Alineamiento de imágenes multiespectrales de teledetección en un clúster de GPUs

Álvaro Ordóñez[1,2], Dora B. Heras[1,2] y Francisco Argüello[2]

Resumen— **El registro o alineamiento de imágenes de teledetección multiespectrales o hiperespectrales es una tarea fundamental tras la captura de dichas imágenes, ya que permite alinear las diferentes bandas de la imagen o diferentes imágenes entre sí para construir escenas de mayor tamaño o para construir imágenes de mayor resolución. En los casos en los que es necesario registrar las bandas de una o varias imágenes y posteriormente diferentes imágenes, el tiempo de ejecución es muy alto debido no solo a la complejidad del proceso de registrado sino también a que es necesario ejecutar muchas veces dicho proceso. En este artículo se presenta una implementación multinodo-multiGPU para el registro de bandas e imágenes multiespectrales utilizando el algoritmo HSI-KAZE. Los distintos conjuntos de datos son distribuidos entre los nodos disponibles de un clúster de GPUs usando MPI y a su vez, cada nodo, distribuye las bandas e imágenes de un mismo conjunto entre las distintas GPUs del nodo utilizando OpenMP. Las GPUs son programadas mediante CUDA.**

Palabras clave— **Multiespectral, registro de imágenes, clúster, MPI, GPU.**

I. Introducción

El gran avance en el desarrollo de sensores permite que hoy podamos obtener imágenes multiespectrales de teledetección a menores costes que antaño. La característica más significativa de estas imágenes es el rango de longitud de onda que abarcan [1]. Las imágenes multiespectrales están formadas por 10 o menos canales espectrales continuos, también llamados bandas. Esto significa que cada píxel es un vector donde cada componente se corresponde con una determinada longitud de onda [2]. Gracias a la alta resolución espectral de estas imágenes podemos distinguir con mayor precisión elementos que no sería posible identificar mediante imágenes RGB.

El registro o alineamiento de imágenes es una tarea que forma parte de la etapa de preprocesado. Este proceso consiste en alinear imágenes de la misma zona o área pero que fueron obtenidas en distintos momentos temporales, desde diferentes perspectivas y/o bajo diferentes condiciones lumínicas. Si restringimos el problema al registro de dos imágenes, lo que se pretende es registrar o alinear una de ellas, llamada imagen objetivo, con respeto de la otra, llamada imagen de referencia. Esto exige localizar y emparejar en ambas imágenes los mismos objetos, estructuras, regiones, etc. Se trata por lo tanto de un proceso clave antes de realizar aplicaciones basadas en el estudio o análisis de como ha afectado el paso del tiempo a una misma región. Hablamos, por ejemplo, de detección automática de cambios [3], monitorización medioambiental [4] o detección de anomalías, entre otros.

Los algoritmos de registro pueden categorizarse en dos tipos: basados en área y basados en la detección de características [5]. Los métodos basados en área explotan directamente los datos de las imágenes sin ningún análisis estructural, es decir, no buscan correspondencias entre las imágenes buscando puntos o regiones comunes [6]. Destacan entre ellos los métodos que usan la transformada rápida de Fourier para computar la correlación entre las imágenes.

Los métodos basados en características se basan en buscar características distintivas o de interés en las imágenes a registrar. Estas características pueden ser puntos, contornos, regiones o líneas que cumplen con una serie de requisitos: ser invariantes a transformaciones geométricas, disponer de información suficiente para ser distinguidas de otras características de la imagen y ser insensible a pequeñas variaciones en los píxeles. Conociendo un cierto número de características comunes de ambas imágenes, se puede calcular la transformación geométrica que registre una imagen con respeto a otra. Esta búsqueda de características comunes a alto nivel hace que este tipo de métodos se vean menos afectados por cambios de iluminación, perspectiva o intensidad, o por la presencia de ruido, en comparación con los basados en área [7], pero al mismo tiempo hace que sean computacionalmente más costosos (construcción de espacios-escala de ambas imágenes, búsqueda de características en ellos, cómputo de los descriptores, entre otros). Si a esto se le añade el coste computacional de procesar la gran cantidad de información espectral disponible para este tipo de imágenes, el tiempo de cómputo necesario aumenta considerablemente. Es por eso, que en la literatura se han propuesto implementaciones en GPU para acelerar el registro de imágenes multidimensionales [8], [9], [10].

