Phys. Rep. 986 (2022) 1–128, http://dx.doi.org/10.1016/j.physrep.2022.08.003, The Variational Quantum Eigensolver: A Review of Methods and Best Practices.

[253] Y. Wang, L.M. Sager-Smith, D.A. Mazziotti, Quantum simulation of bosons with the contracted quantum eigensolver, New J. Phys. 25 (10) (2023) http://dx.doi.org/10.1088/1367-2630/acf9c3.

[254] P. Gokhale, O. Angiuli, Y. Ding, K. Gui, T. Tomesh, M. Suchara, M. Martonosi, F.T. Chong, Minimizing state preparations in variational quantum eigensolver by partitioning into commuting families, 2019, http://dx.doi.org/10.48550/arXiv.1907.13623, arxiv preprint.

[255] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran, Evaluating analytic gradients on quantum hardware, Phys. Rev. A 99 (2019) 032331, http://dx.doi.org/10.1103/PhysRevA.99.032331.

[256] J. Stokes, J. Izaac, N. Killoran, G. Carleo, Quantum natural gradient, Quantum 4 (2020) 269, http://dx.doi.org/10.22331/q-2020-05-25-269.

[257] J.D. Viqueira, D. Faílde, M.M. Juane, A. Gómez, D. Mera, Density matrix emulation of quantum recurrent neural networks for multivariate time series prediction, 2023, http://dx.doi.org/10.48550/arXiv.2310.20671, arXiv.

[258] D. Wierichs, J. Izaac, C. Wang, C.Y.-Y. Lin, General parameter-shift rules for quantum gradients, Quantum 6 (2022) 677, http://dx.doi.org/10.22331/q-2022-03-30-677.

[259] D. Faílde, J.D. Viqueira, M. Mussa Juane, A. Gómez, Using differential evolution to avoid local minima in variational quantum algorithms, Sci. Rep. 13 (1) (2023) 16230, http://dx.doi.org/10.1038/s41598-023-43404-3.

[260] M.S. Alvarez-Alvarado, F.E. Alban-Chacón, E.A. Lamilla-Rubio, C.D. Rodríguez-Gallegos, W. Velásquez, Three novel quantum-inspired swarm optimization algorithms using different bounded potential fields, Sci. Rep. 11 (1) (2021) 11655, http://dx.doi.org/10.1038/s41598-021-90847-7.

[261] E.R. Anschuetz, B.T. Kiani, Quantum variational algorithms are swamped with traps, Nat. Commun. 13 (1) (2022) 7760, http://dx.doi.org/10.1038/s41467-022-35364-5.

[262] Y. Du, Y. Qian, X. Wu, D. Tao, A distributed learning scheme for variational quantum algorithms, IEEE Trans. Quantum Eng. 3 (2022) http://dx.doi.org/10.1109/TQE.2022.3175267.

[263] S.Y.C. Chen, S. Yoo, Federated quantum machine learning, Entropy 23 (4) (2021) http://dx.doi.org/10.3390/e23040460.

[264] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, A. Li, EQC: Ensembled quantum computing for variational quantum algorithms, in: Proc. of the 49th Int. Symp. on Computer Architecture, ACM, 2022, pp. 59–71, http://dx.doi.org/10.1145/3470496.3527434.

[265] P. Das, S.S. Tannu, P.J. Nair, M. Qureshi, A case for multi-programming quantum computers, in: Proceedings of the 52nd Annual IEEE/ACM Int. Symp. on Microarchitecture, 2019, pp. 291–303, http://dx.doi.org/10.1145/3352460.3358287.

[266] A. Ash-Saki, M. Alam, S. Ghosh, Analysis of crosstalk in NISQ devices and security implications in multi-programming regime, in: Proc. of the ACM/IEEE Int. Symp. on Low Power Electronics and Design, 2020, pp. 25–30, http://dx.doi.org/10.1145/3370748.3406570.

[267] L. Liu, X. Dou, QuCloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment, in: IEEE Int. Symp. on High-Performance Computer Architecture, HPCA, IEEE, 2021, pp. 167–178, http://dx.doi.org/10.1109/HPCA51647.2021.00024.

[268] A.A. Saki, S. Ghosh, Qubit sensing: A new attack model for multi-programming quantum computing, 2021, http://dx.doi.org/10.48550/arXiv.2104.05899, arxiv preprint.

[269] S. Niu, A. Todri-Sanial, Multi-programming cross platform benchmarking for quantum computing hardware, 2022, http://dx.doi.org/10.48550/arXiv.2206.03144, arxiv preprint.

[270] S. Niu, A. Todri-Sanial, How parallel circuit execution can be useful for NISQ computing? in: 2022 Design, Automation & Test in Europe Conference & Exhibition, DATE, 2022, pp. 1065–1070, http://dx.doi.org/10.23919/DATE54114.2022.9774512.

[271] S. Niu, A. Todri-Sanial, Enabling multi-programming mechanism for quantum computing in the NISQ era, Quantum 7 (2023) 925, http://dx.doi.org/10.22331/q-2023-02-16-925.

[272] S. Niu, A. Todri-Sanial, Multi-programming mechanism on near-term quantum computing, in: Quantum Computing: Circuits, Systems, Automation and Applications, Springer, 2023, pp. 19–54, http://dx.doi.org/10.1007/978-3-031-37966-6_2.

[273] L. Liu, X. Dou, QuCloud+: A holistic qubit mapping scheme for single/multi-programming on 2D/3D NISQ quantum computers, ACM Trans. Archit. Code Optim. 21 (1) (2024) 1–27, http://dx.doi.org/10.1145/3631525.

[274] J.K. Lee, O.T. Brown, M. Bull, M. Ruefenacht, J. Doerfert, M. Klemm, M. Schulz, Quantum task offloading with the openmp API, 2023, http://dx.doi.org/10.48550/arXiv.2311.03210, arxiv preprint.

[275] The CUDA Quantum development team, CUDA Quantum, 2025, URL https://github.com/NVIDIA/cuda-quantum. (Accessed 21 January 2025).

[276] A.J. Abhari, A. Faruque, M.J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, F. Chong, M. Martonosi, M. Suchara, K. Brown, M. Pedram, T. Brun, Scaffold: Quantum Programming Language, Tech. Rep., Princeton University, Dept. of Computer Science, 2012, URL https://www.cs.princeton.edu/techreports/2012/934.pdf. (Accessed 21 January 2025).

[277] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, M. Roetteler, Q#: Enabling scalable quantum computing and development with a high-level DSL, in: Proc. of the Real World Domain Specific Languages Workshop, ACM, 2018, pp. 1–10, http://dx.doi.org/10.1145/3183895.3183901.

[278] J. Guo, H. Lou, J. Yu, R. Li, W. Fang, J. Liu, P. Long, S. Ying, M. Ying, isQ: An integrated software stack for quantum programming, IEEE Trans. Quantum Eng. 4 (2023) http://dx.doi.org/10.1109/TQE.2023.3275868.

[279] S. Liu, X. Wang, L. Zhou, J. Guan, Y. Li, Y. He, R. Duan, M. Ying, Q|SI⟩: A quantum programming environment, in: Symposium on Real-Time and Hybrid Systems: Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday, Springer International Publishing, 2018, pp. 133–164, http://dx.doi.org/10.1007/978-3-030-01461-2_8, Chapter 8.

[280] Qiskit contributors, Qiskit: An open-source framework for quantum computing, 2023, http://dx.doi.org/10.5281/zenodo.2573505.

[281] Cirq Developers, Cirq, 2023, http://dx.doi.org/10.5281/zenodo.4062499.

[282] Y. Suzuki, Y. Kawase, Y. Masumura, Y. Hiraga, M. Nakadai, J. Chen, K.M. Nakanishi, K. Mitarai, R. Imai, S. Tamiya, T. Yamamoto, T. Yan, T. Kawakubo, Y.O. Nakagawa, Y. Ibe, Y. Zhang, H. Yamashita, H. Yoshimura, A. Hayashi, K. Fujii, Qulacs: A fast and versatile quantum circuit simulator for research purpose, Quantum 5 (2021) http://dx.doi.org/10.22331/Q-2021-10-06-559.

[283] T. Häner, D.S. Steiger, T. Hoefler, M. Troyer, Distributed quantum computing with QMPI, in: Int. Conf. for High Perf. Computing, Networking, Storage and Analysis, IEEE, 2021, http://dx.doi.org/10.1145/3458817.3476172.

[284] Y. Shi, T. Nguyen, S. Stein, T. Stavenger, M. Warner, M. Roetteler, T. Hoefler, A. Li, A reference implementation for a quantum message passing interface, in: Proc. of the SC'23 Workshops, ACM, 2023, pp. 1420–1425, http://dx.doi.org/10.1145/3624062.3624212.

[285] R. Wakizaka, Towards reliable distributed quantum computing on quantum interconnects, ACM Int. Conf. Proc. Ser. (2023) 114–116, http://dx.doi.org/10.1145/3594671.3594691.

[286] F. Chow, Intermediate representation, Queue 11 (2013) 30–37, http://dx.doi.org/10.1145/2542661.2544374.

[287] A. McCaskey, T. Nguyen, A MLIR dialect for quantum assembly languages, in: Int. Conf, on Quantum Computing and Engineering, QCE, IEEE, 2021, pp. 255–264, http://dx.doi.org/10.1109/QCE52317.2021.00043.

[288] T. Lubinski, C. Granade, A. Anderson, A. Geller, M. Roetteler, A. Petrenko, B. Heim, Advancing hybrid quantum–classical computation with real-time execution, Front. Phys. 10 (2022) 940293, http://dx.doi.org/10.3389/FPHY.2022.940293.

[289] A. Peduri, S. Bhat, T. Grosser, QSSA: An SSA-based IR for quantum computing, in: Proc. of the 31st ACM SIGPLAN Int. Conf. on Compiler Construction, ACM, New York, NY, USA, 2022, pp. 2–14, http://dx.doi.org/10.1145/3497776.3517772.

[290] D. Ittah, T. Häner, V. Kliuchnikov, T. Hoefler, QIRO: A static single assignment-based quantum program representation for optimization, ACM Trans. Quantum Comput. 3 (3) (2022) http://dx.doi.org/10.1145/3491247.

[291] K. Hietala, R. Rand, S.-H. Hung, X. Wu, M. Hicks, Verified optimization in a quantum intermediate representation, 2019, http://dx.doi.org/10.48550/arXiv.1904.06319, arxiv preprint.

[292] X.-Z. Luo, J.-G. Liu, P. Zhang, L. Wang, Yao.jl: Extensible, efficient framework for quantum algorithm design, Quantum 4 (2020) 341, http://dx.doi.org/10.22331/q-2020-10-11-341.

[293] S. Nishio, R. Wakizaka, InQuIR: Intermediate representation for interconnected quantum computers, 2023, http://dx.doi.org/10.48550/arXiv.2302.00267, arxiv preprint.

[294] A.W. Cross, L.S. Bishop, J.A. Smolin, J.M. Gambetta, Open quantum assembly language, 2017, http://dx.doi.org/10.48550/arXiv.1707.03429, arxiv preprint.

[295] N. Khammassi, G.G. Guerreschi, I. Ashraf, J.W. Hogaboam, C.G. Almudever, K. Bertels, cQASM v1.0: Towards a common quantum assembly language, 2018, http://dx.doi.org/10.48550/arXiv.1805.09607, arxiv preprint.

[296] X. Fu, L. Riesebos, M.A. Rol, J. Van Straten, J. Van Someren, N. Khammassi, I. Ashraf, R.F. Vermeulen, V. Newsum, K.K. Loh, J.C. De Sterke, W.J. Vlothuizen, R.N. Schouten, C.G. Almudever, L. Dicarlo, K. Bertels, EQASM: An executable quantum instruction set architecture, in: Proc. 25th IEEE Int. Symp. on High Performance Computer Architecture, HPCA, IEEE, 2019, pp. 224–237, http://dx.doi.org/10.1109/HPCA.2019.00040.

[297] K.M. Svore, A.V. Aho, A.W. Cross, I. Chuang, I.L. Markov, A layered software architecture for quantum computing design tools, Computer 39 (1) (2006) 74–83, http://dx.doi.org/10.1109/MC.2006.4.

[298] A. Dahlberg, B. van der Vecht, C.D. Donne, M. Skrzypczyk, I.t. Raa, W. Kozlowski, S. Wehner, NetQASM - a low-level instruction set architecture for hybrid quantum-classical programs in a quantum internet, Quantum Sci. Technol. 7 (3) (2022) 035023, http://dx.doi.org/10.1088/2058-9565/ac753f.

[299] M. Ying, Y. Feng, An algebraic language for distributed quantum computing, IEEE Trans. Comput. 58 (6) (2009) 728–743, http://dx.doi.org/10.1109/TC.2009.13.

[300] X. Qiu, L. Chen, Quantum cost of dense coding and teleportation, Phys. Rev. A 105 (6) (2022) 062451, http://dx.doi.org/10.1103/PhysRevA.105.062451.

[301] C. Piveteau, D. Sutter, Circuit knitting with classical communication, IEEE Trans. Inform. Theory 70 (4) (2024) 2734–2745, http://dx.doi.org/10.1109/TIT.2023.3310797.

[302] D.W. Wall, Global register allocation at link time, ACM SIGPLAN Not. 21 (1986) 264–275, http://dx.doi.org/10.1145/13310.13338.

[303] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, Y. Okamoto, Algorithmic theory of qubit routing, in: Algorithms and Data Structures, Springer Nature, 2023, pp. 533–546, http://dx.doi.org/10.1007/978-3-031-38906-1_35.

[304] A. Paler, On the influence of initial qubit placement during NISQ circuit compilation, in: Quantum Technology and Optimization Problems, Springer, 2019, pp. 207–217, http://dx.doi.org/10.1007/978-3-030-14082-3_18.

[305] M. Bandic, C.G. Almudever, S. Feld, Interaction graph-based characterization of quantum benchmarks for improving quantum circuit mapping techniques, Quantum Mach. Intell. 5 (2023) 1–30, http://dx.doi.org/10.1007/S42484-023-00124-1.

[306] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, S. Sivarajah, On the qubit routing problem, in: 14th Conf. on the Theory of Quantum Computation, Communication and Cryptography, TQC, Vol. 135, 2019, pp. 5:1–5:32, http://dx.doi.org/10.4230/LIPIcs.TQC.2019.5.

[307] H. Barnes, A Survey of Qubit Routing Algorithms (Ph.D. thesis), Rochester Institute of Technology, 2023, URL https://repository.rit.edu/theses/11428.

[308] Y. Mao, Y. Liu, Y. Yang, Qubit allocation for distributed quantum computing, in: Conf. on Computer Communications, IEEE, 2023, pp. 1–10, http://dx.doi.org/10.1109/INFOCOM53939.2023.10228915.

[309] Y. Mao, Y. Liu, Y. Yang, Probability-aware qubit-to-processor mapping in distributed quantum computing, in: 1st Workshop on Quantum Networks and Distributed Quantum Computing, ACM, 2023, pp. 51–56, http://dx.doi.org/10.1145/3610251.3610554.

[310] M. Nakai, Qubit Allocation For Distributed Quantum Computing (Bachelor's thesis), Keio University, 2021, URL https://aqua.sfc.wide.ad.jp/publications/dave_bthesis.pdf.

[311] Z. Chen, X. Chen, Y. Jiang, X. Cheng, Z. Guan, Routing strategy for distributed quantum circuit based on optimized gate transmission direction, Internat. J. Theoret. Phys. 62 (12) (2023) 255, http://dx.doi.org/10.1007/s10773-023-05489-4.

[312] S. Rodrigo, D. Spanò, M. Bandic, S. Abadal, H. Van Someren, A. Ovide, S. Feld, C.G. Almudéver, E. Alarcón, Characterizing the spatio-temporal qubit traffic of a quantum intranet aiming at modular quantum computer architectures, in: 9th ACM Int. Conf. on Nanoscale Computing and Communication, 2022, pp. 1–7, http://dx.doi.org/10.1145/3558583.3558846.

[313] S.B. Rached, I.L. Agudo, S. Rodrigo, M. Bandic, S. Feld, H. van Someren, E. Alarcón, C.G. Almudéver, S. Abadal, Characterizing the inter-core qubit traffic in large-scale quantum modular architectures, 2023, http://dx.doi.org/10.48550/arXiv.2310.01921, arxiv preprint.

[314] N. Khammassi, I. Ashraf, J. Someren, R. Nane, A. Krol, M.A. Rol, L. Lao, K. Bertels, C.G. Almudever, OpenQL: A portable quantum programming framework for quantum accelerators, ACM J. Emerg. Technol. Comput. Syst. 18 (1) (2021) 1–24, http://dx.doi.org/10.1145/3474222.

[315] L. Lao, H. Van Someren, I. Ashraf, C.G. Almudever, Timing and resource-aware mapping of quantum circuits to superconducting processors, IEEE Trans. Comput.- Aided Des. Integr. Circuits Syst. 41 (2) (2021) 359–371, http://dx.doi.org/10.1109/TCAD.2021.3057583.

[316] A. Fagan, R. Duncan, Optimising clifford circuits with quantomatic, Electron. Proc. Theor. Comput. Sci. 287 (2019) 85–105, http://dx.doi.org/10.4204/eptcs.287.5.

[317] M. Xu, Z. Li, O. Padon, S. Lin, J. Pointing, A. Hirth, H. Ma, J. Palsberg, A. Aiken, U.A. Acar, Z. Jia, Quartz: superoptimization of quantum circuits, in: Proc. of the Conf. on Programming Language Design and Implementation, PLDI, ACM, 2022, pp. 625–640, http://dx.doi.org/10.1145/3519939.3523433.

[318] S. Haider, S.A.R. Kazmi, An extended quantum process algebra (eQPAlg) approach for distributed quantum systems, 2020, http://dx.doi.org/10.48550/arXiv.2001.04249, arxiv preprint.

[319] M. Lalire, P. Jorrand, A process algebraic approach to concurrent and distributed quantum computation: Operational semantics, 2004, http://dx.doi.org/10.48550/arXiv.quant-ph/0407005, arXiv.

[320] Y. Feng, S. Li, M. Ying, Verification of distributed quantum programs, ACM Trans. Comput. Log. 23 (3) (2022) http://dx.doi.org/10.1145/3517145.

[321] C.A.R. Hoare, An axiomatic basis for computer programming, Commun. ACM 12 (1969) 576–580, http://dx.doi.org/10.1145/363235.363259.

[322] Y. Wang, Verification of distributed quantum protocols, 2021, http://dx.doi.org/10.48550/arXiv.2110.11416, arxiv preprint.

[323] C.H. Bennett, G. Brassard, Quantum cryptography: Public key distribution and coin tossing, Theoret. Comput. Sci. 560 (2014) 7–11, http://dx.doi.org/10.1016/j.tcs.2014.05.025.

[324] D. Ferrari, S. Nasturzio, M. Amoretti, Poster: A software tool for mapping and executing distributed quantum computations on a network simulator, in: QCrypt, 2021, URL https://2021.qcrypt.net/posters/QCrypt2021Poster185Ferrari.pdf. (Accessed 21 January 2025).

[325] D. Ferrari, S. Carretta, M. Amoretti, A modular quantum compilation framework for distributed quantum computing, IEEE Trans. Quantum Eng. 4 (2023) 1–13, http://dx.doi.org/10.1109/TQE.2023.3303935.

[326] D. Cuomo, Architectures and Circuits for Distributed Quantum Computing (Ph.D. thesis), Univ. of Naples Federico II, Univ. of Camerino, National Research Council of Italy, 2023, URL http://www.fedoatd.unina.it/961/1/THESIS_DQC.pdf.

[327] Y. Ohkura, T. Satoh, R. Van Meter, Simultaneous execution of quantum circuits on current and near-future NISQ systems, IEEE Trans. Quantum Eng. 3 (2022) 1–10, http://dx.doi.org/10.1109/TQE.2022.3164716.

[328] T. Tomesh, Z.H. Saleem, M.A. Perlin, P. Gokhale, M. Suchara, M. Martonosi, Divide and conquer for combinatorial optimization and distributed quantum computation, in: Int. Conf. on Quantum Computing and Engineering, QCE, Vol. 1, IEEE, 2023, pp. 1–12, http://dx.doi.org/10.1109/QCE57702.2023.00009.

[329] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, R. Duncan, T|ket): A retargetable compiler for NISQ devices, Quantum Sci. Technol. 6 (1) (2020) 014003, http://dx.doi.org/10.1088/2058-9565/ab8e92.

[330] J.C. Boschero, N.M.P. Neumann, W. van der Schoot, T. Sijpesteijn, R. Wezeman, Distributed quantum computing: Applications and challenges, 2024, http://dx.doi.org/10.48550/arXiv.2410.00609.

[331] C. Gidney, M. Ekerå, How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, Quantum 5 (2021) http://dx.doi.org/10.22331/q-2021-04-15-433.

[332] M. Ekerå, J. Håstad, Quantum algorithms for computing short discrete logarithms and factoring RSA integers, in: T. Lange, T. Takagi (Eds.), Post-Quantum Cryptography, in: LNCS, no. 10346, Springer, 2017, pp. 347–363, http://dx.doi.org/10.1007/978-3-319-59879-6_20.

[333] L. Xiao, D. Qiu, L. Luo, P. Mateus, Distributed quantum-classical hybrid shor's algorithm, 2023, http://dx.doi.org/10.48550/arXiv.2304.12100, arxiv preprint.

[334] N.M. Neumann, R. van Houte, T. Attema, Imperfect distributed quantum phase estimation, in: ICCS, in: LNCS, vol. 12142, Springer, 2020, pp. 605–615, http://dx.doi.org/10.1007/978-3-030-50433-5_46.

[335] R. Van Meter, W.J. Munro, K. Nemoto, K.M. Itoh, Arithmetic on a distributed-memory quantum multicomputer, ACM J. Emerg. Technol. Comput. Syst. 3 (4) (2008) http://dx.doi.org/10.1145/1324177.1324179.

[336] J. Tan, L. Xiao, D. Qiu, L. Luo, P. Mateus, Distributed quantum algorithm for Simon's problem, Phys. Rev. A 106 (3) (2022) http://dx.doi.org/10.1103/PhysRevA.106.032417.

[337] H. Li, D. Qiu, L. Luo, Distributed exact quantum algorithms for Deutsch–Jozsa problem, 2023, http://dx.doi.org/10.48550/arXiv.2303.10663, arxiv preprint.

[338] X. Zhou, D. Qiu, L. Luo, Distributed Bernstein–Vazirani algorithm, Phys. A 629 (2023) 129209, http://dx.doi.org/10.1016/j.physa.2023.129209.

[339] X. Zhou, D. Qiu, L. Luo, Distributed exact Grover's algorithm, Front. Phys. 18 (5) (2023) 1–25, http://dx.doi.org/10.1007/s11467-023-1327-x.

[340] J. Avron, O. Casper, I. Rozen, Quantum advantage and noise reduction in distributed quantum computing, Phys. Rev. A 104 (2021) http://dx.doi.org/10.1103/PhysRevA.104.052404.

[341] K. Fujii, K. Mizuta, H. Ueda, K. Mitarai, W. Mizukami, Y.O. Nakagawa, Deep variational quantum eigensolver: A divide-and-conquer method for solving a larger problem with smaller size quantum computers, PRX Quantum 3 (2022) http://dx.doi.org/10.1103/PRXQuantum.3.010346.

[342] Z. Zhou, Y. Du, X. Tian, D. Tao, QAOA-in-QAOA: Solving large-scale MaxCut problems on small quantum machines, Phys. Rev. Appl. 19 (2) (2023) 024027, http://dx.doi.org/10.1103/PhysRevApplied.19.024027.

[343] S.C. Marshall, C. Gyurik, V. Dunjko, High dimensional quantum machine learning with small quantum computers, Quantum 7 (2022) http://dx.doi.org/10.22331/q-2023-08-09-1078.

[344] K. Zhang, P. Rao, K. Yu, H. Lim, V. Korepin, Implementation of efficient quantum search algorithms on NISQ computers, Quantum Inf. Process. 20 (2021) 233, http://dx.doi.org/10.1007/s11128-021-03165-2.

[345] G. Park, K. Zhang, K. Yu, V. Korepin, Quantum multi-programming for Grover's search, Quantum Inf. Process. 22 (1) (2023) http://dx.doi.org/10.1007/s11128-022-03793-2.

[346] L.K. Grover, Trade-offs in the quantum search algorithm, Phys. Rev. A 66 (2002) http://dx.doi.org/10.1103/PhysRevA.66.052314.

[347] K. Li, D. Qiu, L. Li, S. Zheng, Z. Rong, Application of distributed semi-quantum computing model in phase estimation, Inform. Process. Lett. 120 (2017) 23–29, http://dx.doi.org/10.1016/j.ipl.2016.12.002.

[348] T. Tanaka, Y. Suzuki, S. Uno, R. Raymond, T. Onodera, N. Yamamoto, Amplitude estimation via maximum likelihood on noisy quantum computer, Quantum Inf. Process. 20 (2021) 1–29, http://dx.doi.org/10.1007/s11128-021-03215-9.

[349] S. DiAdamo, M. Ghibaudi, J. Cruise, Distributed quantum computing and network control for accelerated VQE, IEEE Trans. Quantum Eng. 2 (2021) 1–21, http://dx.doi.org/10.1109/TQE.2021.3057908.

Chapter 6

Quantum compilation process and intermediate representations for quantum computing

The core content of this chapter is from the research presented in the following peer-reviewed articles:

1. F. J. Cardama, J. Vázquez-Pérez, T. F. Pena, J. C. Pichel, and A. Gómez, "Quantum Compilation Process: A Survey", *Lecture Notes in Computer Science*, vol. 15385 LNCS, pp. 100–112, 2025, ISSN: 16113349. DOI: 10.1007/978-3-031-90200-0_9
   - Impact Summary: Article published in a SJR 2024 second quartile (Q2) journal.
2. F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of intermediate representations for quantum computing", *The Journal of Supercomputing 2025 81:2*, vol. 81, no. 2, pp. 418–, Jan. 2025, ISSN: 15730484. DOI: 10.1007/s11227-024-06892-2
   - Impact Summary: Article published in a JCR 2024 second quartile (Q2) journal.

This publication specifically addresses and fulfills Objectives O1 a) and O1 c) of this thesis. For a comprehensive overview of the publication's quality indicators (Impact Factor and Quartile), the reader is referred to the Results: published articles from Contributions section.

Quantum Compilation Process: A Survey

F. Javier Cardama[1( )], Jorge Vázquez-Pérez[1], Tomás F. Pena[1,2], Juan C. Pichel[1,2], and Andrés Gómez[3]

[1] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain
{javier.cardama,jorgevazquez.perez,tf.pena,juancarlos.pichel}@usc.es
[2] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain
[3] Galicia Supercomputing Center (CESGA), Avda. de Vigo S/N, 15705 Santiago de Compostela, Spain
andres.gomez.tato@cesga.es

Abstract. Quantum compilation, critical for bridging high-level quantum programming and physical hardware, faces unique challenges distinct from classical compilation. As quantum computing advances, scalable and efficient quantum compilation methods become necessary. This paper surveys the landscape of quantum compilation, detailing the processes of qubit mapping and circuit optimization, and emphasizing the need for integration with classical computing to harness quantum advantages. Techniques such as Variational Quantum Eigensolver (VQE) exemplify hybrid approaches, highlighting the potential synergy between quantum and classical systems. It is concluded that, while quantum compilation retains many classic methodologies, it introduces novel complexities and opportunities for optimization and verification, essential for the evolving field of quantum computing.

Keywords: quantum computing · compilation process · quantum languages · qubit allocation

1 Introduction

With the current growth in the field of quantum computing in recent years, the need for efficient quantum compilation becomes increasingly apparent as researchers strive to harness the potential of quantum processors. Quantum compilation, like its classic counterpart, plays a pivotal role in bridging the gap between high-level quantum programming languages and physical quantum hardware. However, as quantum computing systems continue to evolve, the demand for scalable and efficient quantum compilation methods arises. This paper delves into the realm of quantum compilation, exploring techniques for mapping qubits to clusters, optimising quantum circuits for parallel execution, and managing the complexity of large-scale quantum programs. The principal aim is to organise the current information pertaining to quantum compilation and guide it into a scheme similar to classic compilation, facilitating a

comprehensive understanding of the quantum computing landscape. However, within this survey, an examination of quantum computing interpreters is omitted because they are not considered to be of interest in terms of efficiency. This is attributed to the fact that each quantum circuit performs a predetermined number of shots. Consequently, the contention is that it is more efficient to compile and execute the circuit N times rather than interpreting it N times.

Section 2 will first give a brief introduction to the modular functioning of a classic compiler, and then, in Sect. 3, we will discuss the main novelties introduced by quantum compilation and how these can be integrated into a compiler scheme. Finally, Sect. 4 shows the conclusions drawn from this survey.

2 Classic Compilation

Compilers are crucial in software engineering, translating high-level programming languages into machine code and optimizing machine resources. They follow a structured process involving analysis and synthesis stages, as depicted in Fig. 1. The emergence of quantum computing has extended these principles to quantum compilers, which adapt these phases for quantum programming languages, highlighting the enduring relevance of classic compiler methodologies in new computing paradigms [4].

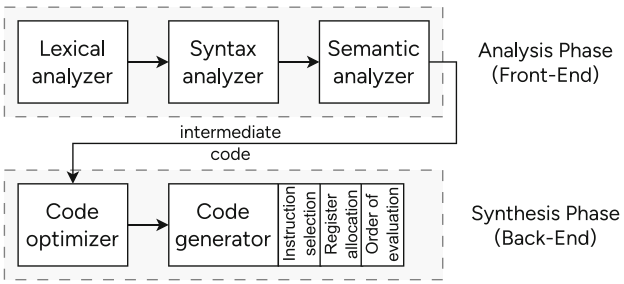


Fig. 1. Sequential phases of classic compiler process - analysis and synthesis stages.

The analysis phase marks the start of the compilation process, focusing on lexical, syntactical, and semantic evaluations to detect errors and form an intermediate representation. It includes tokenization, syntax tree construction, and semantic analysis like type-checking. The front-end generates an Intermediate Representation (IR), crucial for compiler efficiency and adaptability across various languages. This IR facilitates the compilation of languages like C++, Rust, and Java into popular Instruction Set Architectures (ISAs), enhancing code efficiency and quality through machine-independent optimizations.

The final phase in the compilation scheme, the code generator, is crucial as it adapts the code to the target machine's architecture, highlighting significant

differences between classic and quantum computing. This phase ensures the target program adheres to the semantics of the source program while optimizing resource utilization on the target machine. Key tasks included in the code generator phase include the following: instruction selection, register allocation and order of evaluation.

Instruction selection, which chooses the appropriate machine instructions or quantum logic gates, depending on whether the target is a classic or quantum computer; **register allocation**, crucial for managing limited computational resources like registers in classic computing or qubits in quantum computing; and **order of evaluation**, which determines the most efficient execution sequence for independent instructions.

The compilation scheme shown is the one that has been classicly used in compiler development in recent years. Therefore, there are many questions to be asked once quantum computing is incorporated: is the same development scheme being followed? And if so, should this be the scheme to follow? These questions will be addressed in the Sect. 3 on quantum compilation.

3 Quantum Compilation

When discussing quantum compilation, several pressing questions emerge: Is the process carried out in a purely quantum manner? How does it differ from classical compilation? What challenges and bottlenecks are intrinsic to quantum compilation? These questions will be addressed as we explore the landscape of quantum compilation.

To address whether the compilation is done entirely in a quantum fashion, it is enlightening to study the current state of quantum computing. Most quantum computers today are either quantum annealers or small-scale universal quantum computers in the NISQ era. These systems lack the capability to compile a comprehensive programming language.

Understanding this requires revisiting the definition of computation. There is no consensus on what "computing" is due to many advances, so an absolute definition is pending in the literature [15]. What we understand today is rooted in the 1930's concept of Automatic Computation (AC), aiming for machines to perform mathematical calculations automatically. Alan Turing's abstract machine, the Turing machine, forms the basis for all computing systems [49]. Computation, based on Turing's definition, is a process where a formal mathematical system operates according to a set of rules in a given physical architecture [14].

A computer executes automatic calculations, traditionally using binary computation, discerning two states, 0 and 1. From vacuum tubes to multicore processors, this binary model has been the foundation of computational implementations. However, the landscape shifted with quantum computation and Shor's algorithm [43]. Beyond Shor's demonstration of quantum superiority, physical limitations of silicon transistors also spurred interest. Will quantum computing replace the current paradigm? Tentatively, no.

Quantum computers can theoretically perform any task a classical computer can, but with different efficiencies. For some calculations, classical computers

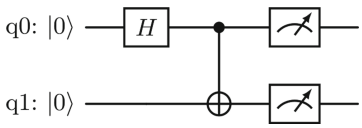
take exponentially more time. This difference is rooted in their principles. Classical computing uses the Cartesian product of systems with two distinct states, while quantum computing employs the tensor product of Hilbert spaces with a basis of two elements, leading to a larger computational space due to superposition and entanglement. However, this does not mean basic operations like NOT are inherently more efficient on a quantum platform. This computational parity explains why quantum computing will not replace the current paradigm. Key parts of a computer-processors, memory, and I/O devices-differ in quantum systems. Specifically, quantum memory and I/O devices pose challenges. Quantum mechanics nature does not align with the needs of I/O devices, which require constant information exchange. Using quantum computers for I/O is inefficient as it reduces qubits to bits.

In quantum computing, memory, often referred to as quantum memory, sparks debate. Quantum memory is volatile, based on quantum Random Access Memory (qRAM) [20,25]. Decoherence, a physical phenomenon, limits non-volatile quantum memory, making full quantum computation inoperable. Qubits are idealized as closed systems, but real quantum experiments can not achieve this, leading to decoherence.

Even if quantum processors, memory, and I/O devices were perfect, would quantum computers replace classical ones? No. Classical computation performs well in many fields with 100% certainty, a major lack in quantum computation. Replacing classical systems does not make sense, but integrating quantum and classical computing offers benefits. Hybrid quantum algorithms like the variational algorithms exemplify the potential synergy between quantum and classical computing, solving specific problems like molecular simulations and optimization [47]. These hybrid algorithms represent a collaborative approach, harnessing the strengths of quantum computing while mitigating challenges like error correction and hardware limitations [17].

After analyzing quantum computation's future, we can answer the first question: no, quantum compilation is not fully quantum. Analyzing the state of the art shows quantum compilation is a classic task with the quantum workload left for execution. Moreover, the workflow mirrors classical compilation. There is an analysis phase, termed the *front-end*, and a synthesis phase, termed the *back-end*, linked via a quantum Intermediate Representation (qIR), analogous to the classical IR. These phases persist throughout quantum computation, with the qIR abstracting multiple target types in the quantum scene.

The following section will describe the analysis phase, similar to its classical counterpart, followed by a section analyzing the synthesis phase, the most complex and divergent from classical counterparts due to its connection with quantum architecture.



(a) Bell circuit with measurement.

```
1 q0 = |0>
2 q1 = |0>
3 H(q0)
4 CNOT(q0, q1)
5 c0 = Measure(q0)
6 c1 = Measure(q1)
```

(b) Pseudocode for the circuit 2a.

Fig. 2. Example of codification of a circuit.

3.1 Compilation Phases for Quantum Computing

Analysis Phase. This phase bears the closest resemblance to the classic compilation world. This similarity should come as no surprise because the problem at hand is exactly the same as in classic compilation: translating a high-level language into an IR (referred to as qIR in this context) that will subsequently be utilized in the synthesis phase. To gain a better understanding of how this problem persists within the quantum workflow, we can engage in an abstraction exercise using the circuit presented in Fig. 2a. For instance, this circuit can be implemented using the structure outlined in the pseudocode found in Fig. 2b. The process of generating an object used for translation into qIR will entail the same steps as those outlined in Sect. 2. It will be presented below with a simplified version of each step, sufficiently profound to illuminate the fundamental concepts.

1. Lexical Analyzer. It detects `q0, q1, c0, c1, H, CNOT` and `Measure` as identifiers, `=` as comparison operator and `|0>` as, for instance, a native token (such as integers or floats). So the lexical result for the line 1 and 5 of Fig. 2b could be:

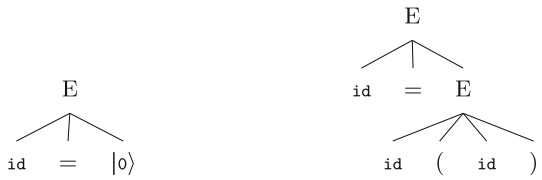
$$\texttt{q0} = |0\rangle \longleftrightarrow \{\texttt{id,0}\}\,\{=\}\,\{|0\rangle\}$$

$$\texttt{c0 = Measure(q0)} \longleftrightarrow \{\texttt{id,3}\}\,\{=\}\,\{\texttt{id,6}\}\,\{(\}\,\{\texttt{id,0}\}\,\{)\}$$

2. Syntax Analyzer. The previous lexical analysis is now fed into this phase in order to generate a parse tree, which will be generated following the grammar defined for the language. For the purpose of this work, a dummy grammar will be defined in order to define the parse tree of lines 1 and 6.

$$E \longrightarrow \texttt{id} = \texttt{ket} \mid \texttt{id} = \texttt{E} \mid \texttt{id}(\texttt{id})$$

   This grammar in conjunction with the lexical analysis defined above, produce the desired parse trees for lines 1 and 6, portrayed in Fig. 3a and Fig. 3b, respectively.
3. Semantic Analyzer: In this phase, the parse tree generated in the previous step is used to verify the semantic consistency of the source code with the language's definition. For instance, a possible check involves determining if the value $|0\rangle$ is valid for the `q0` identifier and which type should be considered



(a) Parse tree of line 1. (b) Parse tree of line 6.

Fig. 3. Example of parse trees.

for it. In this scheme, it is treated as a native value, similar to integers or floats. However, during this stage, it can be represented as a two-integer array, such as $[0, 0]$, or any other suitable format, which can then be translated into an appropriate signal at the physical layer.

This simplistic example suffices to achieve an important task: showing how similar it is the analysis phase in quantum computing in comparison with classic computing. In fact, if someone abstracts from the common concepts of gates, qubits and measurements, and reads the pseudocode with its grammar and parse trees it is impossible for them to tell whether the language purpose is modeling quantum computing or, for example, embedded computing.

This is precisely why, when conducting a current survey of the state-of-the-art in quantum programming languages [1,18,42], numerous libraries emerge, being among them a set of the most used technologies of the quantum computing domain, as it is the case for Qiskit [13], Circ [16], ProjectQ [45], Pennylane [9], among others. Table 1 lists the different software tools for the development of quantum applications. In this survey, these tools have been categorised into four distinct categories:

- **Frameworks and libraries:** offering pre-written code to enhance efficiency. While libraries provide specific functionalities allowing developers to maintain control over the application's flow, frameworks dictate the project's structure through "inversion of control", where the framework calls the developer's code. Despite some software being labeled differently, most tools like Qiskit, Cirq, ProjectQ, and PennyLane, as well as others such as Qulacs, are generally categorized under frameworks and libraries without a strict adherence to the 'inversion of control' principle.
- **Language extensions:** enhance existing languages by adding features, tools, or syntax. These extensions are not standalone languages but augment the base language's capabilities. Notably, QCOR extends C++ and is highlighted for its relevance, while Quipper, although no longer in use, is significant in quantum software literature.
- **Standalone languages:** self-sufficient languages with their own syntax, semantics, and standard libraries, allowing for the development of various applications independently of other languages. It specifically highlights quantum programming languages.

Table 1. Programming software for quantum computing.

Frameworks libraries		Language extensions		Standalone language		Quantum process algebras
Name	Basis	Name	Basis	Imperative	Functional	
Blueqat	Python	QCOR [36]	C++	isQ [22]	cQPL [34]	CQP [36]
Cirq [16]		Proto-Quipper [41]	Haskell	LanQ [37]	Q [10]	eQPAlg [23]
Penny Lane [9]		Quipper [21]		Q#	QFC [7]	QPAlg [27]
ProjectQ [45]		LIQUi|⟩ [51]	F#	QCL [53]	QML [5]	
Perceval [24]		Chisel-Q [31]	Scala	Q|SI⟩ [30]	QPL	
Qiskit [13]		qGCL [52]	GCL	Quingo [2]	QuaFL [29]	
Quantify		qPCF [32]	PCF	QWIRE [39]	Qunity [50]	
Strawberry Fields		Lambda-q [35]	Lambda Calculus	Scaffold [3]		
Tequila		λ_q [48]		Silq [11]		
Quantum++ [19]	C++					
QuEST [26]						
Qulacs [46]						
QuantumOptics.jl [28]	Julia					
Yao.jl [33]						
Cove [40]	C#					

– **Quantum process algebras:** are theoretical constructs aimed at modeling and analyzing quantum computing systems, especially in quantum communication. They build on traditional process algebras by incorporating quantum mechanics concepts like states, operations, entanglement, and superposition. These algebras serve as formal languages for accurately describing and reasoning about quantum processes, aiding in the development and verification of quantum algorithms and protocols. Although these frameworks are not currently in use, they are recognized for their significant contributions to the initial phases of quantum software development.

Understanding the difference between libraries and programming language extensions in quantum computing can be challenging. For instance, Qiskit operates as a library within Python, offering tools like the `QuantumCircuit` for building quantum circuits without altering Python's syntax. In contrast, QCOR extends C++ by introducing quantum-specific constructs, such as the `__qpu__` keyword for quantum kernels, directly integrating quantum capabilities into the language's syntax. This example is shown clearly in Fig. 4.

Synthesis Phase. In classic compilation, each line of code translates to machine-level instructions. However, in quantum languages, most components are classical except for the quantum circuit part, which is generated by the classical elements of the program. Quantum computing languages typically exist as libraries or extensions of classical languages to optimize quantum circuits for

```
__qpu__ void ghz(qreg q) {
  auto first_qubit = q.head();
  H(first_qubit);
  for (auto i : range(q.size()-1))
    X::ctrl(q[i], q[i+1]);
  Measure(q);
}
```

(a) QCOR code (C++ extension).

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)

