UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica y Computación

Centro de Investigación en Tecnoloxías da Información (CiTIUS)

Tesis Doctoral

Spectral-spatial classification of n-dimensional images in real-time based on segmentation and mathematical morphology on GPUs

Presentada por:

Pablo Quesada Barriuso

Dirigida por:

Dra. Dora Blanco Heras

Dr. Francisco Argüello Pedreira

Julio de 2015

Dra. Dora Blanco Heras, Profesora Titular de Universidad del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

Dr. Francisco Argüello Pedreira, Profesor Titular de Universidad del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **Spectral-spatial classification of *n*-dimensional images in real-time based on segmentation and mathematical morphology on GPUs** ha sido realizada por **D. Pablo Quesada Barriuso** bajo nuestra dirección en el Departamento de Electrónica y Computación y en el Centro Singular de Investigación en Tecnoloxías da Información de la Universidad de Santiago de Compostela, y constituye la Tesis que presenta para obtar al título de Doctor.

Y autorizan la presentación de la tesis indicada, considerando que reune los requisitos exigidos en el artículo 34 de la regulación de Estudios de Doctorado, y que como directores de la misma no incurren en las causas de abstención establecidas en la ley 30/1992.

Santiago de Compostela, Julio de 2015

Dora Blanco Heras
Directora de la tesis

Francisco Argüello Pedreira
Director de la tesis

To I. L. V.
My star, my perfect silence.

"If you spend too much time thinking about a thing, you'll never get it done. Make at least one definitive move daily toward your goal."

Bruce Lee.

"With great power there must also come – great responsibility!."

Amazing Fantasy #15 – The first Spider-Man story.

Acknowledgments

I would like to express my gratitude to all the people who have supported me during my thesis. Special thanks goes to my thesis advisors, Dora Blanco Heras and Francisco Argüello Pedreira, for their guidance, support and patience. Their continuous confidence in my work and their motivation have been decisive in the completion of this dissertation.

I would also like to acknowledge the Department of Electronics and Computer Science, specially to the Computer Architecture Group, and to the Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS) at the University of Santiago de Compostela, for providing the necessary resources and technical support required by this project.

My gratitude also goes to Prof. Jon Atli Benediktsson, University of Iceland, and Prof. Lorenzo Bruzzone, University of Trento, for their advise and kind support during my stage in their research groups.

My deepest thanks to my family and close friends for their never ending patience and encouragement during hard moments through all these years.

Finally, I am also thankful to the following institutions for the funding provided for this work: Ministry of Science and Innovation, Government of Spain, cofounded by the FEDER funds of European Union, under contract TIN 2010-1754, and by Xunta de Galicia under contracts 08TIC001206PR and 2010/28.

Julio de 2015

Contents

List of Acronyms

AA Average Accuracy

AWGN additive white Gaussian noise

AP Attribute Profile

API Application Program Interface

AVIRIS Airborne Visible-infrared Imaging Spectrometer

BA Block–Asynchronous

CA Cellular Automata

CDF97 Cohen-Daubechies-Feauveau $9/7$ wavelet

CC compute capability

CCL Connected Component Labelling

CMG Color Morphological Gradient

CS class-specific accuracy

CUBLAS CUDA Basic Linear Algebra Subroutines

CUDA Compute Unified Device Architecture

DAFE Discriminant Analysis Feature Extraction

DBFE Decision Boundary Feature Extraction

DWT discrete wavelet transform

EAP Extended Attribute Profile

EM Expectation Maximization

EMAP Extended Multi-Attribute Profile

EMP Extended Morphological Profile

FE Feature Extraction

FPGA Field Programmable Gate Array

FS Feature Selection

GPU Graphics Processing Unit

HPC High Performance Computing

HSEG Hierarchical Image Segmentation

ICA Independent Component Analysis

MM Mathematical Morphology

MNF Minimum Noise Fraction

MP Morphological Profile

MV Majority Vote

***n*D** *n*-dimensional

NPP NVIDIA Performance Primitives

NWFE Non-parametric Weighted Feature Extraction

OA Overall Accuracy

OAO One-Against-One

PC Principal Component

PCA Principal Component Analysis

PR post-regularization

RBF Radial Basis Function

RCMG Robust Color Morphological Gradient

ROSIS-03 Reflective Optics System Imaging Spectrometer

SE structuring element

SM Streaming Multiprocessor

SP Scalar Processor

SVM Support Vector Machine

WT Wavelet Transform

Resumen

El propósito de esta tesis es desarrollar esquemas eficientes para clasificar imágenes *n*-dimensionales usando técnicas de segmentación y morfología matemática (MM), y máquinas de soporte vectorial (SVM) como clasificadores. Por esquemas eficientes entendemos aquellos que producen buenos resultados en términos de precisión así como los que se pueden ejecutar en tiempo real en arquitecturas de bajo coste. En la búsqueda de estos esquemas establecemos pues un doble objetivo; uno, en referencia a la precisión y, otro, al tiempo de ejecución. El primer de ellos se logra mediante el diseño de nuevos esquemas, mientras que, el segundo, se consigue mediante el desarrollo de técnicas que permiten su ejecución eficiente en hardware disponible en ordenadores personales, como las CPUs multi-hilo y las unidades de procesador gráfico (GPU, *Graphics Processing Unit*, en inglés). Las imágenes *n*-dimensionales engloban tanto a las imágenes de dos y tres dimensiones, ejemplo de ello son las utilizadas en el ámbito médico, como también a las imágenes desde diez hasta cientos de dimensiones, como las imágenes multi- e hiperespectrales adquiridas en teledetección. En el análisis de imágenes multi- e hiperespectrales, un pixel se representa como un vector de valores espectrales o características al que llamamos *pixel vector*. Las imágenes hiperespectrales se adquieren mediante sensores ópticos que capturan diferentes longitudes de onda a la vez, desde el espectro visible hasta cerca del infrarrojo. Esta colección de datos se puede ver como un cubo hiperespectral formado por varias bandas espectrales. [66].

El primer sensor multiespectral a bordo de un satélite, el Landsat-1 en 1972, era capaz de recoger 4 bandas espectrales con una resolución espacial de 80 metros por píxel [95]. Los *cientos de bandas* hiperespectrales llegaron en noviembre del año 2000 con el espectrómetro Hyperion [123]. Desde entonces, la teledetección ha sido un área de investigación muy activa para el mapeo de minerales [92], la identificación de zonas urbanas; por ejemplo, para la detección de cambios urbanísticos [107], y para el análisis en la degradación de los bosques,

entre otras [66, 128]. En este trabajo se han utilizado imágenes hiperespectrales capturadas por el Airborne Visible-infrared Imaging Spectrometer (AVIRIS) [71] y el Reflective Optics System Imaging Spectrometer (ROSIS-03) [113]. Estos dos sensores hiperespectrales cubren una gama de 0,4 a 0,86 μm (ROSIS-03) y de 0,4 a 2,4 μm (AVIRIS) utilizando 115 y 224 canales espectrales, respectivamente, con una resolución espacial que varía entre 1 y 20 m / pixel.

Las características especiales de las imágenes hiperespectrales, que proporcionan información detallada para cada píxel, permiten distinguir entre materiales físicos y objetos incluso a nivel de un píxel. La alta dimensionalidad de los datos presenta nuevos desafíos en técnicas como desmezclado de información espectral (*unmixing*) [89, 23, 24], detección de objetivos y anomalías [107, 14], extracción de características [14], o caracterización y clasificación de la superficie terrestre [96, 128]. Este último ha sido un campo muy investigado en las últimas décadas. Entre los diferentes retos, esta tesis se ocupa de la clasificación de imágenes n-dimensionales.

La clasificación consiste en agrupar y etiquetar elementos que tengan en común una o más propiedades o características. El éxito de una clasificación depende de la habilidad para categorizar y/o discrimar cosas, así como en el conjunto del características usadas para tal fin; por lo tanto, usar las características más relevantes es un requisito imprescindible. En el análisis de imágenes, la clasificación es un método usado con regularidad para extraer información en medicina, vigilancia, manufacturación y en teledetección [128, 146]. Las características que intervienen en el proceso de aprendizaje para clasificar un píxel en una imagen están principalmente relacionadas con la intensidad y el color de dicho pixel, por ejemplo, los canales rojo, verde y azul en una imagen en RGB.

En las imágenes hiperespectrales tenemos muchas más características que podemos usar en este proceso; sin embargo, dicho conjunto de datos espectrales resulta en ocasiones redundante, por lo que se usan técnicas para extracción de características como el análisis en componentes principales (PCA) con el objetivo de reducir la dimensionalidad y extraer las principales características que representan a estas imágenes [146]. Hemos investigado diferentes técnicas para extraer características como análisis de componentes independientes (ICA), fracción mínima de ruido (*Minimum Noise Fraction* en inglés), DAFE (*Discriminant Analysis Feature Extraction*), DBFE (*Decision Boundary Feature Extraction*) y NWFE (*Nonparametric Weighted Feature Extraction*).

Para sacar el máximo provecho a las imágenes *n*-dimensionales, se han propuesto un gran número de métodos de clasificación [72, 55, 146], como máxima verosimilitud, redes neuronales, árboles de decisión y métodos basados en máquinas de soporte vectorial (*Support Vector Machines* (SVM) en inglés). En un estudio exhaustivo de clasificadores presentado en [60] los autores concluyeron que las SVMs estaban entre los mejores métodos. En el campo de teledetección, SVM ha demostrado que puede obtener buenos resultados en clasificación de imágenes hiperespectrales, incluso cuando el número de muestras de entrenamiento es pequeño [72, 55]; por estos motivos, hemos usado SVM como método de clasificación en este trabajo.

Independientemente de la robustez de las SVMs, estos métodos procesan cada pixel de la imagen de forma independiente considerando únicamente la información espectral; sin embargo, se ha verificado que la información espacial que se puede extraer de una imagen hiperespectral mejora la precisión de la clasificación cuando se incorpora dentro de un esquema espectral-espacial [164, 15, 54, 154, 155, 41, 19, 29, 56, 57]. Por lo tanto, los esquemas de clasificación de imágenes hiperespectrales han cambiado de clasificadores a nivel de pixel hacia esquemas de clasificación espectral-espacial. Un estudio de estos avances se recoge en [58, 24, 65]. En particular, en esta tesis estamos interesados en los esquemas que extraen la información espacial en base a técnicas de segmentación y perfiles morfológicos.

Las técnicas de segmentación han sido investigadas en teledetección para extraer información de las imágenes hiperespectrales e incorporarla en los esquemas de clasificación [117, 12, 154, 155, 157, 132, 160]. Al usar las regiones creadas mediante segmentación tenemos en cuenta las estructuras espaciales que pueden estar presentes en la imagen, por ejemplo, las técnicas de segmentación usadas en [12] están basadas en métodos de conjunto de nivel (*level-set* en inglés). Algoritmos basados en clustering (creación de particiones en la imagen agrupando píxeles similares) se han usado en [154], y técnicas basadas en evolución de autómatas celulares fueron diseñadas y aplicadas en [132] para segmentación de imágenes hiperespectrales sin reducir la dimensionalidad de los datos. La transformada watershed se ha aplicado en los esquemas espectrales-espaciales presentados en [117, 155, 156, 158, 78]. Tilton [160] propuso un algoritmo de segmentación jerárquica denomidado HSEG (hierarchical segmentation) que combina dos métodos de segmentación para unir regiones y mejorar los resultados. Entre todas las técnicas, la transformada watershed [168] ha despertado mayor interés en los esquemas de clasificación basados en la segmentación, a pesar de que no se puede aplicar directamente a una imagen *n*-dimensional. El enfoque más común para la aplicación

de esta técnica en imágenes hiperespectrales consiste en reducir el número de dimensiones espectrales a uno, por ejemplo, mediante reducción de características (PCA) o algoritmos de gradiente vectorial (RCMG) [155]. En esta tesis hemos desarrollado un esquema de clasificación basado en segmentación (CA–WSHED–MV) [137] usando autómatas celulares para calcular de forma eficiente la transformada watershed (CA–Watershed). Este algoritmo, CA–Watershed [139, 135, 138], es apropiado para arquitecturas multi-hilo como las CPUs y las unidades de procesador gráfico (GPUs), ya que el autómata se puede dividir en bloques que se actualizan de forma independiente.

La morfología matemática (MM) se define como la teoría para el análisis de estructuras espaciales [152] y se ha usado con éxito en clasificación de imágenes *n*-dimensionales en el campo de la teledetección, a través de perfiles morfológicos (Morphological Profiles (MP) en inglés) [124] y perfiles morfológicos basados en atributos (attribute profiles) [41] con los que es posible analizar diferentes tipos de estructuras. Los perfiles morfológicos eliminan objetos que no encajan dentro de la estructura de un elemento de análisis, llamado structuring element (SE). Aplicando sucesivamente operaciones morfológicas con elementos estructurales de distinto tamaño se crea una representación de la imagen con varios niveles de detalle.

El uso en imágenes *n*-dimensionales es posible aplicando los perfiles a cada banda (o a un conjunto representativo) a través de perfiles morfológicos extendidos (EMPs) [121, 15], y de perfiles basados en atributos extendidos (EAPs) [41]. Los EAPs se pueden crear con diferentes atributos, por ejemplo, el área o la desviación estandard del color de una región, extrayendo más información espacial y dando lugar a los perfiles morfológicos basados en multi-atributos (EMAPs) [42]. En general, los esquemas de clasificación basados en MM usando perfiles extendidos (EMP, EAP, EMAP) han demostrado ser más eficientes en términos de precisión que los esquemas basados en segmentación [58, 65], a cambio de incrementar el coste computacional. En esta tesis hemos desarrollado un esquema de clasificación espectral-espacial basado en EMP (WT–EMP), teniendo en cuenta su posterior ejecución en hardware de bajo coste para procesamiento en tiempo real. Este esquema crea un EMP a partir de un conjunto representativo de los datos y los concatena con la información espectral creando un nuevo conjunto nuevo de características para cada pixel.

Independientemente de las técnicas utilizadas en los esquemas de clasificación, fallos en la calibración de los sensores o fenómenos atmosféricos pueden afectar a la calidad de los datos [146]. La consecuencia más común es la presencia de ruido en la imagen. Por lo tanto, normalmente se requiere un preprocesado de los datos para reducción de ruido [164, 166]

o corrección de dispersión [140]. La transformada *wavelet* es una herramienta matemática para procesamiento de señales que se ha aplicado en teledetección para filtrado de ruido, así como otros preprocesados de la imagen como compresión de datos [61] y reducción de características [86]. En el esquema WT–EMP propuesto en esta tesis, usamos la transformada *wavelet* para extracción de características antes de crear el perfil morfológico extendido, y también para eliminar ruido en cada banda espectral de la imagen original.

A pesar de que existen una gran variedad de esquemas de clasificación espectral-espacial, la mayoría resultan computacionalmente poco eficientes en términos de tiempos de ejecución debido al gran volumen de datos al que deben hacer frente. Por lo tanto, para el uso de estos esquemas en aplicaciones en tiempo real necesitamos implementaciones eficientes en las arquitecturas usadas para su ejecución. Esto es de especial interés en aplicaciones con tiempo de respuesta crítico como monitorización de desastres naturales o detección de objetivos a bordo para salvamento marítimo, donde la toma de decisiones se hace en tiempo real [129, 77, 18]. Además, como las imágenes n-dimensionales han estado más disponibles en los últimos años debido principalmente a la reducción en tamaño y coste de los sensores espectrales, el número de aplicaciones a mediana y pequeña escala como es el control de la calidad en los alimentos [30], la detección de falsificaciones en obras de arte [100], el diagnóstico de enfermedades [30, 102] y medicina forense [50], también se ha visto incrementado. La necesidad de una computación eficiente en arquitecturas de bajo coste está aumentando a medida que incorporamos esta tecnología en aplicaciones de uso diario. En esta tesis nos hemos centrado en diseñar y desarrollar esquemas de clasificación eficientes aplicados al campo de la teledetección para procesar imágenes de la superficie terrestre en tiempo real.

El campo de la computación de altas prestaciones aplicado a la teledetección abarca desde grandes infraestructuras de servidores [49, 75, 130] hasta hardware de bajo coste como las FPGAs (Field Programmable Gate Arrays, en inglés) [129, 126, 67], pasando por CPUs multihilo y las GPUs [126, 127, 67, 36, 20, 137]. Mucha de la investigación en este área se ha centrado en el campo de desmezclado de información espectral [129, 67] y en la detección de objetivos [77, 18]. El hardware más idóneo depende principalmente de la aplicación final, el presupuesto y el espacio disponible para instalar dicho hardware. En el caso de esquemas de clasificación espectral-espacial muy pocos se han adaptado para la GPU [20, 137] o han sido especialmente diseñados desde el principio para tal fin.

Dadas la complejidad de los sistemas de clasificación espectral-espacial y la gran cantidad de datos disponibles en las imágenes hiperespectrales, nos planteamos la siguiente pregunta

en esta tesis:

> *¿Es posible diseñar esquemas de clasificación espectral-espacial que produzcan buenos resultados en términos de precisión, y que se puedan ejecutar en tiempo real utilizando infraestructuras de computación de bajo coste para el procesamiento a bordo de datos hiperespectrales?.*

El éxito de un esquema eficiente requerirá de un estudio a fondo para encontrar técnicas que mejoren la precisión de la clasificación adaptadas al modelo de computación de este hardware de bajo coste, como una GPU.

Las contribuciones principales de esa tesis son:

1. **Análisis de esquemas de clasificación espectral-espacial basados en la segmentación y perfiles morfológicos**. En particular, nos hemos centrado en distintas formas de incorporar la información espacial en los sistemas de clasificación hiperespectrales basados en SVM. Dedicamos especial atención a técnicas para la extracción de características, así como al procesamiento espacial basado en morfología matemática, técnicas de segmentación como la transformada watershed, y técnicas de fusión de datos para combinar la información espectral y espacial.