En cuanto a recursos computacionales, los requerimientos aumentan si, como en el caso tratado en este artículo, se requiere realizar un gran número de registros. El problema que se aborda en este artículo consiste en la aceleración del proceso de registro de diferentes conjuntos de imágenes multiespectrales de 5 bandas tomadas con un dron. Primero hay que registrar las bandas de cada imagen entre sí y luego, una vez obtengamos las imágenes con las bandas registradas, hay que registrar las diferentes imágenes pertenecientes al mismo conjunto para así obtener la escena final conformada por todas ellas.

El contexto en el que tiene que ser realizado el re-

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, e-mail: {alvaro.ordonez,dora.blanco}@usc.gal.

[2]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, e-mail: francisco.arguello@usc.gal.

gistro condiciona el tipo de plataforma computacional que podemos usar para acelerar los algoritmos de procesamiento [11]. La computación eficiente de este problema ya fue abordada en [12], donde se presentó una implementación que usaba dos GPUs de un mismo nodo (memoria compartida) para registrar dos conjuntos de imágenes diferentes en paralelo. Además, en ese trabajo se llevó a cabo una comparativa para ver qué tipo de método era el más adecuado para resolver el problema en términos de precisión de registro, si uno basado en área o en características. Dependiendo de qué cambios se suelan producir en cada contexto entre las diferentes imágenes, será un método u otro el que ofrezca mejores precisiones de registrado [13].

En este artículo, se trabaja con imágenes que son enviadas desde el dron a tierra sin haber sido procesadas y se propone una nueva implementación para realizar ya en tierra un registro de bandas de cada imagen seguido por uno de imágenes en un clúster que dispone de GPUs en sus nodos. Los conjuntos de datos son distribuidos cíclicamente entre los nodos del clúster usando MPI. Luego, dentro de cada nodo, el registro de bandas y el de las diferentes imágenes multiespectrales son llevados a cabo en paralelo, ya que esta carga de trabajo es distribuida entre las diferentes GPUs disponibles en cada nodo usando OpenMP y CUDA. De esta manera, se explotan tres niveles de paralelismo: nodos (MPI), GPUs (OpenMP) y *Single instruction, multiple threads* (SIMT), es decir, la que nos brinda la propia arquitectura de la GPU.

II. Alineamiento de bandas e imágenes multiespectrales usando tecnologías HPC

En esta sección, se describe el algoritmo de registro utilizado y su primer implementación en una máquina con dos GPUs. A continuación, se detalla la implementación multinodo-multiGPU llevada a cabo para el registro de bandas e imágenes de distintos conjuntos de datos usando MPI, OpenMP y CUDA.

A. Implementación multiGPU

En [12] se presentó una primera implementación para el uso de varias GPUs de una misma máquina para un registro de dos niveles de imágenes multiespectrales de 5 bandas. En esa propuesta se registraban dos conjuntos de imágenes de dos áreas geográficas diferentes en paralelo utilizando las dos GPUs disponibles en el equipo utilizado. La figura 1 muestra el esquema propuesto para una de las GPUs. Cada GPU procesa los dos niveles de registro de un conjunto distinto de manera secuencial. En el ejemplo de la figura 1, primero se registran las bandas de la imagen de referencia, luego las bandas de la imagen objetivo y, finalmente, se registran ambas imágenes para obtener la escena.

Otro objetivo de ese trabajo era comparar dos métodos de registro de imágenes, uno basado en área, Hyperspectral Fourier Mellin (HYFM) [14], y otro basado en detección de características, Hyperspectral KAZE (HSI-KAZE) [15]. Se constató que HYFM era incapaz de completar el segundo nivel de registro en muchas imágenes debido a los cambios de perspectiva que presentan las imágenes tomadas por dron que se querían registrar. Se concluyó, por lo tanto, que HSI-KAZE era el método más idóneo para ese caso debido a su mayor tolerancia a cambios de perspectiva.

HSI-KAZE [15] es un método basado en detección de características especialmente diseñado para el registro de imágenes hiperespectrales de teledetección. Este explota la información espectral contenida en estas imágenes para obtener mejores alineamientos en cuanto a escala y ángulo. El método incluye en una primera etapa una selección de las bandas que contienen la información más relevante y busca en estas puntos de interés comunes en ambas imágenes. Para la detección de los puntos se construye una escala-espacio aplicando un suavizado controlado mediante el uso de una difusión no lineal. Como descriptor de puntos de interés para el posterior emparejado, el método propone un descriptor formado por dos partes: una espacial y una espectral. Para la parte espacial se emplea el descriptor modified-SURF (M-SURF) mientras que para la parte espectral se emplea la firma espectral asociada a ese punto de interés. Finalmente, para estimar la transformación geométrica a partir de los puntos de interés previamente emparejados, HSI-KAZE realiza una búsqueda y selección exhaustiva de las mejores parejas en cuanto a precisión de registro. Estas características hacen de HSI-KAZE un método muy costoso en tiempos de computación por lo que fue necesario proyectarlo para una completa ejecución en GPU consiguiendo aceleraciones de hasta 13× en comparación con una implementación en CPU con OpenMP [10].