qc.measure(1, 0)
```

(b) Qiskit code (Python library).

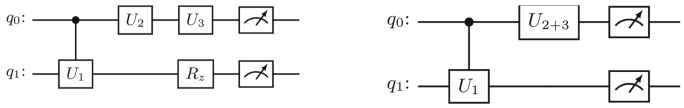
Fig. 4. Example of the difference between a library and a language extension.

minimal gate count and depth. The synthesis phase in quantum compilation is crucial for generating these optimal circuits, whether for actual quantum computers or simulators. This review focuses on the quantum aspects of this phase, highlighting the unique challenges of quantum compilation.

The synthesis phase of quantum compilation can be broadly divided into two main components: **circuit optimization** and **qubit mapping**. Of course, these two parts are not the only ones which can be included in the synthesis phase. Different techniques and processes such as error correction, error mitigation, analysis of metrics extracted from the quantum circuit and so on and so forth, can be included in this phase. But there is not an agreement among the different compilers and software tools whether other techniques are necessary in this phase.

Circuit optimization operates on the principle of modifying a quantum circuit to minimize a specific metric, essential due to the complexity of quantum computers. The choice of metric varies: for NISQ devices, reducing the count of two-qubit gates might be crucial, while for others, minimizing circuit depth due to qubit decoherence may be preferred. The main challenge in circuit optimization is ensuring the modified circuit retains its original functionality. This necessitates verification to confirm the circuit still performs the same computational task. Thus, the optimization phase can be divided into two subphases: **improvement** and **verification**.

The aim of **circuit improvement** is to enhance the performance and efficiency of quantum circuits through techniques such as removing redundant inverse gates, optimizing subcircuits, and employing various algorithms for better circuit synthesis. Figure 5 illustrates an example of circuit improvement using gate fusion and deletion. As can be seen, gates U_2 and U_3 are susceptible to be fused, while the rotations in the Z axis do not modify the measurement in the computational basis, therefore they can be eliminated. Conversely, **circuit verification** ensures the correctness of these optimized quantum circuits, verifying that the transformations applied during optimization preserve the intended functionality, with techniques like [6] and ZX-Calculus [12] maintaining circuit integrity. In quantum compilation, while many compilers like Qiskit, ProjectQ, and ScafCC focus primarily on optimization, the integration of verifica-



(a) Non-optimized circuit. (b) Optimized circuit.

Fig. 5. Example of the optimization of a circuit applying gate fusion techniques.

tion underscores the importance of ensuring circuit correctness throughout the compilation process.

Qubit mapping: allocation and routing is analogous to register allocation in classical computing, aims to efficiently assign logical qubits to physical qubits in a quantum device, considering constraints like device connectivity [44], while minimizing resources and execution time. This process is complicated by the varying error rates and connectivity of qubits, making it an NP-hard problem where exact algorithms are feasible only for small numbers of qubits. Consequently, heuristic and approximation techniques are often employed to find near-optimal solutions. This task involves two main processes:

- **Quantum allocation:** refers to the process of physically assigning specific logical qubits in a quantum processor. For a correct qubit allocation, in most cases, it is necessary to add additional SWAP gates to move the qubit information [38].
- **Quantum routing:** refers to the task of finding efficient paths for communication between qubits in a quantum processor. This is important when mapping gates of two logic qubits that are not interconnected to maximise efficiency [8].

4 Conclusions

This survey has examined the state-of-the-art in quantum compilation, focusing on the processes and challenges unique to quantum systems. Quantum compilation retains many parallels with classical methodologies, including the analysis and synthesis phases, but it introduces significant complexities. The necessity for qubit mapping and circuit optimization, coupled with verification techniques, underscores the distinct nature of quantum compilation. Furthermore, the potential of hybrid algorithms, such as the variational algorithms, demonstrates the benefits of integrating quantum and classical computing. As quantum technology progresses, efficient and scalable compilation methods will be crucial for maximizing the performance and applicability of quantum processors. Future work should continue to explore and refine these methods, ensuring robust and effective compilation processes that can meet the demands of advancing quantum hardware. In the same way, it is also necessary to focus on innovative methodologies, such as distributed quantum compilers.

Acknowledgments. This work was supported by MICINN through the European Union NextGenerationEU recovery plan (PRTR-C17.I1), and by the Galician Regional Government through the "Planes Complementarios de I+D+I con las Comunidades Autónomas" in Quantum Communication. This work was also supported by the Ministry of Economy and Competitiveness, Government of Spain (Grant Numbers PID2019-104834GB-I00, PID2022141623NB-I00 and PID2022-137061OB- C22), Consellería de Cultura, Educación e Ordenación Universitaria (accreditations ED431C 2022/16 and ED431G-2019/04), and the European Regional Development Fund (ERDF), which acknowledges the CiTIUS-Research Center in Intelligent Technologies of the University of Santiago de Compostela as a Research Center of the Galician University System.

Disclosure of Interests. The authors declare no conflict of interest.

References

1. Quantum programming languages (2020). https://doi.org/10.1038/s42254-020-00245-7
2. Fu, X., et al.: Quingo: a programming framework for heterogeneous quantum-classical computing with nisq features. ACM Trans. Quant. Comput. **2**, 1–37 (2021). https://doi.org/10.1145/3483528
3. Abhari, A.J., et al.: Scaffold: Quantum programming language (2012)
4. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
5. Altenkirch, T., Grattage, J.: A functional quantum programming language (2005)
6. Amy, M.: Towards large-scale functional verification of universal quantum circuits. Electron. Proc. Theor. Comput. Sci. EPTCS **287**, 1–21 (2018). https://doi.org/10.4204/EPTCS.287.1. http://arxiv.org/abs/1805.06908
7. Bal, H.E., Steiner, J.G., Tanenbaum, A.S.: Programming languages for distributed computing systems. ACM Comput. Surv. (CSUR) **21**, 261–322 (1989). https://doi.org/10.1145/72551.72552
8. Bandic, M., Almudever, C.G., Feld, S.: Interaction graph-based characterization of quantum benchmarks for improving quantum circuit mapping techniques. Quant. Mach. Intell. **5**, 1–30 (2023). https://doi.org/10.1007/S42484-023-00124-1/TABLES/4. https://link.springer.com/article/10.1007/s42484-023-00124-1
9. Bergholm, V., , et al.: Pennylane: Automatic differentiation of hybrid quantum-classical computations (2018). http://arxiv.org/abs/1811.04968
10. Bettelli, S., et al.: Toward an architecture for quantum programming. Eur. Phys. J. D - Atomic Molec. Opt. Plasma Phys. **25**(2), 181–200 (2003). https://doi.org/10.1140/EPJD/E2003-00242-2
11. Bichsel, B., et al.: Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 286–300 (2020). https://doi.org/10.1145/3385412.3386007
12. Coecke, B., Duncan, R.: Interacting quantum observables. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). LNCS, vol. 5126, pp. 298–310 (2008). https://doi.org/10.1007/978-3-540-70583-3_25/COVER

13. Contributors, Q.: Qiskit: an open-source framework for quantum computing (2023). https://doi.org/10.5281/zenodo.2573505
14. Copeland, B.J.: What is computation? Synthese **108**, 335–359 (1996)
15. Denning, P.J.: The great principles of computing. Am. Sci. **98**(5), 369–372 (2010)
16. Developers, C.: Cirq (2023). https://doi.org/10.5281/zenodo.8161252
17. Endo, S., et al.: Hybrid quantum-classical algorithms and quantum error mitigation. J. Phys. Soc. Jpn. **90**(3), 032001 (2021)
18. Garhwal, S., Ghorani, M., Ahmad, A.: Quantum programming language: a systematic review of research topic and top cited languages. Arch. Comput. Methods Eng. **28**(2), 289–310 (2019). https://doi.org/10.1007/s11831-019-09372-6
19. Gheorghiu, V.: Quantum++: a modern c++ quantum computing library. PLoS ONE **13** (2018). https://doi.org/10.1371/journal.pone.0208073
20. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Phys. Rev. Lett. **100**, 160501 (2008)
21. Green, A.S., et al.: Quipper: a scalable quantum programming language (2013). https://doi.org/10.1145/2499370.2462177
22. Guo, J., et al.: isq: An integrated software stack for quantum programming. IEEE Trans. Quant. Eng. **4** (2023). https://doi.org/10.1109/TQE.2023.3275868
23. Haider, S., Kazmi, D.S.A.R.: An extended quantum process algebra (eqpalg) approach for distributed quantum systems (2020). http://arxiv.org/abs/2001.04249
24. Heurtel, N., et al.: Perceval: a software platform for discrete variable photonic quantum computing. Quantum **7**, 931 (2023). https://doi.org/10.22331/q-2023-02-21-931
25. Jaques, S., Rattew, A.G.: Qram: a survey and critique. arXiv preprint arXiv:2305.10310 (2023)
26. Jones, T., et al.: Quest and high performance simulation of quantum computers. Sci. Rep. **9** (2019). https://doi.org/10.1038/s41598-019-47174-9
27. Jorrand, P., Lalire, M.: Toward a quantum process algebra. In: 2004 Computing Frontiers Conference, pp. 111–119 (2004). https://doi.org/10.1145/977091.977108
28. Krämer, S., Plankensteiner, D., Ostermann, L., Ritsch, H.: Quantumoptics.jl: a julia framework for simulating open quantum systems. Comput. Phys. Commun. **227**, 109–116 (2018). https://doi.org/10.1016/J.CPC.2018.02.004
29. Lapets, A., da Silva, M.P., Thome, M., Adler, A., Beal, J., Roetteler, M.: Quafl: a typed dsl for quantum programming, pp. 19–26. Association for Computing Machinery (2013). https://doi.org/10.1145/2505351.2505357
30. Liu, S., et al.: $Q|SI\rangle$: a quantum programming environment. In: Jones, C., Wang, J., Zhan, N. (eds.) Symposium on Real-Time and Hybrid Systems. LNCS, vol. 11180, pp. 133–164. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01461-2_8
31. Liu, X., Kubiatowicz, J.: Chisel-q: designing quantum circuits with a scala embedded language. In: 2013 IEEE 31st International Conference on Computer Design, ICCD 2013, pp. 427–434 (2013). https://doi.org/10.1109/ICCD.2013.6657075
32. Paolini, L., Zorzi, M.: qPCF: a language for quantum circuit computations. In: Gopal, T.V., Jäger, G., Steila, S. (eds.) TAMC 2017. LNCS, vol. 10185, pp. 455–469. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55911-7_33
33. Luo, X.Z., et al.: Yao.jl: extensible, efficient framework for quantum algorithm design (2020). https://doi.org/10.22331/q-2020-10-11-341
34. Mauerer, W.: Semantics and simulation of communication in quantum programming (2005)
35. Maymin, P.: Extending the lambda calculus to express randomized and quantumized algorithms (1996). https://arxiv.org/abs/quant-ph/9612052v2

36. Mintz, T.M., et al.: Qcor: a language extension specification for the heterogeneous quantum-classical model of computation. J. Emerg. Technol. Comput. Syst. **16**(2) (2020). https://doi.org/10.1145/3380964
37. Mlnarik, H.: Operational semantics and type soundness of quantum programming language lanq (2007). https://arxiv.org/abs/0708.0890v1
38. Paler, A.: On the influence of initial qubit placement during NISQ circuit compilation. In: Feld, S., Linnhoff-Popien, C. (eds.) QTOP 2019. LNCS, vol. 11413, pp. 207–217. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14082-3_18
39. Paykin, J., Rand, R., Zdancewic, S.: Qwire: a core language for quantum circuits. ACM SIGPLAN Not. **52**, 846–858 (2017). https://doi.org/10.1145/3009837.3009894
40. Purkeypile, M.D.: Cove: A practical quantum computer programming framework. arXiv org (2009)
41. Rios, F., Selinger, P.: A categorical model for a quantum circuit description language. Electron. Proc. Theor. Comput. Sci. **266** (2017). https://doi.org/10.4204/EPTCS.266.11
42. Serrano, M.A., Cruz-Lemus, J.A., Perez-Castillo, R., Piattini, M.: Quantum software components and platforms: overview and quality assessment. ACM Comput. Surv. **55**, 1–31 (2023). https://doi.org/10.1145/3548679
43. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
44. Siraichi, M.Y., et al.: Qubit allocation. In: Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018, pp. 113–125. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3168822
45. Steiger, D.S., et al.: Projectq: an open source software framework for quantum computing. Quantum **2** (2018). https://doi.org/10.22331/q-2018-01-31-49
46. Suzuki, Y., et al.: Qulacs: a fast and versatile quantum circuit simulator for research purpose. Quantum **5** (2021). https://doi.org/10.22331/Q-2021-10-06-559
47. Tilly, J., et al.: The variational quantum eigensolver: a review of methods and best practices. Phys. Rep. **986**, 1–128 (2022). https://doi.org/10.1016/j.physrep.2022.08.003
48. van Tonder, A.: A lambda calculus for quantum computation. SIAM J. Comput. **33**, 1109–1135 (2003). https://doi.org/10.1137/S0097539703432165
49. Turing, A.M., et al.: On computable numbers, with an application to the entscheidungsproblem. J. Math **58**(345–363), 5 (1936)
50. Voichick, F., et al.: Qunity: a unified language for quantum and classical computing. Proc. ACM Program. Lang. **7**, 921–951 (2023). https://doi.org/10.1145/3571225
51. Wecker, D., Svore, K.M.: Liqui|spsdoigtsps: a software design architecture and domain-specific language for quantum computing (2014)
52. Zuliani, P.: Non-deterministic quantum programming, pp. 179–195 (2004)
53. Ömer, B.: Classical concepts in quantum programming. Int. J. Theor. Phys. **44**, 943–955 (2005). https://doi.org/10.1007/S10773-005-7071-X/METRICS

The Journal of Supercomputing (2025) 81:418
https://doi.org/10.1007/s11227-024-06892-2

Review of intermediate representations for quantum computing

F. Javier Cardama[1] · Jorge Vázquez-Pérez[1] · César Piñeiro[1,2] · Juan C. Pichel[1,2] · Tomás F. Pena[1,2] · Andrés Gómez[3]

Accepted: 23 December 2024

Abstract
Intermediate representations (IRs) are fundamental to classical and quantum computing, bridging high-level quantum programming languages and the hardware-specific instructions required for execution. This paper reviews the development of quantum IRs, focusing on their evolution and the need for abstraction layers that facilitate portability and optimization. Monolithic quantum IRs, such as QIR (Lubinski et al. in Front Phys 10:940293, 2022. https://doi.org/10.3389/fphy.2022.940293), QSSA (Peduri et al. in Proceedings of the 31st ACM SIGPLAN international conference on compiler construction. CC 2022. Association for Computing Machinery, New York, 2022), or Q-MLIR (McCaskey and Nguyen in Proceedings-2021 IEEE International Conference on Quantum Computing and Engineering, QCE, 2021), their effectiveness in handling abstractions, and their hybrid support between quantum-classical operations are evaluated. However, a key limitation is their inability to address qubit locality, an essential feature for distributed quantum computing (DQC). To overcome this, InQuIR (Nishio and Wakizaka in InQuIR: Intermediate Representation for Interconnected Quantum Computers, 2023. https://arxiv.org/abs/2302.00267) was introduced as an IR specifically designed for distributed systems, providing explicit control over qubit locality and inter-node communication. While effective in managing qubit distribution, InQuIR's dependence on manual manipulation of communication protocols increases complexity for developers. NetQIR (Vázquez-Pérez et al. in NetQIR: An Extension of QIR for Distributed Quantum Computing, 2024. https://arxiv.org/abs/2408.03712), an extension of QIR for DQC, emerges as a solution to achieve the abstraction of quantum communications protocols. This review emphasizes the need for further advancements in IRs for distributed quantum systems, which will play a crucial role in the scalability and usability of future quantum networks.

F. Javier Cardama, Jorge Vázquez-Pérez, César Piñeiro, Juan C. Pichel, Tomás F. Pena and Andrés Gómez these authors contributed equally to this work.

Extended author information available on the last page of the article

Published online: 21 January 2025

Keywords Intermediate representation · Quantum compiling · Distributed quantum computing · Compilers

1 Introduction

The evolution of computing has consistently aimed to abstract hardware complexities to facilitate architecture-agnostic software development. A key milestone in classical computing was the introduction of compilers, which played a central role in translating high-level programming languages into hardware-specific instructions. Without them, every piece of software would have to be tailored to a specific hardware platform, significantly slowing the development and adoption of new technologies. As computing systems became more sophisticated, so did the need for more flexible and efficient compilation processes [6–9].

However, compilers are some of the most complex programs ever developed, often rivaling operating systems in their intricacy. Their development and maintenance require significant resources. To address this, the field of classical computing introduced the concept of intermediate representations (IR) or intermediate languages. These serve as a middle layer between the high-level languages (such as C++ or Python) and the low-level instructions specific to the hardware (such as x86 or ARM instruction sets). The IR allows for a modular compilation process, simplifying the creation of new compilers and facilitating the introduction of new languages and hardware architectures [10–12].

On the other hand, in recent decades, quantum computing has emerged as a groundbreaking field with the potential to revolutionize many areas of science and technology. The development of Shor's algorithm, which promises an exponential speedup in factoring large numbers, shows the immense computational power that quantum systems could bring [13]. This was a direct challenge to classical cryptography, which relies on the difficulty of prime factorization. Since then, the race to develop quantum computers has intensified, with the promise of solving problems currently intractable for classical computers [14–16].

As the field of quantum computing evolves, we are seeing the emergence of new languages, libraries, and frameworks designed to facilitate the programming of quantum hardware and simulators. Like classical computing, quantum computing has to follow a path of abstraction and simplification, building on decades of knowledge of classical compiler design. A key area is the proper development of quantum compilers and the integration of intermediate representations for quantum languages, easing the path toward more accessible quantum programming [17–20].

The need for quantum intermediate representations arises from what has been learned in classical computing, where abstraction has played a critical role in reducing the complexity of compiler development and, by extension, has enabled the proliferation of new languages. Intermediate quantum representations also serve as a bridge between the frontend (high-level quantum languages) and the backend (quantum hardware implementations) and include quantum-specific instructions such as quantum gates, qubit registers, and entanglement operations. These abstractions are essential for the future of quantum computing, as they will enable the development

of more efficient compilers, foster new quantum programming languages, and ultimately make quantum computing more accessible to end users [1, 21].

The main objective of this work is to review the current state of the art related to existing IRs for quantum computing. In order to carry out this review in the most objective and informed manner possible, this paper has the following objectives:

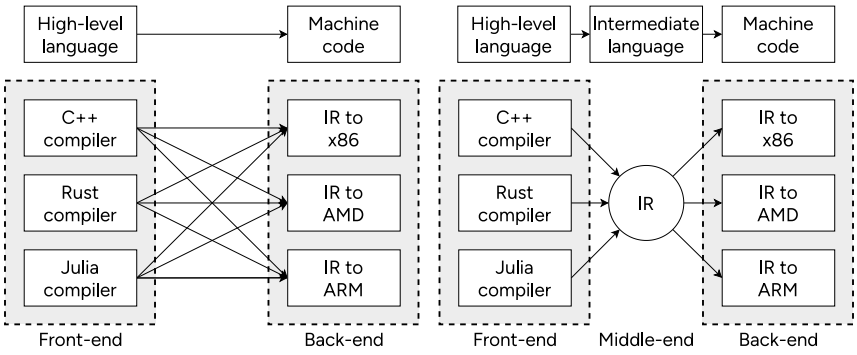
- To review the state of the art of the well-defined characteristics of classical computing IRs.
- To establish a classification of the different existing IRs for quantum computing.
- To define a qualitative comparison of the different representations for quantum computing.
- Further investigate existing representations for distributed quantum computing.

This paper is structured starting with Sect. 2 where we discuss the background or related work of this part of the intermediate representations in the compilation process. Section 3 then deals with the characteristics and subsequent classification of quantum IRs. Section 4 discusses the different representations for the distributed quantum distribution. Finally, Sect. 5 discusses the conclusions and future work drawn through this work.

2 Background

In the early days of computing, programs were written in assembly language, which is highly specific to the underlying hardware architecture. This close coupling between software and hardware severely limited code reusability across different machines. Assembly language, although highly efficient for specific hardware, hindered software portability and presented a problem for developers, who had to manage hardware-specific instructions manually. As a result, the need for higher levels of abstraction in programming languages became evident. Higher-level languages, such as Fortran, COBOL, and later C, emerged to abstract the complexities of hardware-specific details, making programming more efficient and portable [22].

One of the critical innovations that facilitated this shift was the development of compilers. A compiler translates high-level code into machine code, effectively decoupling the programming language from the hardware. This abstraction made it possible to write software once and run it on multiple architectures without modification, fostering code reusability. However, developing a compiler for every programming language and hardware architecture combination was a monumental task. This led to introducing intermediate representations (IR), which bridge the high-level source code and the low-level machine instructions. An IR allows a single high-level language to be translated into an intermediate form, which can then be compiled into machine code for different architectures. This reduced the complexity of compiler design, as developers only needed to target the IR rather than every possible hardware platform directly. The IR allows for a modular compilation process. Figure 1 shows this advantage incorporating an IR into a simple compilation scheme, where the number of compilers required scales from a quadratic number



(a) Front-end to back-end scheme. (b) IR as middle-end in the scheme

Fig. 1 Comparison between integrating IRs into a simple compilation scheme

of $n \cdot m$ (Fig. 1a) to a more manageable linear order of $n + m$ compilers (Fig. 1b), where n is the number of high-level languages and m the number of hardware platforms.

One of the most influential developments in intermediate representations was creating the low-level virtual machine (LLVM) framework [23], initially designed for C and C++ programs. LLVM introduced a flexible, retargetable IR that could be used across multiple hardware platforms, making it a foundational tool in modern compiler design. LLVM abstracts code into a platform-independent form that can later be optimized and translated into architecture-specific machine code. Its modular design has allowed LLVM to support a wide range of languages beyond C++, including Swift, Rust, and Julia, making it a cornerstone in the development of modern compilers [24–26].

However, as computing systems have evolved, the need for more advanced IRs has grown, particularly with the rise of heterogeneous systems. These systems combine different types of processors, such as central processing units (CPUs), graphics processing units (GPUs), and field-programmable gate arrays (FPGAs), each of which excels at different types of computations [27, 28]. In GPUs and FPGAs, where tasks are highly parallelized, traditional IRs had to be adapted for efficient task distribution and execution. OpenCL, CUDA, and other parallel computing platforms introduced specialized IRs to manage the complexity of these devices, ensuring that high-level code could be effectively translated into machine-level instructions capable of running on these specialized architectures.

The multi-level intermediate representation (MLIR) [29] is a significant extension of LLVM, designed to address the complexity of heterogeneous architectures by supporting multiple levels of abstraction. MLIR enables high-level optimizations for domains like machine learning while still providing low-level hardware-specific optimizations. This multi-level approach facilitates parallelism, concurrency, and communication across different devices, making it ideal for handling the needs of heterogeneous systems. This is achieved by implementing the concept of dialects. Each dialect is a specialized language with its own vocabulary—operations and

types—tailored to a specific subject, for instance, as already mentioned, machine learning. Just as you would choose a language that best suits a topic, in MLIR, you choose dialects that best match the domain you are working in.

In addition to MLIR, standard portable intermediate representation (SPIR) [30] is another vital IR in the domain of heterogeneous systems. SPIR was developed for the OpenCL framework to provide a platform-neutral IR that enables code portability across different hardware platforms, including CPUs, GPUs, and FPGAs. SPIR simplifies the compilation of OpenCL kernels into optimized machine code, ensuring performance and compatibility across diverse architectures.

NVVM-IR (NVIDIA virtual machine intermediate representation) plays a similar role for NVIDIA hardware systems. Based on LLVM, NVVM-IR supports CUDA, NVIDIA's parallel computing platform. It abstracts CUDA code into a form that can be optimized and compiled for efficient execution on NVIDIA GPUs, thus improving performance and enabling parallelization on highly specialized hardware.

With the advent of quantum computing, the complexity of hardware has reached an entirely new level, requiring an entirely new class of intermediate representations. Quantum computing platforms, unlike classical ones, are based on principles such as superposition and entanglement, which require fundamentally different instruction sets. Quantum software frameworks such as Qiskit and Cirq were developed to allow high-level quantum programming. These frameworks generate code that can run on quantum hardware, such as superconducting qubits, trapped ions, and photonic systems. However, each of these hardware platforms has unique characteristics, making the development of a unified quantum IR essential for the long-term scalability of quantum programming.

3 Quantum intermediate representations: characteristics and classification

In this section, the characteristics that a standard IR should fulfill are defined by compiling information from different citations in the literature. In the following, the quantum IRs developed in the literature are classified and detailed, and a qualitative comparison of different characteristics important for quantum software is made. Finally, an example code for a quantum teleportation circuit is shown.

3.1 Characteristics of an intermediate representation

An intermediate representation (IR) has to meet specific characteristics that distinguish it from a high-level or machine code language. A fundamental question has to be asked: Why is C language—or any other high-level language— not an intermediate representation?

Different characteristics can be analyzed to define an IR. The most important one we should focus on is that **an IR is created by and for machines**. Therefore, it does not need to be fully readable by the human encoder; in most situations, the IR code is encoded in binary.

First, an IR has to be abstract enough to represent a set of high-level languages (HLLs) rather than just one. This implies that it is at a different level from HLLs and machine codes, so there is a process of compilation, not transpilation.

On the other hand, the compilation process to the IR **must keep the information from the compiler's previous analysis phase**. For example, in quantum computing, it would be a mistake to transpose high-level gates to a set of elementary gates, essentially because hardware particularities such as the supported gate set itself or the error propagated by each gate are not known.

Some of the characteristics of an IR that can be taken into account are the following [31]:

- **Comprehensive representation:** The IR must encapsulate all necessary constructs, abstractions, and concepts from programming languages to ensure precise execution across diverse computing platforms. A key measure of this capability is how easily the IR can be transformed into and from widely used IRs across multiple programming languages.
- **Device independence:** The IR should remain neutral to specific hardware features. Its execution model should reflect the programming language's semantics rather than the underlying hardware, allowing it to be compiled across various devices. This neutrality must be achieved by carefully balancing the abstraction level.
- **Direct programmability:** Like assembly languages, IRs offer programmers the ability to fine-tune their code manually. This is beneficial not only for optimization purposes but also for supporting compiler developers during the construction process. Typically, higher-level IRs make manual programming more straightforward.
- **Forward compatibility:** As programming languages evolve, the IR must be flexible enough to integrate new paradigms without sacrificing backward compatibility. This adaptability is crucial to ensure the IR remains relevant and functional as programming practices change over time.

From the compiler design perspective, the following three attributes are critical for an IR's effectiveness as a program representation tool during the compilation process:

- **Simplicity in design:** The ideal IR should limit the variety of its constructs while still capturing all computations expressible by source languages. This simplicity facilitates the canonicalization process, where source code is standardized before optimization, thereby reducing code variation and easing the compiler's workload.
- **Retention of program details:** The original source code contains the richest information about the program. If critical details are lost during translation, optimization can suffer. Therefore, the IR should include mechanisms to retain important high-level details, such as type information and pointer aliasing, which are essential for effective optimization.

- **Inclusion of analytical data:** Successful program transformations often rely on additional data, such as information on data dependencies and aliasing patterns. Embedding this analytical data in the IR allows it to be used by different parts of the compiler. However, this must be managed carefully to avoid the risk of invalidation by later transformations. Balancing the inclusion of analytical information with the complexity it adds to the IR is a crucial design consideration.

3.2 Intermediate representations for quantum computing

In the compilation process, IR serves as a crucial intermediary between high-level programs and machine-executable instructions, as has already been shown. This representation improves translation efficiency and allows for optimization. When it comes to quantum computing, specialized IR becomes essential to take full advantage of the intrinsic characteristics of quantum computing. A variety of IR languages have been developed to bridge the gap between high-level quantum programming languages and low-level quantum machine code. In this subsection, we will attempt to address the large number of IR languages proposed in the literature. Table 1 shows the classification of the IR of the literature in the following categories:

- **Language extensions:** IRs that extend an intermediate representation of classic computation to add components of quantum features.
- **Standalone languages:** IRs that have been designed for quantum computing without having any basis in another classic IR.
- **IR framework:** An IR that is integrated within a complete compilation scheme or framework and has been created specifically for that scheme.

The initial intermediate language to be examined is **quantum intermediate representation (QIR)** [1]. GIR, developed by the GIR Alliance, which counts

Table 1 Classification of existing IRs in the literature

(a) Language extensions	
Name	**Language**
Q-MLIR [3]	MLIR [29]
QIR [1]	LLVM [23]
QSSA [2]	SSA [32]
QIRO [33]	MLIR and SSA
(b) Standalone	
SQIR [34]	
(c) IR Framework	
Name	**Framework**
XACC IR [19]	XACC
QBIR [35]	Yao
tlket⟩ IR [36]	tlket⟩

Microsoft among its members, should be distinguished from the broader concept of intermediate representation (qIR) previously outlined. It functions as a universal interface between quantum programming languages or frameworks and various quantum computing platforms. Moreover, it delineates a series of protocols for representing quantum programs in a language and hardware-neutral format within the LLVM IR [23]. Concerning the translation from high-level languages, GIR remains non-specific to any particular quantum programming framework, thereby facilitating its adoption for articulating quantum programs. Conversely, regarding the translation of IR to machine-specific instructions, GIR is designed to be hardware-independent, abstaining from prescribing a specific quantum instruction or gate set and instead deferring to the preferences of the target computation environment.

Moving on to other quantum IRs, Q-MLIR, an extension of quantum computing to the IR multi-level intermediate representation (MLIR) described in Sect. 2, is introduced in [3], highlighting the potential of this dialect to conform to the GIR standards recently proposed by Microsoft. This facilitates a shared optimization and execution generation framework across multiple source languages.

Furthermore, additional quantum computing oriented extensions of MLIR, such as **quantum intermediate representation for optimization (QIRO)** detailed in [33], are noteworthy. The QIRO framework is tailored for quantum-classical co-optimization and embeds data flow directly within the IR, enabling a range of optimizations through data flow analysis. It comprises two dialects: one for input and another for optimization. In contrast, Q-MLIR focuses on defining a quantum IR by extending MLIR but does not explore or implement optimizations that leverage MLIR's capabilities for quantum program improvement.

Moreover, IRs can adhere to specific properties or constraints to facilitate optimization and verification processes. One such property, extensively explored in classical compilation literature, is the static single assignment (SSA) form [32, 37, 38]. An SSA form mandates that each variable in the source code is assigned uniquely in the intermediate representation. This implies that additional variables must be introduced if a variable receives assignments at multiple points in the source code to denote the distinct versions required.

Figure 2 exemplifies the LLVM IR translation of C++ source code where the variable `a` undergoes reassignment. In an SSA compliant code, it is imperative to generate a new variable (`%3` as shown in Fig. 2c) to signify the updated version, thereby preserving the original variable (`%1` as depicted in Fig. 2b).

```
int a = 10;
int b = a + 5;
a = a * 2;
int c = a * b;
return c;
```

(a) Example of C++ source code.

```
%1 = 10
%2 = %1 + 5
%1 = %1 * 2
%4 = %1 * %2
return %4
```

(b) Representation in LLVM IR without SSA.

```
%1 = 10
%2 = %1 + 5
%3 = %1 * 2
%4 = %3 * %2
return %4
```

(c) Representation in LLVM IR with SSA.

Fig. 2 Comparison between an LLVM IR code with and without SSA

The usefulness of this property in verification and optimization, together with the extensive study in classic compilation, allows the use of this approach in quantum compilation as a suitable and perfectly tested basis. The IR previously discussed, QIRO, uses this SSA property for the co-optimization of quantum codes.

On the other hand, the work presented in [2] as **quantum static single assignment (QSSA)** performs an extension of the SSA properties to generate a quantum IR that performs a special emphasis on the verification in the compile-time of the physical constraints of quantum nature, such as the no-cloning theorem.

Figure 3 shows a graphical representation of the various quantum IRs that extend a classic representation such as LLVM. It is shown in a set format to characterize those representations that extend more than one language or feature.

Outside the classic IR extensions such as LLVM or MLIR, there are also standalone IR languages like **small quantum intermediate representation (SQIR)**, proposed in [34]. Its primary purpose is to serve as an intermediate language to verify the correctness of the quantum circuit once optimizations have been applied to it. This IR is built on top of the mathematical definition language Coq.

In addition, some frameworks implement the complete compilation scheme or a high-level language that develops its specific IR, such as eXtreme-scale Accelerator programming framework (XACC) [19] that implement the compilation scheme in full, based on a three-level scheme: a frontend that maps the quantum code to its own IR; a middle-end related to the transformation and optimization of the IR; and, finally, a backend that processes the IR to the specific machine code. Another example is the quantum block intermediate representation (QBIR) [35], which is used in the high-level quantum language Yao.jl. The main novelty of QBIR is that it allows a quantum circuit to be specified using "blocks" of nested tree operations.

3.3 Comparison of quantum IRs

The purpose of this section is to provide a comparative analysis of the different IRs discussed in this work. It is important to emphasize that this is a qualitative comparison to maintain objectivity, as a quantitative study is not viable due to the lack of standardized benchmarks and the different levels of maturity of the different IRs. To achieve this, a set of key properties has been established to evaluate how each IR performs in different contexts. This comparison aims not to declare one IR superior to another but to provide insights that will help readers decide which IR is best suited to their particular implementation priorities. This framework allows readers to

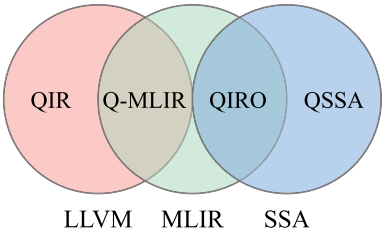


Fig. 3 Venn diagram with the different quantum IRs shown in the "Language extensions" section. Each set represents a language or feature of classic computing, and each element is a quantum IR

make informed decisions based on their specific needs, whether hardware compatibility, optimization efficiency, or support for hybrid quantum-classical operations.

Table 2 presents a qualitative comparison of seven quantum IRs: QIR, SQIR, QSSA, QIRO, XACC, QBIR, and Q-MLIR, based on the next essential features for quantum software development:

1. **Hybrid quantum-classical operations:** Quantum programs often combine classical and quantum computations, especially for hybrid algorithms like the variational quantum eigensolver (VQE). An IR that supports hybrid operations allows efficient execution of both quantum instructions and classical control logic (e.g., conditionals, loops).
2. **Hardware agnosticism:** An IR hardware agnosticism determines how well it abstracts away hardware-specific details, making code portable across various quantum architectures (e.g., superconducting qubits, trapped ions, photonic systems).
3. **Optimizations and compiler passes:** Quantum computations require significant optimizations (e.g., gate reduction, circuit simplifications) to be executable on quantum hardware, where resource constraints like qubit coherence times and gate fidelity are critical.
4. **Expressiveness:** refers to how flexibly the IR can describe quantum programs, supporting modularity, advanced quantum gates, and complex quantum operations.
5. **Simulator compatibility:** The ability to simulate quantum circuits before executing them on actual hardware is a crucial step in quantum program development.
6. **Community support:** IRs community support is critical for its development and adoption. A strong ecosystem ensures access to tools, libraries, and active collaboration.
7. **Ease of use and accessibility:** evaluates how approachable the IR is for developers, particularly those with limited experience in quantum computing. This includes factors such as comprehensive documentation, availability of debugging tools, seamless integration with classical programming languages, and the presence of user-friendly graphical interfaces.

In the context of Table 2, qualifiers such as high or medium are used to provide a qualitative assessment of how well an IR matches the characteristic being assessed. For example, in the case of expressiveness, most IRs are rated high due to their ability to represent a wide range of quantum operations. However, Q-MLIR is rated very high because its multi-level structure allows for the definition of new instructions, providing greater flexibility and extensibility compared to other IRs. With regard to the simulator compatibility section, the medium qualifier is assigned to those IRs that cannot be directly applied by existing simulators.

In the case of ease of use, to obtain a “High” it is necessary that IR information is published as well as execution and debugging tools in a public code repository. In the case of “Medium” these tools are out of date, while “Low” means that only the publication paper has been found.

Table 2 Qualitative comparison of the different IRs existing in the bibliography

Feature	QIR	Q-MLIR	QSSA	QIRO	XACC	QBIR	SQIR
Hybrid quantum-classical ops	Yes (via LLVM)	Yes	Yes	Yes	Yes	No	No
Hardware agnosticism	High	High	High	High	High (via XACC compiler)	High	High (Coq-based)
Optimizations	Built-in (LLVM passes)	Built-in (via MLIR)	Compiler optimizations	Limited	LLVM Optimizations	Circuit Simplifications	Formal Verification
Expressiveness	High	Very High	High	High	Very High (via MLIR)	Medium	Medium
Simulator compatibility	High (LLVM simulators)	High	Medium	High	High	Medium	High
Community support	Strong (LLVM, Microsoft)	Growing (MLIR community)	Growing (Quantum research)	Limited	Strong (XACC framework)	Limited	Growing (Coq community)
Ease of use and accessibility	High	Low	Low	Medium	High	High	High

Each quantum IR brings unique strengths to quantum software development. QIR and Q-MLIR perform particularly well in handling hybrid quantum-classical operations, optimizations, and hardware agnosticism, which makes them an excellent fit for scalable quantum computing applications. SQIR excels in formal verification, while XACC offers solid support across the software stack and full compilation. QSSA and QIRO emphasize optimizations and error correction, which are fundamental to long-term fault-tolerant quantum computing.

By comparing these features, developers can choose the most appropriate IR based on their specific needs, such as hardware compatibility, optimization, and expressiveness.

3.4 Example of code: teleportation circuit

In this subsection, two IRs that extend LLVM, QIR, and Q-MLIR will be compared to analyze the differences between the two models. We aim to demonstrate how different IRs work when implementing a complex circuit, such as quantum teleportation, where several critical aspects are involved, such as quantum gates between qubit pairs, intermediate measurement, or physically separated qubits.

The circuit to be implemented is the one shown in Fig. 4, corresponding to the well-known teleportation circuit between two physically separated qubits. The codes generated for this circuit are shown in Fig. 5a for QIR and Fig. 5b for Q-MLIR.

The QIR code for the teleportation circuit is very similar to the LLVM IR format. It focuses on low-level quantum operations and directly invokes quantum gates like Hadamard and CNOT with explicit quantum instruction calls. Measurements are performed with conditions applied to control subsequent gates (X and Z corrections), reflecting the hybrid nature of QIR, which integrates quantum operations with classical control through conditional branches. This low-level representation ensures detailed control and optimization but requires more explicit operations management.

In contrast, the Q-MLIR code operates at a higher level of abstraction, using MLIR's modular structure. While the circuit structure is similar—applying Hadamard and CNOT gates followed by measurements—the code is more declarative. The quantum gates and measurements are invoked in a more abstract manner, emphasizing flexibility and modularity rather than fine-tuned control. Q-MLIR simplifies the representation, making it easier to map to different

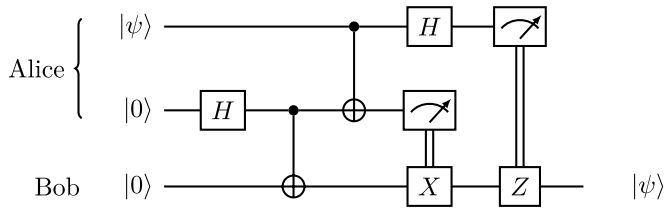


Fig. 4 Circuit for teleporting a quantum state $|\psi\rangle$ between two physically separated qubits (from Alice to Bob)

```
define void @Teleportation__body(%Qubit* %msg, %Qubit* %ent1, %Qubit* %ent2) {
entry:
  ; Create entanglement between ent1 and ent2
  call void @__quantum__qis__h(%Qubit* %ent1)
  call void @__quantum__qis__cnot(%Qubit* %ent1, %Qubit* %ent2)

  ; Apply CNOT between msg and ent1 and next the Hadamard gate to msg
  call void @__quantum__qis__cnot(%Qubit* %msg, %Qubit* %ent1)
  call void @__quantum__qis__h(%Qubit* %msg)

  ; Measure msg and ent1
  %meas_msg = call %Result* @__quantum__qis__mz(%Qubit* %msg)
  %meas_ent1 = call %Result* @__quantum__qis__mz(%Qubit* %ent1)

  ; Apply the classical controlled gates
  call void @__quantum__qis__x(%Qubit* %ent2) if (%meas_ent1 == 1)
  call void @__quantum__qis__z(%Qubit* %ent2) if (%meas_msg == 1)

  ret void
}
```

(a) QIR code

```
module {
  func @Teleportation(%msg: !quantum.qbit, %ent1: !quantum.qbit, %ent2: !quantum.qbit) {
    // Entanglement creation
    "quantum.h"(%ent1) : (!quantum.qbit) -> ()
    "quantum.cnot"(%ent1, %ent2) : (!quantum.qbit, !quantum.qbit) -> ()

    // Teleportation step
    "quantum.cnot"(%msg, %ent1) : (!quantum.qbit, !quantum.qbit) -> ()
    "quantum.h"(%msg) : (!quantum.qbit) -> ()

    %m1 = "quantum.mz"(%msg) : (!quantum.qbit) -> !quantum.result
    %m2 = "quantum.mz"(%ent1) : (!quantum.qbit) -> !quantum.result

    "quantum.x"(%ent2) if %m2 == 1
    "quantum.z"(%ent2) if %m1 == 1
    return
  }
}
```

(b) Q-MLIR code

Fig. 5 Implementation of a teleportation circuit in two quantum intermediate representations (QIR and Q-MLIR)

hardware backends or domain-specific optimizations while maintaining a clearer structure for higher-level quantum algorithms.

These examples show that both QIR and Q-MLIR effectively handle intermediate measurements (hybrid quantum-classical operations) and quantum gates between qubit pairs. However, the major unresolved challenge lies in communication between physically separated qubits. The current IRs do not specify whether qubits are located on different nodes, a crucial factor in distributed quantum computing (DQC). Without addressing this issue, managing the necessary quantum communication and synchronization between distant quantum processors becomes impossible.

To address this limitation, the following section will explore various intermediate languages specifically designed for DQC, which incorporate mechanisms to handle physically separated qubits and quantum communication.

4 IRs for distributed quantum computing (DQC)

In this section, we describe IRs for programming distributed quantum computers, trying to overcome the shortcoming of all quantum IRs presented in the previous section: the inability to define physically separated qubits. In DQC, where quantum systems are distributed over multiple physical locations, it becomes crucial to specify the locality of each qubit and to manage the communication between qubits located in different nodes.

In distributed quantum systems, operations like quantum teleportation and entanglement swapping require an IR that can effectively model both the quantum gates and the communication between distant qubits [39]. In the context of DQC, two intermediate representations (IRs) have emerged to address the challenges posed by qubit locality and inter-node communication: InQuIR [4] and NetQIR [5]. These IRs aim to overcome the limitations of monolithic quantum systems discussed in the previous section, where qubits are assumed to exist within a single physical location.

InQuIR (intermediate representation for interconnected quantum computers) introduces a mechanism for explicitly specifying the location of qubits across distributed nodes. Unlike conventional IRs, InQuIR enables fine-grained control of communication between quantum processors by allowing developers to manually manage entanglement generation and classical communication protocols. This explicit control is achieved through primitives such as:

- `genEnt()` for entanglement generation between nodes,
- `send()` and `recv()` for classical communication of measurement results.

While InQuIR provides a significant step toward enabling distributed quantum programs, its reliance on manual management increases programming complexity and risks introducing errors, especially as the number of nodes grows.

As an illustrative example, consider the teleporting circuit, where the goal is to transfer the state of a qubit from one node to another. In a distributed quantum system, this involves entangling two qubits at different locations and using classical communication to transfer the qubit's state. The InQuIR code is shown in Fig. 6, where it can be seen that there is a different code for each qubit node (Alice

```
0 {
  world = open[0,1];
  q0 = init();
  _cq0 = genEnt[1](10);
  CX q0 _cq0; H q0;
  _m0 = measure _cq0;
  _m1 = measure q0;
  free _cq0; free q0;

  send[1](world, 11:_m0);
  send[1](world, 11_2:_m1);
}
```

(a) InQuIR code, `0` node (Alice).

```
1 {
  world = open[0];

  q1 = init();
  _cq1 = genEnt[0](10);

  recv(world, 11:_m3);
  recv(world, 11:_m4);

  X[_m3] _cq1;
  Z[_m4] _cq1;
}
```

(b) InQuIR code, `1` node (Bob).

Fig. 6 InQuIR code to implement the teleport circuit in the different nodes

or Bob). Therefore, this IR allows to specify the physical separation of a register of qubits and, additionally, to program them in a different way depending on this characteristic.

InQuIR provides important tools for managing qubit locality and inter-node communication in distributed quantum computing. However, its limitations arise from its dependence on manual control over communication processes, which restricts its ability to fully abstract the complexities inherent in quantum networking. In particular, InQuIR's reliance on explicit entanglement generation (`genEnt`) and manual control over the transmission of classical data (`send` and `recv`) requires developers to work directly with the low-level details of the quantum communication process. This lack of abstraction contrasts strongly with the practice in classical distributed computing, where frameworks like MPI effectively hide the details of the underlying communication protocols.

The abstraction of the network layer in traditional distributed systems provides flexibility to work with different network architectures and protocols without exposing their complexity to the user, which allows developers to focus on high-level logic without managing the intricacies of the interconnection network. However, InQuIR's dependence on explicitly managing communication through teleportation or entanglement protocols requires developers to handle the complexities of quantum networking, which can increase the risk of errors and limit scalability. As quantum networks grow in complexity, with more nodes and greater distances between them, this approach becomes less feasible and more cumbersome. The need for a more abstract and flexible system becomes evident, one that can manage communication efficiently and transparently.

To address these issues, NetQIR [5] emerges as an extension of GIR [1] and extends it to support distributed quantum systems. NetQIR addresses the limitations of InQuIR by introducing higher-level abstractions that automate the management of quantum communication. Key features include:

- `qsend()` for transmitting qubits between nodes,
- `expose()` for making qubits accessible to other nodes.

These instructions abstract the communication process, allowing the system to automatically manage the complexities of quantum entanglement, teleportation, and classical communication. This approach mirrors the abstraction seen in classical distributed computing frameworks, where message-passing or data synchronization details are hidden from the user. By integrating such abstraction, NetQIR not only simplifies the development process but also increases the flexibility and scalability of distributed quantum systems.

Figure 7 shows the same teleport circuit discussed throughout this article, highlighting the reduction in code complexity achieved by abstracting the communication protocol. The implementation becomes more agile and efficient by delegating these low-level details to the backend, which has access to more specific information about the connecting network and the hardware involved.

A significant advantage of NetQIR is that, by extending QIR, it inherits a strong foundation from the QIR framework, which itself is built on LLVM. This allows

```
define void @main(i32 noundef %0, ptr noundef %1) #0 {
    entry:
        ; Variable allocation
        %2 = alloca i32, align 4

        ; Init the NetQIR communication and get the rank of the process
        %3 = call i32 @__netqir__init(i32 noundef %0, ptr noundef %1)
        %4 = call i32 @__netqir__comm_rank(%Comm* @netqir_comm_world, ptr %2)

        ; Choose if it is the process receiving or sending the qubit
        %5 = load i32, ptr %2, align 4
        %6 = icmp eq i32 %5, 0
        br i1 %6, label %7, label %9

    ; Process sending
    7:
        %8 = call i32 @__netqir__qsend(%Qubit* null, i32 noundef 1,
                                       %Comm* @netqir_comm_world)
    ; Process receiving
    9:
        %10 = call i32 @__netqir__qrecv(%Qubit** null, i32 noundef 0,
                                        %Comm* @netqir_comm_world)