2. **Propuesta de esquemas de clasificación espectral-espacial**. Hemos propuesto los siguientes esquemas:

   – CA-WSHED–MV [137, 136] es un esquema basado en segmentación, SVM y fusión de datos vía votación mayoritaria. Este esquema está basado en el framework de clasificación espectral-espacial propuesto por Tarabalka [156, 155]. El esquema consiste en calcular un gradiente vectorial (RCMG) que reduce la dimensionalidad de la imagen a una sola banda, la cual es segmentada usando una transformada watershed basada en autómatas celulares. La clasificación se lleva a cabo mediante SVM. Por último, las regiones segmentadas se combinan con los resultados de clasificación usando una técnica de votación. La novedad de este esquema es que el algoritmo de watershed, basado en autómatas celulares, no genera líneas de segmentación las cuales no están asignadas a ninguna región y, por lo tanto, no necesita un procesamiento adicional para incluir dichas líneas en el esquema. Además, este algoritmo sigue un modelo de computación en el cual las

celdas del autómata se pueden dividir en grupos y asignarse a diferentes unidades de computación, donde se pueden actualizar de forma asíncrona.

- WT–EMP [133] es un esquema basado en wavelets, morfología matemática y SVM. Fue diseñado teniendo en cuenta su posterior ejecución en GPU. La transformada wavelet se usa 1) para extraer información espectral usando filtros $9/7$, y 2) para reducción de ruido utilizando un conjunto de tres filtros [148]. La morfología matemática se usa para crear un perfil morfológico extendido a partir de las características extraídas por wavelets. Este nuevo esquema de clasificación espectral-espacial mejora los resultados en términos de clasificación en comparación con otros basados en segmentación y MM.

3. **Desarrollo de técnicas y estrategias para computación eficiente en GPU**. Hemos aplicado diferentes estrategias, y en particular, hemos propuesto una estrategia basada en computación asíncrona (block–asynchronous) con la que es posible ejecutar autómatas celulares en GPU de forma más eficiente. Esta estrategia reduce el número de puntos globales de sincronización y explota de forma eficiente la jerarquía de memoria de esta arquitectura; además, resulta adecuada para arquitecturas multi-hilo y se ha adaptado para su uso tanto en imágenes 2D como en volúmenes en 3D. En particular, la hemos aplicado a:

   - CA–Watershed: éste es el autómata celular asíncrono para calcular la transformada watershed en GPU [139, 135, 138]. El comportamiento asíncrono de esta propuesta introduce irregularidades en los bordes entre regiones que hemos solucionado corrigiendo la velocidad de propagación entre bloques, usando una técnica denominada *wavefront* [112].
   - Operaciones morfológicas de apertura y cierre por reconstrucción basados en el mismo principio de computación asíncrona en GPU [134]. Estas técnicas se usan para crear los perfiles morfológicos empleados en el esquema WT-EMP.
   - Filtrado basado en atributos. Se trata de una nueva propuesta para el filtrado (apertura y cierre) de imágenes en escala de grises en GPU y que además, se puede extender a diferentes atributos. Este tipo de filtrado se aplica en esquemas de clasificación basados en atributos extendidos (EAPs) y multi-atributos (EMAPs).

4. **Implementación eficiente de esquemas de clasificación espectral-espacial en CPUs multi-hilo y GPUs, usando OpenMP y CUDa, respectivamente**.

- CA–WSHED-GPU [136, 137] es la proyección en GPU del esquema CA-WSHED-MV. Se han estudiado distintas estrategias para particionar los datos y diferentes configuraciones de bloques con el fin de explotar al máximo la capacidad computacional de la GPU. Para el RCMG hemos usado un particionamento en el dominio espectral. La segmentación está basada en el autómata celular asíncrono (CA–Watershed) y la fusión de datos por votación se ha hecho con operaciones atómicas.
- WT–EMP–GPU [134] es la implementación en GPU del esquema WT–EMP, con el que hemos conseguido alcanzar ejecución en tiempo real en la GPU. Hemos adaptado la extracción de características por wavelets para aplicarla a miles de pixel vectors en paralelo. Hemos tenido que realizar una nueva implementación de la transformada wavelet en dos dimensiones para manejar los tres filtros que se usan en la etapa de eliminación de ruido. La reconstrucción morfológica asíncrona en GPU la hemos aplicado para construir el perfil morfológico extendido.

A pesar de las diferentes soluciones de GPU para la clasificación basada en SVM encontrados en la literatura, desarrollamos una nueva aplicación para clasificación de problemas multi-clase en GPU [134], la cual es compatible con los modelos de entrenamiento generados por la librería de facto LIBSVM [35].

Abstract

The objective of this thesis is to develop efficient schemes for spectral-spatial n-dimensional image classification. By efficient schemes, we mean schemes that produce good classification results in terms of accuracy, as well as schemes that can be executed in real-time on low-cost computing infrastructures, such as the Graphics Processing Units (GPUs) shipped in personal computers. The n-dimensional images include images with two and three dimensions, such as images coming from the medical domain, and also images ranging from ten to hundreds of dimensions, such as the multi- and hyperspectral images acquired in remote sensing.

In image analysis, classification is a regularly used method for information retrieval in areas such as medical diagnosis, surveillance, manufacturing and remote sensing, among others. In addition, as the hyperspectral images have been widely available in recent years owing to the reduction in the size and cost of the sensors, the number of applications at lab scale, such as food quality control, art forgery detection, disease diagnosis and forensics has also increased. Although there are many spectral-spatial classification schemes, most are computationally inefficient in terms of execution time. In addition, the need for efficient computation on low-cost computing infrastructures is increasing in line with the incorporation of technology into everyday applications.

In this thesis we have proposed two spectral-spatial classification schemes: one based on segmentation and other based on wavelets and mathematical morphology. These schemes were designed with the aim of producing good classification results and they perform better than other schemes found in the literature based on segmentation and mathematical morphology in terms of accuracy. Additionally, it was necessary to develop techniques and strategies for efficient GPU computing, for example, a block–asynchronous strategy, resulting in an efficient implementation on GPU of the aforementioned spectral-spatial classification schemes. The optimal GPU parameters were analyzed and different data partitioning and thread block

arrangements were studied to exploit the GPU resources. The results show that the GPU is an adequate computing platform for on-board processing of hyperspectral information.

Keywords: graphics processing unit (GPU), GPGPU, wavelet transform, denoising, mathematical morphology, morphological profile, feature extraction, image classification, image segmentation, remote sensing, hyperspectral imaging.

Chapter 1

Thesis overview

The objective of this thesis is to develop efficient schemes for spectral-spatial n-dimensional image classification based on segmentation, Mathematical Morphology (MM), and Support Vector Machine (SVM) classifiers. By efficient schemes, we mean schemes that produce good classification results in terms of accuracy, as well as schemes that can be executed in real-time on low-cost computing infrastructures. The first goal is accomplished by designing new schemes, while the second one is achieved by developing techniques that allow their efficient computation in commodity hardware, such as multi-threaded CPUs and many-core architectures, such as the Graphics Processing Units (GPUs) shipped in personal computers. The n-dimensional images include images with two and three dimensions, such as in the medical domain, and also images ranging from ten to hundreds of dimensions, such as the multi- and hyperspectral images acquired in remote sensing.

In the case of multi- and hyperspectral image analysis, a pixel is represented as a vector of spectral values or features (pixel vector). Hyperspectral images are collected by optical sensors which capture different wavelength responses from the visible to the near infrared (NIR) spectrum in the same line scan, and they can be seen as a hyperspectral cube of several images or spectral bands [66]. The first space-based multispectral scanner capable of collecting 4 bands with a spatial resolution of 80 m by pixel was launched in 1972 in the Landsat-1 satellite [95]. The *hundreds of features* came in November 2000 with the Hyperion imaging spectrometer [123]. Since then, remote sensing has been an extensive research area for mineral mapping [92]; the identification of urban areas, such as road extraction and urban change detection [107]; and vegetation applications such as forest degradation analy-

sis, among others [66, 128]. In this work we have used hyperspectral images captured by the Airborne Visible-infrared Imaging Spectrometer (AVIRIS) [71] and the Reflective Optics System Imaging Spectrometer (ROSIS-03) [113] sensors. These two airborne hyperspectral sensors cover a range of 0.4 to 0.86 μm (ROSIS-03) and 0.4 to 2.4 μm (AVIRIS) using 115 and 224 spectral channels, respectively, with a spatial resolution varying from 1 to 20 m/pixel.

The special characteristics of hyperspectral images, which provide detailed information for each pixel, make it possible to distinguish physical materials and objects even at pixel level. The high dimensionality of these data introduces new challenges in spectral unmixing [89, 23, 24], target and anomaly detection [107, 14], feature extraction [14], and background characterization, including land-cover classification [96, 128]. The latter has been a very common topic in recent decades. Among the different challenges, this thesis deals with the classification of n-dimensional images.

A classification process consists in grouping and naming things based on one or more common property or feature. Success in classification depends not only on the ability to categorize and/or discriminate objects, but also on the set of features used for the same. Thus, adding the most relevant features in the process of classifying is mandatory. In image analysis, classification is a regularly used method for information retrieval in medical diagnosis, surveillance, manufacturing and remote sensing, among others [128, 146]. The features that come into play for learning to categorize a pixel in an image are mainly related to its intensity and its color; for example, the red, green and blue values of a pixel in an RGB image. In hyperspectral images there are a larger number of features that can be taken into account for the classification.

Considering that the set of features in a hyperspectral image is often redundant, Feature Extraction (FE) techniques such as Principal Component Analysis (PCA) are usually performed with the objective of reducing the dimensionality of the image, while the main characteristics are kept [146]. Different techniques, such as Independent Component Analysis (ICA), Minimum Noise Fraction (MNF), Discriminant Analysis Feature Extraction (DAFE) and Non-parametric Weighted Feature Extraction (NWFE) have been investigated for extracting Principal Components (PCs), showing their effectiveness in remote sensing classification [48, 31, 93, 96].

To take full advantage of the rich information provided in the hyperspectral images, a large number of classification methods have been proposed [72, 55, 146], such as maximum likelihood, neural networks, decision trees and kernel-based methods. An exhaustive evalu-

ation of classifiers belonging to a wide collection of families was presented in [60]. In their work, the authors concluded that the classifiers most likely to be the best were the random forest and the Support Vector Machine (SVM). In remote sensing, the SVM classifier [1] has shown good classification results even when a small number of training samples are available [72, 55]. For this reason, we have used the SVM classifier in this work. This classification method processes each pixel vector independently considering only the spectral information. However, it has been clearly stated that the spatial information extracted from the hyperspectral image improves the accuracy of the classification when it is incorporated into the scheme [164, 15, 54, 154, 155, 41, 19, 29, 56, 57].

Consequently, the schemes for hyperspectral image classification have changed from pixel-wise classifiers to spectral-spatial classification schemes. A survey of these advances can be found in [58, 24, 65]. In particular, we are interested in the schemes based on segmentation and morphological profiles for extracting spatial information.

The segmentation techniques have been widely investigated in remote sensing for extracting and incorporating spatial information in the classification schemes [117, 12, 154, 155, 157, 132, 160]. When using the regions created by a segmentation technique, the spatial structures that may be present in the scene are taken into account. For instance, the segmentation techniques used in [12] are based on level-set methods. Clustering algorithms were used in [154], and evolutionary Cellular Automata (CA) were designed and applied in [132] for segmenting hyperspectral images without reducing the dimensionality of the data. The watershed transform was applied in [117, 155, 156, 158, 78], and hierarchical segmentation (HSEG) in [160]. Among all the techniques, the watershed transform [168] has drawn more attention in the classification schemes based on segmentation, although it cannot be applied directly to multidimensional images. The most common approach for applying the watershed transform to multidimensional images consists in reducing the number of spectral dimensions to one, for example, by feature extraction and by vectorial or multidimensional gradient computation [155]. In this thesis we have developed a segmentation-based scheme [137] (CA–WSHED-MV) using cellular automata for computing efficiently the watershed transform (CA–Watershed). This CA–Watershed algorithm is adequate for multi-core and many-core architectures [139, 135, 138] as the automaton can be partitioned into blocks of cells that run independently from each other.

Mathematical Morphology is defined as a theory for the analysis of spatial structures [152] and it has been successfully applied in remote sensing for image classification through the use

of Morphological Profiles (MPs) [124] and Attribute Profiles (APs) [41], which are used to model different kinds of spatial structures (objects). The profiles keep objects in the image if a structuring element or attribute fits within the object, otherwise they are removed. Through the sequential application of morphological operations and increasing the size of the structuring element (or filter), the MP and AP create a multilevel characterization of the image. The extension to n-dimensional images is possible by applying the profile to each band (or to a representative subset) through the Extended Morphological Profile (EMP) [121, 15] and the Extended Attribute Profile (EAP) [41], respectively. The latter can be further extended with different attribute filters, (e.g. the area and standard deviation of pixel colors within a region), extracting more spatial information from the image, creating an Extended Multi-Attribute Profile (EMAP) [42]. In general, MM-based classification schemes using extended profiles have shown to be more efficient than segmentation-based approaches in terms of classification accuracy [58, 65], at the expense of increasing the computational cost. In this thesis we have developed a scheme based on Extended Morphological Profiles (WT–EMP) and taking into account its subsequent projection on low-cost computing infrastructures for real-time processing. The scheme creates the EMP from a representative subset of the hyperspectral bands that is combined with the spectral data in a new vector of features.

Regardless of the techniques used in remote sensing, undesirable artifacts such as improper calibration of the sensor or atmospheric phenomena may affect data quality [146]. The most common consequence of these artifacts is the presence of noise in the image. Therefore, a preprocessing step, such as denoising [164, 166] or scatter correction [140], is usually required before classifying the images. Wavelets are mathematical tools for signal processing and have been investigated in remote sensing for filtering the noise introduced in the acquisition of the image, as well as additional preprocessing like data compression [61] and feature extraction [86]. In the WT–EMP scheme proposed in this thesis, wavelets are used for feature extraction to create the EMP, and also for denoising over each band of the original hyperspectral image.

Although there are many spectral-spatial classification schemes, most are computationally inefficient in terms of execution time, as they have to deal with a high number of features resulting in large execution times. Therefore, the computation of these schemes for real-time applications requires their efficient implementation on the adequate computing architecture. This is particularly true in time-critical applications in remote sensing, such as natural disasters monitoring and on-board target detection for maritime rescue where decisions are made

in real time [129, 77, 18]. In addition, as the hyperspectral images have been widely available in recent years owing to the reduction in the size and cost of the sensors, the number of applications at lab scale, such as food quality control [30], art forgery detection [100], disease diagnosis [30, 102] and forensics [50], has also increased. The need for efficient computation on low-cost computing infrastructures is increasing in line with the incorporation of technology into everyday applications. This thesis focuses on developing efficient spectral-spatial schemes for land-cover classification in remote sensing imaging for real-time applications.

The research in High Performance Computing (HPC) for remote sensing applications covers a field ranging from infrastructures of clusters [49, 75, 130] to Field Programmable Gate Arrays (FPGAs) [129, 126, 67] and commodity hardware such as multi-threaded CPUs and many-core GPUs [126, 127, 67, 36, 20, 137]. Most of the research in HPC for remote sensing has been done in the field of spectral unmixing involving endmember extraction [129, 67], and also in target detection [77, 18]. The most suitable hardware for HPC depends mainly on the final application, the budget and the space available for the computing infrastructures. The GPU, with its high computational capacity has not been yet fully exploited. In the case of spectral-spatial classification schemes, only a few have been adapted for GPU [20, 137], and even fewer have been specially designed for that purpose from the outset.

Given the complexity of the spectral-spatial classification schemes, and the high amount of data available in the hyperspectral images, we posed the following question in this thesis:

> *Is it possible to design efficient spectral-spatial classification schemes that produce good classification results and can be executed in real-time, using low-cost computing infrastructures for on-board processing of hyperspectral information?*

The success of an efficient scheme will require a thorough study to find techniques that improve the classification accuracy, while matching the computing model of the low-cost computing infrastructures, such as a GPU.

1.1 Main contributions

As a result of the research conducted in this thesis to find a solution to our main question, the following contributions to the field of remote sensing and HPC have been produced:

1. **Analysis of spectral-spatial classification schemes based on segmentation and morphological profiles**. In particular, we focus on the different ways of incorporating spatial information into the pixel-wise spectral classification schemes based on the SVM classifier. We devote special attention to feature extraction techniques, as well as spatial processing by mathematical morphology, segmentation techniques based on clustering, such as kmeans and quick-shift, and segmentation techniques based on region growing such as the watershed transform, and data fusion strategies for combining the spectral and spatial information. We have taken into account the efficient computation on GPU of the techniques under study.

2. **Proposal of spectral-spatial classification schemes**. The following schemes are proposed:

   – CA-WSHED–MV [137, 136] is a scheme based on segmentation, SVM and Majority Vote. This scheme is based on the spectral-spatial classification framework proposed by Tarabalka et al. [156, 155]. The scheme consists of the calculation of a Robust Color Morphological Gradient (RCMG) which reduces the dimensionality of the hyperspectral image, followed by the calculation of a watershed transform based on cellular automata which produces the spatial results. The classification is carried out by SVM. Finally, the spectral and spatial results are combined with a majority vote. The novelty of this scheme is mainly introduced in the watershed algorithm based on cellular automata used for segmenting the hyperspectral image. This algorithm follows a computational model in which the grid of cells of the automaton is partitioned into regular regions that are assigned to different blocks of threads on the GPU that can be asynchronously updated. In addition, this implementation does not create the so-called watershed lines, and thus it is not necessary to compute a standard vector median [7] for every watershed region as in [155].

   – WT–EMP [133] is a scheme based on wavelets, MM and SVM. This scheme was designed taking into account its efficient computation in a subsequent GPU execution. The Wavelet Transform is used for feature extraction and image denoising using $9/7$ wavelet filters for the former and a set of three filters for perfect reconstruction [148] for the latter. Mathematical Morphology is used for creating the EMP from the features extracted by wavelets. This new spectral-spatial classifi-

cation scheme improves the classification results in terms of accuracy in comparison to other spectral-spatial schemes based on segmentation and Mathematical Morphology.