Sin embargo, en esa primera propuesta multiGPU [10] se observó que el paralelismo a nivel de conjunto de datos estaba limitado ya que cada GPU procesaba las distintas imágenes de cada conjunto de manera secuencial. El siguiente paso natural era trasladar la implementación a un clúster de varios nodos.

B. Implementación propuesta usando un clúster de GPUs

En esta sección se presenta la implementación propuesta para realizar el registro de dos niveles de diferentes conjuntos de imágenes multiespectrales en un clúster de GPUs.

La arquitectura usada es un clúster homogéneo donde cada nodo dispone de varias GPUs. Inicialmente, los diferentes conjuntos de datos (formados por imágenes multiespectrales de 5 bandas) se encuentran almacenados en el disco duro del nodo máster. El nodo máster lee todos los conjuntos de datos, los almacena como un array de enteros en memoria y los distribuye cíclicamente a los distintos nodos del clúster, incluido el mismo, usando `MPI_Send`.

En la figura 2 se muestra el registro de dos niveles llevado a cabo por cada nodo para un conjun-

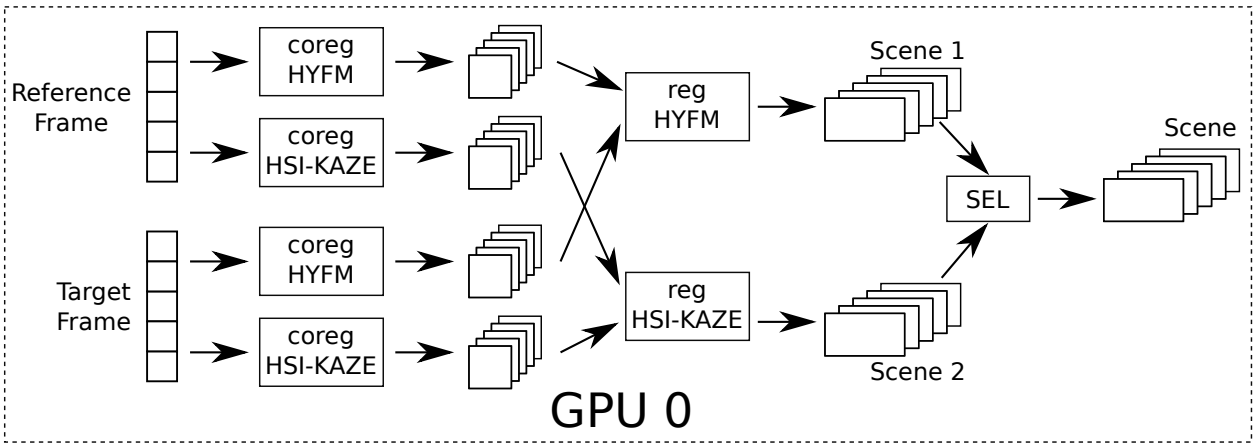


Fig. 1: Esquema de la primera implementación multiGPU. Ejemplo para uno de las GPUs (GPU 0). Figura de [12].