        ; End of the program
        %11 = call i32 @__netqir__finalize()
}

; Function declaration
declare i32 @__netqir__init()
declare i32 @__netqir__qsend(%Qubit*, i32, %Comm)
declare i32 @__netqir__qrecv(%Qubit**, i32, %Comm)
declare void @__netqir__finalize()
```

Fig. 7 NetQIR code for the state teleportation between Alice and Bob

NetQIR to leverage the extensive optimization and tooling infrastructure developed for QIR and LLVM, including compiler passes, simulation tools, and debugging environments. As a result, developers can utilize high-level abstractions for distributed quantum operations and benefit from the robust ecosystem of tools available in the LLVM framework. On the other hand, one of the main disadvantages of NetQIR is that it is currently still in a work-in-progress phase. While the IR has been defined conceptually, there is a significant gap in the availability of practical tools and infrastructure for working with it. Unlike established quantum IRs such as QIR, which benefit from a mature ecosystem of compilers, optimizers, and simulators, NetQIR lacks concrete implementations and tools at this stage. As a result, developers and researchers are not yet able to fully utilize NetQIR in real-world distributed quantum systems.

In summary, Table 3 compares the characteristics discussed in this section. Both InQuIR and NetQIR incorporate instructions for quantum communication—InQuIR with `genEnt` and `send`, and NetQIR with `qsend`. However, InQuIR lacks abstraction from the network layer and communication protocols. By relying on the

Table 3 Comparison between the IRs for distributed quantum computing analyzed

Intermediate representation	Instructions for quantum comm.	Abstraction of network layer	Abstraction of protocols
InQuIR	✓	✗	✗
NetQIR	✓	✓	✓

`genEnt()` function, developers are required to implement their own teledata or telegate protocols, with no built-in consideration for network connectivity or abstraction from the underlying communication layer.

5 Conclusions

This work explored the design, evaluation, and limitations of intermediate representations (IRs) for monolithic and distributed quantum computing. To develop an effective IR, certain core characteristics must be defined, such as hardware agnosticism, optimization capabilities, modularity, and the ability to handle quantum-classical hybrid operations. These features ensure that an IR can bridge the gap between high-level quantum programming languages and the hardware, ensuring efficient performance and adaptability across various quantum architectures. This paper began by reviewing quantum IRs designed for monolithic quantum systems, followed by a more in-depth discussion of the challenges associated with IRs for distributed quantum computing.

The study of monolithic quantum computing focused on IRs like QIR, Q-MLIR, SQIR, or QSSA, which are primarily designed for quantum processors located in a single physical location. These IRs support features such as gate-level abstractions, hardware-agnostic instruction sets, and optimizations targeting performance across various quantum platforms. They are well-suited for tasks that do not involve inter-node communication or distributed architectures. However, they still face challenges in scaling with increasing qubit counts and handling the specificities of different quantum technologies.

The evolution of quantum IRs reflects the need for such tools to facilitate the development of new compilers and a complete quantum software stack. This work highlights the current limitations of quantum IRs, particularly the lack of support for managing qubit locality in systems where quantum processors are distributed across multiple physical locations. Therefore, IRs for DQC, such as InQuIR or NetQIR, have emerged in the literature.

InQuIR takes an important step by allowing programmers to define qubit locality and manage communication explicitly, but this approach has significant drawbacks. Namely, it requires developers to handle low-level communication protocols (e.g., `genEnt`, `send`, `recv`) manually, adding to the complexity and potential for error. In contrast, NetQIR introduces a higher level of abstraction, where instructions like `qsend` and `expose` delegate the details of quantum communication to the backend. This allows for a more flexible and scalable programming model, similar to how classical distributed systems abstract network protocols through frameworks like MPI.

The key strength of NetQIR is its foundation in QIR, which is itself built on LLVM. This allows it to take advantage of LLVM's well-established ecosystem of optimization tools, compilers, and simulators. This strong base ensures that NetQIR not only simplifies the programming model for distributed quantum computing but also benefits from powerful optimizations and debugging capabilities developed for classical computing.

In summary, while InQuIR provides an initial solution for managing distributed qubit locality and communication, its explicit handling of communication protocols limits its scalability. NetQIR, as an extension of QIR, offers a more abstract, scalable solution for distributed quantum systems. By integrating high-level instructions that hide the complexity of quantum communication, NetQIR provides a pathway to more efficient and manageable distributed quantum computing frameworks. However, it is important to note that NetQIR remains in a **work-in-progress phase**, with the intermediate representation defined but without practical tools available yet for real-world implementations. Further development is required to fully realize its potential in distributed quantum systems.

Finally, the development of IRs is essential to bridge the gap between high-level quantum programming languages and diverse quantum hardware. A standardized IR would ensure portability, allowing quantum programs to run seamlessly across different platforms, while enabling critical optimizations such as gate minimization and error mitigation. It would also facilitate the integration of hybrid quantum-classical workflows and promote a unified ecosystem of tools such as simulators, compilers, and debuggers.

Regarding future prospects, a common IR will be key to achieving scalable DQC by addressing the challenges of qubit locality and communication. Standardization efforts will need to balance hardware abstraction and performance. By mirroring the success of classical IRs such as LLVM, a well-defined quantum IR will accelerate the adoption and development of quantum technologies, paving the way for practical and efficient quantum systems. It is crucially important to have the necessary compilation tools and optimal IR for when quantum computing moves beyond gate specification codes to a high-level programming architecture such as currently exists in classical computing.

Acknowledgements This work was supported by MICINN through the European Union NextGenerationEU recovery plan (PRTR-C17.I1), and by the Galician Regional Government through the "Planes Complementarios de I+D+I con las Comunidades Autónomas" in Quantum Communication. This work was also supported by financial support from the Agencia Estatal de Investigación (Spain) (PID2022-141623NB-I00), the Xunta de Galicia - Consellerí a de Cultura, Educación, Formación Profesional e Universidades (Centro de investigación de Galicia accreditation 2024-2027 ED431G-2023/04 and Reference Competitive Group accreditation ED431C-2022/016), and the European Union (European Regional Development Fund - ERDF).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Lubinski T, Granade C, Anderson A, Geller A, Roetteler M, Petrenko A, Heim B (2022) Advancing hybrid quantum-classical computation with real-time execution. Front Phys. https://doi.org/10.3389/fphy.2022.940293
2. Peduri A, Bhat S, Grosser T (2022) QSSA: an SSA-based IR for quantum computing. In: Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. CC 2022. Association for Computing Machinery, New York, pp. 2–14. https://doi.org/10.1145/3497776.3517772
3. McCaskey A, Nguyen T (2021) A MLIR dialect for quantum assembly languages. In: Proceedings—2021 IEEE International Conference on Quantum Computing and Engineering, QCE 2021, pp 255–264. https://doi.org/10.1109/QCE52317.2021.00043
4. Nishio S, Wakizaka R (2023) InQuIR: Intermediate Representation for Interconnected Quantum Computers. https://arxiv.org/abs/2302.00267
5. Vázquez-Pérez J, Cardama FJ, Piñeiro C, Pena TF, Pichel JC, Gómez A (2024) NetQIR: An Extension of QIR for Distributed Quantum Computing. https://arxiv.org/abs/2408.03712
6. Alfred VA, Monica SL, Jeffrey DU (2007) Compilers principles. Techniques & Tools. Pearson Education Inc, London
7. Schneck PB (1973) A survey of compiler optimization techniques. In: Proceedings of the ACM Annual Conference, pp 106–113
8. Tremblay J-P, Sorenson PG (1985) Theory and practice of compiler writing. McGraw-Hill Inc, New York
9. Torczon L, Cooper K (2007) Engineering a compiler. Morgan Kaufmann Publishers Inc., Cambridge
10. Conway ME (1958) Proposal for an UNCOL. Commun ACM 1(10):5–8. https://doi.org/10.1145/368924.368928
11. Ottenstein KJ (1984) Intermediate program representations in compiler construction: a supplemental bibliography. SIGPLAN Not 19(7):25–27. https://doi.org/10.1145/988574.988579
12. Stanier J, Watson D (2013) Intermediate representations in imperative compilers: a survey. ACM Comput Surv 45(3):1–27. https://doi.org/10.1145/2480741.2480743
13. Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 26(5):1484–1509. https://doi.org/10.1137/s0097539795293172
14. Steane A (1998) Quantum computing. Rep Prog Phys 61(2):117. https://doi.org/10.1088/0034-4885/61/2/002
15. Ladd TD, Jelezko F, Laflamme R, Nakamura Y, Monroe C, O'Brien JL (2010) Quantum computers. Nature 464(7285):45–53. https://doi.org/10.1038/nature08812
16. Nielsen MA, Chuang IL (2010) Quantum computation and quantum information: 10th, Anniversary. Cambridge University Press
17. Chong FT, Franklin D, Martonosi M (2017) Programming languages and compiler design for realistic quantum hardware. Nat Publ Group. https://doi.org/10.1038/nature23459
18. Häner T, Steiger DS, Svore K, Troyer M (2018) A software methodology for compiling quantum programs. Ins Phys Publ. https://doi.org/10.1088/2058-9565/aaa5cc
19. McCaskey AJ, Lyakh DI, Dumitrescu EF, Powers SS, Humble TS (2020) XACC: a system-level software infrastructure for heterogeneous quantum-classical computing. Quantum Sci Technol 5(2):024002. https://doi.org/10.1088/2058-9565/ab6bf6
20. Mintz TM, McCaskey AJ, Dumitrescu EF, Moore SV, Powers S, Lougovski P (2020) QCOR: a language extension specification for the heterogeneous quantum-classical model of computation. J Emerg Technol Comput Syst 16(2):1–7. https://doi.org/10.1145/3380964
21. McCaskey AJ, Dumitrescu EF, Liakh D, Chen M, Feng W, Humble TS (2018) A language and hardware independent approach to quantum-classical computing. SoftwareX 7:245–254. https://doi.org/10.1016/j.softx.2018.07.007
22. Knuth DE, Pardo LT (1980) The early development of programming languages. In: A History of Computing in the Twentieth Century, pp 197–273. https://doi.org/10.1016/B978-0-12-491650-0.50019-8
23. Foundation L (2009) LLVM Assembly Language Reference Manual. https://releases.llvm.org/2.6/docs/LangRef.html

24. Lattner C, Adve V (2004) LLVM: a compilation framework for lifelong program analysis & transformation. In: International Symposium on Code Generation and Optimization, 2004. CGO 2004, pp 75–86. https://doi.org/10.1109/CGO.2004.1281665
25. Terei DA, Chakravarty MM (2009) Low level virtual machine for Glasgow Haskell compiler. PhD Thesis, Bachelor's Thesis, Computer Science and Engineering Dept., The University of New South Wales
26. Ştirb I, Gillich GR (2023) A low-level virtual machine just-in-time prototype for running an energy-saving hardware-aware mapping algorithm on C/C++ applications that use pthreads. Energies 16(19):6781. https://doi.org/10.3390/en16196781
27. Weaver G (1995) Compiler representations for heterogeneous processing. Technical Report UM-CS-1995-102, University of Massachusetts
28. Belwal M, TSB, S (2015) Intermediate representation for heterogeneous multi-core: A survey. In: 2015 International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA), pp 1–6. https://doi.org/10.1109/VLSI-SATA.2015.7050496
29. Lattner C, Amini M, Bondhugula U, Cohen A, Davis A, Pienaar J, Riddle R, Shpeisman T, Vasilache N, Zinenko O (2021) MLIR: scaling compiler infrastructure for domain specific computation. In: CGO 2021—Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization, pp 2–14. https://doi.org/10.1109/CGO51591.2021.9370308
30. Kessenich J, Ouriel B, Krisch R (2018) SPIR-V specification. Khronos Group 3:17
31. Chow F (2013) Intermediate representation. Queue 11:30–37. https://doi.org/10.1145/2542661.2544374
32. Cytron R, Ferrante J, Rosen BK, Wegman MN, Zadeck FK (1989) An efficient method of computing static single assignment form. In: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '89. Association for Computing Machinery, New York, pp 25–35. https://doi.org/10.1145/75277.75280
33. Ittah D, Häner T, Kliuchnikov V, Hoefler T (2022) QIRO: a static single assignment-based quantum program representation for optimization. ACM Trans Quantum Comput 3(3):1–32. https://doi.org/10.1145/3491247
34. Hietala K, Rand R, Hung SH, Wu X, Hicks M (2019) Verified Optimization in a Quantum Intermediate Representation. https://arxiv.org/abs/1904.06319
35. Luo XZ, Liu J-G, Zhang P, Wang L (2020) Yao. jl: extensible, efficient framework for quantum algorithm design. Quantum 4:341. https://doi.org/10.22331/q-2020-10-11-341
36. Sivarajah S, Dilkes S, Cowtan A, Simmons W, Edgington A, Duncan R (2020) t|ket⟩: a retargetable compiler for NISQ devices. Quantum Sci Technol 6(1):014003. https://doi.org/10.1088/2058-9565/ab8e92
37. Van Emmerik MJ (2007) Static single assignment for decompilation. PhD Thesis, University of Queensland, Queensland, Australia
38. Rastello F, Tichadou FB (2022) SSA-based compiler design. Springer International Publishing, pp 1–382. https://doi.org/10.1007/978-3-030-80515-9
39. Barral D, Cardama FJ, Díaz G, Faílde D, Llovo IF, Juane MM, Vázquez-Pérez J, Villasuso J, Piñeiro C, Costas N, et al (2024) Review of Distributed Quantum Computing. From single QPU to high performance quantum computing. https://arxiv.org/abs/24004.01265

Authors and Affiliations

F. Javier Cardama[1] · Jorge Vázquez-Pérez[1] · César Piñeiro[1,2] · Juan C. Pichel[1,2] · Tomás F. Pena[1,2] · Andrés Gómez[3]

✉ F. Javier Cardama
javier.cardama@usc.es

Jorge Vázquez-Pérez
jorgevazquez.perez@usc.es

César Piñeiro
cesaralfredo.pineiro@usc.es

Juan C. Pichel
juancarlos.pichel@usc.es

Tomás F. Pena
tf.pena@usc.es

Andrés Gómez
andres.gomez.tato@cesga.es

[1] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Galicia, Spain

[2] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, 15705 Santiago de Compostela, Galicia, Spain

[3] Galicia Supercomputing Center (CESGA), 15705 Santiago de Compostela, Galicia, Spain

Chapter 7

NetQIR: an extension of Quantum Intermediate Representation for distributed quantum computing

The core content of this chapter is from the research presented in the following peer-reviewed article:

> F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, T. F. Pena, J. C. Pichel, and A. Gómez, "NetQIR: An extension of QIR for distributed quantum computing", *Future Generation Computer Systems*, vol. 174, p. 107 989, 2026, ISSN: 0167-739X. DOI: 10.1016/j.future.2025.107989
>
> - Impact Summary: Article published in a JCR 2024 first quartile (Q1) journal.

This publication specifically addresses and fulfills Objective O3 of this thesis. For a comprehensive overview of the publication's quality indicators (Impact Factor and Quartile), the reader is referred to the Results: published articles from Contributions section.

Future Generation Computer Systems 174 (2026) 107989

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

NetQIR: An extension of QIR for distributed quantum computing

F. Javier Cardama [a,*], Jorge Vázquez-Pérez [a,c], César Piñeiro [a,b], Tomás F. Pena [a,b], Juan C. Pichel [a,b], Andrés Gómez [c]

[a] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain
[b] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain
[c] Galicia Supercomputing Center (CESGA), Avda. de Vigo S/N, Santiago de Compostela, 15705, Spain

ARTICLE INFO

Keywords:
Distributed quantum computing
Quantum intermediate representation
Quantum internet
Compilers
Teledata
Telegate
Distributed quantum applications

ABSTRACT

The rapid advancement of quantum computing has highlighted the need for scalable and efficient software infrastructures to fully exploit its potential. Current quantum processors face significant scalability constraints due to the limited number of qubits per chip. In response, distributed quantum computing (DQC) — achieved by networking multiple quantum processor units (QPUs)— is emerging as a promising solution. To support this paradigm, robust intermediate representations (IRs) are needed to translate high-level quantum algorithms into executable instructions suitable for distributed systems. This paper presents NetQIR, an extension of Microsoft's Quantum Intermediate Representation (QIR), specifically designed to facilitate DQC by incorporating new instruction specifications. NetQIR was developed in response to the lack of abstraction at the network and hardware layers identified in the existing literature as a significant obstacle to effectively implementing distributed quantum algorithms. Based on this analysis, NetQIR introduces new essential abstraction features to support compilers in DQC contexts. It defines network communication instructions independent of specific hardware, abstracting the complexities of inter-QPU communication. Although the proposed work allows abstraction of the underlying network, it is important to note that it is intended for the development of high-performance code on future modular quantum architectures. Leveraging the QIR framework, NetQIR aims to bridge the gap between high-level quantum algorithm design and low-level hardware execution, thus promoting modular and scalable approaches to quantum software infrastructures for distributed applications. Furthermore, its design may serve as a foundational component for future implementations of distributed quantum standards such as the Quantum Message Passing Interface (QMPI).

1. Introduction

The evolution of computing has progressed from simple mechanical calculators to modern-day classical computers, that have significantly transformed numerous fields, including science, engineering, and everyday life. Despite these advances, classical computers face limitations in solving certain complex problems efficiently, such as factoring large numbers, simulating quantum systems, or optimizing large-scale systems [1,2]. This has led to the emergence of quantum computing, which leverages the principles of quantum mechanics to process information in fundamentally new ways, offering the potential to solve these intractable problems more efficiently than classical computers can achieve [3,4].

Over the last few years, the development of a comprehensive software stack for quantum computing has gained importance in allowing the programming of quantum devices in a scalable and easy way. This software stack includes quantum high-level languages, compilers, and runtime environments designed to enable the programming and execution of quantum algorithms on quantum devices [5,6]. High-level quantum programming languages such as Q# [7], Quipper [8], or Qiskit [9] facilitate the development of quantum algorithms by abstracting the complexities of quantum hardware [10].

For the efficient execution of these algorithms, quantum code compilers play a crucial role. A compiler is a software program that translates high-level languages into low-level instructions that quantum processors can execute [11]. In classical computing, the concept of IR was introduced as an abstract-machine code to facilitate the development of new compilers [12]. This concept was extended in the world of quantum computing to allow a common IR as an intermediate step between high-level and back-end languages. The main objective of using an IR is to facilitate the optimization of quantum codes and, simultaneously, to ensure their compatibility with different hardware backends [13,14].

* Corresponding author.
E-mail address: javier.cardama@usc.es (F.J. Cardama).

https://doi.org/10.1016/j.future.2025.107989
Received 18 April 2025; Received in revised form 18 June 2025; Accepted 20 June 2025
Available online 3 July 2025

One of the critical challenges in quantum computing remains the scalability and noise of the qubits. Current quantum hardware is limited by the number of qubits that can be reliably maintained and manipulated on a single chip, thus complicating the development of more complex algorithms [15]. These limitations have led to the development of new computing approaches, one of which is the design of modular architectures based on DQC. In these architectures, multiple quantum processing units (QPUs) are networked together to work on a problem collaboratively [16–18]. DQC uses both quantum and classical communications to distribute and synchronize computations across QPUs, thereby potentially overcoming the scalability constraints of individual quantum chips [19–21].

DQC introduces a level of complexity over monolithic quantum computing systems (systems consisting of a single QPU) due to the management of classical and quantum networks and its complexities [22]. Because of this, an abstraction model of the complete process of a DQC algorithm, from its high-level specification to the specific back-end where it is supposed to run, must be defined to facilitate the development of tools according to their specific use. Additionally, this need highlights the importance of defining an IR that not only enables the efficient programming of quantum algorithms within this abstraction framework but also addresses the specific challenges inherent to DQC, such as optimized communication protocols and precise synchronization mechanisms across QPUs [23].

In the context of DQC, it is essential to distinguish it from the concept of the Quantum Internet [24,25]. While DQC uses a quantum interconnection network, its focus is on a form of future distributed quantum computing, thus allowing the user to abstract themselves from both classical and quantum networks [26]. Therefore, it is crucial to abstract away the complexities of quantum networking and, instead, define the appropriate high-level instructions within an IR to ensure its efficient utilization. Currently, a wide range of tools are available for simulating quantum communication networks (network-level simulators) [27], such as SquidASM [28] and Simulaqron [29], as well as discrete event simulation for quantum communication at the physical level, including NetSquid [30] and SeQuence [31]. There are also simulators specifically designed for distributed quantum computing, such as QuNetSim, which follows a more point-to-point model, with network simulation handled by the in-process EQNS simulator [32].

In response to the problems encountered in the literature, this paper proposes two main contributions to the state of the art:

- The definition of a **layered abstraction model** for the correct implementation of IRs for DQC by collecting information from the literature, both from standards followed in classical computing and from attempts at quantum computing. This objective will allow other developers to implement new IRs following a model whose efficiency will be demonstrated later.
- The proposal of an IR that implements the abstraction layer model proposed in the previous objective. Our proposal, NetQIR, extends Microsoft's QIR [33], augmenting it with advanced communication and distributed computation directives to support interoperability and scalability, thereby facilitating robust quantum algorithm development in distributed quantum environments and hybrid programming. As IR, the objective is to be a common language that unites different and future optimization, compilation, or scheduling tools.

It is important to note that both the layered abstraction model and the proposed IR aims to contribute to the set of tools for future distributed quantum computing, abstracting from any type of underlying network. The proposed work is not a compiler or an optimizer; it is a specification of an IR designed to facilitate the integration and use of other tools. Therefore, automatic circuit or task partitioning is not within the scope of NetQIR or the proposed work, in the same way that MPI does not automate data or task parallelism.

Fig. 1 illustrates the advantages of using NetQIR as an IR for DQC, highlighting its extension of QIR and, consequently, LLVM. This figure illustrates DQC tools compiled to various quantum network simulators. In the initial approach, without the use of IRs, the number of required compilers is $n \times m$. However, by leveraging an IR such as NetQIR, this complexity is significantly reduced to $n + m$, streamlining the compilation process.[1]

The paper is structured as follows: initially, Section 2 reviews the related work, focusing on existing IRs and programming languages for DQC. Then, Section 3 introduces a layered abstraction model, essential for DQC. It presents the development and network layers, detailing the specific components required to achieve a modular and interoperable architecture for DQC. In addition, Section 4 presents the topics related to the proposed IR, including its specification and the tools developed to facilitate the design of new software for future developers. Subsequently, Section 5 presents a discussion related to the DQC languages analyzed in the related work, along with an evaluation of the different characteristics of the network layer to justify its abstraction in the proposed abstraction layer model. Finally, Section 6 concludes the work and specifies future work.

2. Related work and background

2.1. Background on distributed systems

At the core of any modern computing system lies the operating system (OS), which provides a set of fundamental abstractions to manage hardware complexity and support application execution. The most relevant abstractions include [34]:

- **Processes:** isolated execution contexts with their own memory space and scheduling policies.
- **Memory management:** virtual memory abstraction, page swapping, and memory protection.
- **Input/Output (I/O):** abstract representations of devices, filesystems, and network interfaces.

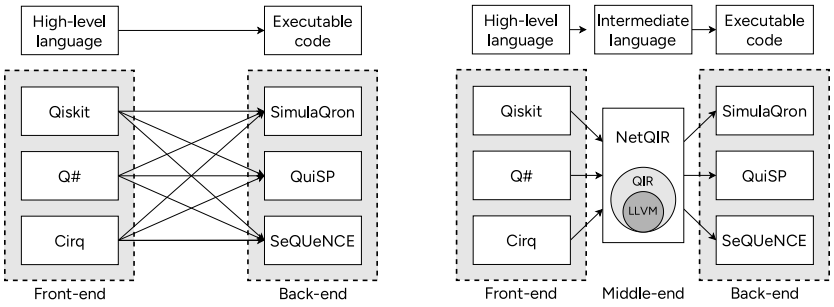
In *distributed systems*, these abstractions are replicated and localized across multiple physical nodes, each typically running its own instance of a full-fledged operating system. This design choice allows each node to manage its own resources — CPU, memory, and I/O devices — independently, leveraging existing OS-level capabilities and simplifying low-level hardware interactions. Fig. 2 shows the levels of a distributed system.

However, distributed applications often require coordination, synchronization, and data exchange across these independent nodes. To support such tasks, a middleware layer is introduced above the OS layer. This middleware provides a programming interface that abstracts away many of the complexities of distributed execution, enabling developers to orchestrate high-level parallel applications without managing low-level networking or hardware-specific details.

One of the most successful and widely adopted middleware standards in classical high-performance computing (HPC) is the *Message Passing Interface* (MPI). In classical HPC, MPI is the *de facto* standard for scalable parallelism across distributed systems. An MPI process is an independent instance of a program with its own local memory, typically mapped to a logical compute unit or hardware core. Each process is assigned a unique *rank* and operates within a *communicator*—a named group of processes that can coordinate using collective or point-to-point communication primitives such as `send` and `recv` or `scatter` and `gather` [36–38].

This abstraction of rank and communicator implies a subsequent mapping by the compiler from logical resource (e.g. rank) to physical

[1] With n as the number of *front-ends* and m of *backends*.



(a) Without IR usage.

(b) Applying NetQIR as IR.

Fig. 1. Comparison between the integration of NetQIR as DQC IR in a compilation scheme between DQC programming frameworks and quantum simulator backends.

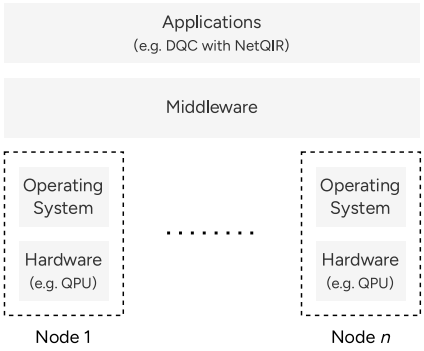


Fig. 2. Levels of a distributed system defined in [35].

resource (e.g. processing unit) typically performed by the compiler or operating system. Work is already underway in the literature on operating systems that allow quantum applications to be run on quantum network nodes such as [39].

Therefore, these abstractions allow the user to exchange messages between the different processes of the distributed system with the aim of collaborating to divide data or tasks. This division is not automatically generated, in any case, by MPI, with the user being responsible for adapting their sequential program to the distributed version.

The goal of the proposed work with NetQIR is the same; NetQIR allows representing circuit partitioning, but it is the user who must modify their sequential circuit to adapt it to a distributed version using the additional features introduced by NetQIR.

NetQIR adapts these ideas to distributed quantum computing, abstracting quantum communication into a clean interface that can coexist with classical workloads and allow for transparent compiler-level optimizations in hybrid quantum–classical applications.

A similar approach is adopted by Quantum Message Passing Interface (QMPI), as proposed by Haner et al. [6]. As its name implies, it is an adaptation of the classical Message Passing Interface (MPI) [36] for quantum communications, achieved by defining analogous point-to-point and collective operations for the quantum pipeline. One of the key differences between NetQIR and QMPI lies in their approach to communication semantics. While NetQIR adopts an MPI-inspired programming style to organize distributed interactions, QMPI replicates classic MPI primitives directly. This direct copying stems from direct problems with the characteristics of quantum computing, in this case, with the no-cloning theorem, as it is not possible to perform copying operations (e.g., broadcast operation). Instead, NetQIR introduces communication abstractions specifically designed to address the unique challenges of distributed quantum computing, as will be detailed in the following sections.

2.2. Related work about intermediate representations in quantum computing

In DQC, the software stack lacks sufficient tools. From high-level languages to lower-level representations — and even development libraries — the literature offers few possibilities, as demonstrated by the state-of-the-art review conducted by Barral et al. [21]. This becomes even more evident when compared to the monolithic case, in which numerous programs, libraries and other software are available for developing quantum applications.

Focusing on the IRs in the monolithic quantum computing case, MLIR [40,41] or SQIR [42] are found, along with QIR [33], backed by the QIR alliance[2] — from now on, it will be referred to as QIR —. The latter is based on the LLVM IR [43] in an attempt to integrate quantum computation into the LLVM infrastructure.[3] In fact, QIR aims to integrate quantum directives with the classical compilation stack, leveraging the advanced LLVM tools to facilitate the generation of highly efficient quantum instructions. In this work QIR will be extended and, therefore, the LLVM IR will be further extended by introducing the necessary directives to perform quantum communications. Throughout this manuscript, this extension will be explained and exemplified.

For DQC, the two most popular specifications in the literature are InQuIR [44] and NetQASM [28]. The first one, InQuIR, is developed from the starting point solely as an IR for DQC. Their primary motivation stemmed from the absence of a dedicated IR for distributed quantum systems. InQuIR stands out for formally defining the grammar of the IR. Using this formalism, InQuIR defines the operational semantics of the IR, which allows it to define and predict how the InQuIR programs will behave under several circumstances. The authors propose some important examples, such as deadlocks and qubit exhaustion, and a roadmap for solving these inconveniences. But InQuIR provides a too low-level approach with explicit generation of the Einstein-Podolsky-Rosen (EPR) pairs and instructions that acknowledge the architecture of the machine, having less control over the form of quantum links.

[2] https://www.qir-alliance.org/.

[3] LLVM is a versatile framework for building compilers and code transformation tools. It lets developers write high-level language code that can be efficiently compiled into machine code for various architectures, with extensive code optimization and analysis support.

As an alternative, as mentioned, NetQASM [28] presents an abstract architecture model composed of an application layer, which is responsible for the classical communications between nodes, and a so-called quantum network processing unit (QNPU), which handles quantum computations and communications. This highlights the scope of NetQASM: the Quantum Internet. It is specifically designed for quantum networks, setting aside *inter-core* communication, which does not require the additional layers that NetQASM introduces. Moreover, NetQASM presents a basic language, called *vanilla*, and a set of variations specially designed for the different quantum architectures, called *flavors*. The authors state that the vanilla version acts as an IR, and the different flavors act as assemblies. The main disadvantage of NetQASM, like IR for DQC, is that its architecture is network-oriented rather than computation-oriented being a proposal closer to the sockets API than to computing communication functions. It also does not consider conditional gates, which are constantly employed in quantum communication protocols, as part of the IR. What is actually done is to perform a measurement, send the result to the application layer and wait until the application layer returns a subroutine with the gate —if the measurement was 1— or without the gate —in the opposite case—.

After discussing the related work about the IRs, it is important to note that NetQIR will not extend QIR arbitrarily. Rather than proposing an ad hoc language, NetQIR builds upon the LLVM-based QIR specification to leverage existing classical optimization and compilation pipelines. This design choice enables hybrid applications to benefit from decades of compiler research, optimization techniques, and mature toolchains originally developed for classical high-performance computing (HPC). By embedding quantum–classical interfaces within an LLVM compatible IR, NetQIR facilitates seamless integration into HPC workflows and heterogeneous computing environments.

In summary, both InQuIR and NetQASM exhibit certain aspects that may represent drawbacks for an IR. Additionally, QMPI defines a high-level standard that is strongly coupled with the classical MPI, introducing various complications and intricacies. NetQIR seeks to address these issues and this manuscript will detail the approach taken to achieve that goal and justify the decisions made in order to do so.

3. Layered abstraction model for distributed quantum computing

Software architectures nowadays strongly rely on abstraction mechanisms as a core principle. These simplify the development of new algorithms, platforms, compilers and tools. DQC architectures follow the same principle. As in the monolithic–and even classical–case, any new IR targeting this paradigm should be hardware-independent and compatible with diverse quantum computing platforms. To achieve this, it is essential to first define a layered abstraction model before designing the DQC IR. To achieve this, it is essential to first define a layered abstraction model before designing the DQC IR.

Fig. 3 depicts the abstraction layers relevant for executing algorithms in a DQC environment. This paper focuses on the computational part of distributed quantum, Therefore, two key layers are defined in our proposed model: the development layer and the network layer. The development layer provides users the necessary tools to design and implement DQC algorithms and software. Meanwhile, the network layer acts as an interface between the development layer and the quantum interconnection network, ensuring seamless interaction while managing its underlying characteristics. Additionally, the network layer interacts with lower layers as needed, further abstracting the physical complexities of the quantum network.

3.1. Network layer

As discussed above, the network layer aims to abstract the particularities of the quantum interconnection network to the development layer. For this purpose, it is necessary to identify this layer's main components, which are the *quantum interconnection network*, the *quantum communication channel* and the *communication protocols*.

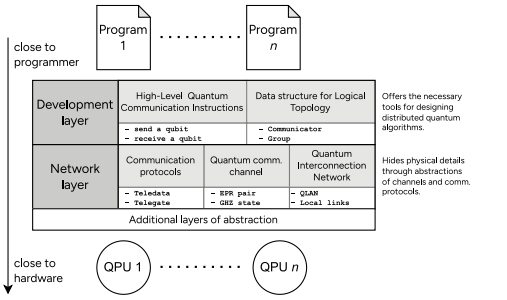


Fig. 3. Abstraction layers relevant to the development of an abstract IR for DQC.

The *quantum interconnection network* abstracts the communication between different quantum computing nodes. This network can be composed of different types of connections, such as quantum network devices, QLANs, or the Quantum Internet. Fig. 4 illustrates a complex example that interconnects quantum computing nodes of different QLANs via the Quantum Internet. This network architecture comprises several QLANs interconnected through the Quantum Internet, allowing quantum computing nodes to communicate with each other. While the quantum interconnection network can abstract a wide spectrum of quantum communication infrastructures — from local optical links to long-range quantum internet protocols — the layered model proposed is specifically designed with a DQC perspective. This implies that, analogously to how the MPI is capable of operating over the Internet but is fundamentally optimized for tightly-coupled HPC clusters, the abstraction model focuses on practical, low-latency interconnections between QPUs within modular or co-located quantum systems.

The main objective of introducing this abstraction layer is not to support long-range quantum communications, but rather to enable efficient and scalable interconnection of modular QPU architectures [45]. In these systems, multiple quantum processors — each with limited qubit capacity — are physically co-located and interconnected to jointly execute a quantum algorithm. Supporting such modular systems requires a flexible software model that can abstract the communication between QPUs without exposing hardware-specific constraints to the developer.

A concrete example of this architectural direction is Xanadu's recent modular quantum computing system, which connects 35 photonic chips using 13 km of optical fiber to construct a distributed quantum processor [46]. Although physically integrated within a local infrastructure, such systems rely on quantum communication channels between QPUs to function as a cohesive unit.

The *quantum communication channel*, as its name indicates, represents the abstraction of the quantum channel responsible of the connection between two quantum nodes. It enables the exchange of quantum information between quantum computing nodes and exploits the principles of quantum mechanics, particularly qubit entanglement. Fig. 4 shows the quantum channel next to a classical channel, which allows operations such as state teleport — an operation that requires both a quantum and a classical channel — to be implemented.

And the last component of the network layer is also its central element: *quantum communication protocols*. They define the fundamental building blocks for exchanging quantum information between quantum computing nodes. Two of the most important communication protocols are *teledata* [47] and *telegate* [48]. These protocols exploit qubit entanglement to enable the exchange of quantum information. Both techniques utilize an entangled EPR pair, where one qubit of the pair resides on a QPU and the other is located on a physically separated QPU. These EPR pairs create a link between the two QPUs, allowing

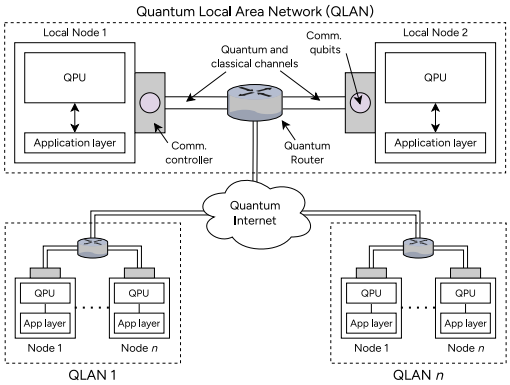


Fig. 4. Complex quantum network architecture interconnecting quantum computing nodes of different QLANs via the Quantum Internet.

quantum data to travel from one QPU to the other by exchanging classical information resulting from measurements of specific qubits. While this work focuses on `teledata` and `telegate`, other communication protocols are also available in the literature [49,50]. Both were selected for this study because they represent two fundamental and widely studied paradigms for distributed quantum communication: quantum state transfer and remote gate application. Their contrasting characteristics make them ideal benchmarks for evaluating abstraction models and compiler decision-making. However, the proposed abstraction model is extensible and can incorporate additional protocols in future work.

Fig. 5 shows the basic structure of the *teledata* (see Fig. 5(a)) and the *telegate* (see Fig. 5(b)) protocols. In both techniques, starting from a state $|a\rangle = \alpha|0\rangle + \beta|1\rangle$ in the local QPU_1, it is necessary that the remote QPU_2 can compute using this information via an EPR pair $|\Phi^+\rangle$. Each protocol is elaborated below:

- **Teledata** protocol transmits the state of the qubit $|a\rangle$ in QPU_1 to an empty qubit in QPU_2. This transmission involves teleportation of the quantum state, causing the original qubit to collapse upon measurement and transferring its state to the destination qubit.
- **Telegate** protocol generates a pair in the state $\alpha|00\rangle + \beta|11\rangle$, where the first qubit is in QPU_1 and the second qubit is in QPU_2. The second qubit is used as a control qubit for a controlled operation. Considering that the control qubit is in the state $|a\rangle = \alpha|0\rangle + \beta|1\rangle$, using the second qubit of the pair achieves the same effect as performing a controlled operation in QPU_2 with the state of the qubit in QPU_1.

The main difference between teledata and telegate is that in teledata, the state is transferred, and computation is performed locally at the receiving QPU, whereas in telegate, the state is not transferred; instead, quantum gates are controlled remotely. Table 1 compares both techniques by evaluating four key characteristics. It is important to understand the difference between performing an operation "Locally" and "Remotely". A **local operation** does not require the use of either quantum or classical communications. On the other hand, a **remote operation** involves the use of quantum or classical communications with other QPUs.

1. *Collapsed qubit*: indicates whether the source qubit collapses once the protocol is executed, requiring a qubit reset.
2. *Entanglement result*: refers to the scope affected by the entanglement generated between the remote and local qubits. This entanglement can be local to the computation node or global to the distributed system.

Table 1
Comparative features between teledata and telegate techniques.

Protocol	Collapsed qubit	Entangl.result	Measures	Numbersyncs
Teledata	Yes	Local	Local - Local	1
Telegate	No	Global	Local - Remote	2

3. *Measurements*: describes how the measurements are performed to implement the protocol.
4. *Number of synchronizations*: the number of synchronizations between the QPUs required to execute the communication protocol.

As observed, in the teledata protocol, the qubit collapses when sending the information, necessitating a reset of the qubit afterwards. This occurs because the quantum state is entirely transferred to the target node; thus, operations are performed locally at the destination, and the resulting entanglement is local to the target QPU. Additionally, measurements are performed simultaneously on two qubits local to the QPU_1, requiring only a single synchronization between the two QPUs.

In contrast, in the telegate protocol, the quantum information is shared as a reference without measuring the original qubit, eliminating the need to reset it. Sharing a reference implies that the generated entanglement is global to the distributed system — this means that qubits from different QPUs have been entangled —. Furthermore, an initial measurement is performed at the QPU_1, and a final measurement is conducted on the remote QPU_2, requiring two separate synchronizations between the QPUs, known as *Cat-Entangler* and *Cat-DisEntangler* (see Fig. 5(b)). The compiler determines the timing of the second synchronization (*Cat-DisEnt*), especially when the qubit is no longer in use.

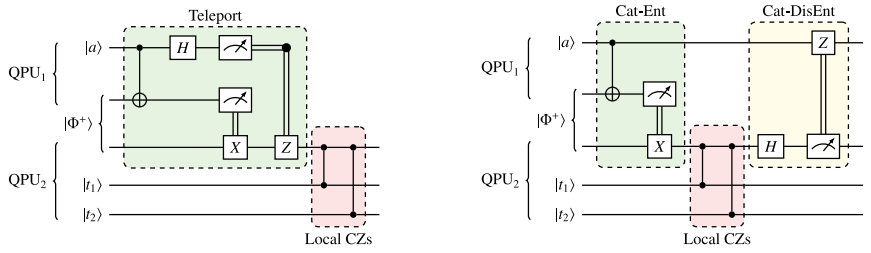
Both protocols have advantages and disadvantages, and there is no clearly superior option. The choice between them depends on the problem to be solved; therefore, the specification of a layered abstraction model will allow the compilation tools to be developed to make an informed decision.

3.2. Development layer

In this subsection, the development layer is introduced, designed to provide users with the necessary instructions to work with DQC algorithms while abstracting away the complexities of the network layer. Specifically, two key components for the development layer, shown in Fig. 3: the *Data Structure for Logical Topology* and the *High-Level Quantum Communication Instructions*. The first component aims to abstract the Quantum Interconnection Network and part of the Quantum Channel by introducing a logical topology data structure that simplifies the development of distributed quantum programs. This logical topology is designed to expose only the number and identity of available quantum processing units (QPUs) to the user, intentionally omitting intermediate network devices such as routers or switches. The second component focuses on abstracting the Quantum Communication Protocols and, to some extent, the Quantum Channel as well, by hiding the implementation details of how communication qubits are generated, through high-level quantum communication instructions.

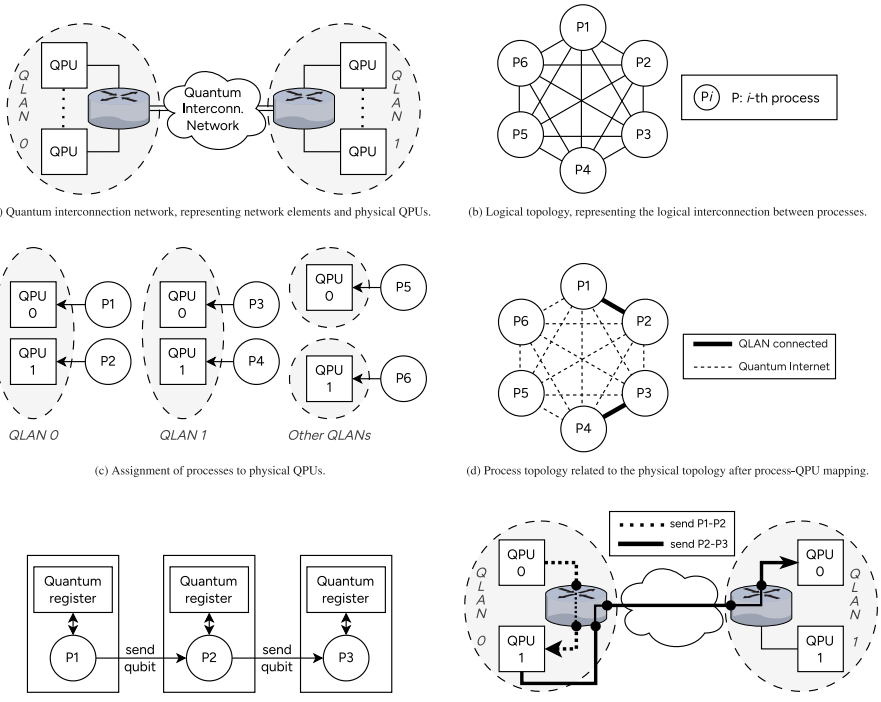
It is important to note that, as in any abstraction model, the compilation tool aims to translate the code from the development layer to physical hardware and the network layer. IRs are developed with the aim of abstracting the underlying hardware and providing a common interface for the development of new applications that can subsequently be used in modern compilers. Therefore, this work seeks to define a model and an IR that performs a complete and correct abstraction of the underlying hardware, providing users with the necessary tools to create their DQC applications. The proposed work is not a compiler.

This allows the different responsibilities involved in running DQC applications to be decoupled, enabling new OSs such as [39] or new application execution environments such as [51] to be developed using a common interface.



(a) Teledata. (b) Telegate.

Fig. 5. Examples of teledata and telegate circuits for the application of CZs.



(a) Quantum interconnection network, representing network elements and physical QPUs.

(b) Logical topology, representing the logical interconnection between processes.

(c) Assignment of processes to physical QPUs.

(d) Process topology related to the physical topology after process-QPU mapping.

(e) Example of communication operations between different logically connected processes.

(f) Flow of interprocess communication operations in the quantum interconnection network.

Fig. 6. Relationship between development layer abstractions and physical network structures of the network layer.

3.2.1. Data structure for logical topology

In this context, a *process* represents a logical execution entity assigned by the OS to a QPU, which participates in the distributed computation. Inspired by the classical HPC model, such as MPI, each process in the abstract model is assigned a rank and can be organized into groups and communicators. These data structures represent logical reorganizations of processes; therefore, a process may belong to multiple communicators or groups. This abstraction enables coordination of distributed tasks independently of the physical network configuration.

Fig. 6 shows the relationship between the network layer and the proposed development layer. Fig. 6(a) shows a simplified quantum interconnection network, which would be used by the development layer to generate a logical process topology, as shown in Fig. 6(b). It is important to note that we are talking about an abstraction of processes, not QPUs or network devices, as these should be abstract to the user.

This abstraction means that users do not need to have a view of the physical topology of the network, which can be highly variable depending on the context and is not always interesting or useful for developing computer programs. In this way, the developer works only with an abstract view of the topology between their running processes. The network layer then manages and optimizes the actual connections within the network, providing an effective interface between the development layer and the physical infrastructure.

Therefore, the IR that implements the development layer abstractions does not need to manage the network layer features, in order to decouple responsibilities. In this case, there are certain operations for translating from the development layer to the network layer that are the subject of the OS.

Fig. 6(c) shows the assignment of processes to processing units, in the case of the DQC, QPUs. This process abstraction is managed

by the OS of each distributed node, as indicated in Section 2.1. Fig. 6(d) shows the logical topology related to the physical topology once processes have been assigned to QPUs, with the aim of showing that some connections between processes are closer than others.

Finally, if we focus on end-user development, Fig. 6(e) shows an example of point-to-point communications between processes, where it can be seen that the instruction does not take into account the physical topology, but simply seeks to comply with the logical topology between processes. Fig. 6(f) shows the physical path followed for the execution of these instructions.

In the following, these high-level instructions for sending quantum messages are cited in more detail.

3.2.2. High-level quantum communication instructions

High-level directives conform the last piece of abstraction of the development layer. With a clearly defined semantic behavior, they are able to abstract the underlying communication protocols in DQC. This approach offers two key advantages: first, it enables the development of distributed quantum algorithms while abstracting the complexity of the underlying communication mechanisms; second, it provides the compiler with precise semantic information for each function, facilitating both optimization and the selection of the most suitable communication protocol. These instructions should prioritize fundamental computational operations, such as data transmission, reception and collective processing, rather than exposing lower-level physical or network mechanisms like entanglement generation, which should remain transparent to the user. It is important to note that blocking communication functions involve synchronization, as do other instructions such as the Barrier instructions. These operations are the responsibility of the OS of the distributed system.

These semantic instructions also allow to improve the management of the OS and the compiler, for example, to improve the fidelities of the result. When using quantum networks, as in the classical counterpart, not only is there an overhead due to communications, but additional noise is incorporated into the result. In the case of quantum computing this is crucial, since an intense use of quantum connections can cause the fidelity of the result to decrease. Using high-level instructions together with the logical topology between processes allows the operating system to better manage its allocation of processes to QPUs, minimizing communications as much as possible.

As an example of how a lack of abstraction can negatively impact both performance and software quality, consider the `entSwap` instruction defined by InQuIR. This instruction explicitly specifies the entanglement swapping procedure, which enables the connection of two quantum nodes that are not directly linked. However, this is fundamentally a low-level problem, as the development layer should not have to manage the connectivity of quantum nodes. Exposing this detail to the development layer might lead users to invoke `entSwap` unnecessarily, resulting in inefficient calls. Allowing the network — and even lower-level layers — to handle connectivity issues would contribute to more robust software, as these unneeded calls would not be performed.

4. NetQIR: a quantum intermediate representation for distributed quantum computing

This section introduces the IR proposed in this paper: NetQIR, an extension of QIR for DQC. NetQIR is defined according to the layered abstraction model presented in Section 3. NetQIR aims to fulfill the development layer by abstracting from the underlying network, being useful for future distributed quantum computing in any classical-quantum interconnection network. The responsibilities of managing the physical resources would be left to the OS, as discussed above. It is important to emphasize that an IR is fundamentally a formal specification intended for future developers. To that end, a specification has been created and a Python Software Development Kit (SDK) developed to test and work with it. This approach allows users to fully understand the specification by experimenting with actual code, thereby facilitating the production of software that employs NetQIR as an IR. Moreover, a grammar has also been developed in ANTLR to allow programmers to translate NetQIR code to specific backends, such as simulators or real systems.

4.1. NetQIR specification

This subsection details the NetQIR specification. In doing so, NetQIR defines both data structures and functions. The data structures include two components: `%Comm` and `%Group`, which correspond to the Communicator and Group described in Section 3.2.1, respectively. Regarding functions, NetQIR defines a set of state functions, data structure functions, and communication functions — the core focus of this work —. It is important to note that the state functions do not relate to the quantum state of the system but rather to the internal state of the NetQIR execution environment. In addition to this document, the authors provide a more comprehensive specification and detailed documentation on GitHub [52].[4]

4.1.1. State functions

State functions serve as breakpoints where the underlying layers of NetQIR's abstraction can be defined. For example, these functions provide a point where the compiler can determine when to query and establish connections between different quantum or classical devices. NetQIR introduces two state functions inspired by similar solutions in classical distributed computing frameworks, such as MPI. These functions are:

- `__netqir__initialize()`, which initializes the execution environment.
- `__netqir__finalize()`, which terminates the environment.

These functions establish a structured workflow for DQC, ensuring proper initialization and finalization of the execution context.

4.1.2. Operate datatypes functions

In order to abstract from the physical topology, as the development layer explained in Section 3 aims, NetQIR needs to implement a logical topology. For this purpose two already mentioned data structures have been added: `%Comm` and `%Group`. These will allow the organization of the processes in groups and the establishment of logical topologies that the processor will then be able to link with its physical version. In this abstract model, a process refers to a logical unit of execution associated with a QPU, responsible for performing computations and participating in distributed tasks. This concept, inspired by the notion of processes in classical HPC frameworks like MPI, enables the grouping and coordination of distributed quantum operations. Additionally, data type functions are defined to create or modify the described types and to obtain information about their content at runtime.

NetQIR, as it has been spurred along this work, works akin to MPI. Here another example of the similarities arises, because two key variables are associated with the so-called `comm_world`: the process `rank` and the communicator `size`. Consequently, both functions will be included:

- `__netqir__comm_rank`: returns the process `rank` inside the specified communicator.
- `__netqir__comm_size`: operation which, from a `%Comm` object, returns the number of nodes in that communicator.

Moreover, there are also functions established to create, modify or delete `%Comm` and `%Group`, and, in addition, operations to establish new logical network topologies. For further information on these functions and their use, the reader is referred to the specification [52].

[4] https://netqir.github.io/netqir-spec/.

Table 2
NetQIR functions: point-to-point and collective.

Point-to-point communication functions			
Sending functions		Receiving functions	
_netqir_qsend_array	(Array*, i32, i32, Comm*)	_netqir_qrecv_array	(Array**, i32, i32, Comm*)
_netqir_qsend_array_teledata	(Array*, i32, i32, Comm*)	_netqir_qrecv_array_teledata	(Array**, i32, i32, Comm*)
_netqir_qsend_array_telegate	(Array*, i32, i32, Comm*)	_netqir_qrecv_array_telegate	(Array**, i32, i32, Comm*)
_netqir_qsend	(Qubit*, i32, Comm*)	_netqir_qrecv	(Qubit**, i32, Comm*)
_netqir_qsend_teledata	(Qubit*, i32, Comm*)	_netqir_qrecv_teledata	(Qubit**, i32, Comm*)
_netqir_qsend_telegate	(Qubit*, i32, Comm*)	_netqir_qrecv_telegate	(Qubit**, i32, Comm*)
_netqir_measure_send_array	(Array*, i32, i32, Comm*)	_netqir_measure_recv_array	(i1*, i32, i32, Comm*)
_netqir_measure_send	(Qubit*, i32, Comm*)	_netqir_measure_recv	(i1*, i32, i32, Comm*)
Collective communication functions			
_netqir_scatter	(Array*, i32, Array*, i32, i32, Comm*)	_netqir_expose	(Qubit*, i32, Comm*)
_netqir_scatter_teledata	(Array*, i32, Array*, i32, i32, Comm*)	_netqir_expose_array	(Array*, i32, i32, Comm*)
_netqir_scatter_telegate	(Array*, i32, Array*, i32, i32, Comm*)		
_netqir_gather	(Array*, i32, Array*, i32, i32, Comm*)	_netqir_reduce	(Array*, i32, Array*, i32, i32, Comm*)
_netqir_gather_teledata	(Array*, i32, Array*, i32, i32, Comm*)	_netqir_reduce_teledata	(Array*, i32, Array*, i32, i32, Comm*)
_netqir_gather_telegate	(Array*, i32, Array*, i32, i32, Comm*)	_netqir_reduce_telegate	(Array*, i32, Array*, i32, i32, Comm*)

4.1.3. Communication functions

NetQIR proposes a large set of semantic instructions to improve the construction of DQC algorithms without the need to know the underlying communication protocols. Operations are defined to send and receive classical data, and, within quantum communications, two large sets are created: point-to-point instructions and collective communication routines. These functions are defined in Table 2 and explained below.

Collective communication operations are particularly relevant in distributed quantum algorithms, as they enable efficient coordination among multiple QPUs. NetQIR extends classical collective patterns, such as scatter and gather, to the quantum domain, while introducing new abstractions tailored for quantum-specific needs. Among them, the expose operation stands out as a novel contribution. This directive allows multiple QPUs to act upon a shared logical qubit without transferring its state explicitly, leveraging global entanglement to minimize resource consumption. By abstracting the communication protocol and leaving the implementation details to the compiler, expose exploits the layered model's advantages to reduce synchronization overhead and optimize the use of communication qubits in distributed computations.

4.1.3.1. Point-to-point communication. Point-to-point communication in quantum computing parallels that in classical computing, where one node sends or receives information to or from another node. The primary difference is that in the classical case, the information is purely classical, whereas in quantum computing, the information can be classical or quantum. Table 2 lists the directives responsible of communication in quantum computing divided into two subgroups: sending and receiving functions. Each sending function corresponds to a receiving one, both of which block the execution of the quantum program. This design ensures that for each send operation at a node there is a corresponding receive operation that unblocks it at the destination node, and vice versa. Mismatches between these could cause an incorrect behavior or compilation errors.

The most basic sending function is `__netqir__qsend`, representing the part of the circuit on the sending QPU. Additionally, the function `__netqir__measure_send` corresponds to sending a classical bit resulting from a measurement, enabling users to develop custom quantum communication protocols. Each of these functions has an `array` variant for sending or receiving arrays of qubits. It is also important to highlight that the abstraction introduced throughout this work enables the compiler to select the most appropriate communication protocol based on the execution context. If the user wishes to specify a particular protocol, they can use the specific version of the selected function, for example `__netqir__qsend_teledata` in case of wanting to use the *teledata* protocol at sending.[5]

4.1.3.2. Collective communication. While the qubit sending and receiving functions are essential primitives, they may not always be the most efficient choice. Collective communication directives address this by involving multiple QPUs in coordinated operations. They resemble those in classical distributed computing, aiding comprehension for HPC computing users. These functions include `scatter`, `gather`, `reduce` and `expose`.

- `__netqir__scatter` function distributes an array of qubits from one QPU to several others, enabling parallel processing.
- `__netqir__gather` function collects qubits from multiple QPUs into a single QPU.
- `__netqir__reduce` directive allows collecting information from multiple remote qubits and applying an operation to obtain a final result. Using `reduce` simplifies code complexity and enhances computational efficiency compared to sequences of `qsend` and `qrecv`.
- `__netqir__expose` directive, which shares a reference to a qubit with other QPUs, allowing modifications visible to the entire distributed system. This is particularly useful in operations where all nodes need to use a qubit as a target or control, such as in the distributed Quantum Fourier Transformation (QFT) algorithm [53], as illustrated in Fig. 8. In this circuit, a sequence of controlled-phase gates is applied between one target qubit and several control qubits located in different QPUs. Using `expose`, the target qubit can be made available to all control units without physically transferring its state, allowing each QPU to apply its operation as if the qubit were local. Additionally, Fig. 9 shows a possible implementation of the expose operation using a GHZ state to connect the QPUs, achieving the state $\alpha|00\ldots00\rangle + \beta|11\ldots11\rangle$ with the exposed state being $\alpha|0\rangle + \beta|1\rangle$. This implementation is not part of the development layer but represents one of the strategies available to the compiler could choose depending on the underlying physically network.

Similar to point-to-point functions, collective directives have *teledata* and *telegate* variants. Fig. 7 illustrates the use of scatter and gather operations using the teledata protocol.

[5] Notably, if a node uses `__netqir__qsend_teledata` to send, the receiving node must use `__netqir__qrecv_teledata` to receive. Mismatched protocols lead to incorrect behavior. The general functions `__netqir__qsend` and `__netqir__qrecv` offer flexibility by not specifying a protocol, allowing the other node to define it.

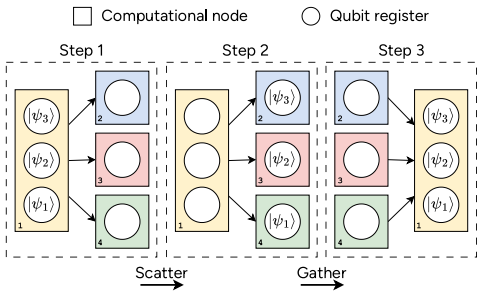


Fig. 7. An example of using a scatter teledata operation on a qubit array (steps 1 to 2) and the inverse gather teledata operation (steps 2 to 3) between 4 QPUs (labeled in each square).

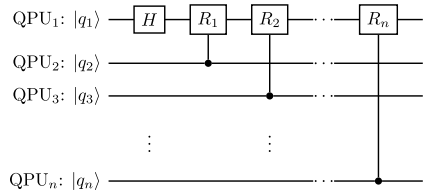


Fig. 8. Use case for the `__netqir__expose` directive on the $|q_1\rangle$ qubit, as it serves as the target for the other remote qubits.

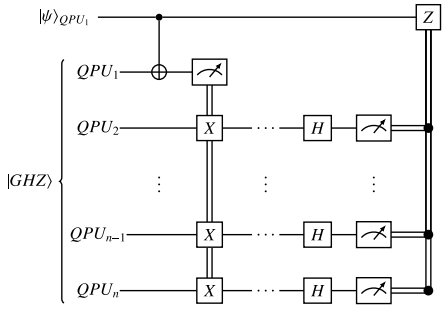


Fig. 9. Possible implementation of the `__netqir__expose` directive on the qubit $|\psi\rangle_{QPU_1}$, which is the target of the rest of the remote qubits. Distributed operations would be performed between `cat-ent` and `cat-dis`.

4.2. NetQIR SDK: PyNetQIR

The NetQIR specification constitutes the central core of an IR, serving as the common starting point for developers in the field of distributed quantum computing. To support developers in building distributed quantum applications, the NetQIR specification is accompanied by an open-source Python SDK designed to facilitate the generation of NetQIR code in Python-based environments. This tool, named PyNetQIR, is available in the project's GitHub repository.[6]

This SDK enables the generation of NetQIR code from high-level Python scripts, following a structured execution model composed of three main components: *Operations*, *Scopes*, and *Executors*. This is provided as a summary, for more information please consult the open source code repository.

- **Operations** represent the basic actions of a NetQIR program. These may include quantum instructions (such as gate applications or qubit transmissions) as well as classical instructions (like conditionals or communicator queries), allowing for hybrid programming. This hybrid approach extends the LLVM framework to integrate quantum semantics while maintaining compatibility with classical logic.
- **Scopes** are hierarchical structures that group operations logically, similar to blocks in traditional programming. The SDK provides builders (e.g., `MainScopeBuilder`) to construct these scopes and manage their contents efficiently.
- **Executors** are responsible for interpreting or compiling the operations within a scope. In the example shown in Fig. 10, a `PrinterExecutor` is used to emit the resulting NetQIR code, although other executors could target simulators or hardware backends in future implementations.

The typical flow of a NetQIR program using this SDK begins by initializing the program and obtaining the global scope and communicator. Fig. 10 shows an example of this process, where a quantum state is transferred from one QPU to another using the `qsend` and `qrecv` directives. Within the main scope, the NetQIR environment is initialized and the rank and size of the communicator are retrieved. By leveraging ranks and communicators, the SDK mirrors the structure of classical distributed frameworks like MPI, allowing developers to adopt familiar patterns when building quantum programs.

A conditional operator based on the process rank is defined: if the rank is 0, the process performs a quantum send (`qsend`); if it is 1, it performs a quantum receive (`qrecv`).[7] The environment is then finalized and the program is executed using the Executor. This example demonstrates how NetQIR supports modular and semantically clear construction of distributed quantum applications using a model inspired by classical distributed computing.

4.3. NetQIR ANTLR grammar

Once the NetQIR specification has been defined and an SDK for translating Python code into NetQIR has been implemented, it becomes essential to provide future developers with a tool for extending NetQIR. This includes developing new high-level languages that compile to NetQIR and translating NetQIR into low-level machine instructions for quantum devices. To facilitate this, a formal grammar definition is introduced, leveraging the ANTLR [54,55] specification to enable structured parsing and transformation of NetQIR code. The primary advantage of defining this grammar is that future developers can choose the target programming language for their NetQIR parser. ANTLR provides automatic code generation from its grammar definitions, supporting a wide range of well-known high-level programming languages. This flexibility facilitates the integration of NetQIR into diverse software ecosystems, enabling seamless adoption across different development environments. The grammar defined for NetQIR is encoded in the GitHub repository `netqir-grammar`.[8] It is important to note that this grammar aims to classify the different categories of NetQIR functions specified earlier. This approach allows developers to generate more specialized listeners for the generated AST tree, providing, for instance, information on whether they are programming a `qsend` function with modifiers such as `teledata` or `array`.

[6] https://github.com/netqir/netqir-sdk.

[7] Several examples, including this one, are available in the repository: https://github.com/NetQIR/netqir-sdk/tree/main/python/examples.

[8] NetQIR grammar: https://github.com/NetQIR/netqir-grammar/.

```python
program = Program()
program.start()  # start the program

gscope = program.get_global_scope()  # get the global scope
comm_world = program.get_comm_world()  # get the communicator world

main = MainScopeBuilder(1, 0) # create a builder for the main scope

main.netqir_initialize() # initialize the NetQIR environment

# Get my rank and the size of the communicator
my_rank = main.get_rank(comm_world)
main.get_comm_size(comm_world, program.get_size_world_register())

# Create a conditional operator
main.conditional(ConditionalType.EQUAL, my_rank, InmediateRank(0),
                 [lambda: main.qsend(comm_world, 0, 1)], # Rank 0
                 [lambda: main.qrecv(comm_world, 0, 0)]) # Rank 1

main.netqir_finalize() # Finalize the NetQIR environment

main.build(gscope) # Build the main scope linked to the global scope

program.run() # Run the program
program.end() # End the program
```

(a) PyNetQIR code.

```
%Qubit = type opaque;
%Result = type opaque;
%Comm = type opaque;
%size_world = alloca i32;

define void @main() {
  %q0 = alloca %Qubit*;

  call i32 @__netqir__initialize();

  %rk0 = alloca i32;
  call i32 @__netqir__comm_rank(%Comm* @netqir_comm_world, i32 %rk0);
  call i32 @__netqir__comm_size(%Comm* @netqir_comm_world, i32 %size_world);

  %t0_result = icmp eq i32 %rk0, 0;
  br i1 %t0_result, label %t1_true, label %t2_false;

%t1_true:                                          Rank 0
  call i32 @__netqir__qsend(%Qubit* %q0, i32 1, %Comm* @netqir_comm_world);
  br label %t3_continue;

%t2_false:                                         Rank 1
  call i32 @__netqir__qrecv(%Qubit* %q0, i32 0, %Comm* @netqir_comm_world);
  br label %t3_continue;

%t3_continue:

  call i32 @__netqir__finalize();