3. **Development of techniques and strategies for efficient GPU computing**. Different strategies are applied and, in particular, a block–asynchronous strategy that maps cellular automata on the GPU is proposed. This strategy reduces the number of points of global synchronization allowing efficient exploitation of the memory hierarchy of this architecture. The block–asynchronous strategy is also adequate for multicore architectures, and it has been tuned to be used with 2D and 3D images. It is applied to:

   – CA–Watershed: this is an asynchronous cellular automaton to compute the watershed transform on GPU [139, 135, 138]. The asynchronous behavior of the CA–Watershed introduces artifacts in the border of the segmented regions that are fixed by correcting the data propagation speed among the blocks using wavefront techniques [112].

   – Opening and closing: an improved algorithm for opening and closing by reconstruction on GPU based on the block-asynchronous approach (*BAR*) [134]. The algorithm is applied to create the Extended Morphological Profile used in land-cover spectral-spatial classification schemes.

   – Attribute filtering: a new proposal for greyscale attribute opening and closing on GPU. This proposal is the first attempt to compute this filter on greyscale images on GPU and can be extended to other attributes. The proposal can be applied to create an Extended Attribute Profile for land-cover spectral-spatial classification schemes.

4. **Efficient implementation of spectral-spatial classification schemes on multi-threaded CPUs and many-core GPUs by using OpenMP and CUDA, respectively**.

   – CA–WSHED-GPU [136, 137] is an efficient projection on GPU of the first proposed scheme (CA-WSHED-MV). Different hyperspectral data partitioning strategies and thread block arrangements are studied in order to effectively exploit the memory and computing capabilities of this architecture. The spectral partitioning is used to compute the RCMG. The watershed transform is based on the asynchronous cellular automaton (CA-Watershed) that efficiently exploits the GPU

architecture, and the majority vote is done by atomic operations to avoid memory race conditions.

– WT–EMP–GPU [134] is a GPU implementation of the WT–EMP scheme that achieves real-time in commodity hardware. We adapted the feature extraction by wavelets computing thousands of pixel vectors in parallel. A new implementation for two dimensional wavelet transforms was required in order to manage the three filters used in the denoising step. Finally, we used the block–asynchronous strategy for morphological reconstruction to compute the EMP used in the scheme.

Despite the different GPU solutions for SVM classification found in the literature, a new implementation was developed for classifying multi-class problems on GPU [134], that is compatible with the one against one trained models produced by the facto LIBSVM library [35].

1.2 Publications

1.2.1 Book Chapters

[137] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Computing efficiently spectral-spatial classification of hyperspectral images on commodity gpus," *in Recent Advances in Knowledge-based Paradigms and Applications* (J. W. Tweedale and L. C. Jain, eds.), vol. 234 of Advances in Intelligent Systems and Computing, Ch. 2, pp. 19–42, Springer International Publishing, 2014.

1.2.2 International Journals

[138] P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2638–2655, 2013.

[78] D. B. Heras, F. Argüello, and P. Quesada-Barriuso, "Exploring ELM-based spatial-spectral classification of hyperspectral images," *International Journal of Remote Sensing*, vol. 35, no. 2, pp. 401–423, 2014.

[133] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Spectral-spatial classification of hyperspectral images using wavelets and extended morphological profiles," *Selected Topics*

in Applied Earth Observations and Remote Sensing, IEEE Journal of, vol. 7, no. 4, pp. 1177–1185, 2014.

[134] P. Quesada-Barriuso, F. Argüello, D. B. Heras, and J. A. Benediktsson, "Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. PP, no. 99, pp. 1–9, 2015 (published online, print edition pending).

[101] J. López-Fandiño, P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on commodity gpus," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. PP, no. 99, pp. 1–10, 2015 (published online, print edition pending).

1.2.3 International Conferences

[135] P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient GPU asynchronous implementation of a watershed algorithm based on cellular automata," *in Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pp. 79–86, 2012.

[136] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Efficient segmentation of hyperspectral images on commodity GPUs," *in 16th International Conference on Knowledge-Based and Intelligent Information & Engineering System*, vol. 243, pp. 2130–2139, 2012.

1.2.4 National Conferences

[139] P. Quesada-Barriuso, J. Lamas-Rodríguez, D. B. Heras, and F. Argüello, "Influencia de las mesetas en la implementación de watershed sobre GPUs," *in XXIII Jornadas de Paralelismo*, pp. 249–254, 2012.

[94] J. Lamas-Rodríguez, P. Quesada-Barriuso, F. Argüello, D. B. Heras and y M. Bóo, "Proyección del método de segmentación del conjunto de nivel en GPU," *in XXIII Jornadas de Paralelismo*, pp. 273–278, 2012.

1.3 Thesis organization

This thesis is organized in six chapters. Chapter 1 introduces the objectives and the main contributions of this thesis. Chapter 2 presents the fundamental concepts involved in this work. First, the main characteristics of the *n*-dimensional images are reviewed. Second, the techniques used in the proposed spectral-spatial classification schemes are detailed. Different feature extraction techniques that have been successfully applied in remote sensing classification are described. The concepts regarding the spatial processing of the images, such as the segmentation, mathematical morphology and attribute filtering are also reviewed in this chapter. Then, cellular automata and the wavelet transform are introduced, and we review the pixel-wise classifiers, in particular the SVM, the one used in our schemes. In addition, the OpenMP and Compute Unified Device Architecture (CUDA) computing model, as well as the hardware resources and the performance measures used in the experiments are introduced in this chapter. Finally, the datasets used in the experiments are described.

Chapter 3 presents our two schemes proposed for efficient *n*-dimensional image classification: CA–WSHED–MV and WT–EMP. The general framework of the spectral-spatial classification schemes and the common data fusion strategies to join the spectral and the spatial information of such schemes are presented in this chapter. The results thereof are compared on real hyperspectral images, in terms of classification accuracy, with the classification schemes found in the literature based on segmentation and MM.

Chapter 4 focuses on the techniques and strategies applied for the efficient computation of the proposed schemes on commodity hardware such as multi-threaded CPUs and many-core GPUs. First, we describe the general strategies related to data partitioning, data movement and data packing, and highlight the challenges of GPU computing. Then, we present the block–asynchronous computation strategy proposed in this thesis. This approach is applied to compute an asynchronous CA–Watershed, opening and closing by reconstruction and attribute filtering by opening and closing. The strategies for efficient projection of wavelets, as well as multi-class SVM classification on GPU are also described in this chapter. The techniques are independently analyzed comparing the execution times to parallel multi-threaded CPU implementations using OpenMP.

In chapter 5 we describe the CUDA implementation of the proposed schemes, named CA–WSHED–GPU and WT–EMP–GPU, applying the techniques described in chapter 4. We also analyze the optimal GPU parameters and the hardware resources that usually limit the occupancy on this architecture. The performance is measured in terms of execution time

and speedup on real hyperspectral images over CPU multi-threaded implementations using commodity hardware.

Finally, the last chapter remarks the contributions, presents the conclusions and proposes future research developments.

CHAPTER 2

FUNDAMENTALS

In this chapter we review the fundamental concepts and present the techniques used in the spectral-spatial classification schemes proposed in this thesis. In particular, the main characteristics of the n-dimensional images are reviewed and the concepts regarding the spatial processing of the images, such as the watershed transform, cellular automata, vectorial gradients, mathematical morphology and wavelet transform are outlined. We then go on to describe the pixel-wise classifiers and the classification map produced by the same, in particular for SVM that is the classifier used in our scheme. In addition, the OpenMP and the CUDA parallel programming models, as well as the hardware resources and the performance measures are introduced in this chapter. Finally, the datasets used in the experiments are detailed.

2.1 n-dimensional images

A n-dimensional (nD) image is considered a dataset where two dimensions represent a spatial location $[x,y]$, while the remaining dimensions represent phenomena occurring per spatial location [3]. Therefore, a pixel is represented as a vector of values or features (pixel vector). For example, a nD image can be produced by optical sensors which capture different wavelength responses (one per spectral band) in the same line scan. Multispectral images usually are composed of a few, usually three to ten, spectral channels, while hyperspectral images have hundreds of spectral bands. These datasets can be seen as spectral cubes of several (hundreds) images where $[x,y]$ varies across the physical space and $[z]$ varies across the electromagnetic spectrum. [66]. A nD image can also be a set of images of the same scene acquired by the

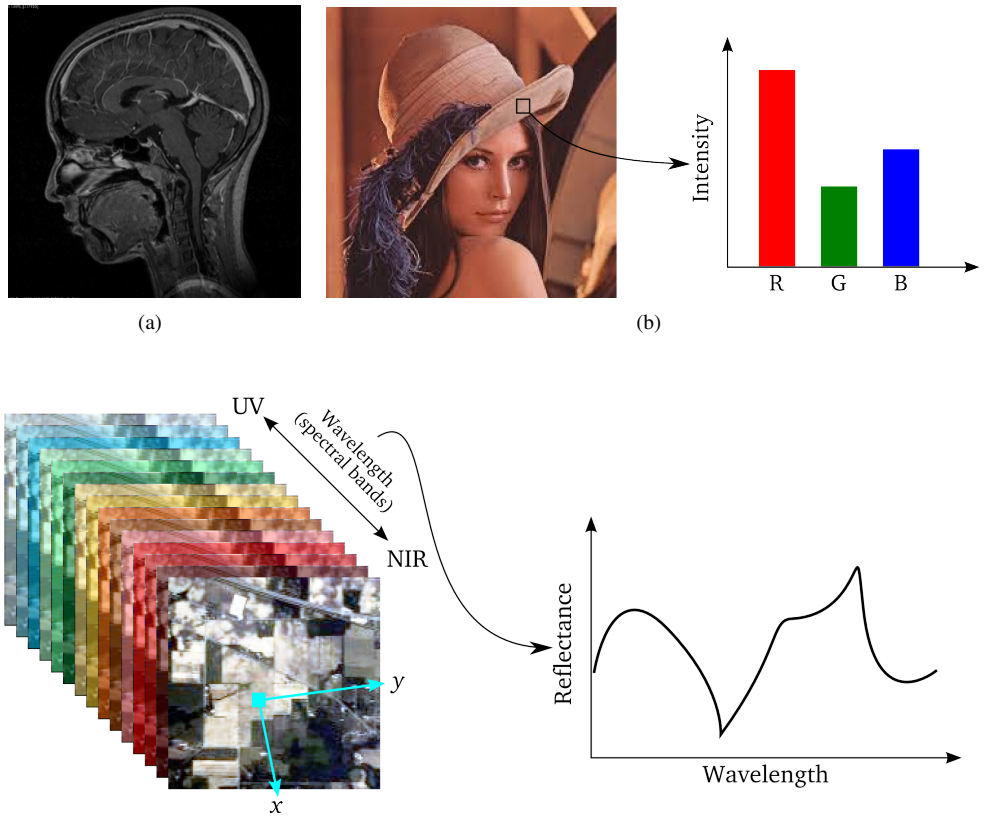


Figure 2.1: *n*-dimensional images. (a) 2D slice of a CT scan, (b) color image and the intensity of the RGB channels, and (c) hyperspectral image and the different wavelengths responses of a pixel vector.

same sensor (hyperspectral or not) at different times or from multiple sensors capturing the same scene at the same time, with both cases resulting in a stack of images.

We also considered color images (RGB) and 3D images (volumes) as *n*-dimensional images as they can be represented as stacked images, although in the case of 3D images, each pixel represents a spatial location $[x, y, z]$ where the pixels are spatially connected in the $[z]$ dimension as well. In the case of RGB, the image has red, green, and blue wavelengths responses that are independently stacked and then combined to produce the color image. A volume, such as the on generated by a computerized tomography (CT) scan, combines a se-

ries of X-ray images to create cross-sectional slices of the human body. Figure 2.1 shows a slice of a CT scan of a human head, thus, it is a 2D image, and a color image indicating the intensity of the RGB channels for a pixel.

By extracting spatial features from a single image, we can also create n-dimensional images by stacking the new features as a pixel vector. Figure 2.1(c) shows a hyperspectral image and the different wavelength responses of a pixel vector. The term n-dimensional image will be used hereinafter for images with more than one stacked feature, and the term pixel vector will be used to refer a pixel from these images.

2.2 Dimensionality reduction

Dimensionality reduction has been used in remote sensing as a preprocessing technique to reduce the dimensionality of the hyperspectral images onto a low-dimensional space, while keeping the meaningful information [146].

Two common approaches are widely used for dimensionality reduction: Feature Selection (FS) and Feature Extraction (FE) [34]. In the former, also referred as best band selection, a subset of relevant features is selected via a certain criterion, while the latter combines and transform the original data in a lower-dimensional space. FE is mainly based on statistical analysis of the Principal Components (PCs). In the following we describe different FE techniques that have been investigated and successfully applied in remote sensing classification. Most of the criteria designed for FE techniques are also applicable to FS methods.

2.2.1 Feature extraction

Feature extraction can be grouped in unsupervised techniques, such as Principal Component Analysis (PCA), Minimum Noise Fraction (MNF) and Independent Component Analysis (ICA), and supervised techniques such as Discriminant Analysis Feature Extraction (DAFE), Decision Boundary Feature Extraction (DBFE) and Non-parametric Weighted Feature Extraction (NWFE). Principal Component Analysis (PCA) is one of the most frequently used unsupervised techniques in remote sensing for extracting the main characteristics of the hyperspectral images.

In the analysis by PCA, data are transformed into a set of uncorrelated components which represents a new low-dimensional space by its new components. PCA is a linear transformation that makes use of eigenvectors to determine the significance of PCs. It computes the

maximum amount of data variance in a new uncorrelated data set extracting the principal components using the eigenvalues in a decreasing order; i.e., the largest values obtained in the analysis are kept.

Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ be a pixel vector of n features, and X be a n-dimensional image with K pixel vectors, whose mean is zero across each dimension, that is, $\mathbf{x}_k = \mathbf{x}_k - \mu$ for each $\mathbf{x} \in X$, with μ the mean position of the pixels in the space.

$$\mu = \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_k. \tag{2.1}$$

Then, the covariance matrix is obtained by:

$$\Sigma_x = \frac{1}{K} \sum_{k=1}^{K} \Big[(\mathbf{x}_k - \mu)(\mathbf{x}_k - \mu)^T \Big]. \tag{2.2}$$

The PCA is based on the following transformation of the covariance matrix:

$$\Sigma_x = EDE^T, \tag{2.3}$$

where D is the diagonal matrix with the eigenvalues $\lambda_1, \lambda_1, \ldots, \lambda_n$, and E is the eigenvector matrix of Σ_x.

The linear transformation of each pixel vector $\mathbf{x}$ is defined by

$$\mathbf{y}_k = E^T \mathbf{x}_k, \qquad k = 1, 2, \ldots, K, \tag{2.4}$$

where $\mathbf{y}_k$ is the transformation of each pixel to the new uncorrelated space. The first principal component is extracted via (2.4) from the first largest eigenvalue. We refer the reader to [151, 34] for a comprehensive description about the eigenvalues and covariance matrix computation.

Regarding other unsupervised FE techniques, MNF [70] is similar to PCA but it extracts the principal components sorting the transformation by descending signal-to-noise ratio, rather than the variance. The ICA attempts to minimize the dependencies in statistical independent components. There are different strategies to define the independence for ICA, for example, by minimization of the mutual information.

Among the supervised methods, DAFE uses the mean vector and the covariance matrix of each class to identify discriminant informative features that increase the separability among classes. It seeks the optimal subspace such that the primary axis maximizes the separation between their means, while the standard deviation within a class should be as small as possible [95]. Thus, it only works well if the distributions of classes are normal-like distributions, and it is less effective when the mean between classes is similar.

DBFE finds the best features and the best dimensionality to be used from a decision boundary using the training samples themselves, rather than statistics from them [96]. Thus, it is effective even when there is no difference in the mean vectors or the covariance matrices, although it is highly dependent on training samples.

The NWFE was proposed in [93] to outperform the disadvantages of DAFE and DBFE. The idea is to compute a weighted mean putting different weights on every training sample within a class, giving more weight to nearby samples, and defining new nonparametric between-class and within-class scatter matrices to estimates of the covariance. The NWFE has been shown to perform better than DAFE and DBFE in classification of hyperspectral data [96].

Other non-statistical-based FE techniques, such as the 1D discrete wavelet transform (1D-DWT), have been shown to be better or comparable to traditional methods such as PCA in reducing the number of spectral bands [26, 162, 86, 68]. The use of the discrete wavelet transform (see Section 2.6) for feature extraction in remote sensing was investigated in [26] yielding better or comparable classification accuracy compared to traditional methods such as PCA. An automatic extraction of features using wavelets was presented in [86], where the dimensionality of the image is reduced several times, and then reconstructed by wavelets. The best level of decomposition is then determined through the similarity between the original pixel vector and the vector reconstructed using wavelets. By reducing the spectral bands by wavelets we can extract band-coefficients representing an approximation of the original data.

Multidimensional and vectorial gradients methods have been used to reduce the number of features to one in order to apply different spatial techniques on one band of the image. Thus, reducing the computational cost of any possible future process.

2.2.2 Vectorial gradients

The boundaries or edges of an object are generally characterized by grey-level intensity transitions. Accordingly, gradient operators are usually applied prior to an image segmentation to enhance these variations [152].

The morphological gradient operator for greyscale images is defined as [141]:

$$\nabla(f) = \delta_g(f) - \varepsilon_g(f) \ , \tag{2.5}$$

where δ_g and ε_g are the dilation and erosion morphological operators described in Sect. 2.5, and g the structuring element which defines the neighborhood of a pixel in the image f.

Eq. (2.5) can be expressed as follows:

$$\nabla(f) = \max_{x \in g}\{f(x)\} - \min_{y \in g}\{f(y)\} = \max(|f(x) - f(y)|) \quad \forall x, y \in g \ , \tag{2.6}$$

giving the greatest intensity difference between any two pixels x, y within the structuring element g.