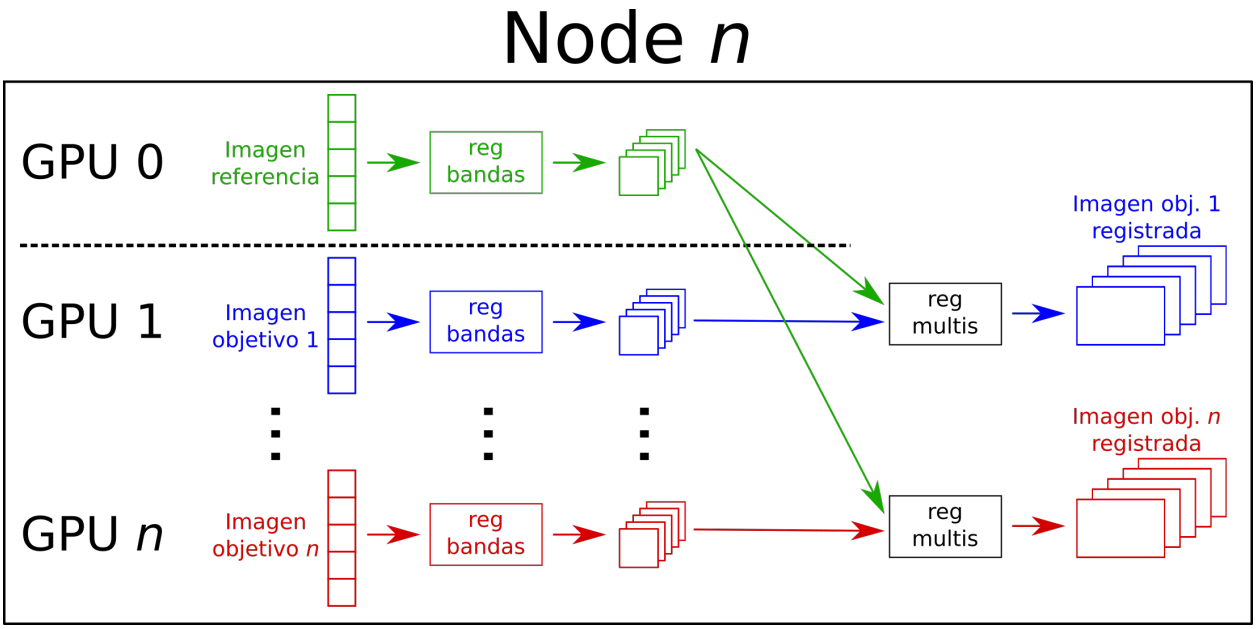


Fig. 2: Registro de dos niveles de imágenes multiespectrales de 5 bandas en un nodo del clúster. En un primer nivel, se registran las bandas de cada imagen. En un segundo nivel, se registran las diferentes imágenes para construir la escena.

to de imágenes de la misma localización geográfica. Dentro de cada nodo, se crean distintos hilos usando OpenMP para gestionar las distintas GPUs que estos contienen. Recibido un conjunto de datos, que consta de varias imágenes de 5 bandas, el nodo envía una imagen multiespectral de 5 bandas a cada GPU, es decir, copia cada imagen desde la memoria principal a la memoria de la GPU correspondiente. Cada GPU empieza registrando las bandas de una imagen multiespectral diferente ("reg bandas" en la figura 2).

Cuando las bandas de todas las imágenes están alineadas, se pasa al segundo nivel de registro en el que cada GPU registrará una imagen multiespectral con respecto a la imagen de referencia, típicamente la primera del conjunto de datos. En la figura 2, la imagen de referencia es la imagen cuyas bandas son registradas en la GPU 0 (marcada en color verde). Para que el resto de GPUs puedan realizar el registro de su imagen objetivo con respecto a la imagen de referencia es necesario realizar una copia de la imagen de referencia almacenada en la memoria de la GPU 0 a la memoria del resto de GPUs. Una vez hecho esto, el registro de cada par de imágenes, el segundo nivel de registro, puede comenzar ("reg multis" en la figura 2).

Una vez finalizado el segundo nivel, las imágenes objetivo ya registradas son copiadas de las memorias de las diferentes GPUs a la memoria principal. El hilo principal empaqueta los resultados y los envía usando MPI al nodo máster.

III. Resultados

En esta sección se detalla el hardware utilizado así como los compiladores, bibliotecas y opciones usadas para obtener el ejecutable de la implementación. También se describe el conjunto de imágenes utilizado para realizar los experimentos. Finalmente, se presentan los tiempos de ejecución para la implementación propuesta y se compara esta con una implementación secuencial multiGPU y con una implementación multinodo-multiCPU.

A. Hardware, software y conjunto de datos

Los experimentos fueron llevado a cabo en tres nodos del supercomputador FinisTerrae-II del Centro de Supercomputación de Galicia (CESGA) [16]. Esta es la mayor cantidad de nodos con GPUs que se pueden solicitar concurrentemente en el FinisTerrae-II. Cada nodo está formado por dos procesadores Haswell 2680v3 con 24 núcleos, dos aceleradores NVIDIA Tesla K80 y 128 GB de memoria principal. Los nodos están interconectados por una red Mellanox Infiniband FDR@56Gbps con topología *fat-tree*.