}

declare i32 @__netqir__comm_rank(%Comm*, i32);
declare i32 @__netqir__comm_size(%Comm*, i32);
declare i32 @__netqir__qrecv(%Qubit*, i32, %Comm*);
declare i32 @__netqir__finalize();
declare i32 @__netqir__initialize();
declare i32 @__netqir__qsend(%Qubit*, i32, %Comm*);
```

(b) Generated NetQIR from PyNetQIR code (reduced for document legibility).

Fig. 10. Generation of NetQIR code of teleportation circuit using PyNetQIR.

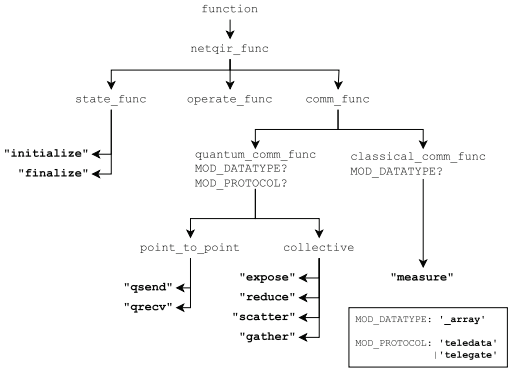


Fig. 11. NetQIR Grammar for the classification of functions in its specification.

Fig. 11 presents a syntax tree illustrating the structure of the defined grammar, where bold elements represent lexical tokens. It can be observed that both quantum and classical communication functions support modifiers, such as "`array`" for sending quantum or classical arrays, and protocol specifications like "`telegate`" or "`teledata`". It is important to note that both modifiers are optional (hence the use of the ? symbol). If no protocol is explicitly specified, the compiler will automatically select the most optimal one based on the execution context.

5. Discussion

This section presents two complementary analyses to support the design rationale behind NetQIR and to evaluate its relevance for distributed quantum computing.

Since it is not feasible to perform a quantitative comparison between IRs — given that they are formal language specifications without associated compilers or optimizers for benchmarking — in this section we conduct a qualitative comparison of various languages designed for distributed quantum computing, followed by a justification of the abstraction layer model through an evaluation of different communication protocols and quantum interconnection networks. In this case, the goal is not to develop a tool that directly improves fidelity, memory management, or execution time. Instead, we propose a model and an IR intended to support the future development of software stack tools for distributed quantum computing.

Therefore, it is proposed a qualitative comparison of different languages designed for distributed quantum and, subsequently, a justification of the abstraction layer model by evaluating different communication protocols and quantum interconnection networks.

First, a **comparative analysis with state-of-the-art languages** is performed to assess how existing DQC intermediate representations and frameworks align with the layered abstraction model proposed in this work. This comparison identifies the extent to which each solution abstracts the complexities of distributed quantum execution at the network and development layers.

Second, a **justification of the layered abstraction model** is provided, with a focus on the benefits introduced by collective communication directives. In particular, this analysis demonstrates how high-level operations, such as `scatter`, `gather`, or the novel `expose`, empower the compiler to select the most appropriate communication strategy depending on the system's topology or the specific requirements of a given algorithm. Rather than prescribing low-level instructions, these abstractions enable optimization and adaptation to the execution context, ultimately leading to more efficient resource usage and scalability. It is important to point out that the objective is to show the reader how the compiler and the OS can use one communication protocol or another depending on the problem to be solved.

Together, these two analyses highlight the practical and conceptual value of adopting a layered abstraction model and the role of NetQIR in bridging high-level algorithm design with efficient distributed quantum execution.

Table 3
Qualitative comparison table between the different languages selected for DQC programming.

Language	Network layer			Development layer		Other characteristics	
	Quantum channel	Q. Interconnection network	Communication protocols	High-level quantum comm. instructions	Data structure for logical topology	Real quantum computing inspired	Hybrid programming
NetQASM	∼	∼	✗	✗	✗	✓	✗
InQuIR	∼	✗	✗	✗	✗	✓	✗
QMPI	✓	✓	✓	✓	✓	✗	∼
NetQIR	✓	✓	✓	✓	✓	✓	✓

5.1. Comparison with state-of-the-art languages

This section presents a comparative analysis of existing DQC languages and IRs, focusing on their alignment with the layered abstraction model proposed in Section 3. This model defines essential elements across two layers — the network layer and the development layer — designed to facilitate the efficient development of DQC tools. The evaluation criteria include key aspects such as the ones outlined below:

Network layer: the network layer contains the necessary characteristics to define a correct quantum–classical connection between different nodes, abstracting from the underlying physical complexities, like:

- **Quantum Channel Abstraction**: evaluates whether the language abstracts the quantum channels used for inter-node communication, essential for managing quantum information transfer.
- **Quantum Interconnection Network**: assesses the language's ability to abstract the structure of quantum interconnection networks connecting multiple QPUs, a foundational aspect for scalable QPU architectures.
- **Communication Protocols**: identifies whether the abstraction of the communication protocol is allowed or has to be defined by the user when programming. Abstraction is essential to allow the compiler to optimize techniques according to the context.

Development layer: programming languages for DQC must incorporate features that provides users to perform distributed quantum computing while abstracting away the complexities of the underlying quantum network.

- **High-Level Quantum Communication Instructions**: considers whether the language provides high-level commands to simplify distributed quantum operations, facilitating programming efficiency and code readability. These instructions allow the compiler to provide functions with semantic context, allowing it to perform optimizations between the different possible physical implementations.
- **Data Structures for Logical Topology**: determines the language's support for data structures that abstract the physical topology of the distributed system over a logical topology, allowing easier specification of quantum node relationships and delegating to the compiler the responsibility for matching the code to the target topology.

Other characteristics: key features for an IR to enable the correct and efficient development of new tools.

- **Real Quantum Computing Inspired:** specifies whether the language is intended to perform real quantum computation or only simulated quantum computation. This feature aims to eliminate all languages that include instructions that are not allowed in the quantum model, such as perfect copying of generic states.
- **Hybrid programming:** this feature is crucial for enabling the orchestration between classical and quantum computing devices, facilitating tasks such as optimization problems.

Table 3 shows the comparison between the different languages discussed in Section 2 (related work) together with the IR proposed in this work: NetQIR.

In particular, NetQASM and InQuIR do not fully abstract the quantum channel. InQuIR requires the programmer to explicitly call operations such as `genEnt` to create entangled pairs and `entSwp` to perform entanglement swapping between QPUs. This exposes the physical routing of qubits and limits the flexibility of the compiler to adapt to different network configurations. Similarly, NetQASM uses `create_epr` to establish entanglement between nodes, providing only a minimal abstraction and restricting the channel model to EPR-based links, without considering other communication resources such as GHZ states or multi-party entanglement.

Regarding the quantum interconnection network, both approaches offer at best a partial abstraction. Although entanglement-based communication is used, the fact that the programmer must manage the flow of entangled pairs (e.g., through manual swapping or addressing specific qubits) means that the logical network structure is not decoupled from the physical implementation. For instance, in InQuIR, if a QPU wants to communicate with a non-adjacent node, the user must explicitly define a chain of entSwp instructions. This introduces rigid dependencies on the physical topology and prevents the compiler from transparently handling the routing of quantum information.

Concerning the communication protocols, there is no abstraction in this feature because the programmer has to decide how to interact with the communication qubits. Furthermore, the development layer is not implemented as neither quantum communication instructions nor structures to define a logical topology are defined. Hybrid programming is not allowed in languages such as NetQASM or InQuIR which are specific to quantum device programming.

With respect to QMPI, it allows abstraction in the fields indicated, except that it is not intended for real quantum computation because it has collective operations that are not meaningful due to the non-cloning theorem, such as the `Allscatter` or `Allgather` operation. It is also important to specify that QMPI is a message passing interface, so it is not an IR, just as MPI is not an IR. On the other hand, QMPI, being a message passing interface and not an IR, could allow this type of programming depending on its future implementations.

Finally, NetQIR, the intermediate representation for DQC proposed in this paper, would meet the above requirements by abstracting each part of the layered model. NetQIR, by extending QIR, which in turn extends LLVM, ensures hybrid programming by integrating quantum and classical programming in the same IR.

5.2. Justification of the layered abstraction model

This section aims to analyze the use of different communication protocols such as teledata or telegate against collective operations such as "`expose`". Additionally, it also looks at the comparison between different network topologies. This will allow us to analyze how the computational resources consumed can vary depending on the communication protocol used on different network topologies.

The analysis focuses on how resource consumption varies depending on the selected communication protocol, the physical topology of the quantum network, and the number of QPUs involved. By abstracting these aspects through semantic directives and delegating the decision-making to the compiler, the model allows for adaptation to the underlying infrastructure and algorithmic needs, improving overall efficiency.

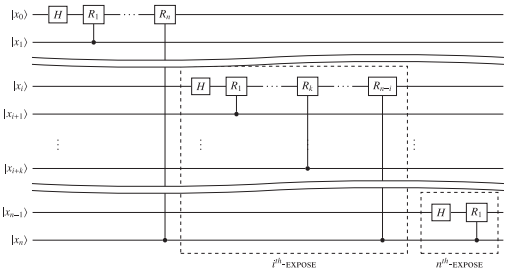


Fig. 12. Circuit for the calculation of the Quantum Fourier Transformation (QFT), where n calculations are performed on n qubits.

Two main metrics are considered: the total number of **communication qubits consumed** during circuit execution, and the number of **communication qubits each QPU must reserve** to support the communication process. These results serve to highlight the benefits of decoupling communication details from the algorithmic description, as promoted by the layered abstraction model.

To do this, the circuit in Fig. 12 was used, which represents the computation of a QFT using $n+1$ qubits. This process can be separated into n epochs, where in epoch i controlled gates are applied to qubit $|x_i\rangle$, equivalent to an `expose` function. In this use case, the QPU i is assigned the qubit $|x_i\rangle$. Therefore, the growth of QPUs will imply a growth in the number of qubits in the circuit.

It is important to note that this partitioning is done on an "ad hoc" basis for the evaluation. No circuit partitioning strategy is used. The objective is to assign a qubit to a QPU in order to maximize the number of communications in the circuit. In this case, QFT is selected as the circuit to be tested as it is resource intensive in terms of gating all the qubits. Therefore, by having each qubit in a distributed QPU, it forces the consumption of communication qubits by having to perform the controlled gate remotely.

On the other hand, the topologies to be tested are shown in Fig. 13, which are the direct connection option and the interconnection option via a communicator.

- **Direct interconnection**: this type of connection is peer-to-peer so that each QPU is connected to each of the other QPUs. This allows a direct connection between each pair of QPUs without unnecessary hops but has the disadvantage of having a tightly coupled network, making it problematic to add new nodes and requiring a large number of communication qubits.
- **Topology via one communicator**: In this case, access to the distributed system is managed through a quantum communicator (or quantum router). Each QPU is connected only to this central node, which is responsible for relaying quantum information between them. One of the main advantages of this abstraction is that it allows the compiler to choose the most suitable strategy depending on the context. In some situations, the communicator may assist the routing process by performing entanglement swapping between QPUs. In other cases, the communicator might directly establish entanglement links with the target QPU and transfer the qubit's state or apply remote operations, acting as an active participant in the communication. This flexibility highlights the benefit of leaving implementation decisions to the lower layers of the stack.

Continuing with the communication protocols evaluated, the analysis considers the use of `teledata`, `telegate`, and `expose` in both topologies. For the `expose` function, the implementation shown in Fig. 9 is used, where a GHZ state is generated to connect all involved QPUs.

In the case of `teledata`, the source QPU must send the qubit to the destination node, allow the operation to be performed, and then retrieve the updated state — requiring two EPR pairs per operation. The `telegate` protocol, which executes the operation remotely without moving the qubit, requires only a single EPR pair. Lastly, the `expose` directive leverages a shared GHZ state, allowing multiple QPUs to access the same logical qubit. This significantly reduces the number of required communication qubits, especially as the number of QPUs increases, since the cost of GHZ generation is shared across the participants.

Fig. 14 shows the results obtained, always from the perspective of resource consumption on a single QPU in the system. As can be seen in Figs. 14(a) and 14(b), the number of qubits consumed for a QPU increases with the QPUs connected, as it implies a higher number of communications between the circuit. With regard to protocols, for both topologies, the protocol with the highest qubit consumption is `teledata`, followed by `telegate`, and finally `expose`, which shows the lowest consumption. This highlights the importance of selecting an appropriate communication protocol for each specific task. By using high-level semantic functions, the compiler gains the flexibility to optimize communication by selecting the most efficient strategy based on the topology and algorithmic context. This case is particularly comprehensive as QFT has been selected as the test circuit, as it has controlled gates between all the qubits.

On the other hand, Fig. 14(c) shows the number of communication qubits each QPU must reserve to communicate with the distributed system. As expected, this number grows linearly in the case of a directly connected topology, requiring one or two additional communication qubits for each new QPU added to the system. In contrast, in the via one communicator topology, this requirement remains constant, as only one or two communication qubits are needed per QPU, regardless of the system size, since the quantum communicator handles the interconnections. As shown in Figs. 14(a) and 14(b) above, this also implies that the consumption of communication qubits is higher, as the information has to flow through the communicator.

It is important to contextualize the resource usage of NetQIR or the IRs in general. While custom low-level code can, in theory, achieve superior performance, this comes at the cost of portability, maintainability, and development complexity. The primary advantage of adopting an IR lies in its ability to act as a unifying abstraction across multiple frontends and backends. This enables the reuse of mature, optimized compiler infrastructures and promotes rapid prototyping and code generation from high-level languages.

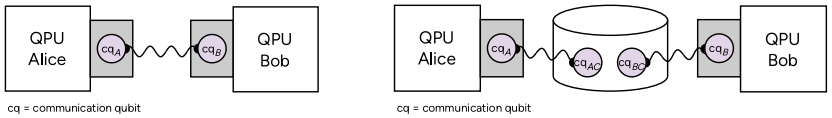
NetQIR inherits these advantages by offering a structured IR for distributed quantum computation, allowing the integration of quantum-specific optimizations without sacrificing compatibility with classical toolchains (due to the extension of QIR and LLVM). In practice, the abstraction introduced by NetQIR empowers developers to generate code more efficiently and consistently across architectures, while still enabling backend-specific optimizations through lower layers. Therefore, while the raw performance of hand-optimized code may remain unmatched, the productivity and correctness benefits of IR-based development — especially in complex distributed environments — make NetQIR a scalable alternative.

Overall, these results reinforce the value of the proposed layered abstraction model. By abstracting protocol and topology details through collective operations, the compiler is empowered to make optimal decisions, improving scalability and resource efficiency in distributed quantum systems.

6. Conclusions

This work addresses two key objectives aimed at mitigating some of the challenges identified in the literature regarding the development of compilation frameworks and software tools for distributed quantum computing.

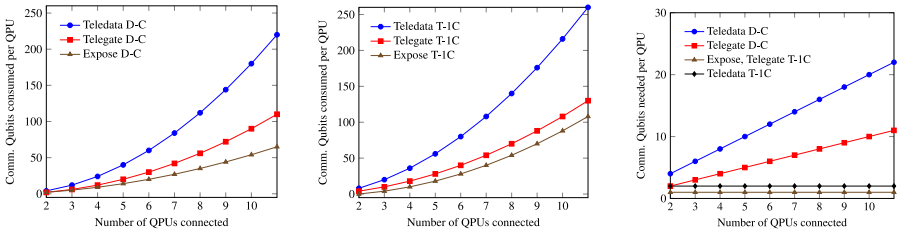
Firstly, the need to establish a common framework for the development of new IRs related to DQC is addressed by proposing the *layered*



(a) Direct interconnection.

(b) Interconnection via a communicator.

Fig. 13. Examples of different quantum network topologies for connecting QPUs within a QLAN.



(a) Consumed qubits in Directly Connected Topology (D-C). Refers to the number of communication qubits that have been measured for each single QPU (per-QPU).

(b) Consumed qubits in Topology via one communicator (T1-C). Refers to the number of communication qubits that have been measured for each single QPU (per-QPU).

(c) Communication qubits needed in the quantum register per-QPU. Refers to how many are needed at the same time for each single QPU.

Fig. 14. Comparison of the number of communication qubits consumed and needed for the QFT circuit, in function of the number of connected QPUs, the communication protocol used and the existing physical topology. It is important to note that, because of how the partitioning of the distributed circuit has been defined, one qubit has been assigned to one QPU, therefore the number of connected QPUs is equal to the number of qubits used exclusively for the QFT.

abstraction model. This model not only provides a scalable architecture but also establishes a foundation for optimization opportunities in DQC. By implementing functions that facilitate quantum data distribution across logical topologies, NetQIR reduces the complexity of DQC programming, allowing compilers to dynamically optimize based on high-level semantic directives. This model effectively addresses challenges observed in other IRs, such as NetQASM and InQuIR, which either lack flexibility in protocol handling or are too closely tied to specific network assumptions.

Secondly, an IR has been proposed in this work that meets the requirements objectively specified in the abstraction layer model, called NetQIR. It is an innovative extension of QIR for DQC, designed to address current limitations in scalability and interoperability in distributed quantum environments, allowing the hybrid programming and the use of LLVM common tools. NetQIR offers a flexible IR designed to handle quantum and classical communications across multiple QPUs by introducing high-level abstractions and communication directives. Unlike previous solutions, NetQIR integrates high-level quantum communication functions — such as point-to-point (`qsend`, `qrecv`) and collective operations (`scatter`, `gather`, `reduce`, `expose`) — enabling developers to program complex distributed algorithms with ease. This design abstracts the underlying network layer, allowing NetQIR to efficiently map communication protocols such as teledata and telegate based on the topology and specific requirements of the quantum network.

In this work, an abstraction model and IR are proposed, aiming to improve the software stack for DQC. Therefore, a compiler or optimizer that improves results such as fidelities or computational resource usage is not being proposed. This implies that, instead of working with empirical results, the focus has been on a discussion of the different existing languages for communication or quantum computing and, on the other hand, the evaluation of the impact on computational resource consumption of different communication protocols in a real problem.

The comparison between languages allows us to observe that not all of them define a direct abstraction like the one proposed in the abstraction layer model in this work. This is important, as the goal is to propose a system similar to the one already used in classical HPC environments, but incorporating the characteristics and constraints of quantum computing (especially since alternatives like QMPI propose operations that do not comply with the no-cloning theorem).

Moreover, being able to evaluate different communication protocols in various interconnection networks has also made it possible to highlight how important it is to abstract away from these features — belonging to the network layer — so that the operating system or compiler can manage which protocol or routing strategy to use.

Although hand-optimized low-level implementations may achieve maximum performance, the use of an IR such as NetQIR offers a more practical trade-off between abstraction and efficiency. By enabling the reuse of mature compilation toolchains and supporting code generation from high-level languages, NetQIR accelerates development and enhances portability across platforms. This abstraction layer is particularly valuable in DQC, where complexity and heterogeneity make manual low-level programming impractical at scale.

It is important to point out that NetQIR aims to establish an IR to develop the necessary tools for the future DQC, which in the NISQ era is not yet available due to the accumulated errors in quantum communication networks for computing. The main objective is to work on having the right languages and tools for when the FTQC era is reached.

Future work on NetQIR should prioritize developing tools that simplify its integration into new software projects. Additionally, testing its interoperability with various quantum backends and exploring advanced optimization techniques would be valuable. A well-designed toolchain could improve the management of distributed resources in NetQIR, potentially reducing communication costs and enhancing qubit allocation strategies to further boost efficiency and scalability in distributed quantum systems.

CRediT authorship contribution statement

F. Javier Cardama: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Jorge Vázquez-Pérez:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **César Piñeiro:** Visualization, Validation, Supervision. **Tomás F. Pena:** Writing – review & editing, Validation, Supervision, Resources, Project administration,

Investigation, Funding acquisition, Conceptualization. **Juan C. Pichel:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Funding acquisition. **Andrés Gómez:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve language and readability. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Tomas F. Pena reports financial support and article publishing charges were provided by European Union. Tomas F. Pena reports financial support was provided by Government of Spain MINECO. Tomas F. Pena reports financial support was provided by Government of Galicia. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by MICINN through the European Union NextGenerationEU recovery plan (PRTR-C17.I1), the Galician Regional Government through "Planes Complementarios de I+D+I con las Comunidades Autónomas" in Quantum Communication, MINECO (grants PID2019-104834GB-I00, PID2022-141623NB-I00 and PID2022-137061OB-C22), Consellería de Cultura, Educación e Ordenación Universitaria Galician Research Center accreditation 2024–2027 ED431G-2023/04, and the European Regional Development Fund (ERDF).

Data availability

All code is open source and has been indicated by links to the GitHub repository in the manuscript. Everyone is welcome to contribute to open source and to contact us with any questions.

References

[1] R.P. Feynman, Simulating physics with computers, Internat. J. Theoret. Phys. 21 (1982) 467–488, http://dx.doi.org/10.1007/BF02650179/METRICS, URL https://link.springer.com/article/10.1007/BF02650179.

[2] I.L. Markov, Limits on fundamental limits to computation, Nat. 2014 512: 7513 512 (2014) 147–154, http://dx.doi.org/10.1038/nature13570, URL https://www.nature.com/articles/nature13570.

[3] H. Buhrman, R. Cleve, A. Wigderson, Quantum vs. classical communication and computation, in: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, 1998, pp. 63–68.

[4] H. Riel, Quantum computing technology, Tech. Dig. - Int. Electron Devices Meet. IEDM 2021-December (2021) 1.3.1–1.3.7, http://dx.doi.org/10.1109/IEDM19574.2021.9720538.

[5] B. Koen, A. Sarkar, T. Hubregtsen, M. Serrao, A.A. Mouedenne, A. Yadav, A. Krol, I. Ashraf, C.G. Almudever, Quantum computer architecture toward full-stack quantum accelerators, IEEE Trans. Quantum Eng. 1 (2020) http://dx.doi.org/10.1109/TQE.2020.2981074.

[6] T. Haner, D.S. Steiger, T. Hoefler, M. Troyer, Distributed quantum computing with QMPI, Int. Conf. High Perform. Comput. Netw. Storage Anal. SC (2021) http://dx.doi.org/10.1145/3458817.3476172/SUPPL_FILE/TRENDS, URL https://dl.acm.org/doi/10.1145/3458817.3476172.

[7] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, M. Roetteler, Q#: Enabling scalable quantum computing and development with a high-level DSL, in: Proceedings of the Real World Domain Specific Languages Workshop 2018, in: RWDSL2018, Association for Computing Machinery, New York, NY, USA, 2018, http://dx.doi.org/10.1145/3183895.3183901.

[8] A.S. Green, P.L. Lumsdaine, N.J. Ross, P. Selinger, B. Valiron, Quipper: A scalable quantum programming language, ACM SIGPLAN Not. 48 (2013) 333–342, http://dx.doi.org/10.1145/2499370.2462177/SUPPL_FILE/PLDI154.ZIP, URL https://dl.acm.org/doi/10.1145/2499370.2462177.

[9] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F.J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, J.M. Chow, A.D. Córcoles-Gonzales, A.J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S.D.L.P. González, E.D.L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I.F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B.G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, L. ukasz Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F.J. Martín-Fernández, D.T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D.M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L.J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A.R. Davila, R.H.P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J.A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J.A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I.Y. Akhalwaya, C. Zoufal, Qiskit: An open-source framework for quantum computing, 2019, http://dx.doi.org/10.5281/zenodo.2562111.

[10] M.A. Serrano, J.A. Cruz-Lemus, R. Perez-Castillo, M. Piattini, Quantum software components and platforms: Overview and quality assessment, ACM Comput. Surv. 55 (2022) 164, http://dx.doi.org/10.1145/3548679, URL https://dl.acm.org/doi/10.1145/3548679.

[11] A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[12] J. Stanier, D. Watson, Intermediate representations in imperative compilers, ACM Comput. Surv. 45 (2013) http://dx.doi.org/10.1145/2480741.2480743, URL https://dl.acm.org/doi/10.1145/2480741.2480743.

[13] T.S. Metodi, S.D. Gasster, Design and implementation of a quantum compiler, in: E.J. Donkor, A.R. Pirich, H.E. Brandt (Eds.), Quantum Information and Computation VIII, vol. 7702, International Society for Optics and Photonics, SPIE, 2010, p. 77020S, http://dx.doi.org/10.1117/12.852548.

[14] K. Hietala, R. Rand, S.-H. Hung, X. Wu, M. Hicks, Verified optimization in a quantum intermediate representation, 2019, URL https://arxiv.org/abs/1904.06319v4.

[15] C.G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, K. Bertels, The engineering challenges in quantum computing, Proc. the 2017 Des. Autom. Test Eur. DATE 2017 (2017) 836–845, http://dx.doi.org/10.23919/DATE.2017.7927104.

[16] R. Beals, S. Brierley, O. Gray, A.W. Harrow, S. Kutin, N. Linden, D. Shepherd, M. Stather, Efficient distributed quantum computing, Proc. R. Soc. A: Math. Phys. Eng. Sci. 469 (2013) http://dx.doi.org/10.1098/RSPA.2012.0686, URL http://dx.doi.org/10.1098/rspa.2012.0686orvia. http://rspa.royalsocietypublishing.org.

[17] S.W. Loke, From distributed quantum computing to quantum internet computing: An overview, 2022, URL https://arxiv.org/abs/2208.10127v2.

[18] R. Wakizaka, Towards reliable distributed quantum computing on quantum interconnects, ACM Int. Conf. Proceeding Ser. (2023) 114–116, http://dx.doi.org/10.1145/3594671.3594691, URL https://dl.acm.org/doi/10.1145/3594671.3594691.

[19] R.V. Meter, T.D. Ladd, A.G. Fowler, Y. Yamamoto, Distributed quantum computation architecture using semiconductor nanophotonics, Int. J. Quantum Inf. 8 (2009) 295–323, http://dx.doi.org/10.1142/S0219749910006435, URL http://arxiv.org/abs/0906.2686.

[20] S. Rodrigo, S. Abadal, E. Alarcon, C.G. Almudever, Will quantum computers scale without inter-chip comms? A structured design exploration to the monolithic vs distributed architectures quest, 2020 35th Conf. Des. Circuits Integr. Syst. DCIS 2020 (2020) http://dx.doi.org/10.1109/DCIS51330.2020.9268630.

[21] D. Barral, F.J. Cardama, G. Díaz, D. Faílde, I.F. Llovo, M.M. Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas, J.C. Pichel, T.F. Pena, A. Gómez, Review of distributed quantum computing. From single QPU to high performance quantum computing, 2024, URL https://arxiv.org/abs/2404.01265v1.

[22] S.-H. Wei, B. Jing, X.-Y. Zhang, J.-Y. Liao, C.-Z. Yuan, B.-Y. Fan, C. Lyu, D.-L. Zhou, Y. Wang, G.-W. Deng, et al., Towards real-world quantum networks: A review, Laser & Photonics Rev. 16 (3) (2022) 2100219.

[23] F.T. Chong, D. Franklin, M. Martonosi, Programming languages and compiler design for realistic quantum hardware, Nature 549 (7671) (2017) 180–187.

[24] S. Wehner, D. Elkouss, R. Hanson, Quantum internet: A vision for the road ahead, Science 362 (6412) (2018) eaam9288, http://dx.doi.org/10.1126/science.aam9288, URL https://www.science.org/doi/abs/10.1126/science.aam9288.

[25] J. Illiano, M. Caleffi, A. Manzalini, A.S. Cacciapuoti, Quantum internet protocol stack: A comprehensive survey, Comput. Netw. 213 (2022) 109092, http://dx.doi.org/10.1016/j.comnet.2022.109092.

[26] K. Azuma, S.E. Economou, D. Elkouss, P. Hilaire, L. Jiang, H.-K. Lo, I. Tzitrin, Quantum repeaters: From quantum networks to the quantum internet, Rev. Modern Phys. 95 (4) (2023) 045006.

[27] O. Bel, M. Kiran, Simulators for quantum network modelling: A comprehensive review, 2024, URL https://arxiv.org/abs/2408.11993. arXiv:2408.11993.
[28] A. Dahlberg, B.V.D. Vecht, C.D. Donne, M. Skrzypczyk, I.T. Raa, W. Kozlowski, S. Wehner, NetQASM—a low-level instruction set architecture for hybrid quantum–classical programs in a quantum internet, Quantum Sci. Technol. 7 (2022) 035023, http://dx.doi.org/10.1088/2058-9565/AC753F, URL https://iopscience.iop.org/article/10.1088/2058-9565/ac753f.
[29] A. Dahlberg, S. Wehner, SimulaQron—a simulator for developing quantum internet software, Quantum Sci. Technol. 4 (2018) 015001, http://dx.doi.org/10.1088/2058-9565/AAD56E, URL https://iopscience.iop.org/article/10.1088/2058-9565/aad56e.
[30] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J.d. Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, S. Wehner, NetSquid, a NETwork simulator for quantum information using discrete events, Commun. Phys. 2021 4: 1 4 (2021) 1–15, http://dx.doi.org/10.1038/s42005-021-00647-8, URL https://www.nature.com/articles/s42005-021-00647-8.
[31] X. Wu, A. Kolar, J. Chung, D. Jin, T. Zhong, R. Kettimuthu, M. Suchara, SeQUeNCe: A customizable discrete-event simulator of quantum networks, Quantum Sci. Technol. 6 (2020) http://dx.doi.org/10.1088/2058-9565/ac22f6, URL https://arxiv.org/abs/2009.12000v1.
[32] S. Diadamo, J. Nötzel, B. Zanger, M.M. Beşe, QuNetSim: A software framework for quantum networks, IEEE Trans. Quantum Eng. 2 (2021) http://dx.doi.org/10.1109/TQE.2021.3092395.
[33] QIR Alliance: https://qir-alliance.org. URL https://github.com/qir-alliance/qir-spec.
[34] A.S. Tanenbaum, H. Bos, Modern Operating Systems, Pearson Education, Inc., 2015.
[35] G.F. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: Concepts and Design, pearson education, 2005.
[36] M.P. Forum, MPI: A message-passing interface standard, 1994.
[37] J. Bruck, D. Dolev, C.-T. Ho, M.-C. Roşu, R. Strong, Efficient message passing interface (MPI) for parallel computing on clusters of workstations, in: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures, 1995, pp. 64–73.
[38] A. Skjellum, N. Doss, K. Viswanathan, A. Chowdappa, P. Bangalore, Extending the message passing interface (MPI), in: Proceedings Scalable Parallel Libraries Conference, 1994, pp. 106–118, http://dx.doi.org/10.1109/SPLC.1994.376998.
[39] C. Delle Donne, M. Iuliano, B. Van Der Vecht, G. Ferreira, H. Jirovská, T. Van Der Steenhoven, A. Dahlberg, M. Skrzypczyk, D. Fioretto, M. Teller, et al., An operating system for executing applications on quantum network nodes, Nature 639 (8054) (2025) 321–328.
[40] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, O. Zinenko, MLIR: Scaling compiler infrastructure for domain specific computation, in: 2021 IEEE/ACM International Symposium on Code Generation and Optimization, CGO, 2021, pp. 2–14, http://dx.doi.org/10.1109/CGO51591.2021.9370308.
[41] A. McCaskey, T. Nguyen, A MLIR dialect for quantum assembly languages, in: 2021 IEEE International Conference on Quantum Computing and Engineering, QCE, IEEE, 2021, pp. 255–264, http://dx.doi.org/10.1109/QCE52317.2021.00043.
[42] K. Hietala, R. Rand, S.-H. Hung, X. Wu, M. Hicks, A verified optimizer for quantum circuits, Proc. ACM Program. Lang. 5 (POPL) (2021) http://dx.doi.org/10.1145/3434318.
[43] L. Foundation, LLVM Assembly Language Reference Manual, URL https://releases.llvm.org/2.6/docs/LangRef.html.
[44] S. Nishio, R. Wakizaka, InQuIR: Intermediate representation for interconnected quantum computers, 2023, URL https://arxiv.org/abs/2302.00267v1.
[45] I. Quantum, Technology for the quantum future: Development roadmap, 2025, URL https://www.ibm.com/quantum/technology#roadmap. Online, (Accessed 16 June 2025).
[46] S.K. Moore, The future of quantum computing is modular, IEEE Spectr. (2025) URL https://spectrum.ieee.org/quantum-computers. (Accessed 16 June 2025).
[47] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, W.K. Wootters, Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels, Phys. Rev. Lett. 70 (1993) 1895–1899, http://dx.doi.org/10.1103/PhysRevLett.70.1895, URL https://link.aps.org/doi/10.1103/PhysRevLett.70.1895.
[48] D. Gottesman, I.L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, Nature 402 (6760) (1999) 390–393, http://dx.doi.org/10.1038/46503.
[49] D. Ferrari, A.S. Cacciapuoti, M. Amoretti, M. Caleffi, Compiler design for distributed quantum computing, IEEE Trans. Quantum Eng. 2 (2021) http://dx.doi.org/10.1109/TQE.2021.3053921.
[50] D. Ferrari, S. Carretta, M. Amoretti, A modular quantum compilation framework for distributed quantum computing, IEEE Trans. Quantum Eng. 4 (2023) http://dx.doi.org/10.1109/TQE.2023.3303935.
[51] B. van der Vecht, A.T. Yücel, H. Jirovská, S. Wehner, Qoala: An application execution environment for quantum internet nodes, 2025, URL https://arxiv.org/abs/2502.17296. arXiv:2502.17296.
[52] J. Vázquez-Pérez, F.J. Cardama, NetQIR/netqir-spec: v0.0.1, 2024, http://dx.doi.org/10.5281/zenodo.13142521.
[53] N.M. Neumann, R. van Houte, T. Attema, Imperfect distributed quantum phase estimation, in: Computational Science–ICCS 2020: 20th International Conference, Amsterdam, the Netherlands, June 3–5, 2020, Proceedings, Part VI 20, Springer, 2020, pp. 605–615.
[54] T.J. Parr, R.W. Quong, ANTLR: A predicated-LL (k) parser generator, Softw.: Pr. Exp. 25 (7) (1995) 789–810.
[55] T. Parr, The Definitive ANTLR 4 Reference, The Pragmatic Bookshelf, 2013.

Chapter 8

NetQMPI: an MPI-inspired library for programming distributed quantum aplications over quantum networks using NetQASM SDK

The content of this chapter is presented in the format of a scientific article which is currently pending publication:

> F. J. Cardama and T. F. Pena, "NetQMPI: An MPI-Inspired library for programming Distributed Quantum Applications over Quantum Networks using NetQASM SDK", Dec. 2025
>
> - **GitHub code:** `https://github.com/netqir/netqmpi`

This publication specifically addresses and fulfills Objective O2 of this thesis. For further details regarding the publication status of this manuscript, the reader is referred to the Results: unpublished articles section.

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier XX.ZZZZ/ACCESS.YYYY.XXXXXXX

NetQMPI: An MPI-Inspired library for programming Distributed Quantum Applications over Quantum Networks using NetQASM SDK

F. JAVIER CARDAMA[1] (Student Member, IEEE), and TOMÁS F. PENA[1,2] (Senior Member, IEEE)

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain
[2]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain

Corresponding author: F. Javier Cardama (email: javier.cardama@usc.es).

This work was funded by the European Union EuroHPC program with grant agreement 101194491 and by MICIU/AEI/10.13039/501100011033 with project number PCI2025-163229, the Agencia Estatal de Investigación (Spain) (PID2022-141623NB-I00), the Consellería de Cultura, Educación e Ordenación Universitaria Galician Research Center accreditation 2024–2027 ED431G-2023/04, and the European Regional Development Fund (ERDF).

ABSTRACT Distributed Quantum Computing (DQC) is essential for scaling quantum algorithms beyond the limitations of monolithic NISQ devices. However, the current software ecosystem forces developers to manually orchestrate low-level network resources, such as entanglement generation (EPR pairs) and classical synchronization, leading to verbose, error-prone, and non-scalable code. This paper introduces **NetQMPI**, a high-level Python framework that adapts the Message Passing Interface (MPI) standard to the quantum domain using the Single Program Multiple Data (SPMD) paradigm. Built as a middleware over the NetQASM SDK, NetQMPI abstracts the underlying physical topology, automating network initialization and resource management through a unified `Communicator` interface. We propose semantic point-to-point primitives and novel collective operations—such as `expose` and `unexpose`—that address the constraints of the No-Cloning Theorem by leveraging multipartite entanglement for data distribution. Our comparative analysis demonstrates that NetQMPI decouples algorithmic logic from network size, reducing the code complexity for generating an N-node GHZ state from $\mathcal{O}(N^2)$ to constant complexity $\mathcal{O}(1)$. Furthermore, the framework ensures backend agnosticism, enabling the seamless execution of high-level applications on rigorous physical simulators, such as NetSquid (via SquidASM), and future quantum hardware adhering to the NetQASM standard.

INDEX TERMS Distributed Quantum Computing, High-Performance Computing, MPI, Quantum Networks.

I. INTRODUCTION

In recent years, quantum computing has emerged as a computational paradigm capable of offering exponential speedups for problems intractable on classical machines [1], [2], such as integer factorization [3] and unstructured search [4]. However, realizing these algorithms at scale requires a substantial number of qubits and robust error correction capabilities, which remain challenging for current hardware in the Noisy Intermediate-Scale Quantum (NISQ) era [5].

To overcome these scaling limitations, Distributed Quantum Computing (DQC) has emerged as a promising path. By interconnecting multiple Quantum Processing Units (QPUs) via a quantum network, it is possible to create a virtual cluster capable of executing large-scale algorithms [6]–[8]. However, orchestrating computations across such a distributed infrastructure introduces engineering complexities [9]. Unlike classical networks, quantum networks require the management of fragile resources such as entanglement, Einstein-Podolsky-Rosen (EPR) pairs, and the tight synchronization of classical and quantum control planes [10]–[12].

Nevertheless, the transition from monolithic to distributed quantum algorithms introduces significant software engineering challenges [13]. Most existing software tools require researchers to program at the physical or link layer, where they must explicitly manage elementary operations [6], [14], [15]. At this level, developers are often burdened with defining hardware-specific parameters such as photon emission timing, fiber attenuation, and memory decoherence , while also

manually orchestrating network protocols like entanglement generation (EPR pairs), routing, and the tight synchronization of classical and quantum control planes [16].

This low-level approach not only increases development complexity but also couples the algorithmic logic to specific hardware topologies or simulation backends, severely limiting code portability and maintainability. In the realm of classical High-Performance Computing (HPC), this challenge was successfully addressed by the Message Passing Interface (MPI) standard, which unified distributed programming, typically following the Single Program Multiple Data (SPMD) paradigm. While theoretical proposals for a Quantum Message Passing Interface (QMPI) exist [17], the ecosystem still lacks a functional, standardized implementation that bridges the gap between high-level algorithmic design and the execution on rigorous quantum network stacks.

A. CONTRIBUTIONS

In this work, we present **NetQMPI**, a high-level MPI-inspired Python library for programming distributed quantum applications over quantum networks. NetQMPI acts as a middleware layer over the standard NetQASM Software Development Kit (SDK), allowing developers to write distributed quantum algorithms using high-level semantics while ensuring execution on rigorous network simulators.

The main objectives and novel contributions of this work are:

- **Adoption of the SPMD Paradigm:** We propose a shift from role-based scripting to a unified Single Program Multiple Data model. By injecting logical `rank` identifiers, NetQMPI enables a single source file to orchestrate complex protocols across N nodes, thereby decoupling code complexity from network size.
- **Abstraction of Physical Resources:** We introduce the `QMPICommunicator` and semantic primitives (e.g., `qsend`, `qrecv`) that abstract the management of EPR sockets and classical control planes, which allows researchers to focus on algorithmic logic rather than the intricate details of entanglement generation and fidelity management.
- **Novel Collective Operations:** We define and implement quantum-specific collective operations, such as `expose` and `unexpose`. These primitives address the constraints of the No-Cloning Theorem by leveraging multipartite entanglement to share quantum information across the communicator without physical copying.
- **Backend-Agnostic Execution:** By targeting the standardized NetQASM Instruction Set Architecture (ISA), NetQMPI serves as a unified interface that can execute applications on high-fidelity simulators (such as NetSquid via SquidASM) and is forward-compatible with future quantum hardware, bridging the gap between abstract algorithm design and realistic physical simulation.

This paper is organized as follows. Section II introduces the related work including tools for DQC programming and simulation. Section III provides an overview of the NetQASM framework, distinguishing between its ISA and the SDK, while highlighting the programming complexity inherent in the current ecosystem. Section IV details the architecture of NetQMPI as well as the creation process on NetQASM SDK and the high-level semantic functions provided. Section V presents a comparative analysis of the various tools in the literature described in the related work section, and, finally, Section VI concludes the paper.

II. RELATED WORK

Recent advances in quantum algorithms have driven considerable efforts in developing software tools for programming and simulating distributed quantum systems. This section presents the most relevant tools in this domain, distinguishing between high-level abstractions and low-level physical simulators.

A. SOFTWARE TOOLS FOR PROGRAMMING DISTRIBUTED QUANTUM SYSTEMS

To manage the complexity of distributed quantum states, several frameworks have been proposed to abstract the underlying physical layer.

QuNetSim [18] provides a high-level Python API for simulating network protocols (e.g., Quantum Key Distribution (QKD), teleportation) without dealing with hardware physics.

- *Strengths:* It offers an extremely gentle learning curve and built-in implementations of standard protocols.
- *Weaknesses:* It is purely a simulator with no path to physical hardware execution; its abstraction level hides the resource management details (like entanglement fidelity) required for realistic system programming.

Interlinq [19] offers a high-level abstraction layer built upon the QuNetSim communication framework, designed to automate circuit partitioning and message scheduling across distributed nodes.

- *Strengths:* It simplifies distributed execution by automating circuit partitioning and teleportation scheduling, allowing monolithic circuits to be run across multiple nodes (“circuit knitting”, i.e., splitting a quantum program to fit onto smaller devices linked by classical communication [20]) without manual intervention for small-scale systems.
- *Weaknesses:* It suffers from severe scalability issues, where execution time and memory usage grow exponentially with the number of qubits and nodes. Additionally, its automatic partitioner lacks advanced optimization techniques, leading to redundant teleportations and communication overhead, while imposing rigid constraints on node configurations.

QMPI, proposed by Häner et al. [17], introduces the theoretical foundation for applying the Message Passing Interface (MPI) standard to quantum computing.

- *Strengths:* It leverages the mature MPI standard (commands like `Send`, `Recv`, `Bcast`) familiar to the HPC community.

- *Weaknesses:* There is no known implementation or functional application of this proposal, as the work is limited to defining the semantics. Furthermore, it does not address how to execute collective operations such as `Bcast` or `Allgather`; these primitives inherently imply data copying in classical MPI, which conflicts with the No-Cloning Theorem, and the proposal offers no mechanism to resolve this.

B. QUANTUM NETWORK SIMULATORS

This category encompasses tools designed to simulate any quantum network. These tools allow the execution of platform-independent code, often relying on lower-level engines for physical accuracy.

SimulaQron [21] focuses on the distributed nature of the network. It runs a simulation where each network node is a separate process on the host machine, communicating via classical sockets to imitate a real distributed architecture.

- *Strengths:* It enforces a strict separation of memory between nodes, making it ideal for verifying client-server logic and distributed application flows.
- *Integration:* While it acts as a standalone distributed backend, it is part of the QuTech ecosystem (a suite of standardized quantum networking tools developed at TU Delft) and was a precursor to modern execution environments.

SquidASM [22] is the specialized execution tool for NetQASM applications. It acts as a bridge between the application layer and the physical layer.

- *Strengths:* It allows developers to run NetQASM routines on a simulated network with high fidelity.
- *Integration:* SquidASM is explicitly built on top of NetSquid (see next subsection). It translates the high-level instructions from the SDK into discrete events that NetSquid can process, providing a realistic noise model while maintaining a programmable interface.

C. DISCRETE-EVENT SIMULATORS

At the lowest level of abstraction, these tools model the physical transmission of qubits, time-dependent noise, and hardware components (memories, repeaters, fibers).

NetSquid [23] is the state-of-the-art platform for modeling quantum networks at the physical layer.

- *Strengths:* It handles precise timing of photon emission, fiber loss, and quantum memory decoherence. It serves as the physics engine powering higher-level tools like SquidASM.
- *Weaknesses:* The complexity of defining hardware components makes it cumbersome for general application programming; it is designed for network architects rather than algorithm developers.

SeQUeNCe [24] is a customizable discrete-event simulator with a strong focus on the network layer (routing, resource management).

```
# NetQASM Assembly (Alice Side)
set R0 0          # Register for Qubit ID
qalloc R0         # Allocate Control Qubit
init R0           # Initialize to |0>

# -- Entanglement Generation --
store 1 @0        # Store arg: number of pairs
store 0 @1        # Store arg: type
create_epr 0 1 0 0 @0
wait_all @0       # Explicit wait for hardware

set R1 1          # Manually assign ID to EPR qubit
cnot R0 R1        # Local CNOT
meas R1 R2        # Measure EPR into Register R2
qfree R1          # Free EPR qubit resource
```

FIGURE 1. Snippet of NetQASM ISA Assembly corresponding to Alice's operations. It highlights the low-level management of registers (e.g., `R0`, `R1`) and memory addresses (e.g., `@0`) required by the architecture.

- *Strengths:* Excellent for studying network topology and protocol stacks, and available as open-source.
- *Weaknesses:* Like NetSquid, it presents a high barrier to entry for software developers simply wanting to write quantum applications using standard programming paradigms.

III. NETQASM SDK OVERVIEW

The foundation of the work presented in this paper is the NetQASM framework [22], which can be distinguished into two main components: the Instruction Set Architecture (ISA) and the Software Development Kit (SDK).

A. NETQASM ISA

NetQASM defines a low-level, platform-independent ISA explicitly designed for quantum networks. It serves as a common language that abstracts the heterogeneity of underlying hardware. Programs compiled into NetQASM assembly can be executed on various backends, ranging from high-fidelity physical simulators like SquidASM (based on NetSquid) or SimulaQron. This architecture separates the quantum logic from the control hardware, allowing for hybrid quantum-classical execution.

To illustrate the granularity of control required by this architecture, Figure 1 presents a snippet of Alice's operations. The execution flow begins with explicit resource management, where the program manually assigns a value to a classical register (`set R0 0`) to identify, allocate (`qalloc`), and initialize (`init`) a local control qubit. Subsequently, the entanglement generation process requires preparing the arguments in specific memory addresses—storing the number of pairs at `@0` and the type at `@1`—before triggering the `create_epr` instruction. Notably, the code must include a `wait_all` instruction to explicitly block execution until the physical hardware confirms the entanglement generation. Finally, the script manually maps the remote entangled qubit to a register (`set R1 1`), performs a local two-qubit gate (`cnot`), measures the result into a classical register (`meas`), and explicitly frees the quantum memory with `qfree`.

Alice (Control Node)

```
# Create a socket to send classical information
socket = Socket("Bob", "Alice", log_config=log_config)

# Create an EPR socket for entanglement generation
epr_socket = EPRSocket("Bob")

with NetQASMConnection(epr_sockets=[epr_socket]) as conn:
    # Create a qubit to teleport
    q = Qubit(conn)

    # Create EPR pairs
    epr = epr_socket.create_keep()[0]

    # Teleport
    q.cnot(epr)
    q.H()
    m1 = q.measure()
    m2 = epr.measure()

# Send the correction information
m1, m2 = int(m1), int(m2)

socket.send_structured(StructuredMessage("Corrections", (
    m1, m2)))

```

Bob (Target Node)

```
# Create a classical socket
socket = Socket("Alice", "Bob")

# Create an EPR socket
epr_socket = EPRSocket("Alice")

with NetQASMConnection(epr_sockets=[epr_socket]) as conn:
    # 1. Receive EPR qubit
    epr = epr_socket.recv_keep()[0]

    # 2. Flush to sync
    conn.flush()

    # 3. Get and apply the corrections
    m1, m2 = socket.recv_structured().payload

    if m2 == 1:
        epr.X()

    if m1 == 1:
        epr.Z()

    # 4. Final flush to sync
    conn.flush()
```

FIGURE 2. Implementation of a teleport protocol using NetQASM SDK. The developer is responsible for manually orchestrating the EPR socket creation, classical message passing, and synchronization (flush).

B. NETQASM SDK

To facilitate programming without writing raw assembly, the NetQASM SDK provides a Python interface for writing applications for specific network roles (e.g., "Alice" and "Bob"). A central abstraction in the SDK is the *EPR Socket*. By opening an EPR socket between two nodes, the programmer can request the generation of entanglement (e.g., creating Bell pairs) utilizing the underlying network link layer.

Figure 2 demonstrates the implementation of a teleport protocol using the NetQASM SDK, highlighting the steps required to transfer a qubit state from Alice to Bob.

C. PROGRAMMING COMPLEXITY

While the SDK abstracts the physical generation of entanglement, it does not provide high-level primitives for qubit transmission. Crucially, to transfer a quantum state from one node to another, the generation of an EPR pair is merely the first step. The programmer must manually implement the full quantum protocol communication *ad hoc*.

For example, to send a single qubit via teleportation [25] using the SDK, the developer must explicitly:

1) Create or request entanglement via the EPR Socket and wait for completion.
   - Alice: line 12.
   - Bob: line 9.
2) Perform local gates (CNOT, Hadamard) between the payload qubit and the entangled qubit.
   - Alice: lines 15–16.
3) Measure the qubits and obtain the classical outcomes.
   - Alice: Lines 17–18.
4) Send the classical bits to the receiver using a separate classical channel.
   - Alice: Line 23.
5) Receiver side: read these bits and apply the corresponding Pauli corrections (X or Z) to recover the state.
   - (Bob: Line 15–21).

This requirement to manually orchestrate classical and quantum synchronization for every transfer leads to verbose and error-prone code. This limitation is the primary motivation for NetQMPI, which encapsulates these complex workflows into simple, atomic communication primitives.

IV. NETQMPI

This section introduces NetQMPI, a high-level Python library designed to simplify the development of distributed quantum applications. Inspired by the standardized Message Passing Interface (MPI) used in classical HPC, NetQMPI adopts the Single Program Multiple Data (SPMD) paradigm, which enables developers to write a single, platform-independent source code that orchestrates quantum operations across multiple network nodes without manually managing the underlying physical entanglement resources.

A. SOFTWARE STACK

Figure 3 illustrates the architectural position of NetQMPI within the quantum network software ecosystem. NetQMPI acts as a middleware layer that sits directly on top of the NetQASM SDK.

A primary objective of this architecture is to bridge the gap between high-level algorithmic design and platform-specific execution. By abstracting the NetQASM SDK, NetQMPI hides the complexity of connection management and explicit instruction compilation. The user application interacts solely with NetQMPI primitives (e.g., `qsend`, `qrecv`), which are then translated at runtime into the corresponding

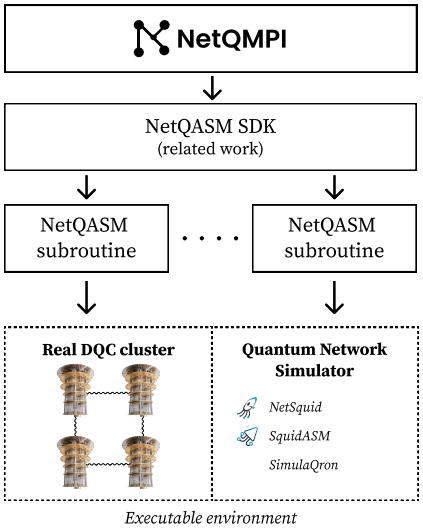


FIGURE 3. The NetQMPI software stack. The library wraps the verbose NetQASM SDK primitives, providing a unified API for the application layer while relying on the standard NetQASM ISA to interface with backend simulators, such as SquidASM or SimulaQron, or real DQC clusters supporting NetQASM.

NetQASM subroutines—handling `EPRSocket` lifecycle, `NetQASMConnection` management, and classical control flow transparently.

Because these primitives compile down to the standardized NetQASM ISA, NetQMPI is inherently backend-agnostic. This abstraction allows developers to write code that is completely decoupled from the underlying infrastructure, enabling the execution of the same source code on both *simulated networks* and *physical quantum networks* without modification.

Furthermore, this design leverages the robust capabilities of the ecosystem. Specifically, it allows users to utilize high-fidelity simulators like NetSquid (via the SquidASM interface) without facing its steep learning curve. While NetSquid is the state-of-the-art for modeling physical noise and network delays, programming it directly for application development is notoriously complex. NetQMPI enables researchers to validate algorithms using NetSquid's realistic physics engine while maintaining a high-level, abstract programming model.

B. CONSTRUCTION OF NETQMPI FROM NETQASM SDK

NetQMPI[1] is constructed as a wrapper that fundamentally alters the workflow of the NetQASM SDK. To achieve the transition from the role-based scripts required by the SDK to the Single Program Multiple Data (SPMD) paradigm, NetQMPI relies on a specific internal architecture. Figure 4 illustrates the organization of the source code.

[1]The complete source code is available at: https://github.com/netqir/netqmpi

```
netqmpi/
|-- sdk/
|   |-- external.py
|   |-- communicator/
|   |   |-- communicator.py
|   |-- primitives/
|   |   |-- collective/
|   |   |-- p2p/
```

FIGURE 4. Directory structure of the NetQMPI library. The `external.py` module orchestrates the multiprocess execution, while `communicator.py` manages the automated entanglement generation.

```python
from netqmpi.sdk.communicator import QMPICommunicator

def main(app_config=None, rank=0, size=1):
    # The rank is injected by NetQMPI
    # and used to initialize the local communicator
    COMM_WORLD = QMPICommunicator(rank, size, app_config)
    print(f"Hello, rank={rank} of {size} processes")
```

FIGURE 5. An example of the rank and size variable injection with the global communicator creation.

The core orchestration is handled by the `external.py` module, which manages the multiprocess environment mapping. The network state and automated entanglement generation are encapsulated within the `communicator` package. Finally, the specific communication logic is organized under `primitives`, divided into `p2p` and `collective` modules, whose specific implementations will be detailed in Subsections IV-C and IV-D, respectively.

1) Single Program Architecture

In the standard NetQASM SDK, developers must write N separate source files (e.g., `app_alice.py` and `app_bob.py`), one for each node in the network. NetQMPI unifies this into a single file, following the SPMD paradigm.

To enable this, NetQMPI utilizes the abstractions of *rank* and *size*. For readers unfamiliar with HPC terminology, the *rank* is a unique integer identifier assigned to each node in the distributed network, ranging from 0 to $size - 1$, while the *size* represents the total number of participating nodes in the communicator. These values are injected automatically by the library into the user's code at runtime, allowing the developer to define the logic for all nodes in a single function signature.

The user script must accept `rank` and `size` as arguments, which are then used to initialize the `QMPICommunicator`, as shown in Figure 5.

When the user executes this script using the Command Line Interface (CLI), the `cli.py` module parses the `-n` argument (number of processes) and triggers the `external.py` dispatcher. This dispatcher spawns the simulation processes using the NetQASM SDK, passing the specific rank to each instance. For example, executing the code with three nodes results in:

```
$ netqmpi -n 3 script.py
Hello, rank=0 of 3 processes
Hello, rank=2 of 3 processes
Hello, rank=1 of 3 processes
```

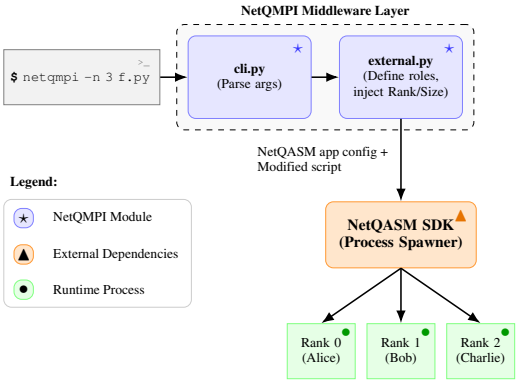


FIGURE 6. Execution flow of a NetQMPI application. The `external.py` module orchestrates the multiprocess environment, injecting rank information into the unified user script and delegating the process management to the NetQASM SDK.

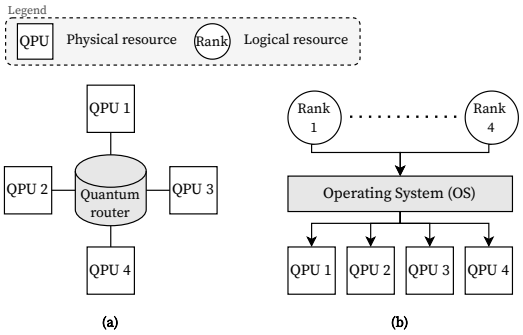


FIGURE 7. Logical abstraction for scheduling physical resources. (a) An example of physical resources interconnected in a physical quantum network. (b) The Communicator provides a logical layer where abstract Ranks are assigned to processes, and the OS/Middleware automatically handles the mapping to the available physical QPUs.

As illustrated in Figure 6, the execution flow is orchestrated by `external.py`. It parses the unified script, defines the network roles based on the requested size, and injects the corresponding rank information into the application's entry point. Finally, it hands over the configuration to the underlying NetQASM SDK, which manages the physical spawning of the runtime processes (e.g., Alice, Bob, Charlie).

2) Automated Network Initialization

The core abstraction that manages the logical connectivity between the distributed nodes is the `QMPICommunicator` class, defined in the `communicator` package.

Conceptually, a communicator acts as a virtualization layer that groups a set of logical processes connected through a logical network. This construct abstracts the underlying complexity of the infrastructure, where logical processes (identified by ranks) are mapped to physical resources (Quantum Processing Units) via a physical interconnection network. This relationship is visually described in Figure 7. Figure 7(a) depicts the physical reality where QPUs are connected via a Quantum Router, while Figure 7(b) illustrates how the software stack (OS/Middleware) maps the high-level Logical Ranks $(1 \ldots n)$ to these physical resources transparently.

This approach mirrors the architecture of classical distributed computing, allowing developers to focus on the algorithmic logic rather than the physical topology. For a comprehensive definition of this abstraction model in distributed quantum computing, we refer the reader to the architectural framework presented in [14], specifically Figure 6.

From an implementation perspective, the communicator automates resource management that is currently handled manually in the NetQASM SDK. In NetQASM, establishing a network requires the user to instantiate a `NetQASMConnection` and define specific `EPRSocket` objects for every pair of communicating nodes, leading to an $O(N^2)$ initialization complexity in fully connected topologies. NetQMPI resolves this by performing the following automated steps during initialization:

1) It assigns a unique `rank` to each logical process.
2) It iterates through the network configuration and automatically instantiates the necessary mesh of `EPRSocket` between all participating ranks.
3) It stores these sockets in an internal routing table within the communicator instance, making the management of entanglement resources transparent to the user.

Furthermore, NetQMPI handles the synchronization of the quantum control flow. While the NetQASM SDK requires manual `flush()` invocations, NetQMPI primitives include implicit flushing mechanisms or offer explicit `flush()` operations to ensure state consistency across the network.

C. POINT TO POINT FUNCTIONS

The fundamental requirement of a distributed quantum system is the ability to transfer quantum states between nodes. NetQMPI encapsulates the standard Quantum Teleportation Protocol into two atomic primitives: `qsend` and `qrecv`.

Crucially, these primitives represent a shift towards **high-level semantic communication**. From the programmer's perspective, the operation is defined simply as sending a specific qubit to a destination rank. All references to low-level hardware resources—such as explicit EPR pair generation, entanglement fidelity, specific QPU identifiers, or classical control signaling—are completely abstracted away. The developer focuses solely on the logical flow of quantum information.

1) The `qsend` Primitive

The `qsend(qubit, dest_rank)` function abstracts the complex role of the sender ("Alice"). When this semantic function is invoked, the library transparently handles the following underlying operations:

```python
def main(app_config=None, rank, size):
    # Initialize the communicator
    comm = QMPICommunicator(rank, size, app_config)

    if rank == 0:
        # Sender Logic (Alice)
        q = Qubit(comm.connection)
        q.H() # Prepare state |+>

        # Semantic send: "Send qubit q to Rank 1"
        # No EPR management required
        comm.qsend(q, 1)
        print("Rank 0: Qubit sent")

    elif rank == 1:
        # Receiver Logic (Bob)
        # Semantic receive: "Get qubit from Rank 0"
        q_received = comm.qrecv(0)

        # The qubit is ready for use
        m = q_received.measure()
        print(f"Rank 1: Qubit received, measure: {m}")
```

FIGURE 8. Example of point to point code using the `qsend` and `qrecv` primitives of NetQMPI.

1) Looks up the pre-established `EPRSocket` associated with the logical `dest_rank` in the routing table.
2) Requests the creation of an EPR pair from the network layer.
3) Performs the local Bell measurement (CNOT and Hadamard gates) between the user's payload qubit and the entangled qubit.
4) Measures the qubits and automatically transmits the classical measurement outcomes (2 bits) to the destination rank using the classical control plane.

2) The `qrecv` Primitive

The `qrecv(source_rank)` function abstracts the role of the receiver ("Bob"). It returns a handle to the received quantum state, allowing the program to continue execution immediately. Internally, the library:

1) Waits for the hardware confirmation of the entanglement generation.
2) Listens for the classical correction bits sent by `source_rank`.
3) Applies the necessary Pauli-X and Pauli-Z corrections to recover the original state fidelity.

This encapsulation drastically simplifies the development process. As shown in Figure 8, the complex coordination required by the raw SDK (discussed in Section III) is reduced to intuitive, single-line commands.

D. COLLECTIVE OPERATIONS

A significant advantage of the MPI paradigm is the ability to perform collective operations involving multiple nodes simultaneously. While point-to-point communication allows for specific data transfer, collective primitives abstract common data movement patterns, reducing code verbosity and potential errors. NetQMPI implements these operations by leveraging the underlying point-to-point primitives.

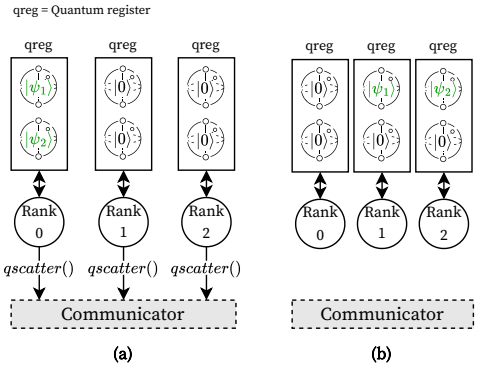


FIGURE 9. Visual representation of the `qscatter` operation. (a) Initial state where all qubits reside in the root process. (b) Distributed state where qubits have been moved to their respective ranks. The operations rely on iterative point-to-point teleportations.

1) Scatter and Gather

The `qscatter` and `qgather` primitives allow for the distribution and collection of quantum states among a group of processes.

The `qscatter` operation takes a list of qubits residing in a source process (*root*) and distributes them sequentially to all other processes in the communicator. Conversely, `qgather` collects qubits from all processes and moves them to a destination register in the root process. From an implementation standpoint, these operations are abstractions of a loop of point-to-point transfers. For instance, a `qscatter` is functionally equivalent to the root node executing a `qsend` inside a `for` loop targeting each rank in the communicator.

Figure 9 illustrates the movement of qubits during these operations. In Figure 9(a), the initial state is shown where the root holds a register of qubits. Figure 9(b) shows the result of a scatter operation (or the start of a gather), where each qubit has been teleported to its corresponding rank.

A practical implementation is shown in Figure 10. In this example, the root process initializes a list of qubits in the superposition state $|+\rangle$ (using Hadamard gates) and distributes one qubit to each node in the network using `qscatter`. Finally, each node measures its local qubit independently.

2) Expose and Unexpose

In classical MPI, the `MPI_Bcast` operation copies data from one process to all others. However, in quantum computing, the *No-Cloning Theorem* prohibits creating independent copies of an arbitrary unknown quantum state. Therefore, a direct quantum broadcast is physically impossible.

To address this, NetQMPI introduces the `expose` and `unexpose` primitives, as proposed in [14]. Instead of copying the state, the `expose` operation creates a shared context where a specific qubit (or set of qubits) becomes conceptually accessible to other processes for collective operations, without violating the no-cloning principle.