Eq. (2.6) can be extended to n-dimensional images through multidimensional and vectorial gradients. Multidimensional gradients calculate on each channel (or spectral band) a gradient (e.g. given by (2.6)), and take the sum or the supremum of the gradients [117].

Vectorial gradients are based on distance between pairs pixel vectors. Let $\mathbf{x}_n$ be a pixel vector of n features, for example $n = 3$ in a RGB image, and $\chi = [\mathbf{x}_n^1, \mathbf{x}_n^2, \ldots, \mathbf{x}_n^k]$ be a set of k pixel vectors in the neighborhood of $\mathbf{x}_n$, where χ contains $\mathbf{x}_n$. The vectorial gradient is defined as:

$$\nabla(\mathbf{f}) = \max_{i \in \chi}\{distance(\mathbf{x}_n, \mathbf{x}_n^i)\} - \min_{j \in \chi}\{distance(\mathbf{x}_n, \mathbf{x}_n^i)\} \ , \tag{2.7}$$

Euclidean, Mahalanobis or chi-squared distances can be used to compute vectorial gradients from (2.7).

Another vectorial gradient is the Color Morphological Gradient (CMG) [52] defined as:

$$\mathrm{CMG} = \max_{i,j \in \chi}\{||\mathbf{x}_k^i - \mathbf{x}_k^j||_2\} \ , \tag{2.8}$$

whose response is the maximum of the distances between all pairs of pixel vectors in the set χ using the Euclidean distance. The CMG is highly sensitive to noise and may produce edges that are not representative of the gradient, so a RCMG was proposed in [52], based on pairwise pixel rejection of Eq. (2.8):

$$\mathrm{RCMG} = \max_{i,j \in \chi - R_s}\{||\mathbf{x}_k^i - \mathbf{x}_k^j||_2\} \ , \tag{2.9}$$

where R_s is the set of s pairs of pixel vectors removed from χ which are the furthest apart.

The result of the Robust Color Morphological Gradient, as well as the multidimensional and other vectorial gradients is a one band gradient image. In this thesis we have used the RCMG for feature reduction of hyperspectral images. The pseudocode of the RCMG is given in Section 5.2.1, together with its GPU implementation.

2.3 Segmentation techniques

In image analysis, the segmentation involves the partitioning of an image into homogeneous groups of pixels or connected regions with respect to some criterion, for example the grey level

values. The techniques for image segmentation can be mainly grouped into clustering-based, edge-based and region-based segmentation techniques [63]. In this section we review different clustering and region growing segmentation techniques that have been applied in spectral-spatial classification schemes. In particular, we devote special attention to the watershed transform, a widely used method for non-supervised image segmentation.

2.3.1 Clustering-based segmentation techniques

Clustering techniques create a partition of the data by grouping feature vectors[1] into clusters based on the similarity between the pixel vector and the cluster center until stability [83]. The similarity between pixel vectors is usually measured by a Euclidean distance.

Clustering algorithms have largely been used for classification, data mining and image segmentation. The k-means is the simplest and most commonly used method for image segmentation based on the square error criterion [105]. The goal is to find the position of the k clusters that minimize the distance from a feature vector $\mathbf{x}$ to the cluster. The square error for a clustering of a set $\mathbf{X}$ is:

$$e^2 = \sum_{j=1}^{k} \sum_{i=1}^{n_j} ||\mathbf{x}_i^{(j)} - \mathbf{c}_j||^2, \tag{2.10}$$

where $\mathbf{x}_i^{(j)}$ is the i-th feature vector belonging to the j-th cluster, and $\mathbf{c}_j$ is the center of the j-th cluster, with n_j the number of pixels assigned to that cluster. The pseudocode in Figure 2.2 summarizes the k-means clustering algorithm. Stability is reached when there is no new reassignment of pixels to new clusters, or the square error (2.10) does not decrease significantly after a certain number of iterations [83]. This algorithm is sensitive to the number of clusters and initial position thereof.

By splitting and merging clusters, it is possible to obtain an optimal partition of the data. The ISODATA [11] algorithm splits a cluster if its variance is above a given threshold and merges clusters when the distance between their center is below another threshold. Other clustering algorithms, such as Expectation Maximization (EM) algorithm [46], apply the same clustering principles but assuming that the data follow a Gaussian probability distribution.

The aforementioned algorithms produce only one partition of the data. Hierarchical clustering methods produce a nested partition of the data by merging clusters from a finer level of detail to a coarser level, creating a tree representation (dendrogram) of the data.

[1] In [83] the term pattern or feature vector is used instead of pixel vector.

Input: input data $\mathbf{X}$
Output: partitioning of the data in k clusters

1: choose the number of k clusters and initialize their position $\mathbf{c}_j$ to k feature vectors
2: **repeat**
3: **for** each feature vector $x \in \mathbf{X}$ **do**
4: assign x to its closest cluster based on the Euclidean distance
5: **end for**
6: **for** each cluster C_j **do**
7: recompute $\mathbf{c}_j$ to the mean of all the points $\mathbf{x}_i^{(j)}$ belonging to that cluster
8: **end for**
9: compute the square error function (2.10)
10: **until** stability

Figure 2.2: Pseudocode for k-means clustering.

Input: input image $\mathbf{X}$
Output: hierarchical image segmentation

1: label each pixel of $\mathbf{X}$ as a separate region
2: **while** the number of regions remaining is greater than two **do**
3: **repeat**
4: calculate the dissimilarity criterion $d_{i,j}$ between each spatially adjacent region
5: find the $\min(d_{i,j})$ and merge all pairs of spatially adjacent regions with meet that value
6: calculate the dissimilarity criterion $s_{i,j}$ between all pairs of non-spatially adjacent regions
7: merge all pairs of non-spatially adjacent regions with $s_{i,j} \leq \min(d_{i,j})$
8: **until** stability
9: save current segmentation map
10: **end while**

Figure 2.3: Pseudocode for HSEG algorithm.

By analogy, hierarchical segmentation can be defined as a family of fine to coarse image partitions [153]. Tilton proposed a Hierarchical Image Segmentation (HSEG) algorithm [161] for exploiting the information content on the segmentation hierarchy. It is a hybrid segmentation technique based on hierarchical step-wise optimization (HSWO) [13] and data clustering. The algorithm alternately performs region growing merging spatially adjacent regions, and spectral clustering merging spatially non-adjacent regions.

The HSEG pseudocode shown in Figure 2.3 is based on the description given in [161, 173]. The inner loop (lines 3–8) merges the segmented regions of the image. Lines 4 and

5 correspond to the region growing step and lines 6 and 7 constitute the spectral clustering part. The dissimilarity criterion is based on the Euclidean distance characterized by the mean vectors of the regions. Stability is reached when the number of regions remaining in the segmentation map is lower than a preset value of regions [161]. The outer loop, lines 2–10 performs the hierarchical segmentation of the image.

In clustering-based image segmentation, the labels in the clustering map are created from k different classes; i.e., there are only k distinct values and the segmented regions are not connected by a unique label. Therefore, a Connected Component Labelling (CCL) algorithm is required to re-label the clusters and produce a segmentation map [83]. The segmentation of nD images aims at finding distinct structures in the spectral domain.

2.3.2 Watershed transform

The watershed transform is a widely used method for non-supervised image segmentation, especially suitable for low-contrast images. The idea behind this method comes from geography. A greyscale image can be represented as a topographic relief, where the height of each pixel is directly related to its grey level. The dividing lines of the catchment basins for precipitation falling over the region are called watershed lines [168]. Various definitions, algorithms and implementations can be found in the literature but, in practice, they can be classified into two groups: those based on the specification of a recursive algorithm by Vincent and Soille [168], and those based on the distance functions defined by Meyer [111]. An intuitive approach is to imagine the terrain being immersed in a lake, with holes pierced in local minima [142]. By flooding the terrain into the water, catchment basins will fill up with water as illustrated in Figure 2.4. At the points where water coming from different basins would meet, dams are built. The process stops when the water level reaches the highest peak. The terrain is partitioned into regions separated by the watershed lines. In Figure 2.4 the image has two minimum grey values representing two valleys in the terrain.

One of the main advantages of the watershed transform is that all regions of the image are well defined at the end of the segmentation process, even if the contrast of the image is poor. Hence it has been widely used in image processing, biomedicine and physics. Nonetheless, the results are over-segmented owing to the large number of regions detected. This problem is overcome by preprocessing the image with the objective of reducing the number of regions, for example by a marked-controlled watershed transform that preselects the regions of interest [22].

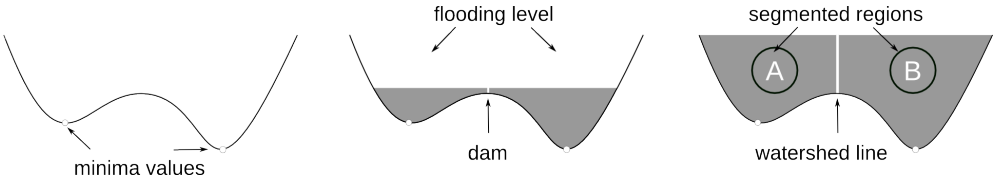


Figure 2.4: Flooding process in a one dimensional image with two minimum, generating a watershed line at the end of the process.

Let us introduce a few concepts and notations about topography in order to continue with the watershed transform. A greyscale image may be considered as a graph $G = (V,A)$ with a finite set of V vertexes (pixels) and a set of arcs $A \subseteq V \times V$ defining the connectivity. Two pixels u and v are connected if $(u,v) \in A$. The pixels connected to u, called neighbors, are denoted by $\mathscr{N}(u)$.

The most widely used connectivity is four, considering the orthogonal neighbors, left, right, up and down, known as Von Neumann connectivity. Another variation is the Moore neighborhood, where the eight neighbors surrounding a pixel are connected. Figure 2.5(a) shows a pixel with 4- and 8-connectivity.

The slope between two neighbors is defined by:

$$\forall u \in V, \forall v \in \mathscr{N}(u), \quad slope(u,v) = h(u) - h(v),$$

where $h(u)$ is the grey value (altitude) of the pixel u. The lower slope is defined as the maximal slope connecting u to any of its neighbors of a lower altitude:

$$LS(u) = max(\, h(u) - h(v)\,) \,|\, v \in \Gamma(u),$$

with $\Gamma(u)$ the set of neighbors v with $h(v) < h(u)$. If $\Gamma(u)$ has more than one element, $\mathscr{N}^F(u)$ represents an arbitrary element of that set.

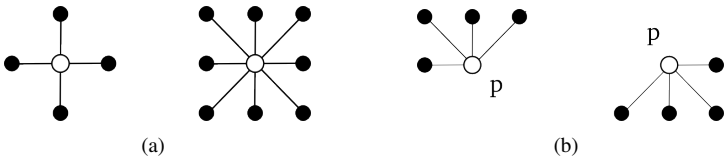


Figure 2.5: (a) Pixel 4– and 8–connectivity, (b) backward $N_G^+(p)$ and forward $N_G^-(p)$ neighborhood of a pixel p with 8–connectivity.

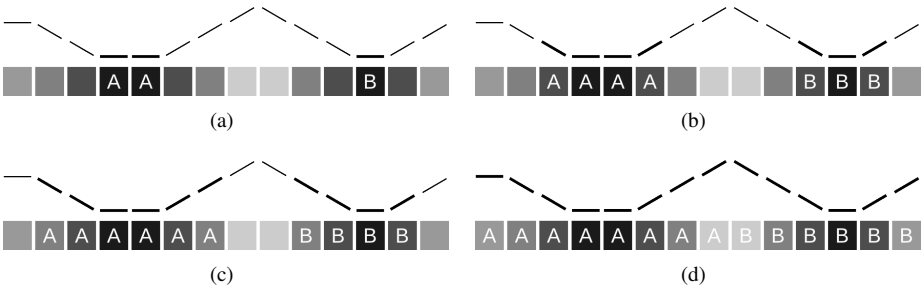


Figure 2.6: Watershed based on Hill-Climbing algorithm. Example of 1D image represented as a terrain by dashed lines and as grey values by the squares: (a) detecting and labelling all minima in the image with unique labels ("A" and "B"), (b) and (c) continues propagating the labels upwards, climbing up the hill, (d) result of the segmentation in two regions.

A plateau is a connected subgraph $P = (V_P, A_P) \subseteq G$, where $\forall u \in V_P, h(u) = c$, and c is the altitude of the plateau. Thus, a plateau is a region of constant grey value within the image. The set $\mathcal{N}^{=}(u)$ denotes the pixels $v \in \mathcal{N}(u)$ with $h(u) = h(v)$. A connected component of a graph G is a subgraph P in which pixels are connected to each other by paths. A plateau P is a level component of the image, considered as a valued graph, i.e., a connected component of pixels of constant grey value h [142].

Finally, the lower border of a plateau P is defined as:

$$\partial_P^- = \{\, u \in V_P \mid \exists v \in \mathcal{N}(u), h(v) < h(u) \,\}.$$

If $\partial_P^- = \emptyset$ the plateau is called a minimum plateau. All the pixels within a minimum plateau are also minimum. In contrast, if $\partial_P^- \neq \emptyset$ the plateau is called a non-minimum plateau.

Different algorithms have been implemented to compute the watershed transform using sequential structures as queues or graphs to simulate the flooding process [142]. Although various implementations can be found in the literature, in this thesis we follow the Hill-Climbing algorithm based on the topographical distance by Meyer [111]. This algorithm starts by detecting and labelling all minima in the image with unique labels, as illustrated in Figure 2.6(a). The process continues by propagating the labels upwards, climbing up the hill, following the path defined by the lower slope of each pixel, Figure 2.6(b) and Figure 2.6(c). The result of the segmentation, as shown in Figure 2.6(d), is a set of regions, each one represented by a catchment basin, with their own label. At the end all the pixels belong to a region and the watershed lines are the limits between these regions.

Problems arise for digital images with plateaus, as it is not possible to know a priori whether a plateau is minimum or non-minimum, so an additional processing is required. The most common solution is to preprocess the image by calculating its lower complete image [111], where each pixel has at least one neighbor with a lower value, except those pixels which are minima. Another alternative is to calculate the distances of an inner pixel to the lower border of the plateau during the watershed processing, which is the strategy selected in this work. The pseudocode of the watershed transform based on the Hill-Climbing algorithm is given in Section 3.2.2 where its implementation based on cellular automata is explained in detail.

2.4 Cellular automata

Cellular Automata (CA) constitute a computing model that has been extensively used for artificial life [51], pattern recognition [37] or image processing [143]. The popularity of CA is mainly due to the simplicity of modelling complex problems with the help of local information only. CA are composed of a set of cells arranged into a regular grid of one, two or three dimensions originally proposed by John Von Neumann [116] as formal models of self-reproducing organisms. In the case of two dimensions, each cell is connected to its four or eight adjacent neighbors, depending on the connectivity, as shown in Fig. 2.7. The most widely used connectivity is the Von Neumann neighborhood, considering the orthogonal neighbors, left, right, up and down. The Moore neighborhood, where the eight neighbors surrounding a cell are connected is a variation also used for connecting cells of the automaton.

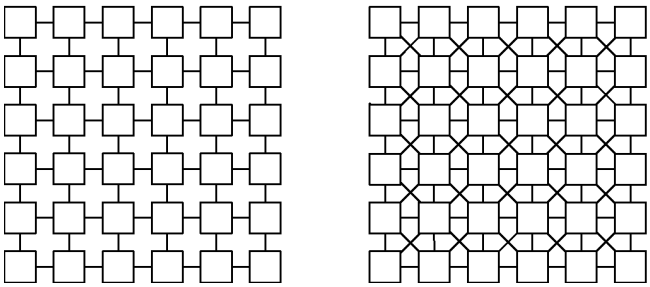


Figure 2.7: Cells arranged in a regular grid of two dimensions. Cells are connected to four (left) and eight (right) adjacent neighbors.

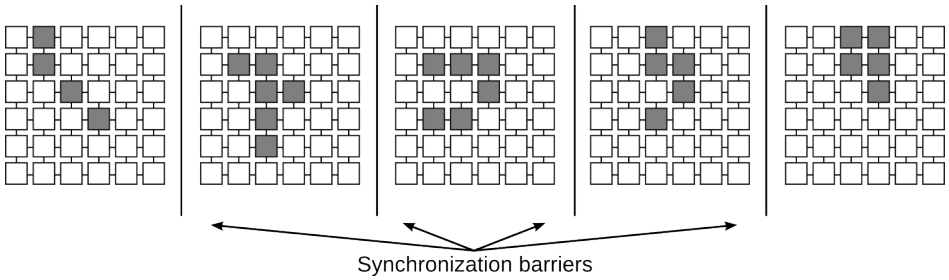


Figure 2.8: Synchronous updating process in a cellular automaton. The updates of the cells are performed synchronously and in discrete time steps. Grey squares represent cells whose state has changed.

Cells in a cellular automaton can be in one of a finite number of possible states. Each cell changes its state depending on the current state and the state of its neighbors. This requires a strict order for updating the automaton, where a cell cannot be updated until all other cells have also been updated. The updates of the cells are usually performed synchronously and in discrete time steps. Figure 2.8 shows an example of a 6×6 automaton, using 8-connectivity, which is updated synchronously. As illustrated in the figure, the updating process takes places in discrete time steps. The cells that are updated in this example are shaded in grey.

If the updates of the cells are not required to take place synchronously, but each one can be updated to its next state an unbounded number of times without synchronization, then we have an asynchronous automaton [115]. In this case, the grid can be partitioned into different regions which can be updated independently. It is possible to ignore the synchronization points associated to the evolution of the automaton, resulting in a so-called asynchronous automaton. In this case, however, the correctness and convergence of the algorithm could be severely affected [4]. In order to efficiently develop asynchronous computing schemes, it is important to investigate the non-deterministic and probabilistic behavior associated to such schemes [2].

Two dimensional CA can be directly mapped into 2D images where each cell in the automaton corresponds to a pixel in the image. The connectivity used in the automaton is denoted in the image by $\mathscr{N}(u)$. The same can be applied for 3D cellular automata and 3D images by modifying the connectivity to take into account the third dimension.