Cada acelerador NVIDIA Tesla K80 está formado por dos GPUs Tesla K80 conectadas directamente en la propia tarjeta [17]. Esto hace que dispongamos de cuatro GPUs Tesla K80 en cada uno de los nodos. Las dos GPUs pertenecientes al mismo acelerador tienen la ventaja en que se pueden realizar copias *peer-to-peer* (P2P) entre la memoria de ambas. Esto significa que la transferencia de datos se realiza directamente a través del bus PCIe. En el caso en que sea necesario hacer movimiento de datos entre GPUs de distintos aceleradores, las copias P2P non son posibles y es necesario emplear un espacio de almacenamiento intermedio en memoria principal. Este caso sucede, por ejemplo, cuando la imagen de referencia se copia de la GPU-0 a la GPU-2. El tipo de copia no es un problema del que el programador se deba preocupar ya que la API de NVIDIA escoge el tipo de copia dependiendo del tipo de conexión disponible entre GPUs. La implementación propuesta ha sido diseñada para que se produzca el menor número de movimientos de datos entre GPUs, y entre memoria principal y GPU con el fin de reducir el tiempo de ejecución.

Las implementaciones multiGPU y multinodo-multiGPU fueron compiladas usando `nvcc` version 11.2.152 con la opción `-O3` y las bibliotecas CUB versión 1.8.0 y OpenMPI versión 11.2.152. CUB es utilizada en el algoritmo HSI-KAZE para computar reducciones paralelas e histogramas en GPU. La versión multinodo-multiCPU fue compilada usando `g++` versión 10.1.0 con la opción `-O3` y OpenMPI versión 11.2.15. Todos los programas usan precisión simple. Los tiempos de ejecución y aceleraciones mostrados en este trabajo se corresponden con la media de cinco ejecuciones independientes.

Para analizar la eficiencia de nuestra propuesta se han empleado tres conjuntos de datos con diferente número de imágenes:

- **Embalse**: 3 imágenes multiespectrales.
- **Parasol**: 4 imágenes multiespectrales.
- **Casa**: 5 imágenes multiespectrales.

Cada conjunto está formado por imágenes de una misma zona geográfica (entorno de la cuenca del río Oitavén, Galicia) tomadas desde localizaciones diferentes tal y como se puede ver en la figura 3. Las imágenes tienen 5 bandas espectrales que van desde los 475 nm a los 842 nm y una resolución espacial de 960×1280 píxeles. Han sido capturadas con el sensor MicaSense RedEdge MX instalado en un dron [1]. Aunque el sensor captura todas las bandas al mismo tiempo es necesario llevar a cabo un registro de las mismas. La utilidad final de estas imágenes es monitorizar la vegetación de ribera de la cuenca del río Oitavén por lo que obtener alineamientos con buenas precisiones es indispensable.

B. Comparación en términos de tiempos de ejecución

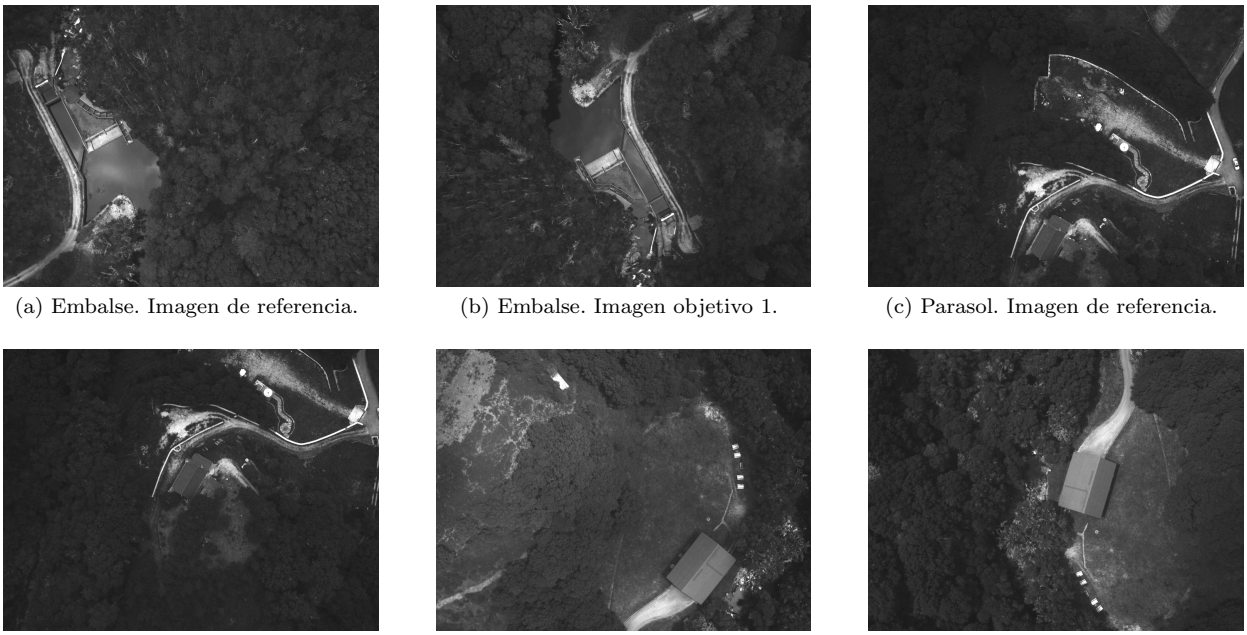
En esta sección se presentan los tiempos de ejecución para la implementación propuesta así como una comparativa con versiones ya desarrolladas anteriormente.