```python
def main(app_config=None, rank, size):
    comm = QMPICommunicator(rank, size, app_config)
    conn = comm.connection
    ROOT_RANK = 0
    qubits = []

    # 1. Root initializes the qubits
    if rank == ROOT_RANK:
        qubits = [Qubit(conn) for _ in range(size)]
        for q in qubits:
            q.H() # Apply Hadamard

    # 2. Scatter: Distribute from Root to all
    # Returns the specific qubit assigned to this rank
    local_qs = comm.qscatter(qubits, ROOT_RANK)

    m = local_qs[0].measure())
```

FIGURE 10. Example of the `qscatter` collective primitive in NetQMPI. The root process initializes a list of qubits in the superposition state $|+\rangle^{\otimes n}$, scatters them to all ranks, and each rank measures its local qubit.

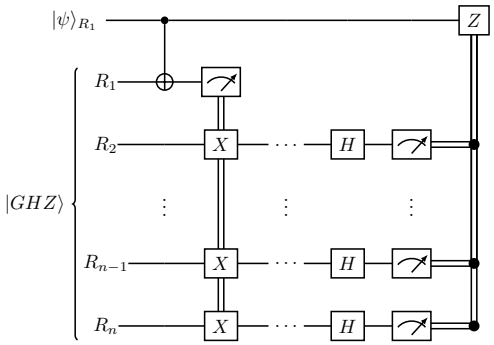


FIGURE 11. Quantum circuit representation of the `expose` and `unexpose` operation. The root qubit is entangled with a pre-shared GHZ state across all nodes, followed by local measurements and classical corrections to distribute the quantum information.

Mathematically, the objective of `expose` is to transition the system from an initial state where the root process R_0 holds an arbitrary superposition

$$|\psi\rangle_{initial} = \alpha|0\rangle_{R_0} + \beta|1\rangle_{R_0}, \quad (1)$$

to a global state where this quantum information is distributed across all n participating ranks $(R_0, \ldots, R_n)$.

$$|\psi\rangle_{exposed} = \alpha(|0\rangle_{R_0}\otimes\cdots\otimes|0\rangle_{R_n}) + \beta(|1\rangle_{R_0}\otimes\cdots\otimes|1\rangle_{R_n}) \quad (2)$$

This shared state enables collective controlled operations on the quantum information distributed across all nodes. The implementation relies on entangling the root qubit with a pre-shared GHZ state across all nodes, followed by local measurements and classical corrections to distribute the quantum information, as illustrated in Figure 11.

Crucially, because this operation alters the accessibility scope of the qubits, it is mandatory to release the resource once the collective computation is finished. The `unexpose` primitive serves this purpose, returning the system to a consistent state where the qubits are once again private to their owner or properly measured.

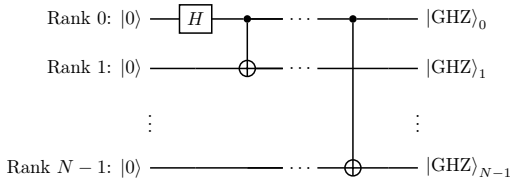


FIGURE 12. Distributed GHZ State Generation across N nodes. The root node (Rank 0) prepares a qubit in superposition and performs distributed CNOTs to entangle the qubits at the target nodes (Ranks $1 \ldots N-1$). Each Remote CNOT involves entanglement distribution and classical communication.

V. COMPARATIVE AND USE CASES

To evaluate the effectiveness of the proposed abstractions, we perform a comparative analysis against the current state-of-the-art tools for distributed quantum programming. We selected the generation of a distributed Greenberger–Horne–Zeilinger (GHZ) state across N nodes as the benchmark scenario. This task is chosen because it requires multi-partite entanglement and coordinated control flow, serving as a standard "Hello World" for the Quantum Internet.

A. BENCHMARK DEFINITION

The goal is to generate the maximally entangled state $|\text{GHZ}\rangle = \frac{1}{\sqrt{2}}(|00\ldots0\rangle + |11\ldots1\rangle)$ shared among N distinct network nodes, where each rank holds exactly one qubit.

As shown in Figure 12, the protocol involves the following steps:

1) One root node (e.g., Rank 0) prepares one qubit in superposition ($|+\rangle$).
2) The root node performs a sequence of distributed CNOT operations targeting the qubits of the remaining $N-1$ nodes.
3) In a distributed setting without shared memory, every "Remote CNOT" implies a complex teleportation protocol or entanglement swapping routine involving classical communication and corrections.

B. QUALITATIVE ANALYSIS: PROGRAMMING COMPLEXITY

We compare the implementation workflow in NetQMPI against three categories of tools: High-level Simulators (QuNetSim), Raw SDKs (NetQASM/SquidASM), and Discrete-Event Engines (NetSquid/SeQUeNCe).

1. NetQMPI (This Work): Following the SPMD paradigm, the solution is written in a *single source file*. The logic scales automatically with the number of processes. The developer uses the `rank` variable to distinguish between the root (Control) and the others (Targets). The distributed CNOT is abstracted and the `expose` primitive is used to create the shared state in a single call. *Complexity:* $O(1)$ in terms of file management. The code remains constant regardless of N.

2. NetQASM SDK (SquidASM / SimulaQron): As these tools operate on a role-based paradigm, the developer must explicitly define the behavior for each node. For a 3-node GHZ state, this requires writing three separate application flows (e.g., `app_alice.py`, `app_bob.py`, `app_charlie.py`). Furthermore, a Remote CNOT is not a native primitive. The developer must manually implement the "telegate" protocol, which involves generating EPR pairs, performing local measurements, sending classical bits via a separate socket, and applying corrections. *Complexity:* $O(N)$ in terms of source files and code maintenance. Extending the system from 3 to 4 nodes requires writing a completely new script for the new node and updating the root script to handle the new connection.

3. NetSquid / SeQUeNCe: These are discrete-event simulators, not programming frameworks. Before implementing the GHZ logic, the user must manually construct the network topology in Python: defining quantum memories, fiber lengths, photon sources, and noise models. The algorithm itself is then defined as a protocol class attached to these physical entities. *Complexity:* Extremely High. The code is dominated by hardware definitions rather than algorithmic logic.

C. CODE IMPLEMENTATION EXAMPLES

To substantiate the complexity analysis, we present code snippets required to implement the GHZ state generation across the evaluated platforms.

1) NetQMPI (SPMD Paradigm)

Listing 1 shows the full implementation in NetQMPI. The logic is defined in a single block. The `expose` primitive automatically handles the underlying distributed CNOTs and entanglement distribution to create the shared state defined in Eq. 2. Changing the size from 3 to 50 nodes only requires changing the command line argument `-n 50`.

```
def main(app_config=None, rank=0, size=3):
    comm = QMPICommunicator(rank, size, app_config)

    # 1. Root creates the base qubit
    qubits = []
    q = Qubit(comm.connection)
    if rank == 0:
        q.H() # Superposition
        qubits.append(q)

    # 2. Collective Operation
    # The root exposes the qubit to all ranks (control of
      remote CNOT)
    comm.expose(qubits, root_rank=0)

    # 3. Perform local operations (if my rank is not root
    )
    if rank != 0:
        qubits[0].cnot(my_qubit)

    comm.unexpose(root_rank=0)
```

Listing 1. NetQMPI: GHZ Generation (Single File)

2) NetQASM SDK (Role-Based)

In the raw SDK, the logic is strictly separated by roles. Listing 2 demonstrates the code required to generate a GHZ state among three nodes (Alice, Bob, and Charlie).

Crucially, note that the developer must explicitly instantiate the connection objects (`EPRSocket` for quantum links and `Socket` for classical communication) *before* the main logic, which reveals the scalability bottleneck: adding a fourth node (Dave) forced us to:

1) Create a completely new file `app_charlie.py`.
2) Modify `app_alice.py` to manually instantiate the new sockets for Charlie (`sock_charlie`, `csock_charlie`) and duplicate the teleportation logic.

```
# --- app_alice.py (Root) ---
# 1. Explicit Socket Creation
# Alice needs separate sockets for each target
sock_bob = EPRSocket("Bob")
sock_charlie = EPRSocket("Charlie")
csock_bob = Socket("Alice", "Bob")       # Classical
csock_charlie = Socket("Alice", "Charlie") # Classical

with NetQASMConnection("Alice", epr_sockets=[sock_bob,
     sock_charlie]) as c:
    q = Qubit(c)
    q.H()

    # 2. Manual Remote CNOT to Bob
    epr_b = sock_bob.create_epr()[0]
    q.cnot(epr_b)
    m_b = epr_b.measure()
    c.flush()
    csock_bob.send(str(m_b)) # Send correction bits

    # 3. Manual Remote CNOT to Charlie
    # Logic must be duplicated for the new node
    epr_c = sock_charlie.create_epr()[0]
    q.cnot(epr_c)
    m_c = epr_c.measure()
    c.flush()
    csock_charlie.send(str(m_c))

# --- app_bob.py (Target 1) ---
sock_alice = EPRSocket("Alice")
csock_alice = Socket("Bob", "Alice")

with NetQASMConnection("Bob", epr_sockets=[sock_alice])
    as c:
    q = Qubit(c)
    epr = sock_alice.recv_epr()[0]
    epr.cnot(q)
    c.flush()

# --- app_charlie.py (Target 2) ---
# New file required for the third node
sock_alice = EPRSocket("Alice")
csock_alice = Socket("Charlie", "Alice")

with NetQASMConnection("Charlie", epr_sockets=[sock_alice
    ]) as c:
    q = Qubit(c)
    epr = sock_alice.recv_epr()[0]
    epr.cnot(q)
    c.flush()
```

Listing 2. NetQASM SDK: GHZ Implementation (Alice, Bob, Charlie)

3) NetSquid (Discrete-Event / Hardware)

NetSquid operates at a lower level of abstraction. Before implementing the GHZ protocol, the user must define the physical hardware. Listing 3 illustrates the verbosity of setting up just the nodes and channels. The actual algorithm

requires defining a class inheriting from `NodeProtocol` for each entity.

Due to the excessive length and technical complexity of the full implementation, the remaining algorithmic code is omitted. It is crucial to emphasize that this comparison does not aim to undermine the utility of discrete-event simulators like NetSquid, but rather to highlight the conceptual gap between physical layer modeling and high-level application development. These tools serve fundamentally different purposes: while discrete-event engines are indispensable for validating hardware fidelity and noise models, they are not designed for algorithmic prototyping. Thus, the level of detail presented here demonstrates that these are not entities directly comparable to NetQMPI, but instead underlying layers that our proposed approach aims to abstract away.

```
# 1. Define Hardware Components
qproc_alice = Node("Alice", num_positions=2,
    mem_noise_models=...)
qproc_bob = Node("Bob", num_positions=2, ...)

# 2. Define Physical Connections
channel_a_b = QuantumChannel("A->B", length=20,
    models={"delay_model": FibreDelayModel()})

# 3. Network Configuration
network = Network("GHZ_Net")
network.add_node(qproc_alice)
network.add_connection(qproc_alice, qproc_bob,
    channel_to=channel_a_b)

# 4. Protocol Implementation (Not shown: ~100 lines)
# Requires defining state machines for photon emission
```

Listing 3. NetSquid: Hardware Setup

4) QuNetSim (High-Level Simulator)

QuNetSim provides a centralized view similar to a network orchestrator. While the algorithmic part is concise, it requires significant configuration code to set up the simulation environment. The developer must manually instantiate hosts, configure the network topology, establish connections, and manage the start/stop lifecycle of the simulation engine (see Listing 4). Furthermore, this code runs purely as a simulation and cannot be compiled to real hardware instructions.

```
# 1. Manual Network Setup
network = Network.get_instance()
alice = Host('Alice')
bob = Host('Bob')
charlie = Host('Charlie')

# 2. Manual Connection Management
alice.add_connection('Bob')
alice.add_connection('Charlie')
bob.add_connection('Charlie') # ... and others ...
alice.start(); bob.start(); charlie.start()
network.start()

# 3. Protocol: Manual Circuit Construction
def ghz_protocol_alice(host):
    # a. Create Root Qubit
    q_root = Qubit(host)
    q_root.H()

    # b. Entangle and Send to Bob
    q_bob = Qubit(host)
    q_root.cnot(q_bob) # Local CNOT

    host.send_qubit('Bob', q_bob)

    # c. Entangle and Send to Charlie
    q_charlie = Qubit(host)
    q_root.cnot(q_charlie)
    host.send_qubit('Charlie', q_charlie)

    # ... Wait for acknowledgments ...

def ghz_protocol_target(host):
    # Targets must explicitly receive the qubit
    q = host.get_data_qubit('Alice')
    print(f"{host.host_id} received part of GHZ")

# 4. Execution
t1 = alice.run_protocol(ghz_protocol_alice)
t2 = bob.run_protocol(ghz_protocol_target)
t3 = charlie.run_protocol(ghz_protocol_target)
t1.join(); t2.join(); t3.join()
network.stop(stop_hosts=True)
```

Listing 4. QuNetSim: Network Setup & Orchestration

Framework	Files	LOC (Network)	LOC (Comp.)	Paradigm
NetQMPI	**1**	$\mathcal{O}(1)$	$\mathcal{O}(1)$	**SPMD (Abstract)**
QuNetSim	1	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	Centralized Sim
NetQASM SDK	N	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	Role-based
NetSquid	N	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	Hardware Descr.

TABLE 1. Comparison of Lines of Code (LOC) complexities required for setting up the network topology and implementing the GHZ generation as the number of nodes N increases.

D. QUANTITATIVE ANALYSIS: LINES OF CODE (LOC)

To evaluate the development effort and scalability of the proposed solution, we analyzed the implementation of the GHZ generation protocol across the selected platforms. We utilize lines of code (LOC) as a metric for software complexity, counting only logical lines (excluding comments and imports).

Table 1 summarizes the asymptotic complexity required for both setting up the network topology and implementing the computational logic.

Trend Analysis and Scalability: The projection of code growth as the network size N increases is illustrated in Figure 13. The analysis reveals three distinct trends:

- **Linear/Quadratic Growth (State of the Art):** In frameworks like the NetQASM SDK or NetSquid, the complexity grows significantly with N. Each new node requires the creation of separate source files, manual instantiation of connection sockets ($\mathcal{O}(N^2)$ for fully connected topologies), and duplication of control logic. This verbose approach forces the developer to write boilerplate code that is essentially identical but structurally necessary, dramatically increasing the probability of introducing copy-paste errors or synchronization bugs.
- **Constant Trend (NetQMPI):** In contrast, NetQMPI maintains a flat complexity curve ($\mathcal{O}(1)$). Thanks to the SPMD paradigm and collective primitives, the code written for $N = 3$ is identical to the code for $N = 100$. The logic is encapsulated in loops (e.g., `expose`) that iterate over the communicator size dynamically.

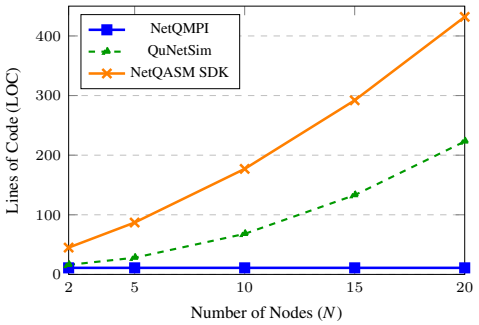


FIGURE 13. Comparison of Lines of Code (LOC) required to implement the GHZ state generation protocol as the network size (*N*) increases. NetQMPI maintains constant complexity due to its SPMD collective operations.

Implications for Quantum Software Engineering: This reduction in code complexity is critical for the development of distributed quantum applications. By decoupling the program logic from the network size, NetQMPI allows for rapid prototyping and ensures a bug-free implementation of complex protocols. In a domain where debugging distributed quantum states is notoriously tricky, minimizing the codebase surface reduces the potential for implementation errors, allowing researchers to focus on algorithmic correctness rather than infrastructure management.

VI. CONCLUSIONS

This work has introduced NetQMPI, a high-level software framework designed to bridge the gap between abstract quantum algorithm design and the low-level management of distributed quantum network resources. By abstracting the verbose primitives of the NetQASM SDK, NetQMPI successfully reduces the complexity of programming distributed quantum applications, offering a scalable and robust development environment.

The primary conclusions extracted from this work are summarized as follows:

- **SPMD paradigm:** NetQMPI facilitates a transition from the labor-intensive, role-based programming model of existing SDKs to a Single Program Multiple Data (SPMD) paradigm. By unifying the control logic into a single source file where roles are defined by the process `rank`, we have demonstrated that the complexity of setting up a quantum network drops from $\mathcal{O}(N^2)$ to $\mathcal{O}(1)$ in LOC number. This decoupling of program logic from network size is a fundamental requirement for scalable quantum software engineering.
- **Abstraction of the Network Layer:** The library introduces the `QMPICommunicator` and semantic point-to-point primitives (`qsend`, `qrecv`) that completely hide the physical layer's intricacies. The developer is no longer required to manually manage `EPRSocket` lifecycles, entanglement fidelity, or the synchronization of classical control signals, allowing them to focus entirely on the algorithmic flow of quantum information.
- **Implementation of Collective Operations:** We have provided practical implementations for collective communication in a quantum context. Notably, the introduction of the `expose` and `unexpose` primitives addresses the physical constraints imposed by the No-Cloning Theorem. These operations allow for the creation of shared multipartite entangled contexts (such as GHZ states) to simulate broadcasting behavior without violating quantum mechanical principles.
- **Operational Implementation of QMPI:** While previous proposal for a QMPI remain theoretical specifications, NetQMPI provides a functional software implementation. This work demonstrates the practical viability of the MPI paradigm for quantum networks, moving beyond abstract definitions to provide a working execution environment. Crucially, the implementation resolves the conflict between classical broadcasting semantics and the No-Cloning Theorem, proving that high-level distributed abstractions can be effectively mapped to rigorous physical instructions via the NetQASM ecosystem.
- **Integration with the quantum ecosystem:** A critical objective of NetQMPI is its seamless integration with the existing quantum stack. By compiling down to the standardized NetQASM ISA, NetQMPI acts as a backend-agnostic middleware. This allows researchers to execute high-level code on state-of-the-art simulators, such as **SquidASM** and **NetSquid**, leveraging their rigorous physical noise models without facing their steep learning curves. Furthermore, this architecture ensures that applications written today in NetQMPI will be executable on future physical quantum hardware that supports the NetQASM standard.

In summary, NetQMPI represents a significant step forward in quantum software engineering. It enables developers to write readable, maintainable, and bug-free distributed quantum applications, paving the way for the implementation of complex distributed algorithms.

REFERENCES

[1] H. Buhrman, R. Cleve, and A. Wigderson, "Quantum vs. Classical Communication and Computation," *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pp. 63–68, 2 1998. [Online]. Available: https://arxiv.org/pdf/quant-ph/9802040

[2] I. L. Markov, "Limits on fundamental limits to computation," *Nature 2014 512:7513*, vol. 512, no. 7513, pp. 147–154, 8 2014. [Online]. Available: https://www.nature.com/articles/nature13570

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999. [Online]. Available: https://epubs.siam.org/doi/10.1137/S0097539795293172

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. Part F129452, pp. 212–219, 7 1996. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/237814.237866

[5] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 8 2018. [Online]. Available: https://quantum-journal.org/papers/q-2018-08-06-79/

[6] D. Barral, F. J. Cardama, G. Díaz-Camacho, D. Faílde, I. F. Llovo, M. Mussa-Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing," *Computer Science Review*, vol. 57, p. 100747, 8 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013725000231

[7] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: A survey," *Computer Networks*, vol. 254, p. 110672, 12 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624005048?via%3Dihub

[8] R. Van Meter, W. J. Munro, and K. Nemoto, "Architecture of a Quantum Multicomputer Implementing Shor's Algorithm," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5106 LNCS, pp. 105–114, 2008. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-89304-2_10

[9] A. Ovide, S. Rodrigo, M. Bandic, H. Van Someren, S. Feld, S. Abadal, E. Alarcon, and C. G. Almudever, "Mapping quantum algorithms to multi-core quantum computing architectures," *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2023-May, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10181589

[10] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, 10 2018. [Online]. Available: https://www.science.org/doi/10.1126/science.aam9288

[11] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti, "Quantum Internet protocol stack: A comprehensive survey," *Computer Networks*, vol. 213, p. 109092, 8 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622002250

[12] K. Azuma, S. E. Economou, D. Elkouss, P. Hilaire, L. Jiang, H. K. Lo, and I. Tzitrin, "Quantum repeaters: From quantum networks to the quantum internet," *Reviews of Modern Physics*, vol. 95, no. 4, p. 045006, 12 2023. [Online]. Available: https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.95.045006

[13] S. W. Loke, "From Distributed Quantum Computing to Quantum Internet Computing: an Overview," *arXiv*, 8 2022. [Online]. Available: https://arxiv.org/pdf/2208.10127

[14] F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, T. F. Pena, J. C. Pichel, and A. Gómez, "NetQIR: An extension of QIR for distributed quantum computing," *Future Generation Computer Systems*, vol. 174, p. 107989, 1 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X25002845

[15] G. Díaz-Camacho, I. F. Llovo, F. J. Cardama, I. Bautista, D. Faílde, M. M. Juane, J. Vázquez-Pérez, N. Costas, T. F. Pena, and A. Gómez, "Benchmarking Distributed Quantum Computing Emulators," *arXiv*, 12 2025. [Online]. Available: https://arxiv.org/pdf/2512.01807

[16] A. S. Cacciapuoti, M. Caleffi, F. Tafuri, F. S. Cataliotti, S. Gherardini, and G. Bianchi, "Quantum Internet: Networking Challenges in Distributed Quantum Computing," *IEEE Network*, vol. 34, no. 1, pp. 137–143, 1 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8910635

[17] T. Haner, D. S. Steiger, T. Hoefler, and M. Troyer, "Distributed quantum computing with QMPI," *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 11 2021. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3458817.3476172

[18] S. Diadamo, J. Nötzel, B. Zanger, and M. M. Beşe, "QuNetSim: A Software Framework for Quantum Networks," *IEEE Transactions on Quantum Engineering*, vol. 2, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9465750

[19] R. Parekh, A. Ricciardi, A. Darwish, and S. Diadamo, "Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing," *Proceedings of QCS 2021: 2nd International Workshop on Quantum Computing Software, Held in conjunction with SC 2021: The International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 9–19, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9651415

[20] C. Piveteau and D. Sutter, "Circuit Knitting With Classical Communication," *IEEE Transactions on Information Theory*, vol. 70, no. 4, pp. 2734–2745, 4 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10236453

[21] A. Dahlberg and S. Wehner, "SimulaQron—a simulator for developing quantum internet software," *Quantum Science and Technology*, vol. 4, no. 1, p. 015001, 9 2018. [Online]. Available: https://iopscience.iop.org/article/10.1088/2058-9565/aad56e

[22] A. Dahlberg, B. Van Der Vecht, C. Delle Donne, M. Skrzypczyk, I. Te Raa, W. Kozlowski, and S. Wehner, "NetQASM—a low-level instruction set architecture for hybrid quantum–classical programs in a quantum internet," *Quantum Science and Technology*, vol. 7, no. 3, p. 035023, 6 2022. [Online]. Available: https://iopscience.iop.org/article/10.1088/2058-9565/ac753f

[23] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, and S. Wehner, "NetSquid, a NETwork Simulator for QUantum Information using Discrete events," *Communications Physics 2021 4:1*, vol. 4, no. 1, pp. 164–, 7 2021. [Online]. Available: https://www.nature.com/articles/s42005-021-00647-8

[24] X. Wu, A. Kolar, J. Chung, D. Jin, T. Zhong, R. Kettimuthu, and M. Suchara, "SeQUeNCe: a customizable discrete-event simulator of quantum networks," *Quantum Science and Technology*, vol. 6, no. 4, p. 045027, 9 2021. [Online]. Available: https://iopscience.iop.org/article/10.1088/2058-9565/ac22f6

[25] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Physical Review Letters*, vol. 70, no. 13, p. 1895, 3 1993. [Online]. Available: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.70.1895

F. JAVIER CARDAMA (Student Member) received the B.S. degree in Computer Engineering from the University of Santiago de Compostela (USC), Spain, in 2021, and the M.S. degree in High-Performance Computing from the same university in 2022. He is currently pursuing the Ph.D. degree in Information Technology Research at the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), USC, holding a Xunta de Galicia fellowship. His research interests focus on distributed quantum computing and high-performance computing, specifically on the development of software frameworks and architectures for the integration of quantum technologies in HPC centers.

TOMÁS F. PENA (Senior Member) received the Ph.D. in Physics from the University of Santiago de Compostela (USC), Santiago, Spain, in 1994. He is currently Full Professor with the Department of Electronics and Computer Science, USC, and a Senior Member of the Research Center in Intelligent Technologies (CiTIUS), Santiago de Compostela, Spain. His research interests include distributed quantum computing, high-performance computing, parallel systems architecture, optimization of performance in irregular codes and with sparse matrices, and Big Data technologies.

Chapter 9

SYCL QPU: an LLVM-based QPU simulation framework built using DPC++

The research presented in this chapter corresponds to the article published in the proceedings of the following international conference:

> M. Leal, F. Javier Cardama, and T. F. Pena, "SYCL QPU: An LLVM-based QPU simulation framework built using DPC++", *Proceedings of the 2025 IEEE International Conference on Cluster Computing Workshops, CLUSTER Workshops 2025*, 2025. DOI: 10.1109/CLUSTERWorkshops65972.2025.11164192
>
> - **GitHub code:** `https://github.com/MiguiLeal8/llvm/tree/feature/qpu_selector`

This publication specifically addresses and fulfills Objective O5 of this thesis. For a comprehensive overview of the conference's quality indicators, the reader is referred to the International conferences from Contributions section.

2025 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops) | 979-8-3315-1256-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/CLUSTERWorkshops65972.2025.11164192

SYCL QPU: an LLVM-based QPU simulation framework built using DPC++

Miguel Leal, F. Javier Cardama, and Tomás F. Pena

Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Galicia, Spain

***Abstract*—Current limitations in quantum hardware have made high-fidelity simulation an essential tool for prototyping and validating quantum algorithms, particularly those involving hybrid quantum-classical workflows. However, building flexible and scalable simulators that support heterogeneous hardware remains a technical challenge. This work introduces a QPU simulation framework based on LLVM and implemented using Intel's open-source DPC++ (Data Parallel C++) compiler, which extends the SYCL programming model for heterogeneous computing. It builds on key abstractions in SYCL to target native CPU backends and CUDA devices while allowing fine-grained control over compilation and execution. This design empowers developers to focus on algorithm development without needing to manage low-level hardware details, while leveraging scalable simulation capabilities to bridge the current gap between classical simulation and future QPU execution. This framework enables hybrid execution on both CPUs and NVIDIA GPUs, supporting seamless integration with Qiskit AER.**

***Index Terms*—Quantum simulation, SYCL, DPC++, QPU, heterogeneous computing, LLVM, Qiskit Aer, GPU acceleration, hybrid quantum-classical workflows.**

I. INTRODUCTION

Quantum computing has witnessed rapid development in recent years, driven by improvements in hardware architectures, compiler toolchains and quantum software ecosystems. However, the practical deployment of scalable quantum algorithms remains hindered by current hardware constraints, including limited qubit counts, short coherence times, and high gate error rates.

As a result, simulation frameworks have become a crucial part of the quantum software development lifecycle, enabling researchers to design, test and optimize quantum classical algorithms before they are deployed in physical devices.

One promising approach to improve the fidelity and scalability of these simulations involves leveraging high-performance computing (HPC) resources in conjunction with heterogeneous hardware backends. To support these capabilities in a flexible and extensible way, modern programming models like SYCL [1] have emerged as compelling tools for targeting diverse hardware from a unified codebase.

II. SYCL QPU OVERVIEW

In this context, we present an LLVM-based [2] QPU simulation framework built using DPC++ [3] that enables hybrid quantum-classical execution across CPU and GPU backends [1]. This framework integrates seamlessly with Qiskit AER and can be used in HPC environments. By abstracting hardware-specific details and providing native support for SYCL kernels, it allows developers to prototype quantum algorithms in a scalable and portable simulation environment, bridging the current gap between classical emulation and near-term quantum hardware.

[1]SYCL QPU simulation framework GitHub

III. SYCL QPU: DESIGN AND FEATURES

At the core of our framework is a modular architecture that leverages SYCL's abstraction model to target different hardware backends. This is achieved by binding high-level quantum-classical workloads to DPC++ based device adapters, which then dispatch workloads to the appropriate backend runtime (e.g., CPU, GPU). These device adapters abstract the complexity of hardware-specific details, providing a unified programming interface for both development and deployment.

The compilation process uses clang++ with SYCL-specific flags such as -fsycl and -fsycl-targets, which direct the compiler to generate device-specific code. For example, targeting native_cpu enables low-overhead local execution, useful for debugging and development. In contrast, targeting nvptx64-nvidia-cuda offloads quantum simulation workloads to NVIDIA GPUs, significantly accelerating execution in HPC settings.

Our framework supports runtime detection and selection of simulated QPU devices through the SYCL device discovery system (sycl-ls). It exposes both real and simulated QPU identifiers, allowing users to explicitly assign a queue to a simulated QPU (sycl::qpu_selector_v). This mechanism is essential for benchmarking quantum workloads and validating performance without requiring physical quantum hardware.

Integration with Qiskit AER is achieved through a C++/Python binding layer built using pybind11 [4], enabling quantum circuit simulations to interface directly with the SYCL runtime. To bridge Python-based quantum logic and SYCL-based execution, we leveraged functions from the CUNQA project [5], which provides utilities to transform QuantumCircuit objects from Qiskit into JSON representations suitable for simulator backends. These JSON-formatted circuits are then executed within the SYCL environment using adapted simulator infrastructure. In addition, our implementation required the use of a modified submodule of the aer-cpp backend from Qiskit Aer. This adaptation was based on a forked version of the Aer repository maintained by CESGA Quantum Spain [6], which provided improved compatibility for integration with custom simulators and SYCL-based workflows. This connection facilitates hybrid workflows where Python-based quantum programming interacts with SYCL-accelerated classical post-processing.

These are some of the features:

- **Multi-Backend Support:** Enables seamless execution on both native CPUs and NVIDIA GPUs using SYCL's target selectors, including simulated QPUs, allowing users to explicitly select a backend using `sycl::qpu_selector_v`.
- **Compiler Integration:** Built with the Intel LLVM-based DPC++ toolchain, this framework supports standard SYCL compilation flags such as `-fsycl` and `-fsycl-targets`.

- **Python Integration via Pybind11:** Allows interaction with Python libraries like Qiskit Aer, facilitating hybrid quantum-classical workflows.
- **Quantum Circuit Abstractions:** Provides C++ wrappers to build and execute circuits using gates like H, X, CX, etc., mapped to Qiskit AER-compatible representations.

IV. EXAMPLE OF USE 1: DEVICE DISCOVERY WITH SYCL-LS

When using a script in a system with a CPU and a Tesla GPU in which our framework is installed, the command `sycl-ls` returns the following device list:

```
[cuda:gpu][cuda:0] NVIDIA CUDA BACKEND, Tesla
    V100S-PCIE-32GB 7.0 [CUDA 12.6]
[cuda:qpu][cuda:1] NVIDIA CUDA BACKEND, Simulated
     QPU on NVIDIA 7.0 [CUDA 12.6]
[native_cpu:cpu][native_cpu:0] SYCL_NATIVE_CPU,
    Native CPU 0.1 [0.0.0]
[native_cpu:qpu][native_cpu:1] SYCL_NATIVE_CPU,
    Simulated QPU 0.1 [0.0.0]
```

Listing 1: Output of `sycl-ls` in an SBATCH script

V. EXAMPLE OF USE 2: FULL GATE SET CIRCUIT ON SIMULATED QPU

This example showcases the construction and execution of a quantum circuit using the simulated QPU backend, integrated with Qiskit Aer. The circuit demonstrates the use of a selection of single-qubit gates (`X`, `H`, `RZ`) and a two-qubit controlled NOT gate (`CX`), providing a realistic simulation scenario involving basic quantum operations, entanglement, and measurement.

The simulation starts by creating a SYCL queue using the `sycl::qpu_selector_v`, which selects the virtual QPU backend. A quantum circuit is then created with three qubits, and initialized to the basis state $|101\rangle$ using the custom `initialize_qubits()` helper function. This function applies `X` gates to the appropriate qubits to match the target bitstring.

A sequence of gates is then applied:

`X` (Pauli-X) flips qubit 0.

`H` (Hadamard) puts qubit 0 into superposition.

`RZ` applies a rotation around the Z-axis to qubit 0.

`CX` entangles qubit 0 with qubit 1, creating quantum correlations based on the control-target pair.

After building the circuit, each qubit is measured, and the results are stored in classical bits. The simulation is configured to run 1024 shots using the `statevector` method, which allows precise tracking of quantum amplitudes. The noise model is left empty, meaning the simulation is ideal (no decoherence or gate errors).

The function `print_circuit_diagram()` is called to visualize the full circuit, aiding debugging and verification. Finally, the circuit is executed via the `executeQuantumCircuit()` function, which interfaces with the backend simulator. The measurement counts are then printed in binary form using `print_results_in_binary()` for human-readable output.

This example demonstrates how a developer can build, simulate, and analyze a basic quantum circuit entirely within the SYCL and DPC++ ecosystem, interfaced with Qiskit Aer.

```
sycl::queue q(sycl::qpu_selector_v);
auto quantumCircuit = sycl::detail::
    createQuantumCircuit(3, q.get_device());
sycl::detail::initialize_qubits(quantumCircuit, "
    101");

sycl::detail::x(quantumCircuit, 0);
sycl::detail::h(quantumCircuit, 0);
sycl::detail::rz(quantumCircuit, 0, M_PI);
sycl::detail::cx(quantumCircuit, 0, 1);

for (int i = 0; i < 3; ++i)
  sycl::detail::measure(quantumCircuit, i, i);

nlohmann::json noise_model_json = {
    {"noise_model", {}},
    {"basis_gates",
    {"x","h", "rz", "cx"}}};

config::RunConfig run_config;
run_config.shots = 1024;
run_config.method = "statevector";
run_config.memory_slots = num_qubits;
run_config.seed = 123;

sycl::detail::print_circuit_diagram(
    quantumCircuit);

nlohmann::json result = sycl::detail::
    executeQuantumCircuit(
    quantumCircuit, noise_model_json, run_config)
        ;

sycl::detail::print_results_in_binary(result,
    num_qubits);
```

Listing 2: Quantum circuit execution on Simulated QPU

VI. CONCLUSIONS

The LLVM-based QPU simulation framework presented in this work demonstrates a flexible and performant approach to executing hybrid quantum-classical workloads using SYCL and DPC++. By decoupling quantum logic from hardware dependencies, the system supports scalable development and prototyping of quantum algorithms across diverse computing environments. Its compatibility with Qiskit AER and support for both CPU and CUDA backends make it a powerful tool for quantum software development in research and HPC scenarios. Looking forward, this framework lays the groundwork for integrating more realistic quantum noise models and optimizing large-scale quantum simulations in production-grade environments.

REFERENCES

[1] Khronos Group, "SYCL™ 2020 Specification." Available: https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html, 2020. Accessed: 2025-04.

[2] LLVM Project, "The LLVM compiler infrastructure." Available: https://llvm.org/, 2025. Accessed: 2025-04.

[3] Intel Corporation, "Intel oneAPI DPC++ Compiler." Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html, 2025. Accessed: 2025-04.

[4] W. Jakob, J. Rhinelander, D. Moldovan, *et al.*, "pybind11 – Seamless operability between C++11 and Python." Available: https://github.com/pybind/pybind11, 2025. Accessed: 2025-05.

[5] C. Q. Spain, "CUNQA: CUDA-based Native Quantum Accelerator." Available: https://github.com/CESGA-Quantum-Spain/cunqa, 2025. Accessed: 2025-05.

[6] C. Q. Spain, "Qiskit aer fork with custom simulator adaptations." https://github.com/CESGA-Quantum-Spain/qiskit-aer/tree/9785243118ddabf42021ce15b8fb1cb70798be39, 2024.

Chapter 10

Benchmarking for distributed quantum computing emulators

The content of this chapter is presented in the format of a scientific article which is currently pending publication:

> G. Díaz-Camacho, I. F. Llovo, F. J. Cardama, I. Bautista, D. Faílde, M. M. Juane, J. Vázquez-Pérez, N. Costas, T. F. Pena, and A. Gómez, "Benchmarking Distributed Quantum Computing Emulators", *arXiv*, Dec. 2025. [Online]. Available: `https://arxiv.org/abs/2512.01807`
>
> - **GitHub code:** `https://github.com/gdiazcamacho/DQC-Emulator-Benchmarking`

This publication specifically addresses and fulfills Objective O4 of this thesis. For further details regarding the publication status of this pre-print, the reader is referred to the Results: unpublished articles section.

Benchmarking Distributed Quantum Computing Emulators: Scalability, Fidelity, and Architectural Trade-offs

Guillermo Díaz-Camacho[a], Iago F. Llovo[a], F. Javier Cardama[b], Irais Bautista[a], Daniel Faílde[a], Mariamo Mussa Juane[a], Jorge Vázquez-Pérez[a], Natalia Costas[a], Tomás F. Pena[b,c], Andrés Gómez[a]

[a]*Galicia Supercomputing Center (CESGA), Avda. de Vigo S/N, Santiago de Compostela, 15705, Spain*
[b]*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain*
[c]*Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain*

Abstract

Scalable quantum computing requires architectural solutions beyond monolithic processors. Distributed quantum computing (DQC) addresses this challenge by interconnecting smaller quantum nodes through quantum communication protocols, enabling collaborative computation. While several experimental and theoretical proposals for DQC exist, emulator platforms are essential tools for exploring their feasibility under realistic conditions. In this work, we introduce a benchmarking framework to evaluate DQC emulators using a distributed implementation of the inverse Quantum Fourier Transform ($QFT^{\dagger}$) as a representative test case, which enables efficient phase recovery from pre-encoded Fourier states. The QFT is partitioned across nodes using teleportation-based protocols, and performance is analyzed in terms of execution time, memory usage, and fidelity with respect to a monolithic baseline.

As part of this work, we review a broad range of emulators, identifying their capabilities and limitations for programming distributed quantum algorithms. Many platforms either lacked support for teleportation protocols or required complex workarounds. Consequently, we select and benchmark four representative emulators: Qiskit Aer, SquidASM, Interlin-q, and SQUANCH. They differ significantly in their support for discrete-event simulation, quantum networking, noise modeling, and parallel execution. Our results highlight the trade-offs between architectural fidelity and simulation scalability, providing a foundation for future emulator development and the validation of distributed quantum protocols. This framework can be extended to support additional algorithms and emulators.

Keywords: distributed quantum computing, quantum intermediate representation, quantum internet, compilers, teledata, telegate, distributed quantum applications

Contents

1. Introduction

Quantum computing has emerged as a promising computational paradigm that uses the principles of quantum mechanics to tackle problems intractable for classical systems. However, the current landscape, commonly referred to as the Noisy Intermediate-Scale Quantum (NISQ) era, is dominated by devices with limited qubit counts and significant susceptibility to noise. One of the most critical challenges in this era is *scalability*: building quantum computers of thousands or even millions of qubits that still maintain high coherence and fidelity. Inspired by classical Distributed Computing and High Performance Computing, this challenge has led to increased interest in distributed architectures that connect multiple smaller quantum processors to act as a unified, larger system. Distributed quantum computing (DQC) offers a possible solution by interconnecting multiple smaller quantum nodes via quantum communication links. This approach enables computational tasks to be split across nodes and coordinated using entanglement and classical control. Rather than scaling-in with a single and larger monolithic device, DQC enables modular architectures that could scale-out more efficiently across heterogeneous platforms and physical environments.

Distinguishing between DQC and the broader concept of the Quantum Internet (QI) becomes crucial. While both involve quantum networks, QI focuses on communication tasks, such as quantum key distribution (QKD), secure transmission, and entanglement distribution, whereas DQC is centered on computational collaboration across nodes. The goal of QI is to transmit quantum information; the goal of DQC is to process it cooperatively. QI wants to maximize the number of Bell pairs as an objective, DQC wants to minimize them as a resource. The boundary between these two fields is subtle but significant, and their technological needs differ in both protocol design and resource management.

DQC protocols are difficult to implement experimentally due to hardware constraints, node synchronization issues, and quantum channel fragility. As a result, *software emulation* has become an indispensable tool for researchers. Emulators allow the prototyping of DQC protocols, exploration of system architectures, and simulation of control mechanisms and physical effects, long before scalable hardware is available. In this way, they accelerate algorithm development and offer insight into tradeoffs between noise, architecture, and performance. However, DQC emulators themselves vary widely in their capabilities and intended use. Some tools model detailed physical layers such as decoherence and photon loss; others focus on logical protocol correctness. Still others distribute the classical simulation effort across HPC resources to simulate larger quantum systems monolithically. This leads to an important distinction:

1. **Distributed (or parallel) emulation of quantum computing:** using classical parallelism to simulate large quantum circuits on distributed memory machines (e.g., MPI-enabled backends).
2. **Emulation of distributed quantum computing:** simulating protocols and behaviors of physically distributed quantum processors.

This study focuses on the latter: benchmarking tools that *emulate distributed quantum computing*. Specifically, we explore whether current emulators are capable of simulating non-trivial distributed algorithms, those that require teleportation, entanglement management, and inter-node gate execution. We categorize emulators into four quadrants based on their architecture (monolithic versus distributed) and their target functionality (monolithic QC versus distributed QC), as shown in Table 1. To the best of our knowledge, no existing software framework provides both a faithful emulation of distributed quantum computing *and* a distributed-memory classical simulation backend. Current tools cover one aspect or the other, but not both simultaneously.

	Monolithic Quantum Computation	**Distributed Quantum Computation**
Monolithic Emulator	Qiskit Aer, PennyLane, myQLM, Qulacs, ProjectQ, IQS, QUEST [...]	NetSquid, SquidASM, SimulaQron, QuNetSim, Interlin-q, QuISP, SeQUeNCe, DQC Executor [...]
Distributed Emulator	mpiQulacs, QuESTlink, IQS (with MPI), Qiskit Aer (multi-GPU) [...]	(??)

Table 1: Classification of quantum emulators by simulation architecture and computation model.

To this end, we implement a distributed version of the Quantum Fourier Transform (QFT), a widely used algorithmic primitive in quantum computing, using gate teleportation to handle non-local interactions. We implement a distributed inverse QFT ($QFT^{\dagger}$), rather than the standard forward QFT, because it allows us to prepare known input states in the Fourier basis and verify correctness by recovering the original phase upon measurement. This approach simplifies fidelity evaluation and avoids the need for complex state preparation across nodes. This benchmark serves to probe how well current emulators support distributed quantum circuits, measuring the computational and memory costs associated with emulating teleportation and coordination. The choice of the QFT comes from its importance for canonical algorithms such as the Quantum Phase Estimation or Shor's factorization algorithm, but also from its dense entanglement between qubits, which makes it hard to distribute.

To our knowledge, this is the first benchmarking study focused specifically on the distributed emulation of quantum algorithms across quantum nodes. While previous works have benchmarked circuit simulators [1, 2], they largely omit inter-node coordination, entanglement management, and protocol-level simulation. Our work highlights both the strengths and limitations of currently available tools in this area.

The remainder of this paper is structured as follows. Section 2

provides a comparative overview of existing distributed quantum emulators. Section 3 introduces the methodology of the benchmark task: a distributed inverse QFT based on gate teleportation. Section 4 presents experimental results on runtime, memory usage, and fidelity across several emulators and configurations. Section 5 discusses limitations, future directions, and opportunities for emulator development[1].

2. Overview and evaluation of Distributed Quantum Computing Emulators

In this section, we review a wide selection of quantum emulators and simulators relevant to the execution of distributed quantum computing (DQC) protocols. Our choice of emulators is informed by the recent survey by Caleffi et al. [3], which provides a comprehensive overview of simulation tools, classifying them for distributed quantum computing into three main categories: hardware-oriented, protocol-oriented, and application-oriented. These tools are listed in Table 2.

Category	Tool	Reference
Hardware-oriented	SQUANCH	[4]
	NetSquid	[5]
Protocol-oriented	SimulaQron	[6]
	SeQUeNCe	[7]
	QuISP	[8]
	QuNetSim	[9]
Application-oriented	NetQASM SDK	[10]
	QNE-ADK	[11]
	DQC Executor	[12]

Table 2: Overview of Distributed Quantum Computing Emulators and Simulators categorized by their primary focus.

The emulators are evaluated using criteria relevant to DQC benchmarking tasks. These include not only native support for essential operations like teleportation or circuit-level logic enabling algorithms such as the Quantum Fourier Transform (QFT), but also deeper characteristics such as discrete-event simulation, realistic quantum networking, noise simulation.

In order to evaluate and compare different distributed quantum computing (DQC) emulators, it is essential to identify and classify the core features that enable or limit their suitability for benchmarking tasks. The emulators in this study are analyzed according to a set of capabilities that reflect their programmability, physical realism, scalability, and usability. This section reviews a set of emulators (and simulators) evaluated for their suitability in modeling DQC tasks. We focus on their ability to simulate teleportation-based logic, node synchronization, noise models, and circuit-level emulation.

[1] The full benchmarking suite, including scripts, data, and plotting utilities, is available at `https://github.com/gdiazcamacho/DQC-Emulator-Benchmarking`.

We group into three categories based on their importance to our benchmarking goals: required (**MUST**) features, important features that significantly enhance the emulator's realism or flexibility, and useful features that improve practicality and performance. A summary of these capabilities is provided in Table 3, and a detailed explanation of each feature follows below.

Category	Feature
MUST	Teleportation Support
	QFT Gate Support
IMPORTANT	Discrete-Event Simulation (DES)
	Multi-Node Simulation
	Quantum Network Modelling
	Noise Modelling
	Emulation Type
	Parallel Execution
USEFUL	Native Performance Metrics
	Language Compatibility and Interface
	Active Development and Documentation

Table 3: Summary of Emulator Capabilities for DQC Benchmarking

MUST-have capabilities. These features are considered essential for the emulators to be included in the benchmarking study:

- Teleportation support (either qubit or gate level): The emulator must support qubit or gate teleportation between nodes. This can be either as a native operation or constructible from lower-level primitives such as Bell pair creation, mid-circuit measurements, feed-forward classical communication and conditional operations.
- QFT gate support: The emulator must provide native or constructible versions of the gates used in the Quantum Fourier Transform. These include Hadamard gates, parameterized Phase gates, and controlled Phase gates. For the teleportation step of the process, more standard gates such as CNOT, X and Z gates may also required.

IMPORTANT capabilities. These features are not strictly required, but are highly valuable for simulating realistic distributed quantum systems and flexibility:

- Discrete-event simulation (DES): allows accurate modeling of time-dependent behaviors, including synchronization between nodes, communication latency and delays. It is also essential to accurately model decoherence and other time-based noise effects. For an accurate emulator, this can be considered a MUST-have, but some non-DES emulators are included in the benchmarking to compare with others.
- Multi-node simulation: The ability to simulate more than two nodes is critical for assessing scalability and architectural behavior. Two-node simulation can be enough

to compare basic scalability in the number of qubits and teleportations in this benchmarking test, but it is not enough for an actual DQC emulator.

- Quantum network modelling: The emulator should support simulation of realistic quantum communication networks, including latency and message passing, photon losses, network topology and routing.
- Noise modelling: Realistic quantum systems are noisy, and noise can completely break the performance of an algorithm. Emulators should allow noise in communication (e.g., depolarizing channels, photon loss) and noise in computation (e.g., gate errors, measurement noise). This could come from a fully open quantum system, a density matrix simulation, or some other technique to simulate noise.
- Emulation type: The underlying simulation model affects expressiveness and performance. Common paradigms include statevector, density matrix, stabilizer or tensor network simulation.
- Parallel execution: The ability to run simulations across multiple threads, cores, or even machines increases scalability and reduces runtime.

USEFUL capabilities. These features are not critical for the simulation itself, but they are convenient to perform the benchmarking.

- Native performance metrics: Built-in measurement of allocated memory and execution time. This is especially useful for automated benchmarking and analysis, but external tools can be used instead.
- Language compatibility and interface: Programming language support and API design affect usability. For instance, Python bindings or integration with quantum SDKs (like Qiskit or NetQASM SDK) may be essential for certain workflows.
- Active development and documentation: tools that are well-maintained with documentation and examples are more accessible for experimental comparison and fair benchmarking.

Descriptions below highlight each emulator's design goals, features, and limitations, and indicate whether they were included in our benchmarking effort. Only a few of them are included in the benchmarking test in the next section, as most of them lacked the necessary support for the chosen test.

From the surveyed tools, we selected Qiskit Aer, SquidASM, Interlin-q, and SQUANCH for benchmarking, as they met the minimum requirements for simulating a gate-teleportation-based distributed inverse QFT, or provided an instructive comparison point despite known limitations. Other tools are briefly reviewed in Section 2.2, but were excluded from benchmarking due to missing capabilities or incompatible architectures.

2.1. Included in Benchmarking

2.1.1. Qiskit Aer

Qiskit Aer [13] is a high-performance circuit-level simulator provided by IBM as part of the Qiskit SDK [14]. Originally intended for monolithic circuit simulation, Qiskit Aer offers a flexible backend interface that includes statevector, density matrix, unitary, and stabilizer simulations. It is fully integrated with Qiskit's transpiler stack and supports realistic noise models, making it suitable for algorithm development and validation under both ideal and noisy conditions.

While Qiskit Aer is not inherently a distributed quantum computing simulator, it can be adapted for DQC prototyping by manually decomposing circuits into separate node registers. Teleportation is not a native primitive but it can be built using a combination of standard operations, such as Bell pair generation, mid-circuit measurements, classical feed-fordward, and post-processing of measurement outcomes. This makes it suitable for validating logical correctness and for performance baselines, even if it does not account for inter-node latency or timing.

Key features:

- Supports multiple backends: statevector, stabilizer, density matrix, unitary simulator.
- Integrates with Qiskit's transpiler for optimized gate decomposition and device-aware routing.
- Allows noise models including depolarizing, amplitude damping, readout error, and custom channels.
- Efficient execution for up to ∼30 qubits depending on hardware resources.
- Active development.

Limitations:

- No notion of distributed nodes or classical message passing beyond manual simulation.
- No discrete-event timing or control over scheduling, cannot model synchronization or qubit lifetimes precisely.
- All teleportation and inter-node logic must be hand-coded using circuit primitives.

Evaluation Context: Used as a baseline for runtime and fidelity comparisons. It provides a best-case scenario against which true DQC emulators can be compared in terms of overhead, noise tolerance, and expressiveness.

2.1.2. SquidASM

SquidASM [15] is a distributed quantum computing (DQC) simulation framework developed by QuTech (TU Delft), built directly on top of NetSquid [5], a modular discrete-event simulator for quantum networks. Unlike abstract protocol-level tools, SquidASM enables explicit modeling of both quantum and classical processes across networked quantum nodes with

time-aware event scheduling. It was designed with realism and low-level control in mind, making it well-suited for algorithmic benchmarking and hardware-aware protocol design.

SquidASM introduces agents that act as quantum processing nodes. Each agent can perform quantum operations, schedule gates, initiate or consume entangled links, and exchange classical messages with other nodes. Classical and quantum control flows are explicitly coded by the user, allowing full freedom to coordinate teleportation protocols, circuit segmentation, and inter-node logic.

Unlike some emulators that abstract node behaviors, SquidASM requires users to explicitly handle gate timing, memory coherence lifetimes, communication latencies, and scheduling queues. This makes it particularly accurate for modeling protocol performance under real-world hardware constraints.

Key Features:

- Full control over discrete-event scheduling and simulation time.
- Native support for teleportation, entanglement generation, swapping, and measurement-driven operations.
- Qubits are explicitly modeled with lifetimes, decoherence, and quantum memory constraints.
- Communication links simulate delays, noise, and entanglement fidelity degradation.
- Multi-node experiments can be executed with custom protocols and arbitrary topologies.
- Active development.

Limitations:

- Requires manual definition of all quantum and classical protocols, including scheduling logic.
- Steep learning curve and limited documentation for advanced usage.
- Not fully open source, access requires a QuTech account and license
- Primarily oriented toward quantum internet protocols but adaptable for algorithmic DQC.

Evaluation Context: In our benchmarking study, SquidASM was the only platform able to support the complete distributed inverse QFT protocol with full phase encoding, teleportation, and classical feedback plus realistic discrete-event simulation.

2.1.3. Interlin-q

Interlin-q [16] is an application-oriented emulator developed by University of Naples Federico II and collaborators in the Quantum Internet Alliance (QIA). Intelin-q takes a monolithic circuit and partitions it for distributed execution using teleportation. It automates circuit partitioning and distributed execution over a QuNetSim [9] backend, which is used for simulating the quantum network.

It supports all the quantum gates necessary for the benchmarking test, with their documentation detailing a distributed version of the Quantum Phase Estimation as an use case, so the programability is straightforward. However, teleportation is handled natively and there was way to use the cat-entangler and cat-disentangler optimizations used in Section 3, potentially increasing teleportation count and overhead as compared to other emulators.

Key features:

- Automates circuit partitioning and
- Provides built-in support for teleportation and distributed execution using QuNetSim.

Limitations: prototype stage

- Automatic partition is hardcoded to use basic gate-teleportation
- No native noise modelling, should be implemented by hand with probabilistic gates.
- No active development.

Evaluation Context: Interlin-q was evaluated as a prototype DQC simulator that handles automatic circuit partitioning and teleportation. Its design is oriented toward high-level distributed execution using the QuNetSim backend. We included Interlin-q in our study primarily for qualitative comparison, noting its current status as an early-stage research tool rather than a mature emulator.

2.1.4. SQUANCH

SQUANCH (Simulator for QUAntum Networks and CHannels) [4] is a hardware-oriented emulator developed by the AT&T Research Labs. It is a Python-based framework designed to simulate quantum networks through an agent-based model. It provides support for density matrix simulation of qubit systems and represents a middle ground between educational tools and low-level network modeling.

It uses classes such as QSystem, QStream, Agents and Channels. QSystem represents the quantum state of a multi-particle entangled system as a complex-valued density matrix. QStream represents a collection of disjoint (mutually unentangled) quantum systems, such as an ensemble EPR pairs. Finally, each agent in SQUANCH can send, receive, and process both quantum and classical information through user-defined channels. These channels simulate time delays and can introduce configurable noise. Internally, SQUANCH uses Python's multiprocessing module and shared memory structures to simulate entangled states across nodes.

Although it provides a usable abstraction for distributed protocols, SQUANCH lacks timing granularity and does not implement discrete-event simulation. Moreover, it only supports den-

sity matrix representation, which limits its scalability in terms of memory footprint.

Key features:

- Agent-based model with classical and quantum memory spaces, running in separate processes.
- Uses shared memory to manage entangled states between agents.
- Provides basic gate library, including teleportation primitives.

Limitations:

- No discrete-event engine, timing is simulated via blocking/sleep delays.
- No native noise modelling, should be implemented by hand with probabilistic gates.
- No active development.

Evaluation Context: SQUANCH was tested for compatibility with DQC-style protocols. Due to its flexible API, it could support basic teleportation logic. We included SQUANCH for qualitative comparison.

2.2. Other Tools Considered

Several other emulators were reviewed but excluded from benchmarking due to missing features, incompatible architectures, or lack of gate-level programmability for the distributed inverse QFT task.

2.2.1. NetSquid

A discrete-event simulator for quantum networks with nanosecond resolution and detailed modelling of noise, latency, and hardware effectsm developed by QuTech, and is part of the European Quantum Internet Alliance (QIA) software stack. NetSquid [5] supports entanglement distribution, teleportation, and complex network topologies. However, it lacks a direct circuit-level interface and requires integration with higher-level tools (e.g., SquidASM, NetQASM) to execute gate-based algorithms. For this study, it was used indirectly via SquidASM but was not benchmarked on its own.

2.2.2. SimulaQron

SimulaQron [6] is a virtual quantum network simulator, also developed by QuTech within the QIA to enable the application-level prototyping of distributed quantum protocols. It provides a network of virtual nodes, each running as a classical process, with simulated quantum registers that can be entangled across nodes. Quantum operations are delegated to pluggable local backends such as ProjectQ or QuTiP, while classical communication between nodes is handled through TCP sockets.

It does not implement discrete-event timing or detailed noise models, and circuit execution is limited so running the full distributed QFT is challenging. Furthermore, we expect it to scale poorly because of the dependence on the backends ProjectQ and QuTiP for the simulation. Furthermore, is is no longer actively developed, and has been deprecated by tools like NetQASM and SquidASM in the QIA ecosystem. While it was useful for early-stage protocol development, it is not appropriate for teleportation-intensive algorithms like our distributed QFT.

2.2.3. NetQASM SDK

A high-level SDK and intermediate representation for distributed quantum applications. Another tool from QuTech and the QIA, NetQASM SDK abstracts hardware details and compiles quantum-classical programs for backends such as SquidASM or SimulaQron, which both use the low-level instruction set architecture NetQASM [10]. Its value lies in portability and code reuse rather than direct simulation, but its performance and fidelity characteristics depend entirely on the backend used. For this reason, we focused on benchmarking SquidASM directly.

2.2.4. SeQUeNCe

A modular discrete-event simulator developed at the University of Chicago and the Argonne National Laboratory (U.S.) for evaluating the performance of quantum communication networks, especially repeater chains and entanglement routing strategies [7]. It models link-layer behaviours, memory decoherence, and signal loss with precision, but its quantum logic support is limited to communication tasks. Lacking native teleportation gates or arbitrary quantum circuit execution, it was unsuitable for our distributed QFT benchmark.

2.2.5. QuNetSim

An educational and prototyping tool for quantum communication protocols [9]. It provides an easy-to-use API for sending qubits, generating entanglement, and exchanging classical messages, but omits timing models, gate execution, and realistic noise. The absence of circuit-level control and teleportation-based computation support made it incompatible with our benchmarking goals, however it is used as a basis for Interlin-q for handling network control.

2.2.6. DQC Executor

A framework for deploying distributed circuits using an OpenQASM input and YAML-based topology descriptions, mapping them to NetSquid simulations [12]. While the tool is straightforward and has a low-entry barrier, it is in early stages and it is not actively developed. Furthermore, as it depends entirely on NetSquid it is more of an intermediate automation layer.

2.2.7. QNE-ADK

. The Quantum Network Explorer Application Development Kit (QNE- ADK) [11] is a high-level, application-oriented tool for setting up experiments and workflows in SquidASM and NetSquid. It automates scaffolding though command line commands, and abstracts away many simulator details. However, it is not a simulator by itself, depending completely on SquidASM and NetSquid. As previously mentioned, we focused on benchmarking SquidASM direcly.

2.2.8. QuISP

A C++-based simulator designed for large-scale quantum communication and network routing analysis [8], developed by the National Institute of Information and Communications Technology (NICT, Japan) in collaboration with Keio University. While it can handle massive numbers of entangled links with simplified error models, it lacks low-level gate fidelity tracking and teleportation logic inside nodes, preventing it from executing distributed quantum algorithms like QFT.

2.3. Emerging tools

We note that several new emulators and software tool for distributed or networked quantum computing have appeared recently, such as Qoala [17] (another tool in the QIA family), NetQMPI [18], or SimDisQ [19]. These projects are promising and reflect the rapid evolution of the DQC software ecosystem, but they were identified only in the final stages of preparing this report and could not be evaluated within our benchmarking framework. We therefore leave their testing to future studies.

2.4. Summary of Emulator Capabilities

This subsection provides a consolidated overview of the capabilities of the distributed quantum computing emulators considered in this work. While these tools span different design goals, ranging from monolithic circuit simulation to full discrete-event modeling of quantum networks, they can be compared along a common set of criteria: teleportation support, multi-node execution, noise and network modeling, event-driven timing, and active development status. Table 4 summarizes these features and highlights which emulators were included in our benchmarking.

Emulator	Telep.	QFT	Event Sim.	Net Model	Noise	Incl.	Active
SquidASM	✓	✓	✓	✓	✓	✓	2025
Qiskit Aer	~	✓	✗	✗	✓	✓	2025
Interlin-q	~	✓	✗	~	✗	✓	2021
SQUANCH	✓	✓	✗	~	~	✓	2018
NetSquid	✓	~	✓	✓	✓	✗	2021
SimulaQron	✓	~	✗	✓	~	✗	2021
NetQASM SDK	~	~	~	~	~	✗	2025
SeQUeNCe	✓	✗	✓	✓	✓	✗	2025
QuNetSim	✓	✗	✗	✗	✗	✗	2023
DQC Executor	~	~	~	~	~	✗	2021
QNE-ADK	~	~	~	~	~	✗	2024
QuISP	✓	✗	✓	✓	~	✗	2025

Table 4: Comparison of distributed quantum emulators by supported features. Symbols: ✓= supported, ~ = partially supported / requires user implementation / depends on other tools, ✗= not supported. The "Included" column indicates whether the emulator was benchmarked in this study. The "Active" column reports the most recent year of public updates (documentation, release, or commit).

3. Benchmarking Methodology

3.1. Distributed Inverse Quantum Fourier Transform

To benchmark the capabilities of distributed quantum emulators, we selected the Quantum Fourier Transform (QFT) algorithm as a testbed, implemented in a distributed fashion via gate teleportation. The QFT is a canonical quantum algorithm and a key subroutine in applications such as Shor's factoring algorithm [20] and quantum phase estimation [21]. It performs a transformation analogous to the classical discrete Fourier transform, but with exponential speedup for certain structured inputs.

The QFT transforms a computational basis state $|x\rangle$ into a superposition with relative phases:

$$QFT\,|x\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i x k/N}\,|k\rangle\,, \tag{1}$$

where $N = 2^n$ for an n-qubit system. It is typically implemented using Hadamard gates and controlled phase rotations, as shown in Figure 1 resulting in high entanglement and global qubit connectivity, making it a stringent test of emulator capabilities. The inverse QFT is particularly well-suited for benchmarking because its output can be deterministically verified when applied to an input state prepared in the Fourier basis. Specifically, if the input encodes a phase θ via local R_z rotations, the inverse QFT will yield a bitstring approximation of θ with high probability in the noiseless case.

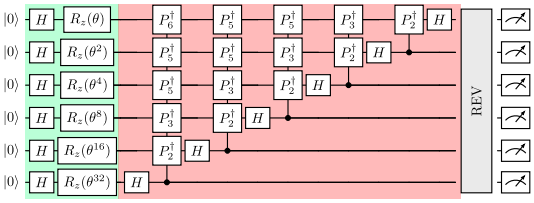


Figure 1: Inverse QFT on 6 qubits (light red) with initial Fourier state preparation (light green). The state preparation consists of a Hadamard state followed by a sequence of phase gates. The QFT circuit applies Hadamard and controlled phase gates, where $P_k^\dagger = \mathrm{P}(e^{-2\pi i/2^k})$. The reverse operator *REV* (which stands for the standard swap gates at the end of the inverse QFT) can be realized by classical postprocessing of the measured bit strings, reducing considerably the entanglement cost [22].

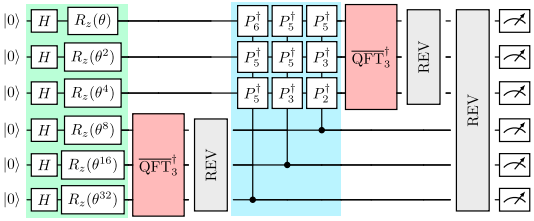


Figure 2: By reordering commuting operations from Figure 1 the inverse QFT can be rewritten into smaller, local inverse QFTs (light red) and a block of controlled phase gradients (light blue). For a bipartite $\mathrm{QFT}^\dagger$ of n qubits, we need $\frac{n}{2}$ EPRs to teleport the phase gradients. The $\overline{\mathrm{QFT}}^\dagger$ represents the usual inverse QFT, without the reverse operator. The local REV operators can be moved forward by reversing the controls of the phase gradients, so that they can be applied classically in post-processing.

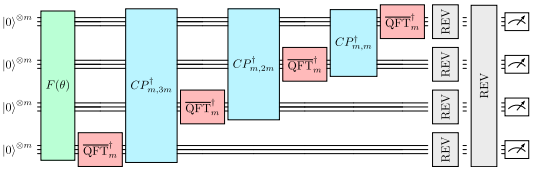


Figure 3: Generalizing the circuit in Fig. 2 to an arbitrary number of nodes k, each one with m qubits, so the whole computation involves $n = km$ qubits. Here $F(\theta)$ is the (local) preparation of the Fourier state, $\mathrm{QFT}_m^\dagger$ is the (local) n-qubit inverse QFT, $CP_{a,b}^\dagger$ is the block of controlled phase gradients with a control qubits and b target qubits. Because we pushed the REV blocks to the end of the circuit as classical post-processing, we need to reverse the order of controls in the controlled phase gradient block. Each phase gradient block can be executed remotely through the use of telegate. The total number of needed EPRs becomes $m\binom{k}{2} = \frac{mk(k-1)}{2}$. If all of the individual CP rotations were to be executed remotely, this number would grow to $\binom{mk}{2}$, that is, quadratic not only in the number of nodes but also the number of qubits per node. Assuming full parallelization of quantum operations, the total number of computing blocks is $2k-1$. Each block of phase gradients involves m layers of phase gates and each $\mathrm{QFT}_m^\dagger$ involves $2m-1$ layers of gates. While the $\mathrm{QFT}_m^\dagger$ are occurring in one node, the rest are executing phase gradients on the rest, with minimal idle time. Measurements can be performed as soon as qubits have finished their gates.

To distribute the QFT across quantum nodes, we employ a recursive factorization of the circuit. Each node handles a local inverse QFT ($\mathrm{QFT}_m^\dagger$) on its m local qubits, while inter-node phase gradients are handled via controlled operations teleported using entangled Bell pairs (EPRs), as shown in Figure 2 for only two nodes. For more than two nodes, we follow a recursive structure shown schematically in Figure 3, where each controlled gradient is sent across nodes using a single EPR per control qubit and receiving node via *cat-entangler/disentangler* circuits [23, 24].

An initial Fourier basis state is prepared locally by each node using Hadamard gates and parameterized R_z rotations of the form $\theta 2^j$. The inverse QFT then recovers the encoded phase, with measurements producing an n-bit approximation of θ.

The inter-node operations are implemented through gate teleportation. Specifically, we use the **telegate** model, in which a control qubit is teleported to the target node using an EPR pair and entangled with a cat-entangler. This allows a series of local controlled phase gates to be applied using the teleported control. A final cat-disentangler circuit restores unitarity and coherence [23].

This method enables the simultaneous teleportation of all controlled-phase gates sharing the same control qubit within a QFT layer. In a fully connected QFT circuit over $N = mk$ qubits (with k nodes of m qubits each), every qubit ideally interacts with all others, leading to $\binom{mk}{2}$ two-qubit gates in total. Each inter-node controlled gate would require a separate teleportation, resulting in a number of required EPR pairs that scales as $O(m^2k^2)$.

By grouping all gates originating from the same control qubit through the *cat-entangle* and *cat-disentangle* procedures, only one teleportation is needed per control qubit and remote node. Consequently, the total number of required EPR pairs is reduced from $\binom{mk}{2}$ to approximately $m\binom{k}{2}$, achieving an overall scaling of $O(mk)$. This linear dependence on both the number of nodes and the number of qubits per node represents a significant improvement over the naïve quadratic scaling in m and k.

3.2. Teleportation-Free Alternatives

We note that an alternative to quantum teleportation is to use deferred measurement and classical control to implement dynamic circuits [25]. In this version, measurements are performed early and classical outcomes are transmitted to other nodes to control local gates. This avoids quantum communication entirely and resembles the circuit structure of iterative quantum phase estimation [26]. Although we focus on gate-teleportation-based QFT in this benchmark, dynamic circuit approaches may offer useful comparisons for future work.

3.3. Performance Metrics

To assess emulator performance, we measure:

- **Execution time:** total time to complete the simulation or emulation of the QFT circuit.
- **Memory usage:** peak memory required during execution, including any inter-node state tracking.
- **Fidelity:** similarity of the final state to the ideal QFT output measured via bitstring distributions (classical fidelity $F = \sum_i \sqrt{p_i q_i}$). Used as a sanity check to make sure the emulator is working as intended.

All metrics are evaluated as a function of the number of qubits n and the number of nodes k. For a strictly statevector emulator, both memory and execution time are expected to grow exponentially with n, as the size of the Hilbert space (and thus the statevectors and unitary matrices) grows accordingly. The scaling with the number of nodes k may uncover the actual capabilities of the emulator for distributed quantum computing. The number of Bell pairs (created and consumed), intermediate measurements and classical messages is expected to grow linearly with k for the test presented in this manuscript.

3.4. Challenges in Benchmarking

Benchmarking emulators poses several practical and theoretical challenges:

- **Heterogeneous APIs:** Each emulator uses a different programming interface, circuit model, and node coordination method.
- **Heterogeneous Environments:** Each emulator has different dependencies, so they need to be installed in isolated environments (e.g. conda) and loaded separately for each job.
- **Gate and message handling:** Emulators vary in their handling of message passing, network latency, mid-circuit measurements, and classical control.

- **Resource tracking:** Not all emulators expose runtime or memory statistics, so custom monitoring has been used for this. We use basic Python instructions from the 'time' and 'tracemalloc' python libraries.
- **Circuit fidelity:** Emulators may implement approximate or symbolic representations of nonlocal gates.
- **Scalability:** Some emulators fail or degrade significantly for moderate node counts or qubit depths due to serialization or backend constraints.

To address these, we structured our benchmarking suite to abstract the benchmark task and use emulator-specific adapters for execution.

3.5. Implementation, Workflow and Automation

All simulations were executed on the QMIO HPC cluster at CESGA [27], a hybrid classical-quantum computing system. Each compute node of QMIO consists of 2x Intel Xeon Ice Lake 8352Y (**ILK**) with 32 cores each (64 cores per node), with 256GB of RAM and 1TB of main memory per node. Jobs were submitted via the SLURM workload manager, ensuring consistent resource allocation and scheduling.

For each emulator, simulations were run in isolated conda environments to manage dependencies and avoid version conflicts. We used fixed software versions (listed below) to ensure reproducibility. The following software versions were used:

- Qiskit Aer v0.17.1, Qiskit v2.1.1 and Python v3.10.16
- SquidASM v0.13.4, NetSquid v1.1.7 and Python v3.8.20
- Interlin-q v0.0.1.post5, QuNetSim v0.1.3.post1 and Python v3.10.18
- SQUANCH v1.1.0 and Python v3.10.16

Additional package dependencies (e.g., `numpy`, `scipy`, `matplotlib`) were managed through `conda`, ensuring reproducibility. Full environment YAML files are provided in the supplementary GitHub repository.

To manage the large-scale parameter sweeps required for benchmarking, we developed an automated workflow. A central configuration file (`sweep_params.yaml`) defines the ranges for qubit counts (n), node counts (k), phase angles (θ), and target emulators. A Python script (`job_creator.py`) parses this configuration and generates SLURM job submission scripts for each emulator and parameter combination by filling a template (`slurm_template_multinode.sh`). This ensures consistent resource allocation and environment setup across all runs. Jobs are submitted either collectively using `job_submit_all.sh` or individually via `job_submit_one.sh` for targeted testing. This modular approach facilitates reproducibility and allows for easy extension of the benchmarking suite.

First we assigned one single core per simulation, with 4GB of RAM per core, and later parallelization behavior was assessed by varying the number of allocated **ILK** cores from 1 to 8. At no point of the simulations there was inter-node communication. Each job was allocated up to 24 hours of walltime, with memory limits scaled according to expected Hilbert space size.

Directory structure of the benchmarking suite. The central configuration (`sweep_params.yaml`) drives automated job generation via `job_creator.py`, which populates per-emulator jobs/ directories. Results are collected in results/ and parsed, analyzed and visualized using `plot_results.py`