(a) (b) (c)

Figure 2.9: (a) Original image, (b) erosion (shrinks brighter objects), and (c) dilation (expands brighter objects). The SE was a square of 5×5 pixels.

2.5 Mathematical morphology

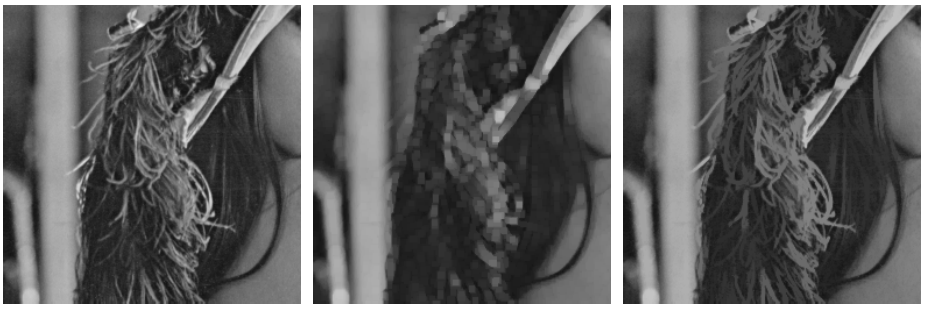
In this section we present an overview of Mathematical Morphology (MM) operators. In particular, we describe morphological operators based on the geodesic reconstruction, which are tools defined in the MM framework [152].

The Mathematical Morphology (MM) is a theory for analyzing and processing spatial structures from images [149]. The techniques are based on two basic operators, erosion (ε) and dilation (δ), from which a set of advanced analysis tools is constructed.

Erosion and dilation transform an image I using a structuring element (SE), giving as output for each pixel p the infimum ($\bigwedge$) or supremum ($\bigvee$), respectively, of the intensity values of the set of pixels included by the SE when it is centered on p. Generally speaking, the erosion operator shrinks objects that are brighter than their surroundings, whereas the dilation operator expands them. Figure 2.9 shows the erosion and dilation of the image of Lena using a square SE of 5×5 pixels.

2.5.1 Opening and closing by reconstruction

The dilation of an eroded image is known as opening, $\gamma(I)$. Conversely, the erosion of a dilated image is known as closing, $\phi(I)$. The opening operator flattens bright objects of an image if the SE fits within the objects, and the closing operator has the opposite effect. Therefore, opening and closing are used to extract information related to the shape and size of the objects. However, opening and closing are not connected filters; i.e., these operators



(a) (b) (c)

Figure 2.10: A subset of 256×256 pixels of the image of Lena. (a) erosion by a SE of 5×5 pixels, (b) opening, and (c) opening by reconstruction.

do not preserve edges between objects as they work on pixels rather than image structures. For instance, adjacent regions can be merged into one, and thus this biases the analysis of the spatial distribution.

It is possible to construct opening and closing operators based on the geodesic reconstruction to completely preserve or remove the spatial structures of an image if the SE fits (or not) within the objects [149]. The geodesic dilation of an image I (the mask) from J (the marker) is defined as:

$$\delta_I^{(1)}(J) = \delta^{(1)}(J) \wedge I, \tag{2.11}$$

where $\delta^{(1)}$ denotes the elementary dilation and $\wedge$ the point-wise minimum.

The opening by reconstruction $\gamma_r^{(n)}(I)$ of an image I is defined as the reconstruction by dilation of I from the erosion with a SE of size n of I. First, the image is transformed by an erosion $\varepsilon^n(I)$ creating a marker image J. Then, reconstruction by dilation R_I^{δ} is an iterative process that applies geodesic dilation on the marker image until stability ($\delta_I^{(n)} = \delta_I^{(n+1)}$):

$$R_I^{\delta}(J) = \delta_I^{(n)}(J) = \delta_I^{(1)}\left[\delta_I^{(n-1)}(J)\right]. \tag{2.12}$$

The reconstruction permits the full retrieval of all those structures that were not completely removed by the erosion. Thus, the morphological reconstruction needs several iterations before stability is attained. Figure 2.10 shows the result of opening and opening by reconstruction over a subset of 256×256 pixels of the image of Lena. It can be observed in Figure 2.10(b) that the edges in the hair or in the feathers of the hat are not preserved in the opening. How-

ever, in Figure 2.10(c) the structures of the hair and feathers that were not completely removed by the erosion are successfully reconstructed by opening by reconstruction.

By duality, the closing by reconstruction $\phi_r^{(n)}(I)$ of an image I is defined as the reconstruction by erosion of I from the dilation with a SE of size n of I.

2.5.2 Attribute filtering

Often, a classic image analysis preprocessing problem consists of filtering out small light (or dark) particles from greyscale images without damaging the remaining structures [170]. Opening and closing by reconstruction are good operators for this task. However, these operators only extract information related to the size of the objects. Therefore, if the structures to be preserved are elongated objects, they can be completely removed.

Morphological attribute filters are connected operators that process an image according to a criterion [25], such as the area or the standard deviation of the pixel intensity, removing plateaus (connected components) that do not satisfy a given criterion. Morphological attribute filters are connected filters. The following definitions are for binary images, although generalization to grey-scale images can be achieved through threshold decomposition [76].

The connected opening Γ_x of a set X at a point x is defined as:

$$\Gamma_x(X) = \begin{cases} X & \text{if } x \in X, \\ \emptyset & \text{otherwise.} \end{cases} \tag{2.13}$$

Figure 2.11 shows an example of a connected opening on a binary image with a set X consisting of three connected components, named C_1^1, C_1^2, and C_1^3. The i-th component is indicated by the superscript, and the background (0) or foreground (1) is indicated by the subscript. The opening Γ_x preserves only that connected component in X which contains the pixel x, as illustrated in Figure 2.11(b).

A trivial opening Γ_T uses an increasing criterion T to filter connected components and it is defined as follows [150]:

$$\Gamma_T(C) = \begin{cases} C & \text{if } C \text{ satisfies criterion } T, \\ \emptyset & \text{otherwise.} \end{cases} \tag{2.14}$$

The trivial opening preserves the connected components for which the increasing criterion T holds. For example, the following is an increasing criterion: must have an area of λ pixels or more [25].

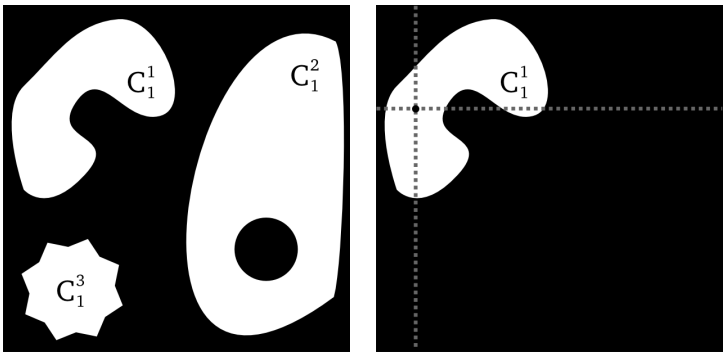


Figure 2.11: Example of a connected opening. (a) Binary image with three connected components named C_1^1, C_1^2, and C_1^3, and (b) the connected opening Γ_x indicated by the intersection of the dot lines.

The attribute opening Γ^T of a set X combines the connected opening (2.13) and the trivial opening (2.14) and is defined as follows:

$$\Gamma^T(X) = \bigcup_{x \in X} \Gamma_T(\Gamma_x(X)). \tag{2.15}$$

The attribute opening is the union of all connected components of X which meet the criterion T. The attribute opening can be explained by using a tree representation of the image, as illustrated in Figure 2.12(b). The root of the tree represents the background of the image, and the leaves corresponds to the connected components in the foreground. The attribute filtering,

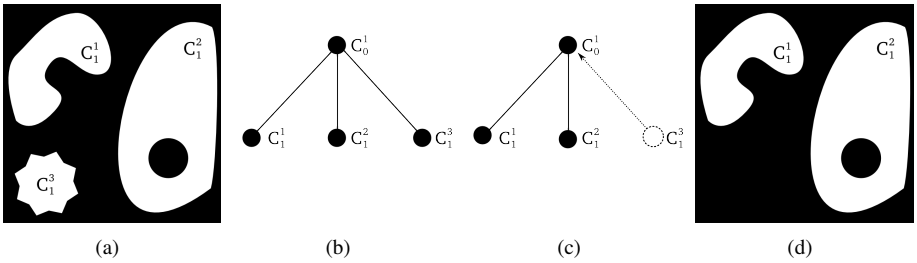


Figure 2.12: Example of a attribute opening. (a) Binary image with three connected components named C_1^1, C_1^2, and C_1^3, (b) tree representation of the image, (c) attribute filtering removing a node of the tree, and (d) the final result.

for example the area, consists in analyzing if each connected component C_1^k has an area of λ pixels or more. In the example given in Figure 2.12, the area of each component is $A(C_1^1) = 49261$, $A(C_1^2) = 22008$, $A(C_1^3) = 111224$. Thus, for a threshold $\lambda = 22009$, the component C_1^3 must be removed as its area is smaller than λ. As can be seen in Figure 2.12(c), the tree links represent the pixels' migration (toward the parent) when a node is removed [144]. As a result, the image illustrated in Figure 2.12(d) keeps the remaining connected components.

2.6 Wavelet transform

Wavelets are mathematical tools for signal processing analysis in the time-frequency domain [44]. A signal can be represented by wavelets from basis functions $\psi(t)$, called mother wavelets:

$$\psi_{s,l}(x) = \frac{1}{\sqrt{s}}\,\psi\left(\frac{x-l}{s}\right)\,, \tag{2.16}$$

where the wavelet ψ is scaled by s and translated by l to generate a family of discrete wavelets, such as a Daubechies wavelet family [69]. Wavelet transforms may be either continuous or discrete. The discrete wavelet transform (DWT) of a signal f is given by:

$$Wf(s,l) = \sum f(x)\;\psi_{s,l}(x)\;. \tag{2.17}$$

When a signal is represented using a DWT, (2.17) can be calculated using filters [165]. The signal is approximated using low-pass filters while high-pass filters are used to bring out details. This wavelet decomposition is calculated as:

$$a(n) = \sum_k h(k)\;f(x-k) = f \star h\,, \tag{2.18a}$$

$$d(n) = \sum_k g(k)\;f(x-k) = f \star g\,, \tag{2.18b}$$

where h, g are the low-pass and high-pass filters of k coefficients, respectively, and f a discrete signal of size n. The decomposition a and d are called approximation coefficients and wavelet details, respectively. Equations (2.18a), (2.18b) are the circular discrete convolution ($\star$) of f with the filters h and g, respectively. As illustrated in Figure 2.13(a), a signal $x(n)$ is passed through each filter, denoted by h and g, where a circular discrete convolution of the signal and the filters is calculated. This wavelet decomposition produces $a(n)$, the approximation coefficients, and $d(n)$, the detail coefficients which are down-sampled at each level to produce half the coefficients, represented by the symbol $\downarrow 2$ in Figure 2.13.

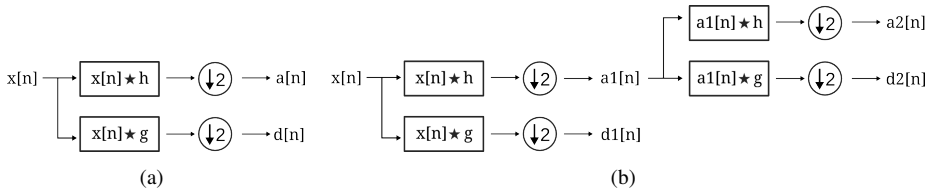


Figure 2.13: 1D-DWT: (a) low-pass h and high-pass g filter diagram, (b) Mallat's tree scheme for 2 levels of decomposition. ($\star$ stands for circular convolution).

Preserving low and high frequency features is an inherent property of the wavelet transforms, as is the ability to compress data. The DWT can be applied recursively to increase the decomposition of the signal, as shown in Figure 2.13(b). The signal is down-sampled at each level to produce half the coefficients. This is called the Mallat algorithm or Mallat-tree decomposition [106]. As the number of wavelet decomposition levels increases, the signal becomes smoother. Figure 2.14 illustrates a 3-level decomposition of a signal representing a pixel vector of 103 features. It is worth observing that a greater level of decomposition implies a smoothing of the original pixel vector. The size of the pixel vector is reduced by a factor of two, from 103 features in the original vector to 13 wavelet coefficients in the third level of decomposition.

Signal denoising is a common task performed by wavelets. This task is known as wavelet thresholding or shrinkage. The smallest high frequency subband coefficients, which are usually considered as noise, might be suppressed without substantially affecting the main features of the signal. These small wavelet coefficients can be removed (hard-thresholding) or attenuated (soft-thresholding). For soft thresholding the following nonlinear transform is used [47]:

$$\eta_t(y) = \mathrm{sgn}(y)(|y| - t)_+ , \tag{2.19}$$

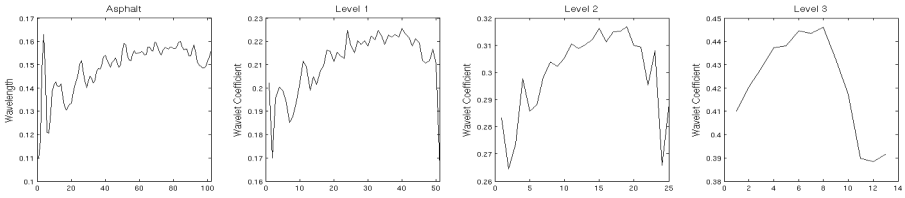


Figure 2.14: Example of 3 levels of decomposition of a signal representing a pixel vector of 103 features.

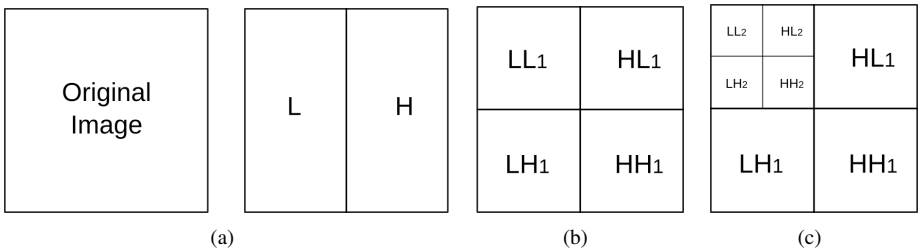


Figure 2.15: Separable 2D-DWT applied to the rows of the image (a) and then, to the columns (b), and the result of 2 levels of decomposition (c).

where t is the threshold and y the signal to be de-noised. The subscript $+$ in (2.19) indicates that the pixels which are greater than the threshold in absolute value, that is, $|y(x)| > t$, are attenuated, whereas the rest are removed.

By applying an inverse wavelet transform (IWT) after hard- or soft-thresholding, it is possible to reconstruct the original signal with less amount of noise [84]. In any case, the idea of shrinkage is to preserve only the details that are above a particular threshold.

The DWT is usually applied to one-dimensional signals, although it can be extended to two dimensions. In the case of separable 2D-DWT, the 1D-DWT is extended by applying the wavelet analysis separately to each dimension. This approach simplifies the mathematics and leads to a faster numerical algorithm [106]. In image processing, the two dimensions correspond to the spatial location $[x, y]$, matching the columns and the rows of the image, respectively. First, the 1D-DWT decomposition is applied to one of the two dimensions, for example, the rows of an image, resulting in two subbands L and H, with the approximation and detail coefficients of the original image. Figure 2.15(a) shows this step where an image is divided into the subband L and the subband H as a result of the convolution by rows with the low-pass and high-pass filters, respectively. Second, the 1D-DWT is applied to the columns of L and H, which results in four subbands LL, HL, LH, HH, corresponding to the low resolution approximation and three subbands of details, as shown in Figure 2.15(b). Like the 1D-DWT, the 2D-DWT can be applied recursively to increase the decomposition of the image, as represented in Figure 2.15(c).

The three subbands HL, LH, HH reveal features related to spatial orientations. Horizontal and vertical structures are highlighted in the LH and HL subbands, respectively, while diagonal features are represented in the third subband, that is HH. However, one disadvantage of the

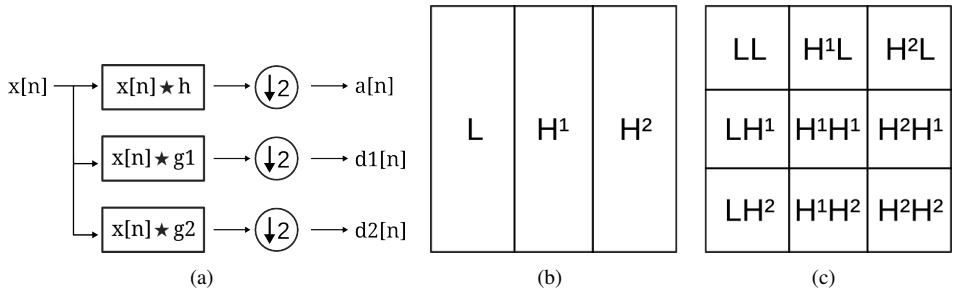


Figure 2.16: 2D Double-Density DWT: (a) low-pass and high-pass filters diagram, (b) separable transform applied to the rows of an image, (c) separable transform applied to the columns of (b). ($\star$ stands for circular convolution).

separable DWT for the application thereof in image processing is that the coefficients reveal only three spatial orientations [59], which results in a poor selectivity of features at different orientations, for example in a curve.

The Double-Density DWT presented in [148] is based on a single scaling function (low-pass filter) and two distinct wavelets (high-pass filters). Although this DWT still suffers from some lack of spatial orientation, with more wavelets than necessary, the Double-Density DWT outperforms the standard 2D-DWT in terms of denoising [148]. Figure 2.16(a) shows the filter decomposition of one dimensional signal $x(n)$ and the resulting three subbands $a(n)$, $d1(n)$ and $d2(n)$ corresponding to the convolution of $x(n)$ with the low-pass filter h and the two wavelets g_1 and g_2, respectively. As in the case of the standard 1D-DWT, the result is down-sampled ($\downarrow 2$) at each level to produce half the coefficients.