Tabla I: Tiempos de ejecución (en segundos) para cada etapa de la implementación multiGPU-multinodo. Se muestran los tiempos requeridos por las diferentes GPUs del nodo más lento.

Etapa	**Tiempo de ejecución (s)**
Overheads	12,40
Inicialización	1,03
Distribución conjuntos (MPI)	0,18
GPU 0 – Reg. Bandas	733,25
GPU 0 – Reg. Multis	76,04
GPU 1 – Reg. Bandas	370,98
GPU 1 – Reg. Multis	60,19
GPU 2 – Reg. Bandas	433,67
GPU 2 – Reg. Multis	87,38
GPU 3 – Reg. Bandas	378,12
GPU 3 – Reg. Multis	59,10
Recibir resultados (MPI)	0,15
Escritura a disco	2,69
Total	**825,75**

La tabla I presenta los tiempos de ejecución detallados por etapa para la implementación multiGPU-multinodo propuesta. Cada nodo registra un conjunto de datos diferente, en concreto, el nodo máster registra *Embalse*, el nodo esclavo 1 *Parasol* y el nodo esclavo 2 *Casa*. Debido a que cada conjunto está formado por un número diferente de imágenes, cada nodo soportará una carga de trabajo diferente. En la tabla I se presenta el tiempo de cada etapa perteneciente al nodo más lento. En el caso de las etapas de registros de bandas e imágenes, el nodo esclavo 2 es el que marca los mayores tiempos ya que tiene que registrar un mayor número de imágenes (5 imágenes). En una primera iteración, el nodo 2 asigna una imagen a cada uno de sus cuatros GPUs para registrar las bandas. Esto hace que hasta la segunda iteración no se complete el registro de bandas. Los otros nodos, cuándo el nodo esclavo 2 aún se encuentra registrando las bandas de la quinta imagen, ya se encuentran ejecutando el segundo nivel de registro, es decir, registrando las imágenes.

[1]Estas imágenes han sido obtenidas en colaboración con la compañía Babcock a través del programa Civil UAVs Initiative de la Xunta de Galicia.



(a) Embalse. Imagen de referencia. (b) Embalse. Imagen objetivo 1. (c) Parasol. Imagen de referencia.

(d) Parasol. Imagen objetivo 1. (e) Casa. Imagen de referencia. (f) Casa. Imagen objetivo 1.

Fig. 3: Primera banda de las imágenes multiespectrales de referencia y de la primera imagen objetivo a registrar.

La etapa *overheads* incluye los tiempos necesarios para realizar las mediciones mostradas así como la creación de los diferentes threads de OpenMP utilizados en la gestión de las diferentes GPUs de cada nodo. En términos de *overheads* es el nodo máster el que presenta mayores tiempos ya que es el encargado de la distribución y recepción de datos.

Es necesario destacar que los tiempos de distribución de datos usando la red y MPI no supera los 0,33 segundos si sumamos los tiempos de envío y recepción. También, que los tiempos asociados con transferencia de datos entre GPUs representa solo 0,01 segundos en el peor de los casos. Existen diferencias entre la copia P2P entre GPUs del mismo acelerador y la copia no P2P entre GPUs de distinto acelerador pero estas son despreciables en este caso particular donde la cantidad de datos a mover (la imagen de referencia con las bandas registradas) no es elevada. Por lo tanto, el tiempo necesario para registrar los tres conjuntos de datos es de un total de 825,75 segundos.

Como se explicó en la sección II-A, en [12] se propuso una primera implementación multiGPU para registrar las bandas e imágenes en las GPUs disponibles de un único nodo. Es decir, cada GPU recibe un conjunto de datos diferente y realiza el registro de las bandas e imágenes de un mismo conjunto, primero registra las bandas de todas las imágenes, y luego las propias imágenes.