```
dqc-emulator-benchmark/
├── configs/sweep_params.yaml
├── scripts/*.py, *.sh
├── emulators/
│   ├── qiskit_aer/
│   │   └── jobs/, results/, scripts/*.py, env/*.yml
│   ├── squidasm/
│   │   └── jobs/, results/, scripts/*.py, env/*.yml
│   ├── interlinq/
│   │   └── jobs/, results/, scripts/*.py, env/*.yml
│   └── squanch/
│       └── jobs/, results/, scripts/*.py, env/*.yml
└── plots/ (auto-generated)
```

4. Results: scalability tests and performance evaluation of distributed QFT implementations

4.1. Description of the scalability test

To evaluate the performance of emulators for distributed quantum computing (DQC), we executed the distributed inverse Quantum Fourier Transform (QFT) on Fourier basis states of increasing size. The benchmark targets execution time, memory consumption, and fidelity as a function of both the number of qubits and the number of computing nodes.

Distributed emulation introduces overheads due to teleportation of nonlocal gates, classical coordination, and synchronization. In the QFT, entangling gates between all qubit pairs make it a stress test for any distributed architecture. Our implementation uses gate teleportation via EPR pairs, and partitions the QFT across k nodes of m qubits each, with $n = km$. As shown in Figure 3, each node executes a local inverse QFT subroutine and participates in remote gate execution through cat-entangler and cat-disentangler primitives.

The theoretical resource requirements grow exponentially with the number of qubits: the Hilbert space dimension doubles per qubit, stressing both memory and compute time. Teleportation introduces further costs proportional to the number of inter-node entangling gates. We used the exact QFT algorithm (no gate truncation), which results in $O(mk^2)$ teleported gate blocks.

An ideal distributed emulator would parallelize across nodes, pipeline computation and communication, and dynamically manage memory and EPRs to maintain high fidelity with efficient resource use. Our benchmarking tests these abilities directly.

4.2. Methods

The benchmark algorithm was implemented in three emulators: Qiskit Aer, SquidASM, and SQUANCH. Each implementation

followed a common high-level structure: preparing a Fourier state with a phase θ, executing the inverse QFT via teleportation across nodes, and measuring output qubit states. The results were compared to a monolithic (non-distributed) reference circuit using fidelity and resource metrics.

The test parameters are summarized in Table 5. We varied:

- The total number of logical qubits n, from 4 to 20.
- The number of nodes k, from 1 to 8. The number of logical qubits n does not need to be a multiple of k, except for Interlin-q.
- The Fourier phase angle $\theta = \{0, 1/3, 2/3\}$. We explore several non-integer angles and average the metrics for them.
- Execution noise models, when available.
- Emulator-specific configurations (message logging, qubit partitioning, etc).

Fidelity between monolithic and distributed runs was computed from measurement outcomes using classical probability overlap. This does only make sense for noiseless emulation.

4.3. Results and Emulator Comparisons

4.3.1. Qiskit Aer.

Qiskit Aer is not a native DQC emulator but supports gate teleportation through manual programming of classical control flow. We implemented teleportation circuits explicitly, resetting and reusing ancillary qubits for EPRs. Despite the overhead, Qiskit performed well up to 16–18 qubits and 4 nodes. Its use of optimized statevector simulation likely accounts for its speed in the noiseless case. However, memory and time scale poorly with increased nodes, and it lacks primitives for quantum network noise (e.g., entanglement fidelity or latency modeling). Furthermore, message passing is entirely implicit, limiting visibility into communication overhead.

4.3.2. SquidASM.

SquidASM is built for distributed execution, compiling circuits into NetSquid simulations. The network topology, qubit layout, and link definitions are programmable via YAML configuration files. Although native support for certain gates (e.g., controlled-phase) was missing, they were constructed from basic gates. SquidASM proved reliable and accurate, producing high-fidelity results even with increasing nodes. Execution times were longer than Qiskit for small problems but scaled more favorably with node count, suggesting better parallelization and concurrency. Importantly, SquidASM simulates full quantum state evolution with noise and messaging delays, making it well-suited for protocol evaluation.

4.3.3. Interlin-q.

Interlin-q provides a higher-level abstraction for distributed quantum computing, automating the circuit partitioning and teleportation processes across multiple nodes. It builds on the QuNetSim communication framework and handles both circuit cutting and message scheduling internally. In our tests, Interlin-q correctly executed small-scale distributed inverse QFTs but exhibited faulty fidelities and poor scaling for larger systems, worsening with the number of nodes. Furthermore, the tool requires the same number of qubits per node, which constrained circuit partitioning and limited automation. Moreover, its automatic partitioner does not implement optimized distribution techniques such as the *cat-entangle* and *cat-disentangle* grouping, leading to redundant teleportations and communication overhead. Execution time and memory usage both grew exponentially with the number of qubits and nodes, and some large configurations failed to complete. While conceptually elegant, the implementation remains at a prototype stage and requires significant optimization for accurate and scalable emulation.

4.3.4. SQUANCH

SQUANCH offers an intuitive distributed programming model but lacks a discrete-event engine. It failed to synchronize teleportation properly, causing fidelity to drop sharply as circuit depth or node count increased. While it showed promising scaling both in time and memory, specially with larger number of nodes, its scaling for monolithic emulation was very steep. While SQUANCH appears to parallelize efficiently its computation between nodes, it does struggle with large node sizes. Known issues with its coherence tracking and message handling were evident in our tests. Furthermore, SQUANCH's simulation did not include any version of noise or realistic behavior. These findings match prior observations that SQUANCH struggles with multi-hop entanglement and gate synchronization.

4.4. Resource Metrics and Trends

- **Execution time** increases exponentially with qubit count, as seen in Fig. 4, which is to be expected. Qiskit Aer is faster for small sizes, but SquidASM's performance improves relative to Qiskit as node count increases. Interlin-q is slower than both, increases with node number, and shows a growing slope for higher qubit numbers. For SQUANCH however, both metrics improve dramatically for the larger number of nodes. Most of the experiments with interlin-q and SQUANCH did not even finish in the allocated time.
- **Memory usage** similarly grows with circuit size, however as seen in the second panel of Fig. 4 Qiskit Aer memory usage grows sub-exponentially with qubit count, unlike SquidASM, SQUANCH and Interlin-q, which both follow exponential trends. SquidASM shows more stable memory behavior with the number of qubits, with a steady exponential increase. However, increasing the number of nodes still increases Qiskit Aer's peak memory consumption, while parallel scalability is better handled in SquidASM, which shows flat or even decreasing time with more nodes in some regimes, unlike Qiskit. Interlin-q shows an exponential growth in both qubit number and node number, while SQUANCH shows great efficiency in memory with more, smaller nodes.

- **Fidelity of noiseless execution** shown in Fig. 5 stays constant for both SquidASM and Qiskit Aer across all tested qubit counts and node numbers, indicating that their distributed execution models preserve the logical structure of the circuit with negligible numerical error (up to shot noise variance). In contrast, Interlin-q and SQUANCH exhibit substantially reduced fidelities that degrade with increasing problem size. For these emulators, fidelity decreases more rapidly as either the number of qubits or the number of nodes grows, even in the absence of noise.

Param.	Description	Values
`Qubits`	Monolithic circuit size	4, 6, 8, 10, 12, . . . , 20
`Nodes`	Count of distributed nodes	1, 2, 4, 8
`Theta`	Fourier state phase param.	0.0, 0.33, 0.66
`Shots`	Circuit repetitions	100
`Backends`	Execution simulators	`squidasm`, `qiskit_aer`, `interlin-q`, `squanch`
`Topology`	Connectivity scheme	All-to-all
`Partition`	Qubit distribution strategy	Evenly partitioned, remainder in last
`Metrics`	Performance outputs	Time, memory, fidelity

Table 5: Summary of parameters used in the benchmarking experiments.

5. Discussion and Conclusions

5.1. Discussion

Our benchmarking of distributed quantum Fourier transform (QFT) implementations across multiple emulators highlights both the current capabilities and the limitations of existing distributed quantum computing (DQC) emulation platforms. The results provide valuable insights into the trade-offs between scalability, accuracy, and resource usage when simulating quantum algorithms in a distributed fashion.

Each emulator illustrates a different point in the design space of DQC simulation: Qiskit Aer as a flexible monolithic baseline, SquidASM as a purpose-built discrete-event simulator, Interlin-q as a partitioning prototype, and SQUANCH as a lightweight conceptual tool

Qiskit Aer shows strong performance as a monolithic simulator adapted for DQC through manual circuit partitioning. However, it has no native notion of a quantum network, as teleportation and classical coordination must be implemented by hand, and there is no built-in support for communication latency, EPR fidelity, or other network-level imperfections. Memory growth in our tests was unexpectedly modest, suggesting Aer employs a memory-efficient representation (e.g., decision diagrams) rather than a pure statevector or density matrix. Runtime benefits from Aer's inherent parallelization, but execution time still grows rapidly with the number of nodes due to teleportation overhead.

SquidASM offers the most complete feature set for realistic DQC emulation, with full discrete-event simulation, network noise modeling, and accurate synchronization. Both time and memory usage scale exponentially with total qubits, as expected, but distributing the same number of qubits across more nodes can improve scaling, indicating efficient memory management and concurrent execution across agents. In contrast to Qiskit Aer, adding CPU cores did not yield substantial performance gains. While SquidASM is primarily oriented toward quantum internet protocols, it can be adapted for algorithmic DQC benchmarking. A major non-technical limitation is that it is not open source, requiring authentication through QuTech's portal to obtain a license.

Interlin-q automates circuit partitioning and teleportation scheduling, but its scaling is strictly exponential and worsens with increased node count. The tool requires equal qubit counts per node, limiting flexibility in partitioning strategies and complicating automation. More critically, its partitioning does not implement our cat-entangler/cat-disentangler optimization, likely resulting in many more teleportations than necessary. This inefficiency, combined with its rigid architecture, constrains its suitability for large-scale DQC algorithm emulation.

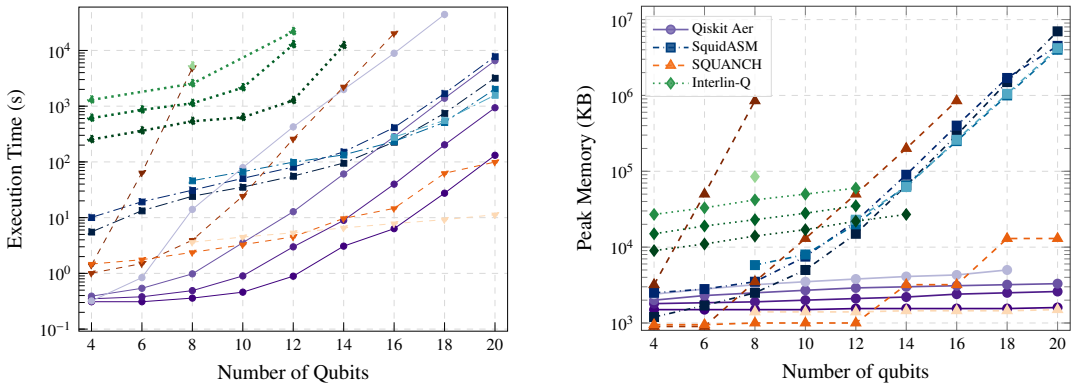
SQUANCH provides an intuitive API for describing distributed quantum protocols, but its lack of a discrete-event simulation engine and limited synchronization support lead to severe fidelity degradation as qubit count or node count increases. In our tests, teleportation could not be consistently synchronized at larger scales, and execution times were significantly higher than with other platforms. While its architecture is suitable for conceptual exploration and education, these limitations make it unsuitable for quantitative benchmarking alongside the other emulators in this study. For this reason, we present SQUANCH's results separately from the main performance figures.

From a methodological perspective, we find that emulators vary significantly in how they handle:

- Mid-circuit measurements and classical feedforward;
- Distributed entanglement and EPR pair management;
- Scheduling and time synchronization across nodes;
- Resource reporting (e.g., message statistics or memory snapshots).

These differences affect not only simulation performance but also the reproducibility and interpretability of results across platforms.

The recursive distributed QFT algorithm used in our benchmark proves to be a useful stress test, as it combines local computation with teleportation-intensive remote gate blocks. It exercises a variety of core features required in distributed systems, such as parameterized gate control, qubit routing, and classical coordination between nodes.



(a) (b)

Figure 4: Execution time (**a**) and peak memory (**b**) usage as a function of the number of qubits, for different numbers of nodes and emulators. Each emulator is represented by a colour gradient: the darkest curve corresponds to single-node (monolithic) execution, while progressively lighter shades indicate increasing node counts.

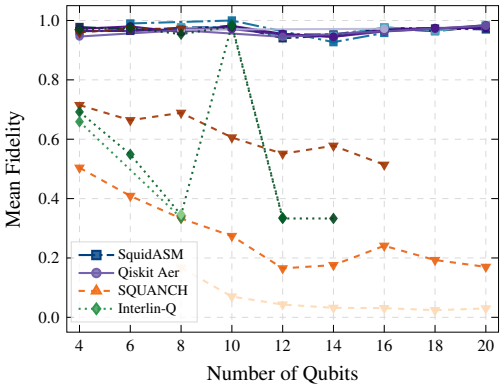


Figure 5: Classical fidelity of noiseless execution as a function of the number of qubits, for different numbers of nodes and emulators. As in 4, lighter shades of the same colour indicates increasing node counts.

It is important to note that the distribution strategies implemented in different emulators are not always equivalent. In particular, tools such as Interlin-q perform automatic gate teleportation for every two-qubit gate whose operands reside on different nodes, which can introduce a significant communication overhead. In contrast, our custom implementation introduces an optimized procedure based on *cat-entangle* and *cat-disentangle* operations, which groups all controlled gates sharing the same control qubit into a single teleportation step. This reduces the number of required EPR pairs from $\binom{mk}{2}$ to $m\binom{k}{2}$, effectively improving the scaling of communication resources from $O(m^2k^2)$ to $O(mk)$.

While this optimization may not be directly comparable to the default behavior of other emulators, it represents a general and transferable approach for efficiently distributing QFT circuits in a multi-node environment. Consequently, we include this variant as an illustration of how algorithm-aware scheduling can significantly reduce the cost of distributed quantum simulations.

5.2. Conclusions

Our study demonstrates that distributed quantum computing emulators are becoming increasingly capable, but there remains substantial variability in feature support, scalability, and performance. The key takeaways are:

- **Emulation of distributed quantum algorithms is possible**, but requires careful algorithm decomposition and tool-specific adaptations.
- **Resource efficiency varies widely across platforms**: low-level discrete-event emulators like SquidASM offer superior memory and runtime scalin, while higher-level tools (e.g. Interlin-q, SQUANCH) show reduced fidelity or fail at larger problem sizes.

- **Fidelity remains a concern at large scales**, especially in platforms that abstract communication or do not natively support gate teleportation.
- **Distributed execution is not yet standard** in many general-purpose simulators. Tools like Qiskit Aer can be extended for DQC, but their internal models are not optimized for teleportation and cross-node messaging.

More broadly, our results highlight a gap in the current ecosystem: to the best of our knowledge, no toll simultaneously emulates physically distributed quantum processors *and* supports distributed-memory classical simulation. Such a *distributed distributed* emulator would allow scalable studies of multi-node architectures and network-aware algorithms, but remains technically challenging and an important direction for future work.

As quantum hardware continues to evolve, emulation and simulation will remain crucial for developing distributed protocols and system architectures. Our benchmarking methodology and test circuits can serve as a foundation for future assessments of DQC platforms.

5.3. Limitations and Future Work

The main limitations of our study are the relatively small number of emulators tested and the fixed benchmark algorithm. Due to time and engineering constraints, only four emulators were fully evaluated, though many more were reviewed in the emulator overview section. NetQASM SDK, while promising as a DQC SDK, was not benchmarked due to compatibility issues between its SquidASM and Simulaqron backends. Simulaqron itself lacks support for general quantum circuits and was excluded. Other emulators (SeQUeNCe, QuNetSim, QuISP) are focused on network-level simulations and were excluded due to lack of native gate-level programmability, circuit construction primitives, or missing teleportation support.

In addition, our analysis did not explore several system-level performance aspects. We did not perform a detailed study of multithreading, GPU acceleration, or backend-specific optimizations, even though these can strongly influence runtime and memory behavior. For example, Qiskit Aer supports GPU backends, but such capabilities were outside the scope of this work. Furthermore, we did not examine compiler optimizations, circuit re-writing strategies, or teleportation scheduling policies beyond our hand-crafted cat-entangle/disentangle approach. These factors can significantly alter both communication overhead and global resource usage.

Our results also reflect default simulation modes, which could impact performance. For example, Qiskit Aer can switch between statevector and density-matrix mode, or use a stabilizer simulator for Clifford-dominated circuits, while SquidASM can configure different physical- and link-level models within NetSquid. These choices can significantly affect runtime, memory usage, and even fidelity, but were beyond the scope of the present work. Exploring alternative formalisms for the emulators (e.g., stabilizer, density-matrix, or tensor-network engines) is an important direction for future benchmarking.

Finally, our benchmark focused exclusively on an all-to-all algorithmic connectivity pattern, as induced by the inverse QFT. This topology does not exercise routing, path selection, or multi-hop entanglement distribution, and therefore does not reveal the strengths or weaknesses of emulators in more realistic network scenarios. For instance, event-driven simulators such as SquidASM or SeQUeNCe are explicitly designed to model routing and link-level contention. Exploring performance under constrained network topologies therefore remains an important direction for future evaluation.

In future work, we plan to extend this benchmarking framework to:

- Include additional emulators as they mature and expand their APIs.
- Benchmark other distributed algorithms such as quantum teleportation networks, distributed Grover's search, or dynamic QKD schemes.
- Explore noise models, link-level imperfections, and error-injection for robustness testing
- Evaluate different simulation formalisms within each emulator.
- Log and analyze quantum/classical message traffic and latency more explicitly.
- Study the influence of classical performance features such as multithreading or GPU acceleration on the scalability of emulation.
- Evaluate emulators under non-trivial network topologies (line, grid, tree, and general graphs), including routing, entanglement swapping, multi-hop scheduling, and path-dependent latency.

Finally, we will release the full benchmarking suite, including scripts, configuration templates, and plotting routines, openly on GitHub, enabling the community to reproduce and expand upon our results.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve language and readability. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgments

This work was supported by MICINN through the European Union NextGenerationEU recovery plan (PRTR-C17.I1), the Galician Regional Government through "Planes Complementarios de I+D+I con las Comunidades Autónomas" in Quantum Communication, MINECO (grants PID2019-104834GB-I00, PID2022-141623NB-I00 and PID2022-137061OB- C22), Consellería de Cultura, Educación e Ordenación Universitaria Galician Research Center accreditation 2024-2027 ED431G-2023/04, and the European Regional Development Fund (ERDF).

References

[1] A. Jamadagni, A. M. Läuchli, C. Hempel, Benchmarking quantum computer simulation software packages: state vector simulators, arXiv preprint ArXiv:2401.09076 (2024).

[2] A. Leonteva, G. Masella, M. Outteryck, A. P. Orioli, et al., Comparative Benchmarking of Utility-Scale Quantum Emulators, arXiv (Apr. 2025). arXiv:2504.14027, doi:10.48550/arXiv.2504.14027.

[3] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, et al., Distributed quantum computing: a survey, Computer Networks 254 (2024) 110672.

[4] B. Bartlett, A distributed simulation framework for quantum networks and channels, arXiv (Aug. 2018). arXiv:1808.07047, doi:10.48550/arXiv.1808.07047.

[5] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, et al., NetSquid, a NETwork Simulator for QUantum Information using Discrete events, Commun. Phys. 4 (164) (2021) 1–15. doi:10.1038/s42005-021-00647-8.

[6] A. Dahlberg, S. Wehner, SimulaQron—a simulator for developing quantum internet software, Quantum Sci. Technol. 4 (1) (2018) 015001. doi:10.1088/2058-9565/aad56e.

[7] X. Wu, A. Kolar, J. Chung, D. Jin, et al., SeQUeNCe: a customizable discrete-event simulator of quantum networks, Quantum Sci. Technol. 6 (4) (2021) 045027. doi:10.1088/2058-9565/ac22f6.

[8] R. Satoh, M. Hajdušek, N. Benchasattabuse, S. Nagayama, et al., QuISP: a Quantum Internet Simulation Package, arXiv (Dec. 2021). arXiv:2112.07093, doi:10.1109/QCE53715.2022.00056.

[9] S. Diadamo, J. Nötzel, B. Zanger, M. M. Beşe, QuNetSim: A Software Framework for Quantum Networks, IEEE Transactions on Quantum Engineering 2 (2021) ArticleSequenceNumber:2502512. doi:10.1109/TQE.2021.3092395.

[10] A. Dahlberg, B. van der Vecht, C. D. Donne, M. Skrzypczyk, et al., NetQASM—a low-level instruction set architecture for hybrid quantum–classical programs in a quantum internet, Quantum Sci. Technol. 7 (3) (2022) 035023. doi:10.1088/2058-9565/ac753f.

[11] qne-adk, [Online; accessed 26. Aug. 2025] (Aug. 2025).
URL https://github.com/QuTech-Delft/qne-adk

[12] dqc-executor, [Online; accessed 26. Aug. 2025] (Aug. 2025).
URL https://github.com/qis-unipr/dqc-executor

[13] qiskit-aer, [Online; accessed 26. Aug. 2025] (Aug. 2025).
URL https://github.com/Qiskit/qiskit-aer

[14] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, et al., Quantum computing with Qiskit (2024). arXiv:2405.08810, doi:10.48550/arXiv.2405.08810.

[15] squidasm, [Online; accessed 26. Aug. 2025] (Aug. 2025).
URL https://github.com/QuTech-Delft/squidasm

[16] R. Parekh, A. Ricciardi, A. Darwish, S. DiAdamo, Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing, arXiv (Jun. 2021). arXiv:2106.06841, doi:10.48550/arXiv.2106.06841.

[17] B. van der Vecht, A. T. Yücel, H. Jirovská, S. Wehner, Qoala: an application execution environment for quantum internet nodes, arXiv preprint arXiv:2502.17296 (2025).

[18] F. J. Cardama, T. F. Pena, Netqmpi: a practical mpi-inspired library for distributed quantum computing over netqasm sdk, in: 2025 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops), IEEE, 2025, pp. 1–2.

[19] S. Zhang, L. Xiong, Y. Liu, B. L. Mark, et al., An end-to-end distributed quantum circuit simulator, arXiv preprint arXiv:2511.19791 (2025).

[20] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[21] A. Y. Kitaev, Quantum computations: algorithms and error correction, Russian Mathematical Surveys 52 (6) (1997) 1191. doi:10.1070/RM1997v052n06ABEH002155.
URL https://dx.doi.org/10.1070/RM1997v052n06ABEH002155

[22] J. Chen, E. M. Stoudenmire, S. R. White, Quantum Fourier Transform Has Small Entanglement, PRX Quantum 4 (4) (2023) 040318. doi:10.1103/PRXQuantum.4.040318.

[23] A. Yimsiriwattana, S. Lomonaco, Generalized ghz states and distributed quantum computing, in: D. Evans, J. Holt, C. Jones, K. Klintworth, et al. (Eds.), CODING THEORY AND QUANTUM COMPUTING, Vol. 381 of Contemporary Mathematics, AMER MATHEMATICAL SOC, P.O. BOX 6248, PROVIDENCE, RI 02940 USA, 2005, pp. 131–147, international Conference on Coding Theory and Quantum Computing, Univ Virginia, Charlottesville, VA, MAY 20-24, 2003.

[24] N. M. Neumann, R. van Houte, T. Attema, Imperfect distributed quantum phase estimation, in: Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part VI 20, Springer, 2020, pp. 605–615.

[25] E. Bäumer, V. Tripathi, A. Seif, D. Lidar, et al., Quantum Fourier Transform Using Dynamic Circuits, Phys. Rev. Lett. 133 (15) (2024) 150602. doi:10.1103/PhysRevLett.133.150602.

[26] J. D. Whitfield, J. Biamonte, A. Aspuru-Guzik, Simulation of electronic structure Hamiltonians using quantum computers, Mol. Phys. (Mar. 2011).
URL https://www.tandfonline.com/doi/abs/10.1080/00268976.2011.552441

[27] J. Cacheiro, Á. C. Sánchez, R. Rundle, G. B. Long, et al., QMIO: A tightly integrated hybrid HPCQC system, arXiv (May 2025). arXiv:2505.19267, doi:10.48550/arXiv.2505.19267.

Chapter 11

Communication-Efficient distributed inverse Quantum Fourier Transform

The content of this chapter is presented in the format of a scientific article which is currently pending publication:

> - **Title:** Communication-Efficient Distributed Inverse Quantum Fourier Transform.
> - **Authors:** F. Javier Cardama, Jorge Vázquez-Pérez, Tomás F. Pena, Andrés Gómez.

This publication specifically addresses and fulfills Objective O6 of this thesis. For further details regarding the publication status of this pre-print, the reader is referred to the Results: unpublished articles section.

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/TQE.2020.DOI

Communication-Efficient Distributed Inverse Quantum Fourier Transform

F. JAVIER CARDAMA[1], JORGE VÁZQUEZ-PÉREZ[3], TOMÁS F. PENA[1,2] and ANDRÉS GÓMEZ[3]

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain
[2]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, 15782, Spain
[3]Galicia Supercomputing Center (CESGA), Avda. de Vigo S/N, Santiago de Compostela, 15705, Spain

Corresponding author: F. Javier Cardama (email: javier.cardama@usc.es).

This work was funded by European Union EuroHPC program with grant agreement 101194491 and by MICIU/AEI/10.13039/501100011033 with project number PCI2025-163229, the Agencia Estatal de Investigación (Spain) (PID2022-141623NB-I00), the Consellería de Cultura, Educación e Ordenación Universitaria Galician Research Center accreditation 2024–2027 ED431G-2023/04, and the European Regional Development Fund (ERDF).

ABSTRACT The scalability of quantum computing is currently limited by physical, technological, and architectural constraints that hinder the integration of a large number of qubits within a single quantum processor. Distributed quantum computing (DQC) has therefore emerged as a viable alternative, aiming to interconnect multiple smaller quantum processing units (QPUs) to jointly operate on a global quantum state. While this paradigm enables scalable architectures, it introduces significant communication overhead due to the cost of non-local quantum operations across distant nodes. In this work, we focus on the inverse Quantum Fourier Transform (iQFT), a fundamental subroutine underlying several cornerstone quantum algorithms, including Quantum Phase Estimation (QPE) and Shor's factoring algorithm. We propose a distributed formulation of the iQFT over a quantum network composed of P nodes, each hosting Q qubits, enabling the execution of the transform on a logical register of size $n = P{\cdot}Q$. Furthermore, we introduce a communication-efficient variant based on a threshold-driven pruning strategy, referred to as a *communication horizon*, which exploits the exponentially decreasing significance of controlled-phase rotations to safely omit remote gates with negligible impact. By reducing the number of inter-node quantum interactions, the proposed approach significantly lowers the quantum communication requirements of the distributed iQFT while preserving its functional correctness. Crucially, we show that this approach fundamentally alters the scaling of the algorithm: the entanglement resource consumption per node saturates to a constant value, reducing the global communication complexity from quadratic $\mathcal{O}(P^2)$ to linear $\mathcal{O}(P)$. As the iQFT constitutes a critical building block in many quantum algorithms, the techniques presented in this paper directly contribute to improving the practicality and scalability of distributed quantum computation.

INDEX TERMS distributed quantum computing, quantum fourier transform, quantum algorithms, quantum communication.

I. INTRODUCTION

DISTRIBUTED quantum computing (DQC) has emerged as a promising architectural approach to overcome the scalability limitations of current monolithic quantum processors [1], [2], which are fundamentally constrained by qubit noise, decoherence, and the increasing complexity of control and calibration as the system size grows [3], [4], [5]. Rather than relying on a single large quantum processor, DQC interconnects multiple smaller quantum processing units (QPUs) through quantum entanglement and classical communication channels, enabling the creation of a global quantum state spanning several physical devices [6], [7], [8].

By entangling qubits across a network of P interconnected QPUs, each hosting Q local qubits, a collective logical register of size $n = P \cdot Q$ can be realized. This modular paradigm allows quantum systems to scale beyond the physical limits of individual processors while relaxing fabrication and control requirements. Such distributed architectures are increasingly regarded as a realistic pathway toward large-scale quantum computation, particularly in the near- to mid-term technological horizon [9], [10].

Quantum computing itself represents a fundamental shift in the computational paradigm, enabling algorithmic speedups for specific problem classes by exploiting quantum superposi-

tion, entanglement, and interference [11]. Several cornerstone quantum algorithms demonstrate this advantage, including the Quantum Fourier Transform (QFT) [12], [13], Quantum Phase Estimation (QPE) [14], Grover's search algorithm [15], and Shor's integer factorization algorithm [16]. These algorithms offer polynomial or even exponential improvements over their classical counterparts for well-defined computational tasks. For instance, Grover's algorithm reduces the complexity of unstructured search from $\mathcal{O}(N)$ to $\mathcal{O}(\sqrt{N})$, while Shor's algorithm leverages quantum phase estimation to factor large integers in polynomial time, an exponential speedup compared to the best known classical methods.

The inverse Quantum Fourier Transform (iQFT) plays a central role in many of these algorithms. In particular, QPE and Shor's algorithm encode their final results in the relative phases of quantum states expressed in the Fourier basis. The iQFT acts as a decoding operation that transforms this phase information back into the computational basis, enabling the extraction of classical outcomes through measurement. Consequently, the efficiency and scalability of the iQFT directly impact the performance of several foundational quantum algorithms.

However, the standard implementation of the iQFT on an n-qubit register requires $\mathcal{O}(n^2)$ two-qubit controlled-phase gates and exhibits an all-to-all interaction pattern, as each qubit must, in principle, interact with every other qubit. This quadratic gate complexity and dense connectivity pose significant challenges for both monolithic and distributed quantum architectures. In distributed settings, the cost of non-local two-qubit gates is particularly high, as they require entanglement generation, quantum communication, and classical synchronization between remote nodes [17], [18], [19].

In this work, we address these challenges by proposing a communication-efficient distributed implementation of the iQFT. The main contributions of this paper are twofold:

- **Distributed iQFT**: we present a systematic decomposition of the iQFT across a quantum network composed of P nodes, each with Q qubits, enabling the execution of the transform over a global register of size $n = P \cdot Q$.
- **Communication-efficient**: we introduce a communication optimized variant of the distributed iQFT based on a threshold-driven pruning strategy. By exploiting the exponential decay of the controlled-phase rotation angles, we define a *communication horizon* that allows remote gates with negligible contribution to be safely omitted, thereby significantly reducing inter-node quantum communication while preserving algorithmic fidelity.

The remainder of this paper is organized as follows. Section II reviews the theoretical background of the iQFT and distributed quantum computing primitives. Section III introduces the proposed distributed iQFT formulation and the communication horizon optimization strategy. Section IV analyzes the impact of the proposed approach in terms of communication cost and accuracy. Finally, Section V and VI conclude the paper and outlines directions for future research.

II. BACKGROUND

A. INVERSE QUANTUM FOURIER TRANSFORM

The Quantum Fourier Transform (QFT) is a linear unitary transformation analogous to the classical discrete Fourier transform. For a quantum system of n qubits, let $N = 2^n$ be the dimension of the Hilbert space. The QFT maps the orthonormal computational basis states $\{|x\rangle\}_{x=0}^{N-1}$ to a new set of orthonormal states, which we refer to as the *Fourier basis* $\{|\widetilde{x}\rangle\}_{x=0}^{N-1}$. This operation is defined as:

$$\mathrm{QFT}|x\rangle = |\widetilde{x}\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i \frac{xy}{N}} |y\rangle. \tag{1}$$

Each state $|\widetilde{x}\rangle$ in the Fourier basis is an equal superposition of all computational basis states, distinguished only by their relative phases. Consequently, the Inverse Quantum Fourier Transform (iQFT) serves as the decoding operation that maps a state from the Fourier basis back to the computational basis:

$$\mathrm{QFT}^{-1}|\widetilde{x}\rangle = |x\rangle. \tag{2}$$

Mathematically, the iQFT is the adjoint of the QFT operator ($\mathrm{QFT}^\dagger$). Its action on an arbitrary computational basis state $|y\rangle$ is given by introducing a negative sign in the complex exponential:

$$\mathrm{QFT}^{-1}|y\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{-2\pi i \frac{xy}{N}} |x\rangle. \tag{3}$$

This transformation is fundamental in quantum algorithms such as Quantum Phase Estimation (QPE) and Shor's algorithm, where the solution is encoded in the relative phases of the qubits (Fourier basis) and must be transformed back to the computational basis to be measured with high probability.

1) Circuit Implementation

The decomposition of the iQFT into a quantum circuit consists of a structured sequence of Hadamard (H) gates and two-qubit Controlled-Phase (CP) rotations, followed by a SWAP network to reverse the order of the qubits. The specific circuit implementation considered in this work is illustrated in Figure 1.

The rotation angle applied to a target qubit $i^\square$, controlled by qubit $i^\bullet$, is determined by the logical index difference k between them, denoted as:

$$\theta(i^\bullet, i^\square) = \theta_{i^\square - i^\bullet} = -\frac{\pi}{2^{i^\square - i^\bullet}}. \tag{4}$$

$CP(\theta_k)$, applies a diagonal phase shift depending on the index difference between the qubits ($k = i^\square - i^\bullet$). For the inverse transform, the rotation angle is negative:

$$CP(\theta_k) = \mathrm{diag}\left(1, 1, 1, e^{-i\frac{\pi}{2^k}}\right), \tag{5}$$

where k represents index difference between the control and target qubits in the register. The exact implementation requires $\mathcal{O}(n^2)$ such gates. However, as k increases, the rotation angle approaches zero.

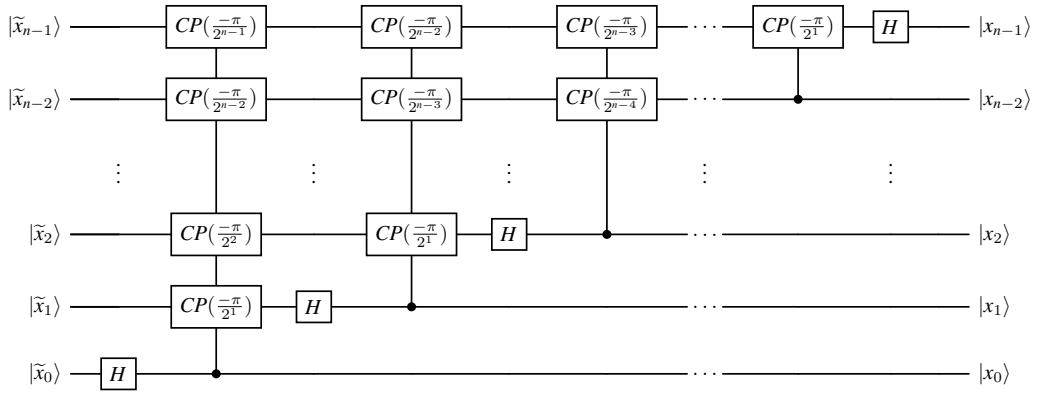


FIGURE 1. Circuit realization of the n-qubit Inverse Quantum Fourier Transform (iQFT). The circuit maps the input state from the Fourier basis $|\widetilde{x}\rangle$ to the computational basis $|x\rangle$ using a sequence of Hadamard gates (H) and controlled phases $CP(\theta_k)$ where $\theta_k = \pi/2^k$.

$$\lim_{k\to\infty} \theta_k = 0. \tag{6}$$

This property is crucial for distributed implementations, as it allows for the truncation of small-angle rotations (approximate iQFT) to reduce inter-node communication, a concept we detail in subsequent sections.

To illustrate the specific connectivity pattern by these operations, Figure 2 describes the interactions between an arbitrary pair qubits, $i^{\bullet}$ as control and qubit $i^{\square}$ as target. The angle θ applied follows the Eq. 4:

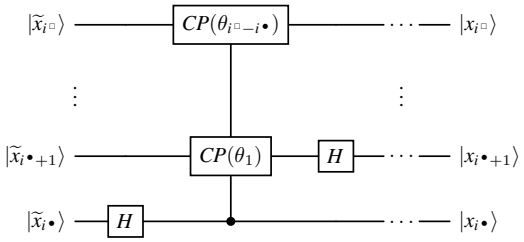


FIGURE 2. Interaction between the $i^{\bullet}$-th qubit as control and the $i^{\square}$-th qubit as target within the iQFT sequence. The diagram highlights the Hadamard gate applied to $|\widetilde{x}_i^{\bullet}\rangle$ and the conditional phase rotation controlled by $|\widetilde{x}_i^{\bullet}\rangle$ acting on $|\widetilde{x}_i^{\square}\rangle$.

2) Gate Complexity and Scaling

The computational cost of the exact iQFT is dominated by the number of two-qubit controlled-phase (CP) rotations required to encode the relative phases. For an n-qubit register, the standard decomposition involves a sequence of operations where the j-th qubit (for $j = 1, \ldots, n$) acts as a control for $n - j$ target qubits.

The total number of controlled-phase gates, denoted as N_{CP}, is determined by the summation of these interactions across the entire circuit:

$$N_{CP}(n) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}. \tag{7}$$

Consequently, the gate complexity scales quadratically with the system size, i.e., $N_{CP} \in \Theta(n^2)$. While the number of Hadamard gates scales linearly as $\Theta(n)$ and the final SWAP network requires $\Theta(n)$ operations, the quadratic growth of the entangling CP gates presents the primary bottleneck for scalability.

Beyond gate count, the iQFT imposes demanding connectivity constraints on the underlying hardware topology. The algorithm dictates that every qubit $i^{\bullet}$ must interact directly via a two-qubit gate with every other qubit $i^{\square}$ (where $i^{\square} > i^{\bullet}$) to apply the necessary phase shifts.

From a graph-theoretic perspective, the interaction graph of the iQFT corresponds to a complete graph K_n, where an edge exists between every pair of vertices.

This disparity between the algorithmic requirement (all-to-all) and physical constraints significantly increases the effective circuit depth and the total operation count in realistic implementations.

B. DISTRIBUTED QUANTUM COMPUTING: TELEDATA AND TELEGATE

Scaling quantum computation beyond the qubit limits of a single processor requires the interconnection of multiple Quantum Processing Units (QPUs) into a distributed network. In such an architecture, the execution of a monolithic quantum circuit is partitioned across distinct nodes. To realize non-local interactions between qubits residing on physically separated QPUs, the system must rely on quantum communication channels and the distribution of entanglement [20], [21], [22].

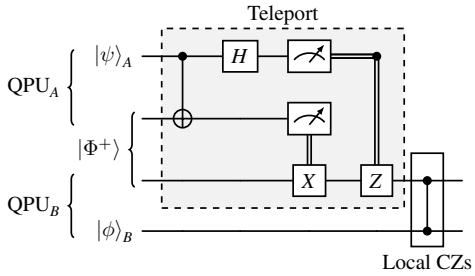


FIGURE 3. Circuit schematic of the *Teledata* protocol. The quantum state $|\psi_A\rangle$ is teleported from QPU$_A$ to QPU$_B$ utilizing a shared EPR pair. Upon receiving the classical measurement outcomes for Pauli corrections (X, Z), the state is reconstructed at the destination. This allows the subsequent interaction with $|\phi_B\rangle$ (depicted here as a CZ gate) to be executed as a local operation, avoiding non-local gate overhead.

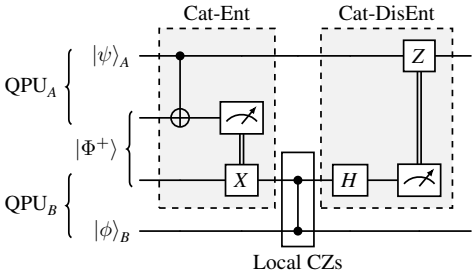


FIGURE 4. Circuit implementation of the *Telegate* protocol. This primitive executes a non-local controlled-Z gate between a control qubit $|\psi_A\rangle$ in QPU$_A$ and a target $|\phi_B\rangle$ in QPU$_B$ using a single shared EPR pair. The process involves an entanglement stage (Cat-Ent) to link the control to the channel, followed by the local interaction, and a disentanglement stage (Cat-DisEnt). Note the bidirectional classical communication required for the X and Z Pauli corrections.

We assume the network links are capable of generating and sharing Einstein-Podolsky-Rosen (EPR) pairs, specifically the Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, between nodes. Utilizing this shared resource, two primary communication primitives are established to support distributed logic: *Teledata* [23] and *Telegate* [24], [25].

1) Teledata: state teleportation

The Teledata strategy, based on the standard quantum teleportation protocol, physically relocates the quantum state of a qubit from a source node A to a destination node B. This approach is necessary when the no-cloning theorem prevents the copying of quantum information, requiring the destruction of the state at the source to reconstruct it at the destination.

Mathematically, let $|\psi\rangle_A = \alpha|0\rangle_A + \beta|1\rangle_A$ be the state to be transmitted. The system begins with the state $|\psi\rangle_A$ and a shared EPR pair $|\Phi^+\rangle_{AB}$ distributed across the nodes:

$$|\Psi_{total}\rangle = |\psi\rangle_A \otimes \frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B). \tag{8}$$

Node A performs a Bell-basis measurement on the qubit to be sent and its half of the entangled pair. This measurement collapses the global state and yields two classical bits, $M_1, M_2 \in \{0, 1\}$. These bits are transmitted via a classical channel to node B. To recover the original state $|\psi\rangle$, node B applies a Pauli correction operation $X^{M_2}Z^{M_1}$ based on the received bits:

$$|\psi\rangle_B = X^{M_2}Z^{M_1}|\phi\rangle_B, \tag{9}$$

where $|\phi\rangle_B$ is the post-measurement state at node B. The circuit implementation for this protocol is depicted in Figure 3.

- **Advantages:** The primary advantage of Teledata is that once the state $|\psi\rangle$ is reconstructed at node B, it becomes local to that node. Consequently, an arbitrary number of subsequent local gates can be executed on $|\psi\rangle$ without further inter-node communication penalties.
- **Disadvantages:** The protocol consumes one EPR pair and requires one round of classical communication. However, if the circuit logic requires the qubit to act as a control for a target back in node A (or another node) after its migration, the state must be teleported back. This "round-trip" operation doubles the cost, consuming a total of 2 EPR pairs and inducing additional latency.

2) Telegate: remote gate application

The Telegate protocol allows the execution of a non-local controlled unitary U between a control qubit q_A in node A and a target qubit q_B in node B. Unlike Teledata, this method does not transport the quantum information of the control qubit. Instead, it temporarily extends the superposition of the control across the network to mediate the interaction.

Let the initial state of the control qubit on QPU A be:

$$q_A : |\psi\rangle_A = \alpha|0\rangle_A + \beta|1\rangle_A, \quad \text{with } \alpha, \beta \in \mathbb{C}, \tag{10}$$

and the network shares an EPR pair

$$e_A, e_B : |\Phi^+\rangle_{AB} = \frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B) \tag{11}$$

The protocol proceeds in three distinct phases:

a: Cat-Entanglement (Cat-Ent)

The goal of this phase is to share the amplitudes of the control qubit into the communication channel. First, a local CNOT is applied between the control q_A and the local EPR half e_A, followed by a Z-basis measurement on e_A (outcome m_A) and a classical feed-forward correction X^{m_A} on e_B.

Mathematically, starting from the joint state:

$$|\Psi_0\rangle = (\alpha|0\rangle_A + \beta|1\rangle_A) \otimes |\Phi^+\rangle_{AB}. \tag{12}$$

1) Apply CNOT$_{A\to e_A}$:

$$|\Psi_1\rangle \propto \alpha|0\rangle_A(|00\rangle_{AB}+|11\rangle_{AB})+\beta|1\rangle_A(|10\rangle_{AB}+|01\rangle_{AB}).$$

Note that the parity of the EPR pair is now correlated with the amplitude β.

2) Measure e_A and correct e_B.
   - If $m_A = 0$, the state collapses to terms with $|0\rangle_A$.
   - If $m_A = 1$, it collapses to terms with $|1\rangle_A$ and the X correction flips e_B.

   In both cases, the system converges to the *cat-entangled* state:

$$|\Psi_{Cat}\rangle = \alpha|0\rangle_A|0\rangle_{e_B} + \beta|1\rangle_A|1\rangle_{e_B}. \tag{13}$$

Interpretation: The probability amplitudes α and β are now *shared* between the original control q_A and the remote EPR proxy e_B. This strictly correlated state $|k\rangle_A|k\rangle_{e_B}$ implies that e_B can be used as control of q_A in the computational basis.

b: Local Interaction

With the amplitudes shared, node B can perform the intended controlled operation using e_B as the control. Let $|\phi\rangle_B$ be the state of the target qubit. The operator CU is applied locally between e_B and $|\phi\rangle_B$:

$$CU_{e_B,q_B} = |0\rangle_{e_B}\langle 0| \otimes \mathbb{I}_{q_B} + |1\rangle_{e_B}\langle 1| \otimes U_{q_B} \tag{14}$$

$$\begin{aligned} |\Psi_{int}\rangle &= CU_{e_B,q_B}\left[(\alpha|0\rangle_A|0\rangle_{e_B} + \beta|1\rangle_A|1\rangle_{e_B}) \otimes |\phi\rangle_B\right] \\ &= \alpha|0\rangle_A|0\rangle_{e_B}(\mathbb{I}|\phi\rangle_B) + \beta|1\rangle_A|1\rangle_{e_B}(U|\phi\rangle_B). \end{aligned} \tag{15}$$

Since e_B is perfectly correlated with A (i.e., e_B is 1 if and only if A is 1), triggering the gate with the proxy e_B is mathematically equivalent to triggering it with the original qubit A. The logical state of the target has evolved correctly according to the superposition of A.

c: Cat-Disentanglement (Cat-DisEnt)

Finally, the protocol must decouple the channel to restore the control qubit. A Hadamard gate is applied to e_B, transforming the basis: $|0\rangle \to |+\rangle, |1\rangle \to |-\rangle$.

$$|\Psi'\rangle = \alpha|0\rangle_A|+\rangle_{e_B}|\phi\rangle_B + \beta|1\rangle_A|-\rangle_{e_B}(U|\phi\rangle_B) - \tag{16}$$

Regrouping terms by the state of e_B:

$$\begin{aligned} |\Psi'\rangle \propto\ & |0\rangle_{e_B}(\alpha|0\rangle|\phi\rangle + \beta|1\rangle U|\phi\rangle) \\ & |1\rangle_{e_B}(\alpha|0\rangle|\phi\rangle - \beta|1\rangle U|\phi\rangle) \end{aligned} \tag{17}$$

A measurement of e_B yields m_B. If $m_B = 1$, a phase flip (minus sign) has occurred on the β term. This is corrected by sending m_B to node A and applying Z^{m_B} on q_A. The final state is:

$$|\Psi_{final}\rangle = (\alpha|0\rangle_A + \beta|1\rangle_A) \otimes \text{controlled-}U(|\phi\rangle_B) \tag{18}$$

Thus, the non-local gate is realized with a single EPR pair, at the cost of two classical communication rounds (m_A and m_B).

- **Advantages:** Telegate is resource-efficient for isolated non-local interactions, as it executes a controlled operation consuming only a single EPR pair (1 EPR). This is half the entanglement cost required to teleport a qubit, perform the gate locally, and teleport it back (Teledata round-trip).