Applied to an image, the wavelet transform presented in [148] creates nine subbands, instead of four as the general case. The Double-Density DWT is applied to the rows of the image resulting in three subbands, L, H^1 and H^2, as illustrated in Figure 2.16(b), and then it is applied to the columns of L, H^1 and H^2, which results in nine subbands as shown in Figure 2.16(c). One of the subbands is the low resolution approximation (LL subband in Figure 2.16(c)), and the other eight subbands correspond to the wavelet details. The Mallat algorithm can be recursively applied with the Double-Density DWT as explained above for the 2D-DWT. For further details on wavelets, we refer the reader to [106].

2.7 Pixel-wise classification by SVM

Given an n-dimensional image, the idea of pixel-wise classification is to assign a name (class of interest) to each pixel based on the spectral similarities among the pixel vectors. The assignment of a class to each pixel requires a priori knowledge of the image (supervised classification) in the form of labelled pixels. Success in classification depends not only in the ability to categorize and/or discriminate objects, but also in the set of training samples used for the same. The supervised classification begins with a training phase where a set of training samples (usually obtained by manually labelling) are used for defining a model of the classes in the spectral space. A second classification phase assigns a class to each pixel based on the model created in the previous phase producing a classification or thematic map [146].

There are a variety of classifiers, such as linear discriminant analysis, maximum likelihood, random forests, nearest-neighbors, artificial neural networks and support vector machines [146, 60]. An exhaustive evaluation of classifiers belonging to a wide collection of families over the whole UCI machine learning classification database[2], was presented in [60]. The classifiers most likely to be the best were the random forest and the SVM.

In remote sensing, the supervised classifier SVM has been generally recognized as the one that offers good results in terms of accuracy even, when the number of training samples is small [72, 55]. This is an important property in remote sensing classification as the number of training samples available in hyperspectral images is usually small. In this work, the SVM is used to classify hyperspectral images. In the following, we briefly describe the general mathematical formulation of SVM for binary classification problems.

The standard two-class SVM method consists in finding the optimal hyperplane which separates two training samples belonging to different classes, maximizing the distance between the closest points of each class. Figure 2.17(a) shows an example of two linearly separable classes $[+1,-1]$ and the maximum margin between them.

Let us consider a set of N training points $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$ with the training vectors $\mathbf{x}_i \in \Re^n$ and the corresponding target $y_i \in \{\pm 1\}$. And let the set N be linearly separable into two classes. The hyperplane can be estimated as:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) > 1 \quad \text{with} \quad i = 1,2,\ldots,N, \tag{2.20}$$

[2] `http://archive.ics.uci.edu/ml/`.

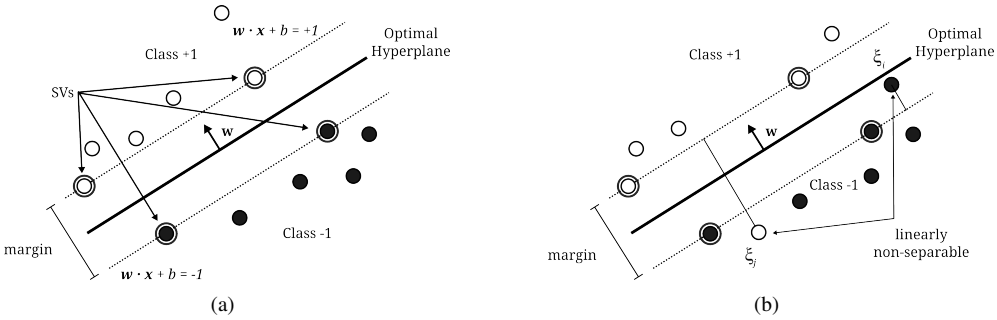


Figure 2.17: SVM classification: (a) optimal hyperplane separating two linearly separable classes $[+1,-1]$, (b) a non-linearly separable case by SVM.

where the hyperplane is defined by the vector $\mathbf{w} \in \Re^n$ and the bias $b \in \Re$. The training samples that maximize the distance are called support vectors (SVs), illustrated in Figure 2.17(a) with double-circle.

Maximizing the distance of samples to the optimal hyperplane is equivalent to minimizing the norm of $\mathbf{w}$. Therefore, the optimal hyperplane can be obtained by minimizing:

$$Q(\mathbf{w}) = \min\left[\frac{||\mathbf{w}||^2}{2}\right], \quad \text{subject to (2.20).} \tag{2.21}$$

The assumption of linear separability means that there exist $\mathbf{w}$ and b that satisfy (2.20). When the data are linearly inseparable, nonnegative slack variables ξ_i are introduced in (2.20) to deal with misclassified samples [1]:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) > 1 - \xi_i, \quad \text{with} \quad \xi_i \geq 0,\ i = 1,2,\ldots,N. \tag{2.22}$$

Figure 2.17(b) shows an example of a non-linearly separable case by SVM. Now, the optimization problem can be described as:

$$Q(\mathbf{w},b,\xi_i) = \min\left[\frac{||\mathbf{w}||^2}{2} + C\sum_{i=1}^{N}\xi_i\right], \quad \text{subject to (2.22),} \tag{2.23}$$

where the constant C is a regularization parameter that controls the amount of penalty (maximizes the margin and minimizes the error). This optimization is usually solved by a quadratic programming problem [1] in the training phase.

The classification phase is performed by computing the sign of the following decision function:

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b, \tag{2.24}$$

where α_i are the nonzero Lagrange multipliers, S is the subset of training samples $\mathbf{x}_i$, and $\mathbf{x}$ is the unknown sample. The parameters (α_i, S, b) are found during the training phase. For each $\mathbf{x}$, the corresponding class is $+1$ if $D(\mathbf{x}) > 0$ and -1 otherwise.

Although the solution may be determined optimally by the SVM when the training samples are not linearly separable, the SVM approach uses kernel tricks to map the non-linearly separable data into a higher dimensional space, in order to enhance linear separability between the two classes. Several types of kernels, such as linear, polynomial, splines or Radial Basis Function (RBF) kernels can be used in SVM classifiers. We refer the reader to [1, 72, 27] for further details on SVM and kernel tricks.

For hyperspectral image classification, the Gaussian RBF has been widely used. The discriminant function (2.24) can be expressed using the RBF kernel as:

$$D(x) = \sum_{i \in S} \alpha_i y_i \exp\Big(-\gamma ||\mathbf{x}_i - \mathbf{x}||^2 \Big) + b, \tag{2.25}$$

where $\exp(-\gamma||\mathbf{x_i} - \mathbf{x}||^2)$ is the RBF kernel and γ is a positive parameter controlling the width of the kernel.

2.7.1 5-fold cross validation

The SVM is mainly a nonparametric method, although some parameters need to be tuned before the optimization. In the RBF kernel case, there are two parameters: C, which controls the penalty term, and γ, which is the width of the kernel. The best parameters are usually found by k–fold cross-validation, where several parameters are tested, for example by a search in a range of values (grid search).

Given a set of training samples, the k-fold cross validation divides the total number of samples into k partitions. One partition is used to validate the accuracy of the classification, and the remaining $k-1$ partitions are used as training data. This validation is repeated k times, using each partition once as a test. The final classification accuracy is determined by the average with the k classification results. In this work we have used 5–fold cross validation for tuning the SVM parameters.

2.7.2 Multi-class SVM classification

The standard SVM classifier is designed for two-class problems. When the data have $K > 2$ classes a method for solving the multiclass problem must be defined. These methods may be of the type one-against-all (OAA), one-against-one (OAO) and all-at-once, where a set of binary classifiers is combined. In the OAA approach, the K class problem is converted into K binary classifiers where the ith classifier separates the class i from the remaining classes. The OAO multiclass approach creates $T = K(K-1)/2$ binary classifiers on each pair of classes, where K is the number of classes. The final classification for each point is given by the highest number of votes obtained for a class in the T classifiers. Hsu and Lin [80] found that the OAO approach is more suitable for practical use than the other methods, mainly because the training time is shorter. This is the approach used in all the SVM classifications carried out in this work.

2.7.3 LIBSVM: the facto library for SVM

In this section we present the facto library for SVM (LIBSVM) and the principal interfaces and extensions that are based on this library. LIBSVM [35] is a library for Support Vector Machine (SVM) written in C, widely used in machine learning. The library implements the One-Against-One approach for multiclass classification. It is currently one of the most widely used SVM software and it has been extended to third-party software such as Weka[3], R[4], Octave and Matlab[5].

Owing to its popularity, it has been also implemented in other languages such as Java[5], Python[5] and C#[6], as well as in other architectures, such as Cell processors and GPUs.

While efficient algorithms for training SVM are available, dealing with large datasets makes training and classification a computationally challenging problem. In [109] the training phase implemented in the LIBSVM is speeded up by parallel computation in Cell processor architectures. LIBSVM has also been accelerated with GPUs using CUDA [8]. This GPU-accelerated LIBSVM implementation is a modification of the original LIBSVM that exploits CUDA with the same functionality and interface of LIBSVM. A comprehensive list of other SVM implementations and wrappers can be found in [60]. In this thesis we have implemented

[3] `https://weka.wikispaces.com/LibSVM/`
[4] `http://cran.r-project.org/web/packages/e1071/`
[5] Matlab, Octave, Java and Python implementations are included in the LIBSVM package.
[6] `https://github.com/ccerhan/LibSVMsharp/`

a new GPU proposal (GPUSVM) for the classification stage, and the LIBSVM [35] has been used as a basis for evaluating the proposed schemes in CPU.

2.8 Parallel programming models

This section describes the OpenMP and CUDA programming models. First, we will describe OpenMP, an API for writing CPU multi-threaded applications on shared memory architectures. Then, we introduce the Compute Unified Device Architecture (CUDA) developed by NVIDIA, a parallel computing platform and a programming model that leverages the high computational throughput of NVIDIA GPUs. We will highlight the major differences found in the different GPU architectures used in this work.

2.8.1 OpenMP

OpenMP is the standard Application Program Interface (API) for multi-threaded parallel programming on shared memory architectures [120]. Communication and coordination between threads is expressed through read/write instructions of shared variables and other synchronization mechanisms. It comprises compiler directives, library routines and environment variables and is based on a fork-join model, as illustrated in Figure 2.18 where a master thread creates a team of threads that work together in a Single Program Multiple Data (SPMD) way (different cores execute different threads operating on different data).

In shared memory architectures, OpenMP threads access the same global memory where data can be shared among them or can be private for each one. From a programming perspective, data transfer for each thread is transparent and synchronization is mostly implicit (see Figure 2.18). When a thread enters a parallel region, it becomes the master, creates a thread team and forks the execution of the code among the threads and itself. At the end of the parallel region, the threads join together and the master resumes the execution of the sequential code.

Different types of worksharing constructs can be used to share the work of a parallel region among the threads [33]. The loop construct distributes the iterations of one or more nested loops into chunks, among the threads in the team. By default, there is an implicit barrier at the end of a loop construct. The way the iterations are split depends on the schedule used in the loop construct [33]. On the other hand, the single construct assigns the work on only one of the threads in the team. The remaining threads wait until the end of the single construct

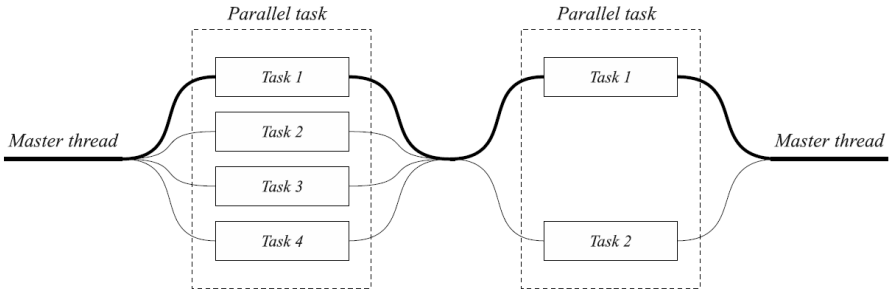


Figure 2.18: OpenMP fork-join parallel model. A master thread creates a team of threads (fork) to work in parallel. When all the threads end their task, they are joined together.

owing to an implicit barrier. This type of construct is also known as non-iterative worksharing construct. Other types of worksharing constructs are available.

Although there are implicit communications between threads through access to shared variables or the implicit synchronization at the end of parallel regions and worksharing constructs, explicit synchronization mechanisms for mutual exclusion are also available in OpenMP. These are critical or atomic directives, lock routines, and event synchronization directives.

OpenMP is not responsible for the management of the memory hierarchy but certain issues regarding cache memory management should be borne in mind. There are two factors that determine whether a loop schedule is efficient: data locality and workload balancing among iterations. The best schedule that we can choose when there are data locality and a good workload balance is static with a chunk size of $q = n/p$, where n is the number of iterations and p the number of threads. In other cases, dynamic or guided schedules may be adequate.

When a cache line, shared among different processors, is invalidated as a consequence of different processors writing in different locations of the line, false sharing occurs. False sharing must be avoided as it decreases performance due to cache trashing. One way to avoid this is to divide the data to be accessed by different processors into pieces whose size is multiple of the cache line size. A good practice for improving cache performance is to choose a schedule with a chunk size that minimizes the requests of new chunks and that is a multiple of a cache line size.

2.8.2 CUDA

The NVIDIA Compute Unified Device Architecture (CUDA) is a parallel computing platform and a programming model. The GPU provides massively parallel processing capabilities with a high computational throughput due to their large number of cores. In particular, CUDA is organized into a set of Streaming Multiprocessors (SMs), each one containing many Scalar Processor (SP) with many cores inside. The NVIDIA's G80 series, introduced in November 2006, had a total of 128 cores in 16 SMs, each one with 8 SPs, as summarized in Figure 2.19(a). This architecture uses a Single Instruction, Multiple Thread (SIMT) parallel programming model. SIMT is the terminology used by NVIDIA to define a hybrid model between vector processing (SIMD) and hardware threading [40]. This approach makes it possible to write single instructions which will be simultaneously executed from multiple threads. The general specification, as well as the number of resources available on the GPU depends on its compute capability (CC), which is represented by a version number, 1.x, 2.x, 3.x and 5.x [119]. The first CUDA capable hardware, codenamed tesla, has a CC of 1.x and defines the main features of the architecture, such as the organization into a set of SMs, SPs and the different types of device memory. A full list of the differences among each compute capability can be found in [119, 40].

The basic compute unit in CUDA is the SM which has fixed and limited resources, such as a set of registers and on-chip memory that are shared among the cores within the same SM. The GPU can manage and schedule thousands of threads in hardware simultaneously, avoiding high thread management overheads. A large number of threads are required to make full use of the GPU computing capabilities. The threads execute the same instruction on different data by grouping 32 threads as the minimum size of collaborative unit, called a warp. The warp size is implementation defined and it is related to shared memory organization, data access patterns and data flow control [90].

The threads are arranged in a 1D, 2D or 3D grid of blocks which are scheduled to any of the available SMs. Figure 2.19(a) shows a 2D grid of 8 blocks each one with 4×4 threads. Each thread has a unique `ID` that identifies which block the thread belongs to, and the thread's position within the block. The grid of blocks is usually designed to match a thread with a value among the different data that will be processed on the GPU.

One of the most important aspects of the architecture is the memory hierarchy as it plays a key role in performance. As shown in Figure 2.19(a), the G80 architecture (CC 1.0) has a global memory, a texture memory and a constant memory which are available for all the

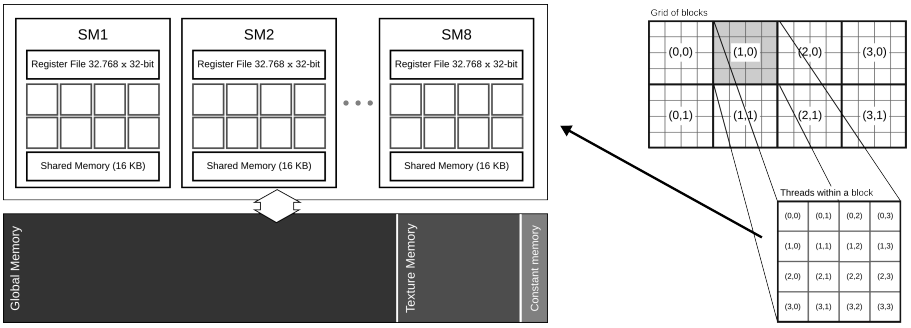


Figure 2.19: CUDA. (a) Overview of G80 architecture, (b) grid of blocks and block of threads scheduled to any of the available SMs.

threads at any Streaming Multiprocessor. There was no a L1 / L2 cache hierarchy for caching global memory data on the early GPU architectures. Therefore, the device memory was designed for specific purposes.

The global memory is the main device memory on the GPU, and has the slowest access time. However, the number of memory accesses to global memory can be reduced for certain memory access patterns by a technique known as coalescing. If threads within the warp[7] request consecutive and aligned values in memory, the device coalesces the global memory transactions (load/store) into as few transactions as possible (one in the best case) [119].

Data allocated in the texture memory space will be automatically cached. There are only 8 KB of cache per SM and it is optimized for 2D spatial locality. This can improve performance when threads access values in some regular spatial neighborhood. Texture memory is generally used for visualization or as input buffers owing to data caching.

The constant memory is another possibility of caching accesses (64 KB in total) to global memory. A single read from constant memory can be broadcast to a warp, if the same value is accessed by all the threads within the warp. As a result, reading from this memory costs one memory transaction from global memory the first time data are accessed, and one read from the constant cache otherwise.