En la tabla II se presentan los tiempos de la versión secuencial multiGPU en comparación con los tiempos de la versión multinodo-multiGPU propuesta en este trabajo. Los tiempos de la versión secuencial [12] fueron obtenidos usando un solo nodo del clúster descrito, es decir, solo tres GPUs fueron usadas, una por cada conjunto de datos. Como se puede ver en la tabla II, en la implementación secuencial, el mayor tiempo de ejecución es obtenido por la GPU que registra el conjunto de datos *Casa* ya que es el que está formado por un mayor número de imágenes. Por lo tanto, el tiempo de esa GPU, 1.910,84 segundos, es el que se corresponde con el tiempo total que necesita la implementación secuencial para registrar los tres conjuntos de datos. Indicar que en esa implementación, al igual que en la multinodo, la gestión de las diferentes GPUs se realiza con hilos OpenMP y que los tiempos necesarios para la gestión de esos hilos están incluidos en la tabla. Por otra parte, la implementación multinodo-multiGPU propuesta consigue realizar el mismo trabajo en 825,75 segundos. Esto significa que se ha conseguido una aceleración de 2,31× gracias a la distribución de los datos entre nodos y GPUs. Esta mayor atomicidad en el reparto de la carga de trabajo nos permite obtener esa mayor aceleración.

Tabla II: Tiempos de ejecución (en segundos) y aceleraciones de la implementación multiGPU-multinodo propuesta en comparación con la implementación multiGPU anteriormente propuesta por los autores en [12].

	multiGPU[12]	**Multinodo-multiGPU**	**Aceleración**
Embalse	1.094,48	825,75	1,33×
Parasol	1.578,33	514,94	3,07×
Casa	1.910,84	823,06	2,32×
Total	**1.910,84**	**825,75**	**2,31**×

En la tabla III se comparan los tiempos de ejecución de la implementación multinodo-multiGPU con una implementación multinodo-multiCPU. En la versión multinodo-multiCPU se ha seguido la misma distribución y paralelización que en la multinodo-multiGPU. En el caso de la versión multinodo-multiCPU, en vez de las 4 GPUs, se han utilizado los 24 núcleos disponibles por nodo (12 por cada una de

Tabla III: Comparación en tiempos de ejecución (en segundos) entre la implementación multinodo-multiCPU y la implementación multinodo-multiGPU.

Implementación	**Tiempo de ejecución (s)**
Multinodo-multiCPU	3.080,24
Multinodo-multiGPU	825,75
Aceleración	**3,73**×

las CPUs) para ejecutar una versión OpenMP del algoritmo HSI-KAZE [10]. En la versión multiGPU se lanzaban 4 hilos para realizar registros en paralelo en las 4 GPUs. En la versión multiCPU se lanzan hasta 12 hilos por cada imagen. Estos hilos son utilizados para paralelizar diferentes etapas del algoritmo HSI-KAZE a bajo nivel. En total, se ejecutan 48 hilos en cada nodo gracias a la tecnología *Hyper-Threading* que nos permite ejecutar 2 hilos por núcleo. La versión multiCPU registra los tres conjuntos de datos en 3.080,24 segundos, un tiempo 3,73× superior a lo que necesita la versión multiGPU.

IV. Conclusiones

En este artículo se presenta una implementación para el registro de bandas e imágenes multiespectrales en un clúster de GPUs utilizando el algoritmo HSI-KAZE. El esquema seguido distribuye cíclicamente los conjuntos de datos reales entre los nodos disponibles usando MPI. Cada conjunto de datos está formado por varias imágenes multiespectrales de 5 bandas. Cada nodo distribuye las imágenes de su conjunto de datos entre las GPUs disponibles con la ayuda de OpenMP. Cada GPU registra en paralelo, primero, las bandas de la imagen, y luego las diferentes imágenes para construir la escena final.

La implementación propuesta ha sido probada utilizando tres conjuntos de datos de tres, cuatro y cinco imágenes multiespectrales capturadas por un dron, respectivamente. El número de imágenes de cada conjunto de datos varía según el conjunto ya que se trata de conjuntos de datos reales. Dicha implementación ha sido comparada con una implementación anterior secuencial multiGPU y con una versión multinodo-multiCPU, obteniendo aceleraciones 2,31× y 3,73× superiores, respectivamente.