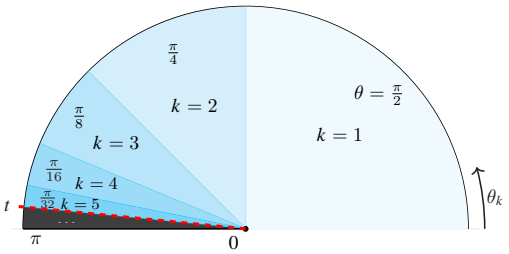


FIGURE 5. Geometric representation of the accumulated phase in the iQFT showing the rapid decay of rotation angles $\theta_k = \pi/2^k$. The explicit sectors for $k = 1$ to $k = 5$ are shown as example. The threshold cutoff t (red dashed line) indicates the truncation point; interactions beyond this limit (black sector) contribute a negligible phase shift bounded by $\varepsilon\pi$.

- **Disadvantages:** The protocol incurs higher synchronization overhead. It requires a bidirectional exchange of classical information to coordinate the sequence: a synchronization signal to initiate the remote sequence and the transmission of measurement results to finalize the gate with Pauli corrections. This exposes the circuit execution to the latency of two distinct classical communication events, potentially stalling the pipeline if not managed efficiently.

C. APPROXIMATE QFT^{-1} WITH PHASE FRACTION ε

The exact formulation of the QFT^{-1} entails a complete interaction graph, requiring a two-qubit controlled-phase rotation $CP(\theta_k)$ for every pair of qubits, where k denotes the index difference between the control and target qubits ($i^{\square} - i^{\bullet}$). As detailed in Section II-A, the rotation angle is given by:

$$\theta_k = -\frac{\pi}{2^k}. \tag{19}$$

A critical observation for optimization is the exponential decay of these rotation angles as k increases. The magnitude of the phase shift is reduced by half for each increment. Consequently, the total phase accumulated by the last target qubit $n-1$, assuming all possible control qubits are in the state $|1\rangle$, follows a geometric progression:

$$\varphi(n-1) = \sum_{k=1}^{n-1}\theta_k = \sum_{k=1}^{n-1}\frac{\pi}{2^k} = \pi\left(1 - \frac{1}{2^{n-1}}\right). \tag{20}$$

As $n \to \infty$, this sum converges to π, representing a maximum phase shift of 180°. However, the contributions from distant qubits (large k) become vanishingly small. This behavior is illustrated in Figure 5, which visualizes how the significance of the rotation diminishes rapidly with k.

To reduce the gate count, we can employ an approximate QFT^{-1} by truncating these negligible rotations. To bound the algorithmic error introduced by this truncation, we define a fractional phase tolerance ε, guaranteeing that the total accumulated phase of the discarded gates strictly does not exceed

$\varepsilon\pi$. Based on this tolerance, we introduce a threshold parameter t, such that all $CP(\theta_k)$ gates with $k > t$ are discarded. The unitary operator for the approximate transform, denoted as $\tilde{U}$, differs from the exact operator U by the omission of these small-angle gates.

To assess the impact of gate truncation, we partition the summation from Eq. 20 into two distinct components based on the selected threshold t. The first component, evaluated as $\varphi(t)$, represents the implemented rotation, while the second component encapsulates the discarded terms. This decomposition allows us to express the exact phase as:

$$\varphi(n-1) = \varphi(t) + \sum_{k=t+1}^{n-1} \theta_k. \tag{21}$$

Consequently, the deviation between the exact and the implemented rotation is determined entirely by the sum of the truncated angles. By substituting the specific rotation values $\theta_k = \pi/2^k$, we obtain:

$$\varphi(n-1) - \varphi(t) = \sum_{k=t+1}^{n-1} \frac{\pi}{2^k}. \tag{22}$$

Using the next property of the geometric series:

$$\sum_{k=t+1}^{n} 2^{-k} = 2^{-t} - 2^{-n}, \tag{23}$$

applying this property to the truncated sum yields the final analytical expression for the approximation error:

$$\varphi(n-1) - \varphi(t) = \pi\left(\frac{1}{2^t} - \frac{1}{2^{n-1}}\right). \tag{24}$$

In addition, an upper bound can be defined by ignoring $1/2^{n-1}$ when $n \to \infty$. This is useful for simplifying subsequent calculations to bound t:

$$\varphi(n-1) - \varphi(t) = \frac{\pi}{2^t}. \tag{25}$$

To ensure that the approximation does not exceed the specified tolerance $\varepsilon\pi$, we require $\varphi(n-1) - \varphi(t) \leq \varepsilon\pi$. Solving for the threshold t, we obtain the cutoff condition:

$$2^t \geq \frac{\pi}{\varepsilon\pi} \implies t \geq \log_2\left(\frac{1}{\varepsilon}\right) = -\log_2(\varepsilon). \tag{26}$$

Thus, to maintain an accuracy of ε, it is sufficient to retain only those controlled-phase gates where

$$k \geq t = \lceil -\log_2(\varepsilon) \rceil. \tag{27}$$

This transforms the gate complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot t)$, offering a significant reduction for large n while preserving algorithmic fidelity.

III. METHODS AND METHODOLOGY

A. DISTRIBUTED IQFT ACROSS N NODES

To distribute the iQFT across a networked architecture, we must restructure the monolithic circuit into a sequence of operations that distinguishes between intra-node (local) and inter-node (distributed) dependencies.

To formalize the distributed architecture, we define a quantum circuit characterized by the following parameters:

- **Network dimension:** The system consists of P nodes, indexed by the identifier $p \in \{0, 1, \ldots, P-1\}$. To explicitly distinguish functional roles during distributed quantum operations, we introduce the modifiers $p^\bullet$ to denote a node acting as a control and $p^\square$ to denote a target node.
- **Node capacity:** We assume a homogeneous distribution where each node hosts exactly Q local qubits. Consequently, the total logical register size is $n = P \cdot Q$.
- **Qubit state notation:** We introduce the notation $|\tilde{x}_q^{(p)}\rangle$ to denote the state of the q-th qubit residing in the p-th node. Here, $q \in \{0, 1, \ldots, Q-1\}$ represents the local index of the qubit within its host node. Similar to the node notation, we use $q^\bullet$ and $q^\square$ to designate a control and target qubit, respectively. To simplify complex mathematical expressions and improve readability, we define the shorthand notation $|\tilde{x}_q^{(p)}\rangle^\bullet$ as strictly equivalent to $|\tilde{x}_{q^\bullet}^{(p^\bullet)}\rangle$, concisely indicating that both the specific qubit and its host node serve the control role. The same equivalence applies to the target notation $|\tilde{x}_q^{(p)}\rangle^\square \equiv |\tilde{x}_{q^\square}^{(p^\square)}\rangle$.
- **Global mapping:** The global index of any given qubit within the complete n-qubit register can be recovered through the linear mapping function $I(q,p) = p \cdot Q + q$.

The standard iQFT decomposition (as seen in Section II-A) requires applying Controlled-Phase (CP) rotations between every pair of qubits. In a distributed setting, these interactions fall into two distinct categories:

1) **Local blocks:** Interactions where both the control and target qubits reside within the same node p (so, $p^\square = p^\bullet$). These operations can be executed entirely within the QPU without utilizing the quantum network.
2) **Communication blocks:** Interactions where the control qubit resides in node $p^\bullet$ and the target qubit resides in node $p^\square$ (with $p^\square \neq p^\bullet$). These require entanglement-assisted communication primitives to realize the non-local phase shifts.

Figure 6 illustrates this segmented architecture for a specific case of $P = 3$ nodes with $Q = 2$ qubits each ($n = 6$). The structure reveals a distinct execution pattern: each node p must first apply a series of *Communication blocks*, interacting sequentially with nodes $p^\bullet < p$. Only after these non-local phases are applied can the node proceed to execute its *Local block*. Crucially, this local component can be identified as a lower-order iQFT instance; specifically, it corresponds to a local iQFT acting on the Q qubits assigned to the node, executed without external dependencies.

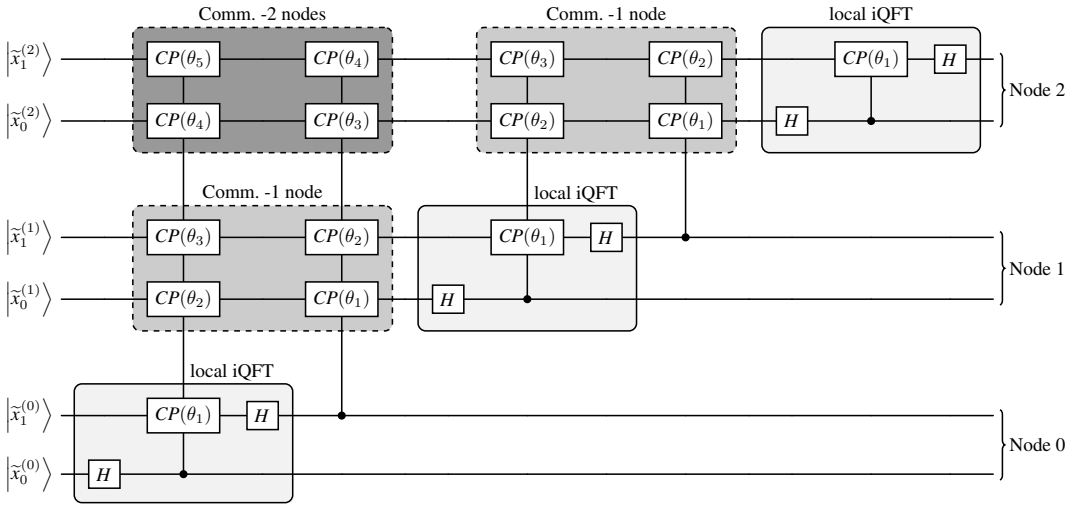


FIGURE 6. Distributed decomposition of a 6-qubit iQFT across 3 nodes ($p = 0, 1, 2$). The circuit is restructured into Local iQFT blocks (light gray), which require no external communication, and Communication Blocks (darker gray), where Node p interacts with qubits from previous nodes $p' < p$. The phase angles follow $\theta_k = \pi/2^k$.

The generalized interaction logic is depicted in Figure 7. For a control qubit $|\tilde{x}_i^{(p)}\rangle^{\bullet}$ in node $p^{\bullet}$ and a target qubit $|\tilde{x}_i^{(p)}\rangle^{\square}$ in a remote node $p^{\square}$, the rotation angle depends on the global index difference between them. Assuming the node index $p^{\square}$ represents the higher-order significance bits, the index difference k between the control qubit $q^{\bullet}$ of node $p^{\bullet}$ and the target qubit $q^{\square}$ of node $p^{\square}$ is derived from their global indices:

$$I^{\bullet} = p^{\bullet} \cdot Q + q^{\bullet}, \quad I^{\square} = p^{\square} \cdot Q + j^{\square}. \tag{28}$$

The rotation index k corresponds to the difference $|I^{\square} - I^{\bullet}|$. In the standard decomposition where controls are applied from distant nodes $p < p'$, the specific rotation angle applying Eq. 4 is given by:

$$\theta(I^{\bullet}, I^{\square}) = \frac{-\pi}{2^{Q(p^{\square}-p^{\bullet})+(q^{\square}-q^{\bullet})}}. \tag{29}$$

This formulation allows us to map the abstract algorithm directly to physical link requirements. The communication blocks effectively group these interactions.

To formally demonstrate that the intra-node operations constitute a standard iQFT of dimension Q (local block), we apply the locality condition $p^{\bullet} = p^{\square}$ to the generalized rotation formula derived in Eq. 29. When both control and target qubits reside in the same node, the term $Q(p^{\square} - p^{\bullet}) = 0$. Consequently, the rotation angle θ simplifies to depend solely on the local indices $q^{\bullet}$ and $q^{\square}$:

$$\theta_{p^{\square}=p^{\bullet}} = \frac{-\pi}{2^{Q(0)+(q^{\square}-q^{\bullet})}} = \frac{-\pi}{2^{q^{\square}-q^{\bullet}}}, \quad \text{for } q^{\square} > q^{\bullet}. \tag{30}$$

This expression is mathematically identical to the phase shift definition of the monolithic iQFT shown in Figure 7 and

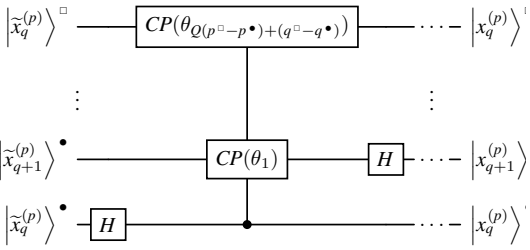


FIGURE 7. Generic interaction between a target qubit $q^{\square}$ in node $p^{\square}$ and a control qubit $q^{\bullet}$ in a distant node $p^{\square}$. The rotation angle is determined by the global index difference, formalized as θ_k where $k = \frac{-\pi}{2^k}$.

Eq. 4, specifically for a difference $k = i^{\square} - i^{\bullet}$. Since the indices i range from 0 to $Q - 1$, the set of operations within this block generates exactly the interaction graph of a Q-qubit iQFT. Thus, the *Local Block* is equivalent to an iQFT(Q), verifying that no inter-node communication is required for these specific gates.

Figure 8 depicts the generalized execution schedule for an arbitrary node p within the network. The quantum circuit is abstracted into a sequence of communicative layers followed by a local computation (as shown in Figure 6).

Instead of treating non-local interactions as isolated gates, the interaction with preceding nodes is structured into communicative blocks denoted by $\mathbf{CP}(\Theta_d)$. Here, d represents the logical distance to the control node (i.e., interacting with node $p^{\square} = p^{\bullet} + d$). Each block encapsulates the complete set of remote controlled-phase rotations required between the

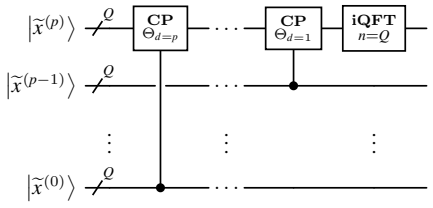


FIGURE 8. Generic interaction between nodes, execution model for an arbitrary node p. The circuit illustrates the sequential application of communication blocks **CP**, starting from the most distant dependency (Node 0, $d = p$) up to the nearest (Node $p-1$, $d=1$). The execution of this sequence assumes that all previous nodes $0,\ldots,p-1$ have already initiated their corresponding operations to act as controls. The process concludes with a local inverse QFT on the Q qubits of node p.

Q qubits of the target node $p^{\square}$ and the Q qubits of the control node $p^{\square}-d=p^{\bullet}$.

Mathematically, Θ_d is defined as the set of all phase rotations to be applied within that specific block. Following the generalized rotation derived in Eq. 29, this set is strictly formulated as:

$$\Theta_d = \{\theta\left(I\left(q^{\bullet},p^{\bullet}\right),I\left(q^{\square},p^{\square}\right)\right) \mid p^{\square}=p^{\bullet}+d\}, \quad (31)$$

where $q^{\square}$ denotes the qubit index in the target register (node $p^{\square}$) and $q^{\bullet}$ the index in the control register (node $p^{\square}-d=p^{\bullet}$). This formulation ensures that all cross-node dependencies are satisfied for the given distance.

As shown in the circuit, the sequence proceeds from the most distant interactions ($d=p$, corresponding to node 0) to the nearest neighbor ($d=1$, corresponding to node $p-1$).

B. COMMUNICATION EFFICIENCY: BLOCK PRUNING STRATEGY

While the standard approximate iQFT reduces the total gate count by discarding small-angle rotations, its impact is particularly transformative in a distributed setting. In monolithic architectures, removing a gate merely saves execution time. However, in a distributed architecture, the primary bottlenecks are the generation of entanglement (EPR pairs) and the latency of classical communication rounds. Therefore, the optimization goal shifts from simply reducing N_{CP} to minimizing the number of active inter-node links.

We leverage the geometric decay of the rotation angles derived in Section II-C to define a *communication horizon*. Recall that for a target accuracy ε, any controlled-phase rotation $CP(\theta_k)$ with distance $k > t$ can be approximated as the identity, where $t=\lceil -\log_2(\varepsilon)\rceil$ (Eq. 27). We apply this threshold to the coarse-grained communication blocks $\mathbf{CP}(\Theta_d)$ (Eq. 31) defined in Section III-A.

A communication block $\mathbf{CP}(\Theta_d)$ encapsulates interactions between a target node $p^{\square}$ and a control node at distance d. The set of rotation indices involved in this block is defined in the Eq. 31. To determine the relevance of the entire block, we consider the *maximum interaction strength* within it, which corresponds to the smallest qubit distance k_{min} (i.e., the largest rotation angle). This occurs between the least significant qubit of the target node ($q^{\square}=0$) and the most significant qubit of the control node ($q^{\bullet}=Q-1$), applying Eq. 29:

$$k_{min}(d) = Q\cdot(p^{\square}-p^{\bullet})+(0)-(Q-1) = Q(d-1)+1. \quad (32)$$

If this strongest interaction falls below the significance threshold (i.e., if $k_{min}(d) > t$), then every other interaction in the block will also satisfy $k > t$. Consequently, the entire communication block becomes negligible and can be pruned from the schedule.

We formalized this by deriving a maximum communication distance, denoted as $d_{\max}$, which bounds the number of remote nodes any given node must communicate with. Setting the condition $k_{min}(d_{\max}) \le t$, we solve for d:

$$Q(d_{\max}-1)+1 \le t \implies d_{\max} \le \frac{t-1}{Q}+1. \quad (33)$$

Substituting t from Eq. 27, we obtain the communication horizon as a function of the error tolerance ε and the node capacity Q:

$$d_{\max} = \left\lfloor \frac{\lceil -\log_2(\varepsilon)\rceil - 1}{Q} \right\rfloor + 1. \quad (34)$$

This derivation implies that a node p only requires entanglement links with its $d_{\max}$ nearest neighbors. Any potential communication block with distance $d > d_{\max}$ is strictly pruned.

This strategy has profound implications for scalability. While the exact distributed iQFT requires an all-to-all communication topology (where node p communicates with all $0\ldots p-1$), our optimized approach bounds the connectivity degree. The number of non-local blocks per node transitions from growing linearly with the network size ($\mathcal{O}(P)$) to being bounded by a constant ($\mathcal{O}(d_{\max})$), effectively localizing the communication overhead regardless of the total number of nodes in the system.

a: Interpretation

To illustrate the practical impact, consider a network composed of nodes with capacity $Q=4$ qubits. Targeting a high algorithmic fidelity with $\varepsilon = 10^{-5}$, the phase truncation threshold is $t \approx 17$. Applying Eq. 34, the communication horizon is bounded by $d_{\max}=5$. This implies that in a large-scale network (e.g., 100 nodes), Node 99 is not required to establish costly long-range entanglement links with Nodes 0 through 93; it strictly needs to coordinate with its 5 nearest predecessors (Nodes 94–98). Interactions beyond this horizon contribute negligible phase shifts and are safely discarded, drastically reducing the demand on quantum repeater networks.

C. METRICS EVALUATED

To rigorously assess the impact of the proposed distributed architecture and optimization strategies, we quantify performance using the following set of metrics. These metrics

are designed to evaluate the trade-offs between algorithmic fidelity, communication resources, and circuit locality.

1) Fidelity vs. Threshold

We evaluate the accuracy of the approximate distributed iQFT by computing the state fidelity, defined as:

$$\mathcal{F} = |\langle \psi_{ideal} | \psi_{approx} \rangle|^2, \tag{35}$$

where $|\psi_{ideal}\rangle$ is the output state of the exact transform and $|\psi_{approx}\rangle$ is the output of the pruned circuit. This metric is analyzed as a function of the pruning threshold t. The objective is to validate the theoretical error model derived in Section II-C by comparing the fidelity obtained from full state-vector simulations against the theoretical lower bound implied by the chosen accuracy parameter ε.

What does it prove? This serves to empirically validate the theoretical error bound and cutoff condition derived in Section II-C (Eq. 27).

2) Scalability of entanglement resources (EPR per Node)

This metric quantifies the aggregate cost of inter-node communication required to execute the algorithm. We measure the total count of EPR pairs consumed across the entire network during the circuit execution. We analyze the behavior of this cost as a function of the pruning threshold t, comparing the baseline distributed iQFT against the optimized implementation.

What does it prove? This metric directly demonstrates the global communication savings achieved by the pruning strategy described in Section III-B.

3) Accuracy under connectivity constraints

Suppose hardware limitation of a fixed communication horizon $d_{\max}$ (the maximum distance between interacting nodes) and a specific node capacity Q, we analyze the achievable algorithmic accuracy. By inverting the relationship established in the block pruning strategy, we determine the minimum error ε that can be guaranteed when the communication topology is restricted.

What does it prove? This analysis confirms the feasibility of the bounded interaction model and the communication horizon ($d_{\max}$) proposed in Section III-B.

4) Coupling Ratio

We define the Coupling Ratio η as the quotient between the number of remote controlled-phase gates (inter-node) and local controlled-phase gates (intra-node). This metric serves as a proxy for the degree of independence of the nodes. We analyze how η evolves with respect to the node capacity Q, the total network size P, and the approximation tolerance ε.

$$\eta(Q, P, \varepsilon) = \frac{N_{CP}^{\text{remote}}}{N_{CP}^{\text{local}}} \tag{36}$$

The objective is to try to achieve a $\eta \to 0$ in the distributed algorithm.

What does it prove? This demonstrates the effectiveness of the distributed decomposition in Section III-A in clustering operations locally to minimize network latency.

5) Communication overhead

To quantify the operational penalty introduced by the network protocols, we define the *Communication Overhead* (γ). This metric is calculated as the ratio between the number of physical operations dedicated exclusively to quantum communication (N_{comm}) and the number of logical computational gates (N_{comp}):

$$\gamma = \frac{N_{\text{comm}}}{N_{\text{comp}}} \tag{37}$$

In the context of the *Telegate* protocol utilized in this work, N_{comm} encompasses all auxiliary operations necessary for state transfer, including Bell pair generation, entanglement swapping operations (CNOTs), and measurement-dependent corrections. Conversely, N_{comp} refers strictly to the logical controlled-phase rotations required by the iQFT algorithm. A visual decomposition distinguishing between these communication-specific overheads and the fundamental computational gates is presented in Figure 4. It is important to emphasize that γ is a global metric, representing the aggregate overhead of the distributed execution rather than a per-node average.

What does it prove? This metric exposes the communication cost of the distributed implementation, quantifying the exact ratio of physical overhead operations required to execute a single useful logical gate.

D. EXPERIMENTAL METHODOLOGY

To validate the proposed distributed architecture and the efficiency of the pruning strategy, we implemented the distributed iQFT algorithm utilizing the Qiskit software development kit. The numerical simulations were performed using the `statevector_simulator` backend provided by Qiskit Aer. This simulator performs an ideal execution of the quantum circuit, computing the exact final quantum state vector $|\psi_{final}\rangle$ rather than a probability distribution of measurement outcomes.

The choice of a state-vector simulation is essential for this study for two primary reasons: first, it allows for the precise calculation of the algorithmic fidelity $\mathcal{F}$ by computing the overlap with the ideal mathematical output, enabling a direct verification of the theoretical error bounds and threshold equations derived in Section II-C. Second, it permits the exact tracking of entanglement resources without the statistical noise associated with shot-based sampling.

All experiments were conducted on the High-Performance Computing cluster of the *Centro Singular de Investigación en Tecnoloxías Intelixentes* (CiTIUS)[1]. Specifically, the simulations were executed on a dedicated compute node designed for memory-intensive workloads, equipped with the following specifications:

- **Processors:** 4 × Intel Xeon Gold 6248 CPUs @ 2.50GHz (20 cores per socket, totaling 80 physical cores).
- **Memory:** 1 TB of RAM.

This high-memory configuration is critical for the study, as the memory requirements for state-vector simulations scale exponentially with the number of qubits (16×2^n bytes). The available 1 TB of RAM enabled the simulation of distributed systems with sufficient qubits to analyze the asymptotic behavior of the communication horizon $d_{\max}$ and the locality ratio η.

IV. RESULTS

A. FIDELITY VS. THRESHOLD VALIDATION

Figure 9 illustrates the infidelity, defined as $1-\mathcal{F}$ (Eq. 35), as a function of the pruning threshold t (Eq. 27) for system size of $Q = 18$ qubits. The data is presented on a semilogarithmic scale to accommodate the wide dynamic range of the error, which spans from approximately 10^{-1} to 10^{-10}.

1) Exponential Convergence

The linearity of the curves in the semilogarithmic plot indicates that the infidelity decays exponentially with a linear increase in the threshold t. This behavior confirms the efficacy of the proposed pruning strategy: a moderate increase in the threshold parameter yields a reduction in the final state error by several orders of magnitude. For instance, in the $Q = 18$ case, increasing the threshold from $t = 4$ to $t = 10$ suppresses the error by nearly four orders of magnitude.

2) Discrepancy between Simulation and Theory

A consistent divergence is observed between the numerical simulations and the analytical predictions. As depicted in Figure 9, the simulated infidelity (solid lines) remains strictly bounded by the theoretical error estimates (dashed lines) derived in Eq. 22.

This discrepancy is attributable to the conservative nature of the theoretical bound, which is derived under a *worst-case scenario* assumption. The theoretical upper bound assumes that every pruned controlled-phase gate, $CP(\theta)$, introduces its maximal possible error. Physically, this corresponds to the scenario where the control and target qubits are invariably in the state $|1\rangle$, maximizing the phase displacement.

However, during the actual execution of the Distributed iQFT, the system occupies a superposition of computational basis states. For any subspace of the wavefunction where the control qubit is in the state $|0\rangle$, the $CP(\theta)$ gate acts as the identity operator (I). Consequently, the removal of the gate in these subspaces introduces zero error.

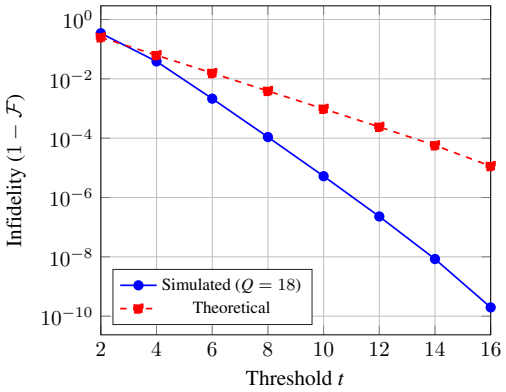


FIGURE 9. Analysis of the algorithmic infidelity ($1-\mathcal{F}$) as a function of the pruning threshold t for local register size $Q = 18$. The data is plotted on a semilogarithmic scale to visualize the exponential decay of the error. Solid lines represent the results obtained from full state-vector simulations, while dashed lines correspond to the theoretical error bounds derived in Eq. 27.

B. REDUCTION OF ENTANGLEMENT CONSUMPTION

Figure 10 presents a comparative analysis of the entanglement resources required for the distributed iQFT, expressed as the number of EPR pairs generated *per node*. The results are depicted as heatmaps varying with the local system size (Q) and the number of computing nodes (P) for three distinct scenarios: the unbounded case (no pruning) and two pruned configurations with thresholds $t = 7$ and $t = 3$.

1) Per-Node Resource Scaling

It is crucial to emphasize that the metric displayed corresponds to the EPR pairs generated *per individual node*. While the total entanglement consumption across the entire network would naturally scale with the number of nodes P, the per-node burden provides a more insightful measure of the local communication overhead and memory requirements. As observed in the leftmost panel (unbounded case), without pruning, the resource demand per node grows linearly with both Q and P, reflecting the dense connectivity of the standard iQFT circuit.

2) Impact of Threshold Pruning

The central and rightmost panels of Figure 10 demonstrate the efficacy of the proposed pruning strategy. By applying a filtering threshold t, the initially quadratic growth of the total EPR count (reflected here as a linear increase per node) is effectively suppressed.

Consistent with the theoretical analysis in Section III-B, the introduction of the threshold acts as a cutoff for long-distance or low-amplitude interactions. For stricter thresholds (e.g., $t = 3$), the number of EPR pairs per node saturates

[1]More information about the HPC cluster: https://wiki.citius.gal/centro:servizos:hpc

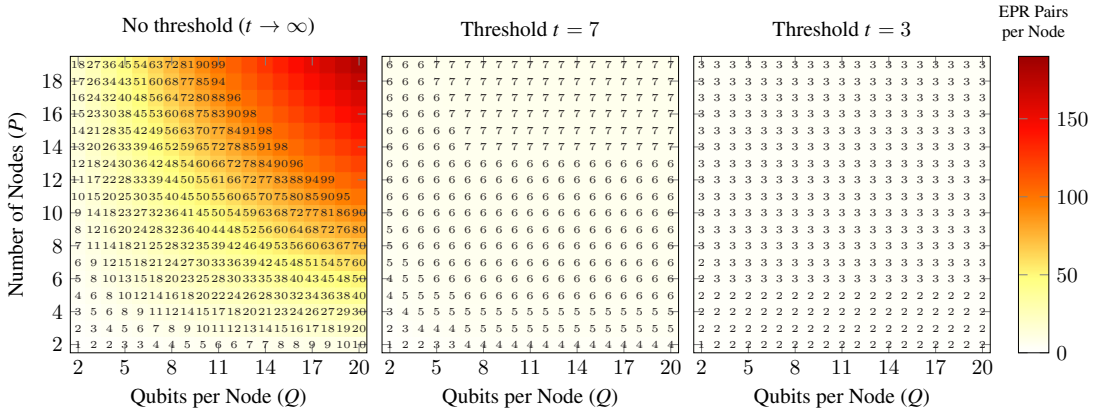


FIGURE 10. **Density of entanglement resources, quantified as the number of EPR pairs generated *per computing node*, as a function of the local register size Q and the total number of nodes P. Note that higher values correspond to increased communication overhead and resource consumption. The leftmost panel (Unbounded) shows a linear increase in local resource demand with respect to P, implying a quadratic global complexity. The central ($t = 7$) and rightmost ($t = 3$) panels demonstrate how the pruning threshold effectively saturates this cost, rendering the resource consumption independent of the network size for sufficiently large distributed systems.**

rapidly, becoming independent of the system size P for large networks. This drastic reduction confirms that the communication complexity can be bounded, transforming the global all-to-all entanglement requirement into a localized resource consumption pattern, thereby making the implementation scalable for large distributed quantum architectures.

C. COUPLING RATIO

To assess the computational efficiency of the distributed implementation relative to the communication overhead, we analyze the *Coupling ratio* (η defined in Eq. 36).

1) High locality

In distributed quantum architectures, the quantum interconnect bandwidth and latency constitute the primary performance bottlenecks, often orders of magnitude slower than local gate operations. Consequently, maximizing data locality is paramount. An ideal distributed algorithm should strive for $\eta \to 0$, indicating that the computational workload is dominated by local processing rather than inter-node data transfer. Minimizing this ratio is essential to prevent the communication network from becoming the limiting factor in the total execution time.

2) Impact of pruning on locality

Figure 11 illustrates the evolution of η as a function of the pruning threshold t. In the standard, unpruned iQFT, the ratio remains constant or grows with system size due to the dense connectivity of the algorithm. However, the application of the threshold t significantly improves this metric.

A central contribution of this work is the establishment of the communication horizon, $d_{\max}(Q, \varepsilon)$ (Eq. 34), which acts as a decisive control parameter for the network overhead the system is willing to tolerate.

Since the amplitude of the controlled-phase rotation $\theta_k = \pi/2^k$ decays exponentially with the logical distance k, the pruning threshold t effectively imposes a hard limit on the interaction range. This mechanism preferentially filters out long-range remote operations—which typically correspond to high k and thus negligible phase shifts. Consequently, $d_{\max}$ serves as a tuning parameter that drastically reduces the numerator N_{CP}^{remote} by enforcing this horizon, while preserving the majority of significant local interactions (N_{CP}^{local}).

D. ACCURACY UNDER CONNECTIVITY CONSTRAINTS

Figure 12 depicts the impact of imposing a fixed communication horizon $d_{\max}$ on the algorithmic accuracy. The heatmaps quantify the theoretical error bound defined in Eq. 22 as a function of the local node capacity (Q) and the maximum allowed communication distance ($d_{\max}$ defined in Eq. 34). This analysis provides a crucial design guide for determining the minimal connectivity required to achieve a target precision.

1) Rapid error convergence

The results demonstrate that the approximation error decays precipitously as the communication horizon increases. Even for moderate horizon values, the fidelity of the distributed iQFT remains high. For instance, in a system with local capacity $Q = 9$ qubits per node, restricting the interaction range to just $d_{\max} = 2$ neighboring nodes yields an error bound of approximately 2^{-10}. This indicates that extending the connectivity beyond a small neighborhood (e.g., $d_{\max} \approx 3$ or 4) provides diminishing returns in accuracy while incurring significant communication costs. Consequently, a highly lo-

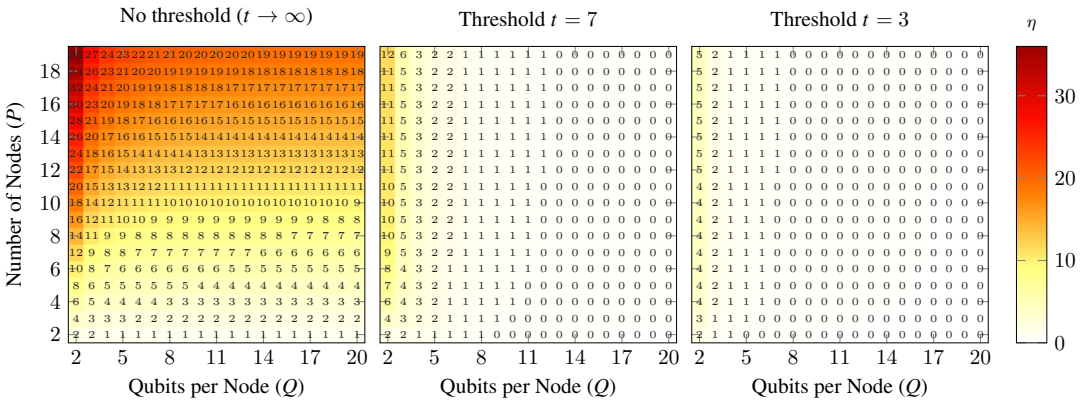


FIGURE 11. Evolution of the Coupling Ratio η, defined as the ratio between remote and local controlled-phase gates ($\eta = N_{remote}/N_{local}$), as a function of the pruning threshold. Note that higher values represent a higher dependency on inter-node communication, which constitutes the primary bottleneck in distributed architectures. The plot demonstrates how the proposed pruning strategy successfully minimizes this ratio ($\eta \to 0$), effectively decoupling the nodes and shifting the computational load towards efficient local processing.

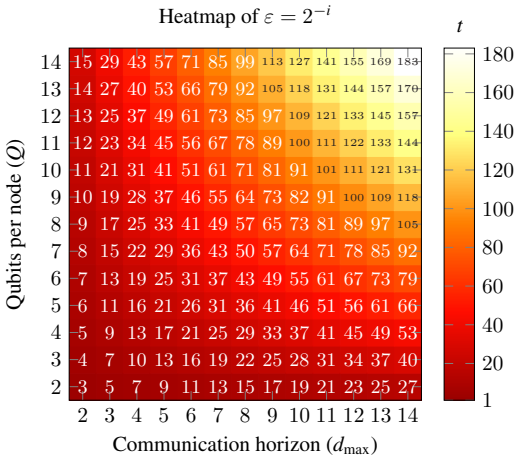


FIGURE 12. Impact of the communication horizon d_{max} and local register size Q on the algorithmic accuracy. The values within the cells represent the negated exponent i of the theoretical error bound (such that $\varepsilon \approx 2^{-i}$). Therefore, higher values indicate lower errors and higher fidelity. A rapid convergence is observed, showing that small horizons are sufficient to achieve high precision.

calized topology is sufficient for most practical applications.

2) Scalability and Network Independence

A fundamental property revealed by this analysis is the independence of the error bound from the total number of nodes P. The accuracy is determined solely by the local register size Q and the communication horizon $d_{\max}$. This implies that a horizon fixed at, for example, $d_{\max} = 3$ ensures the same precision whether the network consists of $P = 20$ nodes or scaled up to $P = 100$ nodes.

This scalability is of paramount importance for the design of large-scale distributed quantum computers. It allows system architects to define a constant connectivity radius that guarantees a specific error tolerance ε, regardless of the global network size.

E. COMMUNICATION OVERHEAD

Figure 13 analyzes the Communication Overhead ratio (γ) across different system sizes and pruning configurations. A crucial observation from these results is that the overhead factor remains relatively invariant with respect to the pruning threshold t.

This stability implies that the application of the threshold does not alter the fundamental architectural efficiency of the distributed protocol. The rationale lies in the fixed operational cost of the *Telegate* mechanism: each executed remote Controlled-Phase gate necessitates a specific, proportional set of auxiliary communication operations (Bell pair generation, entanglement swapping, and corrections). Therefore, when the pruning strategy removes a logical gate, it simultaneously eliminates the associated communication burden.

Consequently, while the *total count* of operations decreases drastically (as detailed in Section IV-C), the *ratio* between communication and computation tasks (γ) remains stable. This confirms that the overhead is an intrinsic property of the chosen quantum interconnect protocol and is not artificially inflated or distorted by the pruning approximation.

V. DISCUSSION

The results presented in this work demonstrate that the communication-aware optimization of the distributed iQFT significantly improves scalability without compromising al-

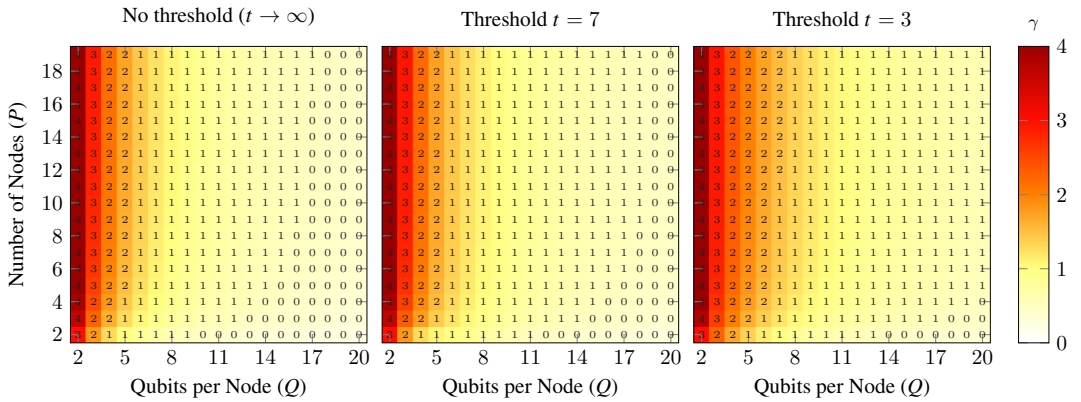


FIGURE 13. Global Communication Overhead γ, defined as the ratio between the number of auxiliary communication operations (entanglement generation, swapping, and corrections) and the logical computational gates ($N_{\text{comm}}/N_{\text{comp}}$). This metric indicates the multiplicative factor by which the communication burden exceeds the useful processing work. Note that higher values are detrimental, representing a regime where the system execution is dominated by network maintenance rather than algorithmic progress.

gorithmic integrity. By restructuring the circuit to exploit the quadratic decay of controlled-phase rotation angles, we effectively transform a global, all-to-all interaction problem into a localized execution model. In this section, we analyze the implications of these findings regarding fidelity bounds, error convergence, and resource decoupling.

A. FIDELITY AND THEORETICAL BOUNDS

A critical observation from the fidelity analysis (Figure 9) is that the empirically simulated infidelity, $1-\mathcal{F}$, is consistently lower than the theoretical upper bound derived in Eq. 22. The theoretical model assumes a worst-case scenario wherein every control qubit involved in a pruned interaction is in the state $|1\rangle$, thereby applying the maximum possible phase shift θ_k. However, in a practical execution environment, the quantum register exists in a superposition of computational basis states. For any component of the wavefunction where the control qubit is $|0\rangle$, the controlled-phase gate acts as the identity $\mathbb{I}$, and its truncation introduces zero error. Consequently, the actual accumulated phase error is statistically lower than the sum of the absolute magnitudes of the discarded angles. This suggests that the proposed pruning thresholds are conservative, offering a safety margin where the realized algorithmic fidelity exceeds the strict design requirements.

B. CONVERGENCE OF THE COMMUNICATION HORIZON

The concept of the communication horizon, $d_{\max}$, emerges as a decisive parameter for distributed architecture design. As evidenced in Figure 12, the algorithmic error converges rapidly with respect to the interaction distance. For a local register size of $Q = 4$, limiting communications to a horizon of $d_{\max} = 3$ nodes yields a theoretical error bound in the order of $\mathcal{O}(2^{-9})$, which is negligible for most fault-tolerant applications. This rapid convergence validates $d_{\max}$ as a sufficient metric to bound the network topology. It implies that for any target precision ϵ, the connectivity graph does not need to scale with the total number of nodes P. Instead, it effectively becomes a constant-degree graph depending only on Q and ϵ. This independence from P is a prerequisite for scalable distributed quantum computing, as it prevents the communication overhead from diverging as the system size increases.

C. ENTANGLEMENT RESOURCE SCALING

The impact of enforcing the communication horizon is most tangible in the consumption of EPR pairs. In the exact, unbounded implementation, the number of EPR pairs required per node grows linearly with the network size, leading to a quadratic aggregate complexity $\mathcal{O}(P^2)$. This is observed in Figure 10 (left panel), where a system of $P = 20$ nodes with $Q = 20$ qubits necessitates ≈ 180 EPR pairs per node. Conversely, applying a pruning threshold of $t = 7$ saturates this growth, capping the requirement at a maximum of approximately 7 EPR pairs per node (Figure 10, center panel), regardless of the network size. This transition from quadratic to linear global scaling (constant per node) indicates that the optimized distributed iQFT is resource-efficient, shifting the bottleneck from entanglement generation rates to local gate fidelities.

D. DECOUPLING VIA THE COUPLING RATIO

Finally, the efficiency of the distributed decomposition is quantified by the Coupling Ratio, $\eta = N_{CP}^{remote}/N_{CP}^{local}$ (Eq. 36). As illustrated in Figure 11, the standard algorithm exhibits a high dependency on remote operations, with η values reaching ≈ 35 for large systems, indicating that inter-node communication dominates the execution time. The application of the optimization strategy drastically suppresses

this ratio. For thresholds $t \leq 7$, η approaches asymptotic zero ($\eta \to 0$). This reduction signifies a fundamental decoupling of the nodes: the computational workload is almost entirely confined to the local QPU, with inter-node links serving only for sparse, nearest-neighbor synchronization. This decoupling ensures that the latency of the quantum channel does not exponentially compound, thereby maintaining the coherence of the global state across the distributed processor.

VI. CONCLUSIONS

In this work, we have presented a communication-aware optimization of the Inverse Quantum Fourier Transform (iQFT) designed specifically for distributed quantum architectures. By rigorously analyzing the geometric decay of the controlled-phase rotation angles, we introduced a pruning strategy that transforms the algorithm's connectivity requirements from a global all-to-all topology to a bounded nearest-neighbor interaction model.

Our theoretical and empirical analyses demonstrate that establishing a communication horizon, $d_{\max}$, allows for a drastic reduction in the consumption of entanglement resources. Specifically, we showed that the number of EPR pairs per node transitions from a linear scaling $\mathcal{O}(P)$ with respect to the network size to a constant saturation level $\mathcal{O}(1)$, effectively making the resource overhead independent of the total number of nodes in the system. Furthermore, the analysis of the Coupling Ratio (η) confirms that our method successfully decouples the processing nodes, reducing the dependency on high-latency inter-node links to near-zero values while maintaining algorithmic fidelity well within the theoretical error bounds derived.

Ultimately, this approach provides a scalable pathway for implementing large-scale Quantum Phase Estimation and other Fourier-based primitives on near-term networked quantum processors, offering a tunable trade-off between execution fidelity and communication bandwidth.

A. FUTURE WORK

While the current study utilized state-vector simulation to validate the theoretical error models and resource scaling exactitude, several key areas for future investigation. First, it would be valuable to extend the validation of this algorithm to specialized distributed quantum computing (DQC) simulators such as NetQMPI, CUNQA, or CUDA-Quantum. These platforms would allow for a more granular analysis of network-specific effects, including link latency, generation rates, and the impact of classical control delays on decoherence.

Additionally, future work should address the integration of this optimized iQFT schedule with quantum error correction codes suitable for distributed environments, such as surface codes over noisy interconnects, to further suppress the effective error rates in the presence of realistic channel noise.

REFERENCES

[1] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: A survey," *Computer Networks*, vol. 254, p. 110672, 12 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624005048

[2] D. Barral, F. J. Cardama, G. Díaz-Camacho, D. Faílde, I. F. Llovo, M. Mussa-Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing," *Computer Science Review*, vol. 57, p. 100747, 8 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013725000231

[3] H. Bluhm and L. R. Schreiber, "Semiconductor spin qubits - A scalable platform for quantum computing?" *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2019-May, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8702477

[4] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 8 2018. [Online]. Available: https://quantum-journal.org/papers/q-2018-08-06-79/

[5] J. Cacheiro, Ã. C. Sánchez, R. Rundle, G. B. Long, G. Dold, J. Friel, and A. Gómez, "QMIO: A tightly integrated hybrid HPCQC system," *Arxiv 2505.19267*, 5 2025. [Online]. Available: https://arxiv.org/pdf/2505.19267

[6] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, 10 2018. [Online]. Available: /doi/pdf/10.1126/science.aam9288?download=true

[7] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler Design for Distributed Quantum Computing," *IEEE Transactions on Quantum Engineering*, vol. 2, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9334411

[8] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti, "Quantum Internet protocol stack: A comprehensive survey," *Computer Networks*, vol. 213, p. 109092, 8 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622002250

[9] P. Escofet, A. Das, S. B. Rached, S. Rodrigo, J. Domingo, F. Sebastiano, M. Babaie, B. Keskin, E. Charbon, P. H. Bolívar, M. Palesi, E. Blokhina, B. Staszewski, A. Nag, A. Garcia-Sáez, S. Abadal, E. Alarcón, and C. G. Almudéver, "On the Impact of Classical and Quantum Communication Networks Upon Modular Quantum Computing Architecture System Performance," *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 984–995, 8 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/11249763

[10] M. Palesi, E. Russo, G. Ascia, H. Rafique, D. Patti, V. Catania, S. Abadal, A. Das, P. Escofet, E. Alarcon, and C. G. Almudéver, "Assessing the Role of Communication in Modular Multi-Core Quantum Systems," *arXiv:2510.11053*, vol. 1, 10 2025. [Online]. Available: https://arxiv.org/pdf/2510.11053

[11] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Information 2016 2:1*, vol. 2, no. 1, pp. 15 023–, 1 2016. [Online]. Available: https://www.nature.com/articles/npjqi201523

[12] Y. S. Weinstein, M. A. Pravia, E. M. Fortunato, S. Lloyd, and D. G. Cory, "Implementation of the Quantum Fourier Transform," *Physical Review Letters*, vol. 86, no. 9, p. 1889, 2 2001. [Online]. Available: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.86.1889

[13] L. Hales and S. Hallgren, "Improved quantum Fourier transform algorithm and applications," *Annual Symposium on Foundations of Computer Science - Proceedings*, pp. 515–525, 2000. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/892139

[14] D. Jiang, X. Liu, H. Song, and H. Xie, "An survey: Quantum Phase Estimation Algorithms," *IEEE Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2021*, pp. 884–888, 10 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9587010

[15] L. K. Grover, "Quantum Mechanics Helps in Searching for a Needle in a Haystack," *Physical Review Letters*, vol. 79, no. 2, p. 325, 7 1997. [Online]. Available: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.79.325

[16] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pp. 124–134, 1994. [Online]. Available: https://ieeexplore.ieee.org/document/365700

[17] C. G. Almudever, R. Wille, F. Sebastiano, N. Haider, and E. Alarcon, "From Designing Quantum Processors to Large-Scale Quantum Computing Systems," *Proceedings -Design, Automation and Test in Europe, DATE*, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10546849

[18] D. Ferrari, S. Carretta, and M. Amoretti, "A Modular Quantum Compilation Framework for Distributed Quantum Computing," *IEEE*

Transactions on Quantum Engineering, vol. 4, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10214316
[19] F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, T. F. Pena, J. C. Pichel, and A. Gómez, "NetQIR: An extension of QIR for distributed quantum computing," *Future Generation Computer Systems*, vol. 174, p. 107989, 1 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X25002845
[20] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, "Distributed quantum computation over noisy channels," *Physical Review A*, vol. 59, no. 6, p. 4249, 6 1999. [Online]. Available: https://journals.aps.org/pra/abstract/10.1103/PhysRevA.59.4249
[21] F. V. Mendes and R. V. Ramos, "Schemes for teleportation of quantum gates," *Quantum Information Processing 2010 10:2*, vol. 10, no. 2, pp. 203–212, 7 2010. [Online]. Available: https://link.springer.com/article/10.1007/s11128-010-0189-7
[22] D. Main, P. Drmota, D. P. Nadlinger, E. M. Ainley, A. Agrawal, B. C. Nichol, R. Srinivas, G. Araneda, and D. M. Lucas, "Distributed quantum computing across an optical network link," *Nature 2025 638:8050*, vol. 638, no. 8050, pp. 383–388, 2 2025. [Online]. Available: https://www.nature.com/articles/s41586-024-08404-x
[23] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Physical Review Letters*, vol. 70, no. 13, p. 1895, 3 1993. [Online]. Available: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.70.1895
[24] D. Gottesman and I. L. Chuang, "Quantum Teleportation is a Universal Computational Primitive," *Nature*, vol. 402, no. 6760, pp. 390–393, 8 1999. [Online]. Available: http://arxiv.org/abs/quant-ph/9908010http://dx.doi.org/10.1038/46503
[25] ——, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations," *Nature 1999 402:6760*, vol. 402, no. 6760, pp. 390–393, 11 1999. [Online]. Available: https://www.nature.com/articles/46503