The on-chip memory, known as shared memory, enables extremely rapid load/store accesses to the data but within the lifetime of the block. There are 16 KB of shared memory

[7] On hardware of compute capability 1.x, memory transactions are coalesced within half warp.

within each SM but it is only "shared" for the threads of the same block. Therefore, sharing data among different blocks is not possible through this memory and becomes a challenge when programming for the GPU. It is up to the programmer loading data from global (texture) memory to shared memory. The main feature of the shared memory is reusing data within a block, sharing data among the threads of the same block. This way, the shared memory can be managed as an explicit cache defined by the programmer. Although the amount of shared memory per SM is only 16 KB, the effective use of this memory can lead to speedups of 7× compared to a naive implementation in global memory [40].

Each thread on the GPU has its own local memory and a set of registers where the computation takes place. The maximum number of registers that can be used by a thread is limited by the CC of the GPU, as well as the number of threads configured per block [119].

Different aspects must be taken into consideration to efficiently exploit all the memory on the GPU [90]. For example: (1) data transfer between the CPU and the GPU should be minimized; (2) the memory access pattern must be coalesced to consecutive memory locations; (3) data loaded in shared memory should be reused to reduce load/store accesses to the global memory; (4) the number of threads per block must be optimized to run the maximum concurrent threads allowed in each SM; and (5) minimizing the global synchronization among blocks leads to a reduction in the execution time of the program. Paying attention to these aspects is vital for GPU programming.

In Fermi (CC 2.x), Kepler (CC 3.x) and Maxwell (CC 5.x) architectures, there is also an L1 and L2 cache hierarchy, as shown in Figure 2.20 for Kepler. The accesses to global memory are cached in this memory hierarchy. The L2 cache is fully available for all the threads and the L1 cache only for the threads running in the same Streaming Multiprocessor (in Kepler, the SMs are called SMXs). Note that the L1 and L2 caches are managed by the GPU, unlike the shared memory, but the programmer can take advantage of this cache hierarchy by exploiting the memory access pattern when reading data from the global memory (the same when writing data to the global memory). The L1 cache is placed in the same chip as the shared memory, so the size of the on-chip memory per SMX (see Figure 2.20) is split between these two kind of memories.

In the following, we will cover the major characteristics found in the CC 2.x and 3.x corresponding to the GPUs used in this work. The main differences are summarized in Table 2.1. The main changes introduced in CC 2.x are: (i) extension in size of the shared memory from 16 KB up to 48 KB per SM, (ii) configurable 16 KB or 48 KB of L1 cache on each SM,

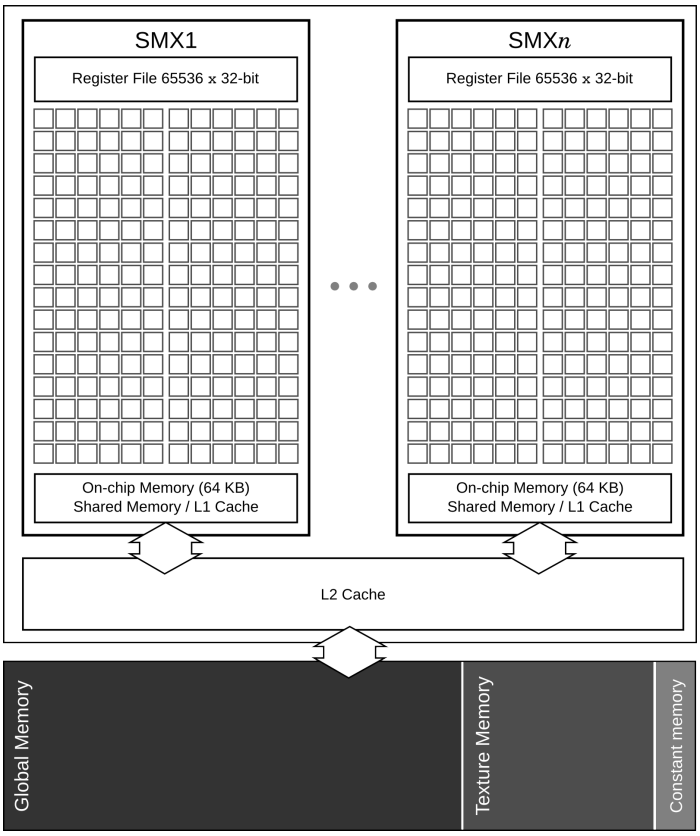


Figure 2.20: Overview of the Kepler architecture incorporating a L1 and L2 cache hierarchy.

and (iii) shared L2 cache for all SMs. The CC 3.x mainly increases the number of resources available per thread, SP and SMX compared to CC 2.x. The CC 3.5 has a read-only cache of 48 KB shared by every three SMXs.

In all the architectures, threads within the same block can be synchronized, for example to communicate intermediate results in the shared memory as part of a parallel computation. However, it is not possible to synchronize threads among different blocks. Owing to this restriction, the communication among all the threads must be through the global memory and this becomes another challenge when a thread needs data which have been generated outside its block.

	G80	Fermi GF110	Kepler GK104	Kepler GK110
Compute capability	1.0	2.0	3.0	3.5
Max SMs	8	16	8	15
CUDA cores per SM	8	32	192	192*
Max threads per block	512	1024	1024	1024
Max total threads per SM	768	1536	2048	2048
Max number of blocks per SM	8	8	16	16
Max Shared Mem	16 kB	48 kB	48 KB	48 KB
Max L1 cache	N/A	48 kB	48 KB	48 KB
Max L2 cache	N/A	768 KB	512 KB	1536 KB
Max Memory	1536 MB	1536 MB	2048 MB	6144 MB

** Plus an additional 64 dual-precision units per SM.*

Table 2.1: Max GPU resources defined by the compute capability.

A CUDA program is called a kernel and it is executed on the GPU by thousands of threads in parallel. A kernel is configured by specifying the grid size, the threads within the block, and the amount of shared memory used by a block. When a kernel is called, the computation on the GPU takes place. The blocks, which are arranged into a grid, are scheduled to any of the available cores enabling automatic scalability for future architectures.

2.9 Experimental setup

In this thesis we have taken into account the efficient computation on GPU of the techniques under study. In recent years, GPUs (and the CUDA API) have evolved so rapidly that we have evaluated our work on different architectures (and CUDA versions). In this section we present the main specifications of the CPU and GPUs used in our experiments. The experiments are executed under Linux using the gcc compiler version 4.6.3 for the OpenMP implementations, and the nvcc compiler for the case of the CUDA implementations, respectively, with full optimization flags (-O3) in both cases. We describe the performance measures to evaluate the classification results in terms of accuracy, as well as in terms of execution time and speedup. Different datasets have been used in this work and will be also detailed in this section. The names and number of known samples (reference map) of the hyperspectral images are summarized at the end of this section.

2.9.1 Hardware used in the experiments

In this work we have used an Intel quad-core i7-860 microprocessor (8MB Cache, 2.80 GHz) and 8 GB of RAM as the base architecture for comparison. Each core has a separated L1 cache for instructions and data, and a unified L2 cache. The unified L3 cache is common to all the cores. The main characteristics of the memory hierarchy are described in detail in [82]. Table 2.2 summarizes the main specifications of this CPU.

We have used the NVIDIA GTX 580, GTX 680 and GTX TITAN devices. Table 2.3 shows the GPU model, the compute capability of the graphic cards, available resources, and CUDA versions used in this thesis.

	# cores	Core clock (GHz)	RAM (GB)	L1 (KB)	L2 (KB)	L3 (KB)
Intel core i7-860	4	2.80	8	64/64	256	8192

Table 2.2: Main characteristics of the CPU used in this thesis.

	GTX 580	GTX 680	GTX TITAN
Compute capability	2.0	3.0	3.5
Streaming Multiprocessors	16	8	15
CUDA cores per SM	32	192	192*
Threads per block	1024	1024	1024
Total threads per SM	1536	2048	2048
Number of blocks per SM	8	16	16
Active warps per SM	48	64	64
Registers per SM	32768	65536	65536
Registers per thread*	63	63	63
Device Memory	1536 MB	2048 MB	6144 MB
Shared Memory	48 KB	48 KB	48 KB
L1 cache	48 KB	48 KB	48 KB
L2 cache	768 KB	512 KB	1536 KB
CUDA version	4.0	5.0	5.5
** Plus an additional dedicated zero register.*			

Table 2.3: GPU model, compute capability, resources and CUDA version of the graphic cards used in this thesis.

2.9.2 Performance measures

We now describe the performance measures used for evaluating the spectral-spatial classification schemes proposed in this thesis in terms of accuracy, as well as in terms of execution time and speedup.

Assessing the Accuracy

The classification process consists in labelling pixel vectors from a hyperspectral image creating a final classification map. The accuracy is based on a reference map, also known as a ground-truth where the set of training samples are taken for the SVM training phase. In order to quantitatively evaluate the classification accuracies, we have used the Overall Accuracy (OA), the class-specific accuracy (CS), the Average Accuracy (AA) and the Kappa coefficient of agreement (k) [167] as the criteria for assessing the accuracy of the spectral-spatial classification schemes.

Let C_i represent the class i, C_{ij} be the number of pixels classified to the class j and referenced as the class i, and K the number of classes.

– The OA is the percentage of correctly classified pixels:

$$OA = \frac{\sum_i^K C_{ii}}{\sum_{ij}^K C_{ij}} \times 100\% \,. \tag{2.26}$$

– The CS (or producer's accuracy) is the percentage of correctly classified pixels for a given class i:

$$CS_i = \frac{C_{ii}}{\sum_j^K C_{ij}} \times 100\% \,. \tag{2.27}$$

– The AA is the mean of the class-specific accuracy for all the classes:

$$AA = \frac{\sum_i^K CS_i}{K} \times 100\% \,. \tag{2.28}$$

– k is the percentage of agreement corrected by the amount of agreement that could be expected due to chance alone, which ranges from 0 to 1. A value of k below 0.6 is interpreted as a moderate agreement, whereas a value above 0.8 can be understood as an almost perfect agreement [167].

These criteria are computed from a confusion matrix. In the field of supervised learning, a confusion matrix is a table, where each column represents the instances in a predicted class, while each row represents the instances in an actual class (reference map) [173]. Table 2.4

	Predicted class				
Reference data	Asphalt	Water	Trees	$\sum_j^K C_{ij}$	CS_i
Asphalt	28	1	1	30	93.3%
Water	14	15	1	30	50.0%
Trees	15	5	20	40	50.0%
$\sum_{ij}^K C_{ij}$	57	21	22	100	

Table 2.4: Confusion matrix for a problem with three classes and class-specific accuracy (CS).

shows an example of a confusion matrix for a problem with three classes: *Asphalt*, *Water* and *Trees*. The number of values correctly classified are represented in the main diagonal. The OA is 63% and the AA 64%. The class-specific accuracy is shown in the last column of the table. It can be observed from the values below the main diagonal of the matrix that the scheme is "confusing" the classes *Water* and *Trees* with the class *Asphalt*. In this example, 14 values of the class *Water* and 15 values of the class *Trees* were incorrectly classified as *Asphalt*. The k coefficient is only 0.454, so there is a moderate agreement in this result.

In order to obtain a reliable evaluation of the results, the accuracies are calculated excluding the samples used for training, that is, from the reference map, a group of samples is used for training the SVM and the remaining samples for testing the classification accuracy.

Speedup

The proposed GPU implementations are also evaluated in terms of execution times and speedups comparing to optimized CPU multi-threaded OpenMP implementations. The speedup is a metric for relative performance improvements and it is calculated as the ratio between the old execution time (base for comparison) and the new (improved) execution time:

$$S = \frac{T_{\text{old}}}{T_{\text{new}}}\,. \tag{2.29}$$

The execution time is measured in seconds. The times T_{old} and T_{new} are the wall clock times (elapsed time from the start to end of the computation) excluding the file I/O time for both, the CPU and the GPU. The reason to exclude the harddisk reading data is that the implementations are part of a scheme where the different stages are concatenated in a pipeline processing, where the output data in one stage are used as input in the following stage. Therefore, the data are kept in memory during the process. For GPU timing, the CPU–GPU data transfer times are included in the analysis as an associated overhead for using that architecture.

Occupancy

When launching a kernel, different configurations are possible, and based on these configurations as well as the hardware requirements by the kernel, the performance may vary. The limit in the hardware resources required to execute a kernel can be achieved in several ways. For example, if a kernel is configured to launch blocks of 512 threads, the maximum number of concurrent blocks per SM will be 3 in Fermi architecture, and 4 in Kepler, although the theoretical maximum number is 8 (Fermi) and 16 (Kepler), as detailed in Table 2.3. The reason is that the maximum number of threads per SM is 1536 in Fermi, and 2048 in Kepler. Therefore, this limit is reached before the concurrent number of blocks, $8/16$ in Fermi/Kepler, respectively. Another example is the number of registers per thread. If we create a kernel that requires 63 registers per thread, that is, the maximum allowed in both architectures, the maximum number of threads per SM will be $512/1024$ in Fermi/Kepler because there are not available registers for more threads.

One factor used to measure the performance on the GPU is the occupancy, which is defined as:

$$\text{Occupancy} = \frac{\text{number of active warps per SM}}{\text{maximum number of possible active warps}}. \tag{2.30}$$

In Kepler we have 64 active warps per SM, see Table 2.3. With a kernel configured with 128 threads per block and with no other limit in the hardware requirements, the occupancy for this kernel is:

$$\text{Occupancy} = \frac{\text{2048 threads max. / 32 threads per warp}}{64} = 1\,.$$

This metric can be also studied in the basis of the total number of concurrent blocks per SM:

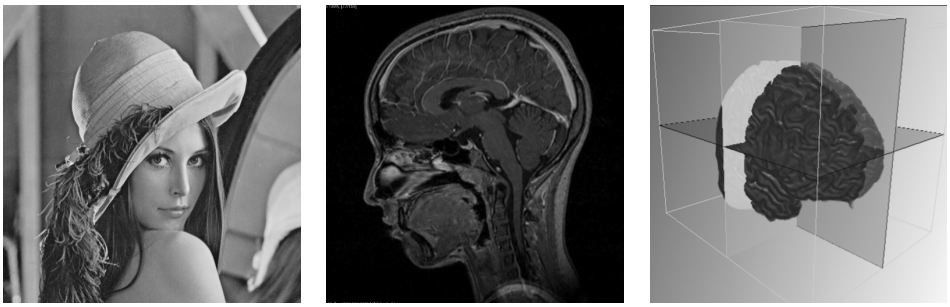
$$\text{Occupancy}_{\text{by blocks}} = \frac{\text{number of active blocks per SM}}{\text{maximum number of possible active blocks}}\,, \tag{2.31}$$

that is also 1 for the same example. With 128 threads per blocks and without any other limit in the hardware requirements, the occupancy is:

$$\text{Occupancy}_{\text{by blocks}} = \frac{\text{2048 threads max. / 128 threads per block}}{16} = 1\,.$$

In both (2.30) and (2.31), the occupancy is 100%.

The highest number of concurrent blocks is desired as it maximizes the occupancy whilst it hides the latency of memory accesses. The hardware resources that usually limit the occupancy on the GPU are the registers usage, the shared memory requirements and the block size [104].



(a) (b) (c)

Figure 2.21: 2D / 3D datasets used in this work. (a) Lena, (b) CT Scan Head, (c) simulated MRI volume.

2.9.3 Datasets used in the experiments

In this thesis we have used the following *n*-dimensional datasets: 2D images, 3D images and hyperspectral images. The following sections describe these datasets in detail.

2D / 3D images

The 2D datasets include the well-known image of Lena widely used in image analysis (grey-scale version), and a computed tomography scan of a human head. These images are shown in Figure 2.21(a) and Figure 2.21(b), respectively .

We have used a simulated MRI volume from the BrainWeb database [38, 9] for the experiments related to 3D image processing. This database contains a set of realistic MRI data volumes produced by a MRI simulator. The dataset used is the *phantom_1.0mm_normal_gry* with dimensions of $181 \times 217 \times 181$. Figure 2.22 shows the image produced using Voreen

Dataset name	Dimensions*	Size (MB)
Lena	512×512	0.25
CT Scan Head	512×512	0.25
BrainWeb	$181 \times 217 \times 181$	6.7

* *These images are used at different resolutions.*

Table 2.5: Name, dimensions and size in MB of the 2D / 3D images used in this thesis.

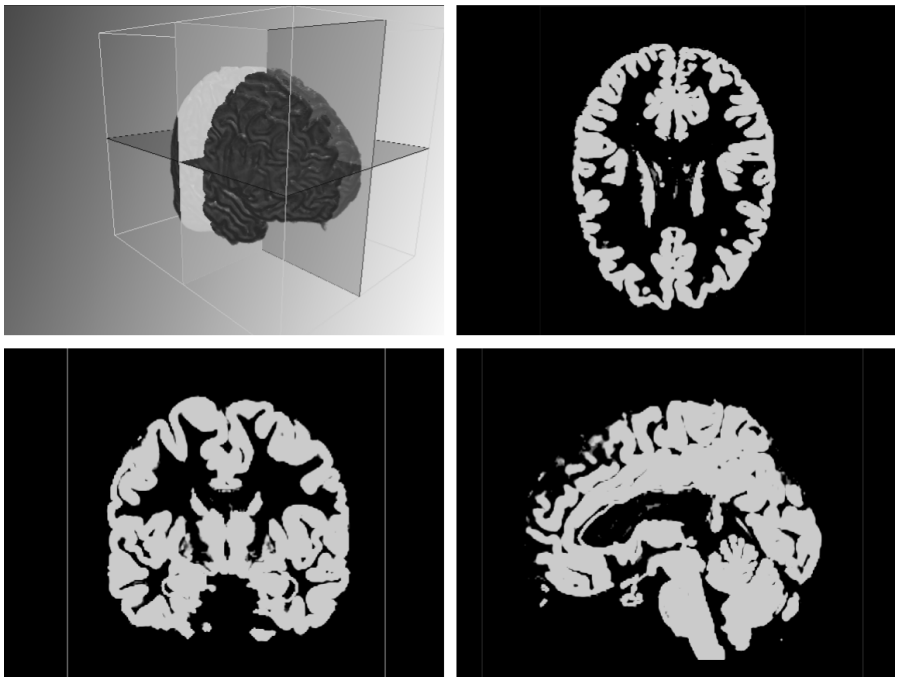


Figure 2.22: Simulated MRI volume from the BrainWeb database (BrainWeb dataset), and XY (axial), XA (coronal) and YZ (sagittal) planes. Images produced using Voreen software.

software[8]. Table 2.5 summarizes the name of these images, the original size in pixels and the size in MBs.