En cuanto a trabajo futuro, sería interesante evaluar el uso de la biblioteca NVIDIA Collective Communication Library (NCCL), especialmente optimizada por NVIDIA para realizar implementaciones multinodo-multiGPU, como una alternativa a MPI. Otras implementaciones serán necesarias dependiendo de cual sea la plataforma computacional más adecuada para los requerimientos de tiempo de ejecución y de precisión del registro en cuanto a la aplicación de teledetección en la que se trabaje.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia e Innovación, Gobierno de España [proyecto PID2019-104834GB-I00] y la Consellería de Cultura, Educación e Universidade, Xunta de Galicia [proyectos ED431C 2018/19 y 2019-2022 ED431G-2019/04]. También hemos recibido financiación de la Junta de Castilla y León (proyecto PROPHET-2, VA226P20). Todos ellos cofinanciados por el Fondo Europeo de Desarrollo Regional (FEDER).

Referencias

[1] Chein-I Chang, *Hyperspectral imaging: techniques for spectral detection and classification*, vol. 1, Springer Science & Business Media, 2003.

[2] Pedram Ghamisi, Javier Plaza, Yushi Chen, Jun Li, and Antonio J Plaza, "Advanced spectral classifiers for hyperspectral images: A review," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 1, pp. 8–32, 2017.

[3] Javier López-Fandiño, Alberto S Garea, Dora B. Heras, and Francisco Argüello, "Stacked autoencoders for multiclass change detection in hyperspectral images," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2018, pp. 1906–1909.

[4] Hernan Flores, Sandra Lorenz, Robert Jackisch, Laura Tusa, I. Cecilia Contreras, Robert Zimmermann, and Richard Gloaguen, "UAS-Based Hyperspectral Environmental Monitoring of Acid Mine Drainage Affected Waters," *Minerals*, vol. 11, no. 2, 2021.

[5] Barbara Zitova and Jan Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[6] Jacqueline Le Moigne, Nathan S Netanyahu, and Roger D Eastman, *Image registration for remote sensing*, Cambridge University Press, 2011.

[7] A. Ardeshir Goshtasby, *Image registration: Principles, tools and methods*, Springer Science & Business Media, 2012.

[8] Yunsheng Zhang, Peilong Zhou, Yue Ren, and Zhengrong Zou, "GPU-accelerated large-size VHR images registration via coarse-to-fine matching," *Computers & Geosciences*, vol. 66, pp. 54–65, 2014.

[9] Sudhakar Sah, Jan Vanek, YoungJun Roh, and Ratul Wasnik, "GPU accelerated real time rotation, scale and translation invariant image registration method," in *Image Analysis and Recognition*, pp. 224–233. Springer, 2012.

[10] Álvaro Ordóñez, Francisco Argüello, Dora B Heras, and Begüm Demir, "GPU-accelerated registration of hyperspectral images using KAZE features," *The Journal of Supercomputing*, pp. 1–15, 2020.

[11] Gabriele Cavallaro, Dora B Heras, Zebin Wu, Manil Maskey, Sebastian Lopez, Piotr Gawron, Mihai Coca, and Mihai Datcu, "High-Performance and Disruptive Computing in Remote Sensing: HDCRS-A New Working Group of the GRSS Earth Science Informatics Technical Committee," *IEEE Geoscience and Remote Sensing Magazine*, 2022.

[12] Álvaro Ordóñez, Dora B. Heras, and Francisco Argüello, "Comparing Area-Based and Feature-Based Methods for Co-Registration of Multispectral Bands on GPU," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, 2021, pp. 1575–1578.

[13] Álvaro Ordóñez Iglesias, *Efficient Registration of Multi and Hyperspectral Remote Sensing Images on GPU*, Ph.D. thesis, Universidade de Santiago de Compostela, 2021.

[14] Álvaro Ordóñez, Francisco Argüello, and Dora B. Heras, "Fourier–mellin registration of two hyperspectral images," *International Journal of Remote Sensing*, vol. 38, no. 11, pp. 3253–3273, 2017.

[15] Álvaro Ordóñez, Francisco Argüello, and Dora B. Heras, "Alignment of hyperspectral images using KAZE features," *Remote Sensing*, vol. 10, no. 5, 2018.

[16] CESGA, "FinisTerrae-II supercomputer," `https://www.cesga.es/infraestructuras/computacion/`, [En línea, accedido: 03-Jun-2022].

[17] NVIDIA, "Board Specification of the Tesla K80 GPU Accelerator," `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/Tesla-K80-BoardSpec-07317-001-v05.pdf`, [En línea, accedido: 03-Jun-2022].