F. JAVIER CARDAMA (Student Member) received the B.S. degree in Computer Engineering from the University of Santiago de Compostela (USC), Spain, in 2021, and the M.S. degree in High-Performance Computing from the same university in 2022. He is currently pursuing the Ph.D. degree in Information Technology Research at the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), USC, holding a Xunta de Galicia fellowship. His research interests focus on distributed quantum computing and high-performance computing, specifically on the development of software frameworks and architectures for the integration of quantum technologies in HPC centers.

JORGE VÁZQUEZ-PÉREZ is currently a Predoctoral Researcher at the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), University of Santiago de Compostela (USC), Spain. He is also affiliated with the Galicia Supercomputing Center (CESGA) as a Research Technician. His research activity focuses on the intersection of High-Performance Computing (HPC) and Quantum Computing, with a particular interest in distributed quantum architectures, intermediate representations (such as QIR), and the development of compilation tools and software frameworks for heterogeneous quantum-classical systems.

TOMÁS F. PENA (Senior Member) received the Ph.D. degree in physics from the University of Santiago de Compostela (USC), Santiago, Spain, in 1994. He is currently an Full Professor with the Department of Electronics and Computer Science, USC, and a Member of the Research Center in Intelligent Technologies (CiTIUS), Santiago de Compostela, Spain. His research interests include distributed quantum computing, high-performance computing, parallel systems architecture, optimization of performance in irregular codes and with sparse matrices, and Big Data technologies.

ANDRÉS GÓMEZ received the Ph.D. degree in physics from the University of Santiago de Compostela (USC), Spain. He is currently the Head of the Applications and Projects Department at the Galicia Supercomputing Center (CESGA). He leads the Quantum Computing activities at CESGA, overseeing the strategic deployment of the QMIO quantum infrastructure and coordinating various R&D initiatives, including the Quantum Spain project. His research interests encompass high-performance computing, cloud technologies, and the practical implementation of distributed quantum computing algorithms and applications in industrial and scientific environments.

Chapter 12

General Conclusions

The research presented in this thesis addresses one of the most critical challenges to the scalability of quantum information science: transitioning from isolated, monolithic quantum processors to modular, interconnected architectures fully integrated into the HPC ecosystem. This transition is not solely a hardware engineering requirement imposed by the physical limitations of single-chip scaling; rather, it constitutes a fundamental paradigm shift that necessitates a comprehensive re-evaluation of the entire software stack. Throughout this investigation, a substantial gap was identified between the physical protocols that govern quantum networks and the software tools currently available to algorithm designers. Although the underlying physics of quantum teleportation and entanglement swapping is well established, the lack of an appropriate abstraction layer capable of transparently managing these resources has historically posed a major obstacle to the design of complex distributed quantum algorithms.

The central hypothesis guiding this work posits that it is possible to define a hardware-agnostic software stack, based on the SPMD paradigm, that can integrate distributed quantum resources into HPC environments. Furthermore, it was hypothesized that such a stack could effectively manage the complexity of quantum networks–particularly the generation, distribution, and consumption of EPR pairs–in a manner that is transparent to end users. The theoretical analyses, software implementations, and experimental validations presented in this thesis collectively provide substantive support for this hypothesis. The results show that the complexity inherent in physical network protocols can be encapsulated within high-level software primitives, thereby enabling a seamless integration of classical supercomputing workflows with distributed quantum processing.

The fulfillment of the research objectives, as defined in Chapter 2, proceeds along a logical trajectory from theoretical characterization to practical implementation and algorithmic validation.

The initial phase of this work consisted of a rigorous analysis of the state of the art (Objective O1). This comprehensive review revealed a highly fragmented ecosystem

in which low-level imperative tools, such as the NetQASM SDK, require developers to manually manage network topology and entanglement lifecycles. The identification of this development gap provided the primary motivation for the core architectural contribution of this thesis: a layered software stack designed to decouple algorithmic logic from its underlying physical realization. An important outcome of this analysis was the formulation of a clear taxonomy for DQC, which systematically and rigorously characterizes the distinctions among parallel execution, circuit cutting, and fully networked quantum computing. This thesis explicitly positions itself within the framework of the latter paradigm.

To mitigate the lack of compiler-level standardization, this thesis introduces NetQIR (Objective O3), a formal extension of the QIR built on the LLVM infrastructure. By elevating network operations to first-class constructs within the IR, NetQIR establishes a unified abstraction layer that intrinsically accommodates quantum operations spanning multiple nodes. This contribution is fundamental to enhancing compiler scalability, as it permits the application of network-specific optimizations, such as entanglement scheduling and routing, at an earlier stage in the toolchain, prior to machine-code generation. By restructuring the compilation flow in this manner, it effectively reduces the semantic disparity between high-level quantum programming abstractions and low-level network control mechanisms.

Building upon this standardized IR, the research addressed the identified development gap through the design and implementation of NetQMPI (Objective O2). This high-level Python library constitutes a realization of the SPMD paradigm in the quantum computing domain. By adapting both the syntax and semantics of the classical MPI, NetQMPI enables the development of distributed quantum algorithms using high-level semantic primitives (e.g., `qsend`, `qrecv`) rather than low-level physical instructions. The library functions as a middleware layer that automatically incorporates topological context and manages process lifecycles, thereby abstracting the underlying complexity of the quantum network. Furthermore, by adopting NetQASM–an established instruction set and programming language for quantum networks–as its foundational abstraction layer, the library guarantees native interoperability with widely used low-level quantum network simulators reported in the literature, such as NetSquid.

To establish a rigorous experimental foundation, we conducted a comprehensive benchmarking study of the DQC emulation ecosystem (Objective O4). This investigation systematically assessed and compared state-of-the-art emulators reported in the literature, including SeQUeNCe and NetSquid. By quantitatively characterizing their computational performance–specifically execution time and memory consumption–this comparative analysis delineated the operational limits and trade-offs inherent to current simulation frameworks, thereby providing a well-defined baseline for the computational resources required to emulate distributed quantum networks. In addition, the stack's interoperability was demonstrated through its integration with heterogeneous programming models, specifically SYCL (Objective O5). As a foundational

step towards enabling effective CPU-QPU orchestration, a simulated backend based on Qiskit Aer was implemented within the SYCL standard.

Finally, the algorithmic capabilities of the distributed paradigm were examined from a theoretical perspective by optimizing the distributed iQFT (Objective O6). This case study provided a foundation for the development of novel algorithmic methods specifically tailored to networked quantum architectures. In this context, a communication efficient optimization strategy based on establishing a communication horizon ($d_{\max}$) was proposed. Theoretical analyses indicate that the entanglement resource consumption can be reduced from a quadratic scaling, $\mathcal{O}(N^2)$, to a constant-scaling saturation regime, $\mathcal{O}(1)$, by employing distance-based gate pruning, without compromising algorithmic fidelity. These results underscore the fundamental importance of network-aware algorithmic design in optimizing resource utilization. Furthermore, since the iQFT serves as the computational backbone of fundamental quantum subroutines such as QPE and Shor's algorithm, the proposed optimization strategies may significantly affect the practical feasibility and scalability of large-scale distributed quantum applications.

In conclusion, this thesis establishes the fundamental software infrastructure necessary to support the forthcoming era of DQC. Analogous to the role of the MPI standard, which enabled classical HPC to evolve from isolated mainframes into large-scale compute clusters, the NetQIR and NetQMPI frameworks introduced in this work are intended to serve as foundational abstractions and interfaces for networked quantum computation.

The results presented herein indicate that the intricate physical processes governing entanglement distribution and quantum teleportation can be systematically encapsulated within suitably designed software abstraction layers. This abstraction allows computer scientists and algorithm developers to construct sophisticated distributed quantum algorithms without necessitating in-depth expertise in the underlying quantum network physics. Through the proposed infrastructure, the incorporation of quantum computing into HPC data centers shifts from a predominantly conceptual consideration to a technically achievable objective, thereby delineating a concrete trajectory toward scalable, networked quantum–classical HPC architectures.

12.1 Future Work

The research contributions articulated in this thesis delineate several promising trajectories for subsequent inquiry and technological innovation. To further enhance the robustness, scalability, and practical readiness of the distributed quantum software stack, the following research directions are proposed:

- **Integration of NetQIR into the LLVM Toolchain:** Future efforts will focus on embedding the NetQIR specification directly into the mainline LLVM compiler infrastructure. This would ensure direct compatibility with the broader

ecosystem of classical compilers (e.g., Clang) and quantum-classical runtime environments, facilitating a seamless hybrid compilation flow.

- **Adoption of NetQIR as the Intermediate Layer:** A key architectural objective is to formally establish NetQIR as the mandatory intermediate step between the high-level development library (NetQMPI) and the low-level control instructions (NetQASM). This refinement would decouple the application logic from hardware-specific details, enhancing portability and modularity.

- **Expansion of the Benchmarking Framework:** Future research will extend the proposed benchmarking suite by extending its scope beyond conventional computational resources like memory consumption and execution time. Specifically, this extension will incorporate specialized quantum networking performance metrics, including entanglement fidelity, EPR pair generation time, and network latency, thereby enabling a more comprehensive and quantitatively rigorous assessment of distributed quantum communication and computation protocols.

- **Performance Profiling via Benchmarking:** The benchmarking framework developed in this thesis will be applied to perform a rigorous performance analysis of the NetQMPI library itself. This involves identifying computational and communication bottlenecks within the middleware layer to optimize the overhead of entanglement management and process synchronization.

- **Experimental Validation of the iQFT Optimization:** While the optimization of the distributed iQFT was presented as a theoretical proposal, future work aims to conduct a comprehensive empirical validation. This will involve executing the proposed algorithm across various distributed simulation backends using the developed software tools, such as NetQMPI and CUNQA, to verify the theoretical fidelity bounds and resource savings in dynamic network environments.

- **Impact of Error Propagation (iQFT) in Complex Algorithms:** An important direction for future research is to analyze how the approximation errors and fidelity losses identified in the distributed iQFT propagate through larger-scale quantum algorithms, including Shor's factoring algorithm and QPE. By employing the proposed benchmarking methodologies, we intend to quantitatively characterize the relationship between network-induced noise in the iQFT and the final success probability of these higher-level quantum applications.

Bibliography

[1] W. Heisenberg, "Über quantentheoretische Umdeutung kinematischer und mechanischer Beziehungen.", *Zeitschrift für Physik*, vol. 33, no. 1, pp. 879–893, Dec. 1925, ISSN: 14346001. DOI: 10.1007/BF01328377.

[2] E. Schrödinger, "Quantisierung als Eigenwertproblem", *Annalen der Physik*, vol. 384, no. 4, pp. 361–376, Jan. 1926, ISSN: 15213889. DOI: 10.1002/andp.19263840404.

[3] P. A. M. Dirac, *The principles of quantum mechanics*. Oxford university press, 1981.

[4] R. P. Feynman, "Simulating physics with computers", *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, 1982.

[5] D. Deutsch, "Quantum theory, the Church–Turing principle and the universal quantum computer", *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, Jul. 1985, ISSN: 00804630. DOI: 10.1098/rspa.1985.0070.

[6] M. A. Nielsen and I. L. Chuang, "Quantum Computation and Quantum Information: 10th Anniversary Edition", *Quantum Computation and Quantum Information*, Dec. 2010. DOI: 10.1017/cbo9780511976667.

[7] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring", *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pp. 124–134, 1994, ISSN: 02725428. DOI: 10.1109/SFCS.1994.365700.

[8] L. K. Grover, "A fast quantum mechanical algorithm for database search", *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. Part F129452, pp. 212–219, Jul. 1996, ISSN: 07378017. DOI: 10.1145/237814.237866.

[9] A. Montanaro, "Quantum algorithms: An overview", *npj Quantum Information*, vol. 2, no. 1, 2016, ISSN: 20566387. DOI: 10.1038/npjqi.2015.23.

[10] J. Preskill, "Quantum Computing in the NISQ era and beyond", *Quantum*, vol. 2, p. 79, Aug. 2018, ISSN: 2521327X. DOI: 10.22331/q-2018-08-06-79.

[11] F. Arute et al., "Quantum supremacy using a programmable superconducting processor", *Nature 2019 574:7779*, vol. 574, no. 7779, pp. 505–510, Oct. 2019, ISSN: 14764687. DOI: 10.1038/s41586-019-1666-5.

[12] H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software", *Dr. Dobb's Journal*, vol. 30, no. 3, pp. 202–210, 2005. [Online]. Available: http://www.gotw.ca/publications/concurrency-ddj.htm.

[13] C. R. Moore et al., "Data Processing in Exascale-Class Computing Systems", in *2011 Salishan Conference on High-Speed Computing*, Apr. 2011.

[14] K. Rupp et al., *Microprocessor Trend Data*, Commit f2121ab, Feb. 2022. [Online]. Available: https://github.com/karlrupp/microprocessor-trend-data.

[15] B. Davari, R. H. Dennard, and G. G. Shahidi, "CMOS Scaling for High Performance and Low Power—The Next Ten Years", *Proceedings of the IEEE*, vol. 83, no. 4, pp. 595–606, 1995, ISSN: 15582256. DOI: 10.1109/5.371968.

[16] L. Johnsson and G. Netzer, "The impact of Moore's Law and loss of Dennard scaling: Are DSP SoCs an energy efficient alternative to x86 SoCs?", *Journal of Physics: Conference Series*, vol. 762, no. 1, p. 012 022, Oct. 2016, ISSN: 1742-6596. DOI: 10.1088/1742-6596/762/1/012022.

[17] S. Borkar, "Design challenges of technology scaling", *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul. 1999, ISSN: 02721732. DOI: 10.1109/40.782564.

[18] N. Karavadara, M. Zolda, V. T. Nga Nguyen, J. Knoop, and R. Kirner, "Dynamic power management for reactive stream processing on the SCC tiled architecture", *EURASIP Journal on Embedded Systems 2016 2016:1*, vol. 2016, no. 1, pp. 14–, Jun. 2016, ISSN: 1687-3963. DOI: 10.1186/S13639-016-0035-9.

[19] P. P. Gelsinger, "Microprocessors for the new millennium: Challenges, opportunities, and new frontiers", *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, pp. 22–25, 2001, ISSN: 01936530. DOI: 10.1109/ISSCC.2001.912412.

[20] T. Mudge, "Power: A first-class architectural design constraint", *Computer*, vol. 34, no. 4, pp. 52–58, Apr. 2001, ISSN: 00189162. DOI: 10.1109/2.917539.

[21] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H. S. P. Wong, "Device scaling limits of Si MOSFETs and their application dependencies", *Proceedings of the IEEE*, vol. 89, no. 3, pp. 259–287, 2001, ISSN: 00189219. DOI: 10.1109/5.915374.

[22] S. H. Lo, D. A. Buchanan, Y. Taur, and W. Wang, "Quantum-mechanical modeling of electron tunneling current from the inversion layer of ultra-thin-oxide nMOSFET's", *IEEE Electron Device Letters*, vol. 18, no. 5, pp. 209–211, May 1997, ISSN: 07413106. DOI: 10.1109/55.568766.

[23] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: RISC-V Edition, The Hardware Software Interface*. 2017.

[24] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Elsevier, 2011, ISBN: 978-0-12-383872-8.

[25] M. Homewood, D. May, D. Shepherd, and R. Shepherd, "The IMS T800 Transputer", *IEEE Micro*, vol. 7, no. 5, pp. 10–26, 1987, ISSN: 02721732. DOI: 10.1109/MM.1987.305012.

[26] J. P. Hayes, T. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "A Microprocessor-based Hypercube Supercomputer", *IEEE Micro*, vol. 6, no. 5, pp. 6–17, 1986, ISSN: 02721732. DOI: 10.1109/MM.1986.304707.

[27] S. Akhter and J. Roberts, *Multi-Core Programming: Increasing Performance through Software Multi-threading*. Hillsboro, Oregon: Intel Press, 2006, vol. 33, ISBN: 0-9764832-4-6.

[28] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors: A review of their common attributes", *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 26–37, 2009, ISSN: 10535888. DOI: 10.1109/MSP.2009.934110.

[29] D. Patterson et al., "A case for intelligent RAM", *IEEE Micro*, vol. 17, no. 2, pp. 34–43, Mar. 1997, ISSN: 02721732. DOI: 10.1109/40.592312.

[30] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th. Pearson, 2014.

[31] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th. John Wiley & Sons, 2018.

[32] P. J. Denning, "Virtual Memory", *ACM Computing Surveys*, vol. 28, no. 1, pp. 213–216, Mar. 1996, ISSN: 15577341. DOI: 10.1145/234313.234403.

[33] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd. Pearson Education, 2003.

[34] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories", *Proceedings of the 11th annual international symposium on Computer architecture - ISCA '84*, pp. 348–354, 1984. DOI: 10.1145/800015.808204.

[35] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming", *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998, ISSN: 10709924. DOI: 10.1109/99.660313.

[36] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, 2010, p. 330, ISBN: 978-1-4398-1192-4.

[37] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI, 2nd Edition: Portable Parallel Programming with the Message Passing Interface*, 2nd ed. MIT Press, 1999, p. 371, ISBN: 9780262571326.

[38] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes", *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2009*, pp. 427–436, 2009. DOI: 10.1109/PDP.2009.43.

[39] J. Nickolls and W. J. Dally, "The GPU computing era", *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar. 2010, ISSN: 02721732. DOI: 10.1109/MM.2010.41.

[40] S. Rodríguez Alcaraz, "Portable heterogeneous computing on distributed HPC systems", Ph.D. dissertation, University of Santiago de Compostela, Dec. 2025.

[41] Top500 contributors, *Top500 Supercomputer*, 2026. [Online]. Available: https://top500.org/.

[42] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA", *ACM SIGGRAPH 2008 Classes*, p. 16, 2008. DOI: 10.1145/1401132.1401152.

[43] C. Augonnet, S. Thibault, R. Namyst, and P. A. Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5704 LNCS, pp. 863–874, 2009, ISSN: 03029743. DOI: 10.1007/978-3-642-03869-3{_}80.

[44] A. Duran et al., "OmpSs: A proposal for programming heterogeneous multicore architectures", *Parallel processing letters*, vol. 21, no. 2, pp. 173–193, Nov. 2011, ISSN: 01296264. DOI: 10.1142/S0129626411000151.

[45] H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns", *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202–3216, Dec. 2014, ISSN: 07437315. DOI: 10.1016/j.jpdc.2014.07.003.

[46] The Khronos Group, *SYCL 2020 Specification (revision 6)*, 2022. [Online]. Available: https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf.

[47] J. Reinders, B. Ashbaugh, J. Brodman, M. Kinsner, J. Pennycook, and X. Tian, *Data parallel C++: Mastering DPC++ for programming of heterogeneous systems using C++ and SYCL*. Apress Media LLC, Nov. 2020, pp. 1–548, ISBN: 978-1-4842-5574-2. DOI: 10.1007/978-1-4842-5574-2.

[48] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1936. DOI: 10.1112/plms/s2-42.1.230.

[49] D. E. Muller, "Application of Boolean algebra to switching circuit design and to error detection", *Transactions of the I.R.E. Professional Group on Electronic Computers*, vol. EC-3, no. 3, pp. 6–12, Jul. 2013, ISSN: 2168-1740. DOI: 10.1109/irepgelc.1954.6499441.

[50] R. Landauer, "Information is physical", *Physics Today*, vol. 44, no. 5, pp. 23–29, 1976, ISSN: 19450699. DOI: 10.1063/1.881299.

[51] I. M. Ross, "The invention of the transistor", *Proceedings of the IEEE*, vol. 86, no. 1, pp. 7–28, 1998, ISSN: 00189219. DOI: 10.1109/5.658752.

[52] H. Iwai and D. Misra, "The Transistor was Invented 75 Years Ago: A Big Milestone in Human History", *Electrochemical Society Interface*, vol. 31, no. 4, pp. 65–72, Dec. 2022, ISSN: 19448783. DOI: 10.1149/2.F13224IF.

[53] P. W. Shor, "Quantum Computing", *Documenta Mathematica*, vol. 1, no. 1000, pp. 467–486, 1998.

[54] S. P. Jordan, "Quantum Computation Beyond the Circuit Model", Ph.D. dissertation, Massachusetts Institute of Technology, 2008. DOI: 10.48550/arXiv.0809.2307.

[55] D. P. DiVincenzo and D. Loss, "Quantum information is physical", *Superlattices and Microstructures*, vol. 23, no. 3-4, pp. 419–432, Jan. 1998, ISSN: 07496036. DOI: 10.1006/spmi.1997.0520.

[56] C. H. Bennett and D. P. Divincenzo, "Quantum information and computation", *Nature 2000 404:6775*, vol. 404, no. 6775, pp. 247–255, Mar. 2000, ISSN: 00280836. DOI: 10.1038/35005001.

[57] C. J. Isham, *Lectures on Quantum Theory: Mathematical and Structural Foundations*. Imperial College Press, 1995, ISBN: 1-86094-001-3.

[58] J. J. Sakurai and J. Napolitano, *Modern Quantum Mechanics*. Cambridge University Press, 2020, ISBN: 9781108587280. DOI: 10.1017/9781108587280.

[59] C. H. Bennett, "Logical Reversibility of Computation", *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973, ISSN: 00188646. DOI: 10.1147/rd.176.0525.

[60] M. Born, "Quantenmechanik der Stoßvorgänge", *Zeitschrift für Physik*, vol. 38, no. 11-12, pp. 803–827, Nov. 1926, ISSN: 14346001. DOI: 10.1007/BF01397184.

[61] J. Von Neumann, *Mathematical foundations of quantum mechanics: New edition*. Princeton university press, 2018.

[62] A. Einstein, B. Podolsky, and N. Rosen, "Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?", *Physical Review*, vol. 47, no. 10, p. 777, May 1935, ISSN: 0031899X. DOI: 10.1103/PhysRev.47.777.

[63] D. E. Deutsch, "Quantum computational networks", *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 425, no. 1868, pp. 73–90, Sep. 1989, ISSN: 0080-4630. DOI: 10.1098/rspa.1989.0099.

[64] A. Barenco et al., "Elementary gates for quantum computation", *Physical Review A*, vol. 52, no. 5, p. 3457, Nov. 1995, ISSN: 10502947. DOI: 10.1103/PhysRevA.52.3457.

[65] J. S. Bell, "On the Einstein Podolsky Rosen paradox", *Physics Physique Fizika*, vol. 1, no. 3, p. 195, Nov. 1964, ISSN: 0554-128X. DOI: 10.1103/physicsphysiquefizika.1.195.

[66] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels", *Physical Review Letters*, vol. 70, no. 13, p. 1895, Mar. 1993, ISSN: 00319007. DOI: 10.1103/PhysRevLett.70.1895.

[67] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation", *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, Dec. 1992, ISSN: 0962-8444. DOI: 10.1098/rspa.1992.0167.

[68] T. Toffoli, "Reversible computing", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 85 LNCS, pp. 632–644, 1980, ISSN: 16113349. DOI: 10.1007/3-540-10003-2_104.

[69] E. Bernstein and U. Vaziranit, "Quantum complexity theory", *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. Part F129585, pp. 11–20, Jun. 1993, ISSN: 07378017. DOI: 10.1145/167088.167097.

[70] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, "Quantum algorithms revisited", *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, pp. 339–354, Jan. 1998, ISSN: 13645021. DOI: 10.1098/rspa.1998.0164.

[71] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and Weaknesses of Quantum Computing", *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1510–1523, Jul. 2006, ISSN: 00975397. DOI: 10.1137/S0097539796300933.

[72] W. Diffie and M. E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976, ISSN: 15579654. DOI: 10.1109/TIT.1976.1055638.

[73] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, ISSN: 15577317. DOI: 10.1145/359340.359342.

[74] V. S. Miller, "Use of Elliptic Curves in Cryptography", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 218 LNCS, pp. 417–426, 1986, ISSN: 16113349. DOI: 10.1007/3-540-39799-X{_}31.

[75] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987, ISSN: 00162523. DOI: 10.1090/s0025-5718-1987-0866109-5.

[76] A. K. Lenstra and H. W. Lenstra, "The development of the number field sieve", in *The development of the number field sieve*, A. K. Lenstra and H. W. Lenstra, Eds., ser. Lecture Notes in Mathematics, vol. 1554, Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, ISBN: 978-3-540-57013-4. DOI: 10.1007/BFb0091534.

[77] M. Cerezo et al., "Variational quantum algorithms", *Nature Reviews Physics 2021 3:9*, vol. 3, no. 9, pp. 625–644, Aug. 2021, ISSN: 25225820. DOI: 10.1038/s42254-021-00348-9.

[78] A. Peruzzo et al., "A variational eigenvalue solver on a photonic quantum processor", *Nature Communications 2014 5:1*, vol. 5, no. 1, pp. 4213–, Jul. 2014, ISSN: 20411723. DOI: 10.1038/ncomms5213.

[79] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm", pp. 1–16, Nov. 2014.

[80] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'brien, "Quantum computers", *Nature*, vol. 464, no. 7285, pp. 45–53, 2010. DOI: 10.1038/nature08812.

[81] W. Gerlach and O. Stern, "Der experimentelle Nachweis der Richtungsquantelung im Magnetfeld", *Zeitschrift für Physik*, vol. 9, no. 1, pp. 349–352, Dec. 1922, ISSN: 14346001. DOI: 10.1007/BF01326983.

[82] D. P. Divincenzo, "The Physical Implementation of Quantum Computation", *Fortschritte der Physik: Progress of Physics*, vol. 48, pp. 771–783, 2000. DOI: 10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E.

[83] D. P. DiVincenzo, "Quantum computation", *Science*, vol. 270, no. 5234, p. 255, Oct. 1995. DOI: 10.1126/science.270.5234.255.

[84] M. Kjaergaard et al., "Superconducting Qubits: Current State of Play", *Annual Review of Condensed Matter Physics*, vol. 11, pp. 369–395, Apr. 2020. DOI: 10.1146/annurev-conmatphys-031119-050605.

[85] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-Ion Quantum Computing: Progress and Challenges", *Applied Physics Reviews*, vol. 6, no. 2, Apr. 2019. DOI: 10.1063/1.5088164.

[86] P. Wang et al., "Single ion qubit with estimated coherence time exceeding one hour", *Nature Communications 2021*, vol. 12, no. 1, pp. 233–, Jan. 2021, ISSN: 20411723. DOI: 10.1038/s41467-020-20330-w.

[87] S. Slussarenko and G. J. Pryde, "Photonic quantum information processing: a concise review", *Applied Physics Reviews*, vol. 6, no. 4, Dec. 2019. DOI: 10.1063/1.5115814.

[88] L. S. Madsen et al., "Quantum computational advantage with a programmable photonic processor", *Nature*, vol. 606, no. 7912, pp. 75–81, 2022, ISSN: 1476-4687. DOI: 10.1038/s41586-022-04725-x.

[89] A. Chatterjee, P. Stevenson, S. De Franceschi, A. Morello, N. P. de Leon, and F. Kuemmeth, "Semiconductor qubits in practice", *Nature Reviews Physics 2021 3:3*, vol. 3, no. 3, pp. 157–177, Feb. 2021, ISSN: 25225820. DOI: 10.1038/s42254-021-00283-9.

[90] M. W. Doherty, N. B. Manson, P. Delaney, F. Jelezko, J. Wrachtrup, and L. C. Hollenberg, "The nitrogen-vacancy colour centre in diamond", *Physics Reports*, vol. 528, no. 1, pp. 1–45, Jul. 2013, ISSN: 03701573. DOI: 10.1016/j.physrep.2013.02.001.

[91] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons", *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[92] J. Sau, "A roadmap for a scalable topological quantum computer", *Physics*, vol. 10, p. 68, 2017.

[93] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware", *Nature 2017*, vol. 549, no. 7671, pp. 180–187, Sep. 2017, ISSN: 14764687. DOI: 10.1038/nature23459.

[94] A. V. Aho, M. S. Lam, R. S. Avaya, and J. D. Ullman, *Second Edition*, 2nd ed. Pearson Addison Wesley, 2007, ISBN: 0-321-48681-1.

[95] B. Heim et al., "Quantum programming languages", *Nature Reviews Physics*, vol. 2, no. 12, pp. 709–722, Nov. 2020, ISSN: 25225820. DOI: 10.1038/s42254-020-00245-7.

[96] S. Garhwal, M. Ghorani, and A. Ahmad, "Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages", *Archives of Computational Methods in Engineering*, vol. 28, no. 2, pp. 289–310, Dec. 2019, ISSN: 18861784. DOI: 10.1007/s11831-019-09372-6.

[97] S. J. Gay, "Quantum programming languages: survey and bibliography", *Mathematical Structures in Computer Science*, vol. 16, no. 4, pp. 581–600, Aug. 2006, ISSN: 09601295. DOI: 10.1017/S0960129506005378.

[98] C. Developers, *Cirq*, Aug. 2025. DOI: 10.5281/zenodo.16867504.

[99] A. McCaskey, T. Nguyen, A. Santana, D. Claudino, T. Kharazi, and H. Finkel, "Extending C++ for Heterogeneous Quantum-Classical Computing", *ACM Transactions on Quantum Computing*, vol. 2, no. 2, Jul. 2021, ISSN: 26436817. DOI: 10.1145/3462670.

[100] K. Svore et al., "Q#: Enabling scalable quantum computing and development with a high-level DSL", *ACM International Conference Proceeding Series*, Feb. 2018. DOI: 10.1145/3183895.3183901.

[101] A. JavadiAbhari et al., "Scaffcc: A framework for compilation and analysis of quantum computing programs", *Proceedings of the 11th ACM Conference on Computing Frontiers, CF 2014*, vol. 14, 2014. DOI: 10.1145/2597917.2597939.

[102] S. J. Gay and R. Nagarajan, "Communicating quantum processes", *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pp. 145–157, 2005, ISSN: 07308566. DOI: 10.1145/1040305.1040318.

[103] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, "Quantum circuits for isometries", *Physical Review A*, vol. 93, no. 3, p. 032 318, Mar. 2016, ISSN: 24699934. DOI: 10.1103/PhysRevA.93.032318.

[104] M. Y. Siraichi, S. Collange, V. F. Dos Santos, and F. M. Q. Pereira, "Qubit allocation", *CGO 2018 - Proceedings of the 2018 International Symposium on Code Generation and Optimization*, vol. 2018-February, pp. 113–125, Feb. 2018. DOI: 10.1145/3168822.

[105] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters", *npj Quantum Information 2018 4:1*, vol. 4, no. 1, pp. 23–, May 2018, ISSN: 20566387. DOI: 10.1038/s41534-018-0072-4.

[106] G. Li, Y. Ding, and Y. Xie, "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices", *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, vol. 19, pp. 1001–1014, Apr. 2019. DOI: 10.1145/3297858.3304023.

[107] A. Zulehner, A. Paler, and R. Wille, "Efficient mapping of quantum circuits to the IBM QX architectures", *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, vol. 2018-January, pp. 1135–1138, Apr. 2018. DOI: 10.23919/DATE.2018.8342181.

[108] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation", *International Symposium on Code Generation and Optimization, CGO*, pp. 75–86, 2004. DOI: 10.1109/CGO.2004.1281665.

[109] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open Quantum Assembly Language", Jul. 2017.

[110] A. Cross et al., "OpenQASM 3: A Broader and Deeper Quantum Assembly Language", *ACM Transactions on Quantum Computing*, vol. 3, no. 3, Sep. 2022, ISSN: 26436817. DOI: 10.1145/3505636.

[111] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A Practical Quantum Instruction Set Architecture", Feb. 2017.

[112] C. Lattner et al., "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation", *CGO 2021 - Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization*, pp. 2–14, Feb. 2021. DOI: 10.1109/CGO51591.2021.9370308.

[113] A. McCaskey and T. Nguyen, "A MLIR Dialect for Quantum Assembly Languages", *Proceedings - 2021 IEEE International Conference on Quantum Computing and Engineering, QCE 2021*, pp. 255–264, 2021. DOI: 10.1109/QCE52317.2021.00043.

[114] QIR Alliance, *Quantum Intermediate Representation (QIR) Specification*, 2022. [Online]. Available: https://qir-alliance.org/.

[115] C. Monroe et al., "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects", *Physical Review A - Atomic, Molecular, and Optical Physics*, vol. 89, no. 2, Feb. 2014, ISSN: 10502947. DOI: 10.1103/PhysRevA.89.022317.

[116] L. M. Vandersypen et al., "Interfacing spin qubits in quantum dots and donors—hot, dense, and coherent", *npj Quantum Information 2017 3:1*, vol. 3, no. 1, pp. 34–, Sep. 2017, ISSN: 20566387. DOI: 10.1038/S41534-017-0038-Y.

[117] M. Caleffi, A. S. Cacciapuoti, and G. Bianchi, "Quantum internet: From communication to distributed computing!: Invited paper", in *Proceedings of the 5th ACM International Conference on Nanoscale Computing and Communication, NANOCOM 2018*, 2018. DOI: 10.1145/3233188.3233224.

[118] T. Peng, A. Harrow, M. Ozols, and X. Wu, "Simulating Large Quantum Circuits on a Small Quantum Computer", Mar. 2019. DOI: 10.1103/PhysRevLett.125.150504.

[119] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, "CutQC: Using small Quantum computers for large Quantum circuit evaluations", *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, vol. 21, pp. 473–486, Apr. 2021. DOI: 10.1145/3445814.3446758.

[120] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, "Distributed quantum computation over noisy channels", *Physical Review A*, vol. 59, no. 6, p. 4249, Jun. 1999, ISSN: 10941622. DOI: 10.1103/PhysRevA.59.4249.

[121] V. Bužek and M. Hillery, "Quantum copying: Beyond the no-cloning theorem", *Physical Review A*, vol. 54, no. 3, p. 1844, Sep. 1996, ISSN: 10941622. DOI: 10.1103/PhysRevA.54.1844.

[122] D. Gottesman and I. L. Chuang, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations", *Nature 1999 402:6760*, vol. 402, no. 6760, pp. 390–393, Nov. 1999, ISSN: 00280836. DOI: 10.1038/46503.

[123] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio, "Optimal local implementation of nonlocal quantum gates", *Physical Review A*, vol. 62, no. 5, p. 052 317, Oct. 2000, ISSN: 10502947. DOI: 10.1103/PhysRevA.62.052317.

[124] A. Yimsiriwattana and S. J. Lomonaco, "Distributed quantum computing: A distributed Shor algorithm", *Quantum Information and Computation II*, vol. 5436, p. 360, Mar. 2004. DOI: 10.1117/12.546504.

[125] R. V. Meter, "Quantum networking and internetworking", *IEEE Network*, vol. 26, no. 4, pp. 59–64, 2012, ISSN: 08908044. DOI: 10.1109/MNET.2012.6246754.

[126] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead", *Science*, vol. 362, no. 6412, Oct. 2018, ISSN: 10959203. DOI: 10.1126/science.aam9288.

[127] I. F. Llovo, D. C. Guillermo, N. C. Lago, and A. G. Tato, "Network-Assisted Collective Operations for Efficient Distributed Quantum Computing", *IEEE Transactions on Quantum Engineering*, vol. 6, 2025, ISSN: 26891808. DOI: 10.1109/TQE.2025.3619387.

[128] S. Nishio and R. Wakizaka, "InQuIR: Intermediate Representation for Interconnected Quantum Computers", Feb. 2023.

[129] A. Dahlberg et al., "NetQASM—a low-level instruction set architecture for hybrid quantum–classical programs in a quantum internet", *Quantum Science and Technology*, vol. 7, no. 3, p. 35 023, Jun. 2022. DOI: 10.1088/2058-9565/ac753f.

[130] F. Darema, "The SPMD Model: Past, Present and Future", in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, C. Yiannis and J. Dongarra, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, p. 1, ISBN: 978-3-540-45417-5.

[131] T. Coopmans et al., "NetSquid, a NETwork Simulator for QUantum Information using Discrete events", *Communications Physics 2021 4:1*, vol. 4, no. 1, pp. 164–, Jul. 2021, ISSN: 23993650. DOI: 10.1038/s42005-021-00647-8.

[132] A. Dahlberg and S. Wehner, "SimulaQron—a simulator for developing quantum internet software", *Quantum Science and Technology*, vol. 4, no. 1, p. 15 001, Sep. 2018. DOI: 10.1088/2058-9565/aad56e.

[133] S. Diadamo, B. Qi, G. Miller, R. Kompella, and A. Shabani, "Packet switching in quantum networks: A path to the quantum Internet", *Physical Review Research*, vol. 4, no. 4, p. 043 064, Oct. 2022, ISSN: 26431564. DOI: 10.1103/PhysRevResearch.4.043064.

[134] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: A survey", *Computer Networks*, vol. 254, p. 110 672, Dec. 2024, ISSN: 13891286. DOI: 10.1016/j.comnet.2024.110672.

[135] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "QuEST and High Performance Simulation of Quantum Computers", *Scientific Reports 2019 9:1*, vol. 9, no. 1, pp. 10 736–, Jul. 2019, ISSN: 20452322. DOI: 10.1038/s41598-019-47174-9.

[136] H. De Raedt et al., "Massively parallel quantum computer simulator, eleven years later", *Computer Physics Communications*, vol. 237, pp. 47–61, Apr. 2019, ISSN: 00104655. DOI: 10.1016/j.cpc.2018.11.005.

[137] X. Wu et al., "SeQUeNCe: a customizable discrete-event simulator of quantum networks", *Quantum Science and Technology*, vol. 6, no. 4, p. 045 027, Sep. 2021, ISSN: 20589565. DOI: 10.1088/2058-9565/ac22f6.

[138] A. Y. Kitaev, "Quantum measurements and the Abelian Stabilizer Problem", Nov. 1995.

[139] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring", Jan. 2002.

[140] D. Barral et al., "Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing", *Computer Science Review*, vol. 57, p. 100 747, 2025, ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2025.100747.

[141] F. J. Cardama, J. Vázquez-Pérez, T. F. Pena, J. C. Pichel, and A. Gómez, "Quantum Compilation Process: A Survey", *Lecture Notes in Computer Science*, vol. 15385 LNCS, pp. 100–112, 2025, ISSN: 16113349. DOI: 10.1007/978-3-031-90200-0_9.

[142] F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of intermediate representations for quantum computing", *The Journal of Supercomputing 2025 81:2*, vol. 81, no. 2, pp. 418–, Jan. 2025, ISSN: 15730484. DOI: 10.1007/s11227-024-06892-2.

[143] F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, T. F. Pena, J. C. Pichel, and A. Gómez, "NetQIR: An extension of QIR for distributed quantum computing", *Future Generation Computer Systems*, vol. 174, p. 107 989, 2026, ISSN: 0167-739X. DOI: 10.1016/j.future.2025.107989.

[144] F. J. Cardama and T. F. Pena, "NetQMPI: An MPI-Inspired library for programming Distributed Quantum Applications over Quantum Networks using NetQASM SDK", Dec. 2025.

[145] M. Leal, F. Javier Cardama, and T. F. Pena, "SYCL QPU: An LLVM-based QPU simulation framework built using DPC++", *Proceedings of the 2025 IEEE International Conference on Cluster Computing Workshops, CLUSTER Workshops 2025*, 2025. DOI: 10.1109/CLUSTERWorkshops65972.2025.11164192.

[146] G. Díaz-Camacho et al., "Benchmarking Distributed Quantum Computing Emulators", *arXiv*, Dec. 2025. [Online]. Available: https://arxiv.org/abs/2512.01807.

List of Figures

List of Tables

List of Listings

List of Acronyms

ALU Arithmetical Logical Unit. 40

ANTLR ANother Tool for Language Recognition. 90

API Application Programming Interface. 51

CPU Central Processing Unit. 1, 4, 7, 11, 14, 16, 17, 19, 21, 24, 40, 42, 44–47, 51, 52, 239, 256

CQP Communicating Quantum Processes. 81

CUDA Compute Unified Device Architecture. 51

DQC Distributed Quantum Computing. xx, 2–8, 12–26, 37, 38, 73–75, 77, 78, 84, 85, 89, 90, 92, 95–97, 103, 238, 239, 257, 261

DRAM Dynamic Random Access Memory. 43

EPR Einstein-Podolsky-Rosen. 2, 3, 6, 8, 12, 13, 16, 18, 20, 21, 23, 25, 86–88, 93, 96, 103, 237, 240

FPGA Field-Programmable Gate Array. 1, 11, 19, 50, 96

GNFS General Number Field Sieve. 65

GPU Graphics Processing Unit. 1, 7, 11, 16, 17, 19, 24, 50, 51

HDD Hard Disk Drive. 43, 46

HIP Heterogeneous-compute Interface for Portability. 51

HPC High-Performance Computing. 1–8, 11–16, 18–24, 26, 37, 38, 40, 46, 49, 73–75, 77–79, 83, 84, 90, 92, 94, 96, 237, 239, 257

I/O Input/Output. 44–46

ILP Instruction Level Parallelism. 40

IPC Inter-Process Communication. 46

iQFT Inverse Quantum Fourier Transform. 4, 7, 8, 14, 17, 18, 22, 25, 74, 75, 78, 97, 99, 103, 239, 240, 258

IR Intermediate Representation. 3–5, 13–15, 21–23, 38, 74, 75, 78–80, 83, 84, 90, 91, 238, 257, 258

LLVM Low-Level Virtual Machine. 2, 4, 5, 12–15, 20–22, 78, 83, 84, 90, 238, 239

LNCS Lecture Notes in Computer Science. 27

MESI Modified, Exclusive, Shared, Invalid. 48

MPI Message Passing Interface. 2, 3, 6, 12, 13, 15, 20, 21, 23, 49, 50, 73, 74, 78, 92, 94, 96, 97, 238, 239, 263

NISQ Noisy Intermediate-Scale Quantum. 1, 11, 19, 66, 80

OS Operating System. 44–47, 255, 261

PCIe Peripheral Component Interconnect Express. 52, 256

QAOA Quantum Approximate Optimization Algorithm. 66

QC Quantum Computing. 1, 11, 19, 77–79, 83, 84, 96, 97, 257, 261

QFT Quantum Fourier Transform. 38, 65, 66, 88

QIR Quantum Intermediate Representation. 2, 4, 5, 12, 14, 15, 20, 22, 38, 78, 84, 90, 238

QPE Quantum Phase Estimation. 7, 17, 25, 78, 97, 239, 240

QPU Quantum Processing Unit. 1, 2, 4, 7, 11, 12, 14, 17, 19–21, 24, 38, 53, 66, 67, 77, 81, 83–86, 88, 97, 98, 239, 256

RAM Random Access Memory. 43, 46–48, 96

RMA Remote Memory Access. 49

RSA Rivest, Shamir y Adleman. 65

SDK Software Development Kit. 90–93, 258

SPMD Single Program Multiple Data. 3, 5, 6, 8, 13, 15, 16, 18, 21, 23, 24, 26, 49, 50, 78, 92, 237, 238, 263

SRAM Static Random Access Memory. 43

SSD Solid State Drive. 43, 46

VLIW Very Long Instruction Word. 40

VQA Variational Quantum Algorithm. 66, 67, 78, 256

VQE Variational Quantum Eigensolver. 66

Appendix A

Copyright and Permissions

This thesis is structured as a compendium of publications. The core contributions presented in Chapters 5, 6, 7, and 9 are derived from articles published in international peer-reviewed journals and conference proceedings.

All the articles included in this thesis were published under an **Open Access** model. Under this model, the authors retain the copyright of their work, granting the publisher a license to publish. Consequently, the authors have the right to reuse the content, including figures, tables, and extensive text passages, in their dissertation or thesis without requiring further explicit permission from the publisher, provided the original source is fully cited.

Below are the specific details for each publication and the publisher's policies regarding thesis inclusion.

A.1 Elsevier Publications

The following articles were published in Elsevier journals:

1. **Future Generation Computer Systems**
   Article: NetQIR: a extension of QIR for Distributed Quantum Computing.
   License: Open Access (CC-BY 4.0)

2. **Computer Science Review**
   Article: Review of Distributed Quantum Computing: from QPU to High Performance Quantum Computing.
   License: Open Access (CC-BY 4.0)

Permissions Policy:
Elsevier explicitly states that authors can include their articles in full in a thesis or

dissertation. As stated on their "Author Rights" page:

> *"Authors can include their articles in full or in part in a thesis or dissertation for non-commercial purposes."*

Reference Link:
Elsevier Copyright and Permissions Policy.

A.2 Springer Nature Publications

The following articles were published in Springer journals and proceedings:

1. **Lecture Notes in Computer Science (LNCS)**
   Article: Quantum Compilation Process: a Survey.
   License: Open Access (CC-BY 4.0)

2. **The Journal of Supercomputing**
   Article: Review of intermediate representations for quantum computing.
   License: Open Access (CC-BY 4.0)

Permissions Policy:
Articles published under the Creative Commons Attribution 4.0 International License (CC-BY 4.0) allow authors (and third parties) to copy, distribute, and transmit the work, as well as to adapt the work, provided the original author and source are attributed. Springer Nature confirms that authors retain the copyright to their work for Open Access articles.

Reference Link:
Rights and Permissions of Lecture Notes and Computer Systems and Rights and Permissions of The Journal of Supercomputing.

A.3 IEEE Publications

The following article was published in the proceedings of an IEEE international conference:

1. **2025 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)**
   Article: SYCL QPU: an LLVM-based QPU simulation framework built using DPC++.

Permissions Policy:
The IEEE does not require individuals working on a thesis to obtain a formal reuse license, as stated in `https://magazines.ieeeauthorcenter.ieee.org/choose-a-publishing-agreement/avoid-infringement-upon-ieee-copyright/`.

> *In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of the University of Santiago de Compostela's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to* `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` *to learn how to obtain a License from RightsLink.*

A.4 Figures

All figures, diagrams, and schematics contained in this dissertation are original works created by the author. Figures reproduced or adapted from external sources are explicitly cited and accompanied by the necessary copyright permissions in their respective captions.

This thesis explores the integration of distributed quantum computing into high-performance computing (HPC) environments, addressing the current challenges and opportunities offered by this emerging paradigm. Despite recent advancements in quantum computing, there are still significant limitations, such as high levels of error and noise in existing systems, as well as limited scalability in the number of qubits per chip, the fundamental unit of quantum information analogous to the classical bit.

Furthermore, due to the probabilistic nature of quantum computing, there are many areas where classical systems outperform quantum ones, particularly in terms of stability and precision. Therefore, just as GPUs and FPGAs have been integrated as heterogeneous devices in HPC systems, this thesis explores the use of quantum processing units (QPUs) as another specialized resource within these environments. The technical implications of this integration are analyzed, and software solutions are developed to incorporate QPUs into the most widely used frameworks in the field.

Potential applications of this integration include molecular simulation, advanced optimization, quantum cryptography, and machine learning, where quantum computing can offer significant improvements when combined efficiently and complementarily with classical resources.