Hyperspectral images from ROSIS-03 sensor

The hyperspectral remote sensing scenes from the Reflective Optics System Imaging Spectrometer (ROSIS-03) sensor used in the experiments are two urban areas of Pavia. The ROSIS-03 sensor provides 115 spectral bands with a nominative spectral coverage ranging from 0.43 to 0.86 μm with a very high (1.3 m) spatial resolution per pixel.

The University of Pavia dataset was taken near the Engineering School of the University of Pavia in Italy. It is a moderately dense urban area, with some buildings and large meadows. The spatial dimension of this hyperspectral image is 610×340 pixels with 103 spectral bands.

[8] `http://www.voreen.org/`

University of Pavia			Pavia City		
Classes	Train	Test	Classes	Train	Test
Asphalt	548	6083	Water	824	65147
Meadows	540	18109	Trees	820	6778
Gravel	392	1707	Meadows	824	2266
Trees	524	2540	Bricks	808	1877
Metal	265	1080	Bare Soil	820	5764
Bare Soil	532	4497	Asphalt	816	8432
Bitumen	375	955	Bitumen	808	6479
Bricks	514	3168	Tiles	1260	41566
Shadows	231	716	Shadows	476	2387
Total	3291	42776	Total	7456	140696

Table 2.6: Training and test samples for ROSIS-03 datasets: Pavia University and Pavia City. Training and test samples are disjoint sets.

The 12 most noisy channels were removed due to noise and the remaining 103 bands are available for experiments in this scene. Figure 2.23(a) shows the true color representation of the University of Pavia. The nine classes of interest available for this dataset are shown in Figure 2.23(b) (reference map). The number of training samples used in the experiments is fixed and was taken from [54], and are presented in Table 2.6. We recall that the number of training samples are not included in the test set.

The second dataset from the ROSIS-03 sensor is a dense urban area of the center of Pavia, with spatial dimensions of 1096×715 pixels, and 102 spectral bands (13 bands were removed due to noise). This dataset is commonly known as Pavia City. The reference map contains nine classes of interest, which are detailed in Table 2.6 and shown in Figure 2.24(b). The number of training samples for this scene is also fixed and was taken from [54]. The true color representation of this scene is shown in Figure 2.24(a).

Hyperspectral images from AVIRIS sensor

The hyperspectral remote sensing scenes from the Airborne Visible-infrared Imaging Spectrometer (AVIRIS) sensor used in the experiments are two agricultural areas over the Indian Pines in north-western Indiana and over the Salinas Valley, California, and a volcanic scene over the Hekla volcano of Iceland. The AVIRIS sensor operates in the visible to mid infrared wavelength range, i.e. from 0.4 to 2.4 μm, collecting 224 spectral bands [92, 72].

Indian Pines		Salinas Valley	
Classes	Number of samples	Classes	Number of samples
1-Alfalfa	54	1-Brocoli_green_weeds_1	2009
2-Corn-notill	1434	2-Brocoli_green_weeds_2	3726
3-Corn-mintill	834	3-Fallow	1976
4-Corn	234	4-Fallow_rough_plow	1394
5-Grass/pasture	497	5-Fallow_smooth	2678
6-Grass-trees	747	6-Stubble	3959
7-Grass/mowed	26	7-Celery	3579
8-Hay-windrowed	489	8-Grapes_untrained	11271
9-Oats	20	9-Soil_vinyard_develop	6203
10-Soybean-notill	968	10-Corn_senesced_green_weeds	3278
11-Soybean-mintill	2468	11-Lettuce_romaine_4wk	1068
12-Soybean-clean	614	12-Lettuce_romaine_5wk	1927
13-Wheat	212	13-Lettuce_romaine_6wk	916
14-Woods	1294	14-Lettuce_romaine_7wk	1070
15-Bld-Grs-Trs-Drs	380	15-Vinyard_untrained	7268
16-Stone-Steel	95	16-Vinyard_vertical_trellis	1807
Total	10366	Total	54129

Table 2.7: Total number of samples for AVIRIS datasets: Indian Pines and Salinas Valley.

Hekla Volcano			
Classes	Number of samples	Classes	Number of samples
1-Andesite lava 1970	342	7-Hyaloclastite formation	684
2-Andesite lava 1980 I	708	8-Lava covered	700
3-Andesite lava 1980 II	1496	9-Rhyolite	404
4-Andesite lava 1991 I	2739	10-Scoria	550
5-Andesite lava 1991 II	410	11-Firn and glacier ice	458
6-Andesite lava with moss	1023	12-Snow	713
Total = 10227			

Table 2.8: Total number of samples for AVIRIS dataset: Hekla Volcano.

The Indian Pines scene was acquired over a mixed agricultural/forested region in northwestern Indiana with a moderate spatial resolution of 20 m. This image represents a very challenging land-cover classification scenario. It consists of 145×145 pixels and 220 spectral

bands. The four bands covering the region of water absorption were removed. This dataset is available through Purdue's University MultiSpec site. Figure 2.25(a) shows the true color representation of the scene. The sixteen classes of interest available in the reference map are shown in Figure 2.25(b). The total number of samples are presented in Table 2.7. The number of training samples for this scene is usually randomly taken from the reference map as 5% or 10% of the available data.

The Salinas dataset has 512×217 pixels and it was captured over the Salinas Valley in California. It is characterized by a high spatial resolution (3.7 m) owing to a low-altitude flight during the acquisition [72]. None of the hyperspectral bands was removed in this dataset. The total number of samples are presented in Table 2.7, and like the Indian Pines, the training samples are randomly taken from the reference map as 5% or 10% of the available data. Figure 2.26(a) shows the true color representation of the scene, and Figure 2.26(b) shows the reference map.

The Hekla volcano scene has spatial dimensions of 560×600 pixels with a spatial resolution of 20 m. During data collection of this dataset, spectrometer four was not properly working. This particular spectrometer operates in the near-infrared wavelength range, from 1.84 μm to 2.4 μm (64 data channels). These 64 data channels were deleted from the data set along with the first channels for all the other spectrometers, but those channels were blank. Once the noisy and blank data channels had been removed, 157 data channels were left [16]. Figure 2.27(a) shows a false color representation of the scene. The twelve classes of interest available in the reference map are shown in Figure 2.27(b). The names and the number of samples per class are available in Table 2.7. For this dataset, a fixed number of fifty samples per class is used for training, and the rest of the samples are used for testing.

Table 2.9 summarizes the dimensions and the size in MB of the five hyperspectral images used in the experiments carried out in this thesis. We have considered different spatial resolutions (1.3 μm, 3.7 μm and 20 μm), different dimensions in the spatial domain (from 145×145 to 1096×715 pixels), and different numbers of spectral bands (from 102 to 224 bands). The Pavia City scene is the largest in the spatial domain with 1096×715 pixels, while the scenes of Indian Pines and Salinas Valley are the scenes with more spectral bands, 220 and 224, respectively.

The Pavia University, Pavia City, Indian Pines and Salinas Valley datasets (hyperspectral image and reference map) are publicly available online[9] at the research webpage of the

[9] http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes

Dataset name	Dimensions	Size (MB)	Sensor
Pavia University	$610 \times 340 \times 103$	162.9	ROSIS-03
Pavia City	$1096 \times 715 \times 102$	609.8	ROSIS-03
Indian Pines	$145 \times 145 \times 220$	35.3	AVIRIS
Salinas Valley	$512 \times 217 \times 224$	189.9	AVIRIS
Hekla Volcano	$560 \times 600 \times 157$	402.5	AVIRIS

Table 2.9: Hyperspectral images used in this work.

Computational Intelligence Group from the Basque University (UPV/EHU). The Indian Pines scene is originally available through Purdue's University MultiSpec site[10].

We would like to thank prof. Benediktsson from the University of Iceland, for providing the hyperspectral dataset of Hekla.

[10] `http://engineering.purdue.edu/~biehl/MultiSpec/hyperspectral.html`

Figure 2.23: Pavia University dataset. True color representation (a), reference map with nine classes of interest (b), and name of the classes (c).

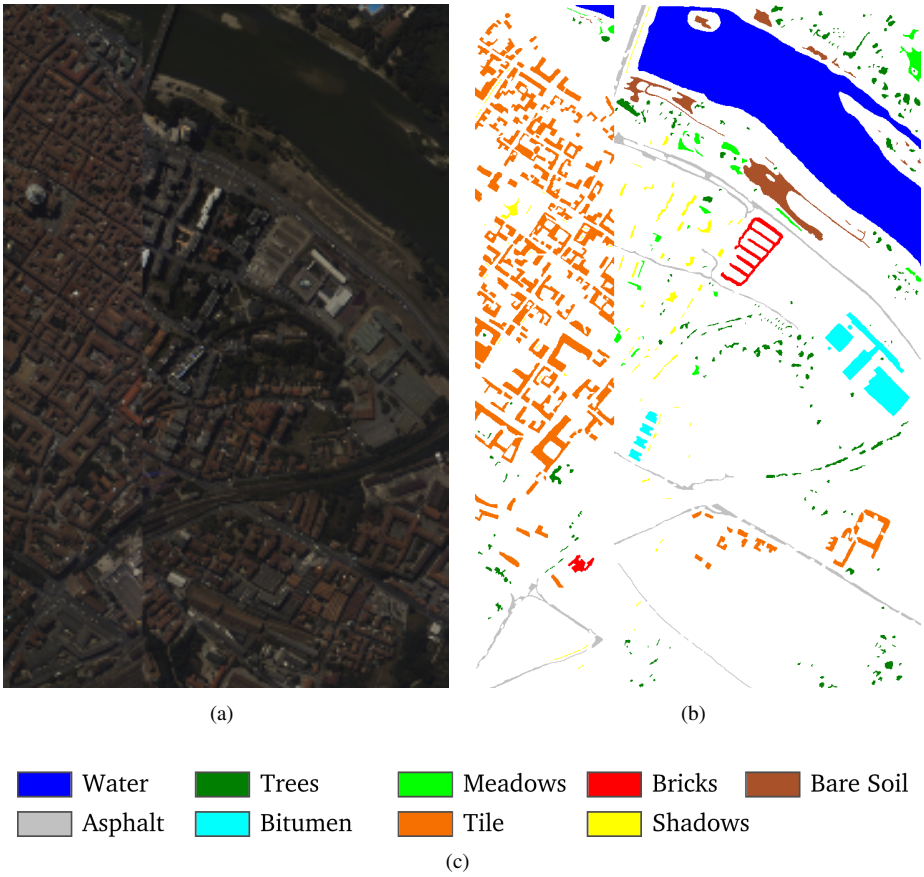
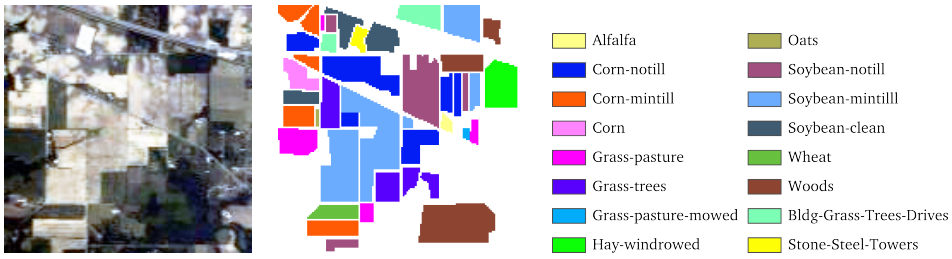
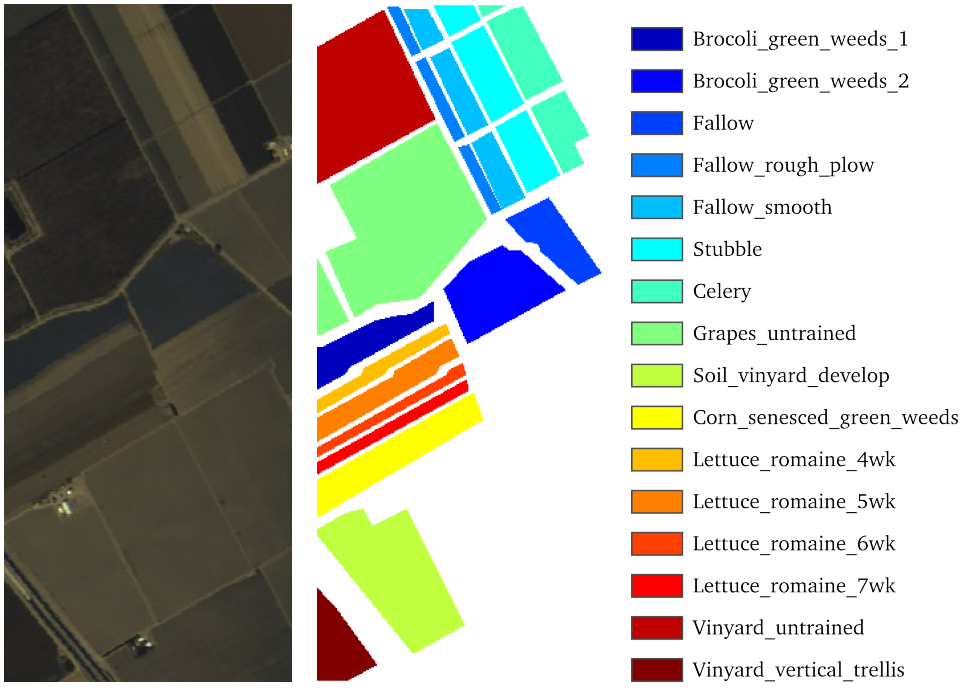


Figure 2.24: Pavia City dataset. True color representation (a), reference map with nine classes of interest (b), and name of the classes (c).



(a) (b) (c)

Figure 2.25: Indian Pines dataset. True color representation (a), reference map with sixteen classes of interest (b), and name of the classes (c).



(a) (b) (c)

Figure 2.26: Salinas Valley dataset. True color representation (a), reference map with sixteen classes of interest (b), and name of the classes (c).

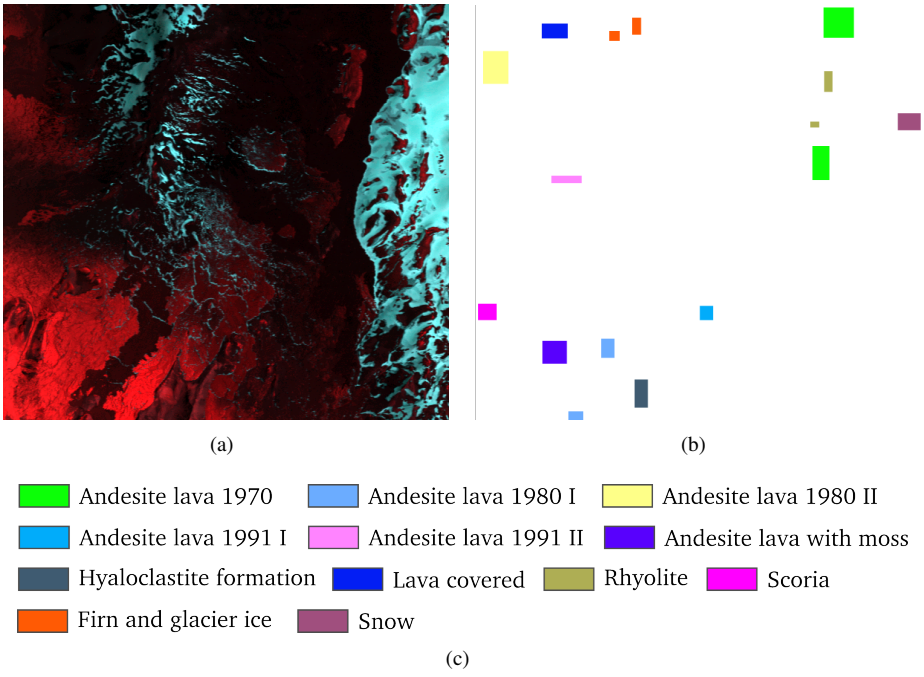


Figure 2.27: Hekla Volcano dataset. False color representation (a), reference map with twelve classes of interest (b), and name of the classes (c).

CHAPTER 3

SPECTRAL-SPATIAL CLASSIFICATION SCHEMES BASED ON SEGMENTATION AND MATHEMATICAL MORPHOLOGY

3.1 Introduction

In this chapter we present the two schemes we proposed for efficient spectral-spatial nD image classification, based on segmentation, Mathematical Morphology (MM), and SVM classifiers, called CA–WSHED–MV and WT–EMP.

First, we describe in Section 3.1.1 the general framework for spectral (pixel-wise) classification schemes as a basis for designing new schemes. This framework is a starting point for hyperspectral image classification. Second, a general scheme for spectral-spatial classification, as well as the common data fusion strategies for joining the spectral and the spatial information of such scheme are described in this chapter in Section 3.1.2. This framework incorporates spatial information to improve the results of the final classification. Based on the approach used for extracting the spatial information, different data fusion techniques can be employed.

The spectral-spatial CA–WSHED–MV scheme, originally proposed in [155], is presented in Section 3.2. In this scheme extracts the spatial information by a watershed transform based on cellular automata (CA–Watershed), resulting in a more efficient step for GPU processing, combining the spectral results of the classifier by a Majority Vote (MV).

The spectral-spatial WT–EMP scheme is a new proposal for land-cover classification in remote sensing imaging for real-time applications. The second scheme proposed in this thesis extracts spatial information using morphological profiles and creates a new vector of features prior to the classification. The scheme is described in Section 3.3.

The classification accuracy of both schemes is evaluated on real hyperspectral images, and the results thereof are compared to classification schemes found in the literature based on segmentation and MM.

3.1.1 Framework for spectral classification schemes

The classification schemes for hyperspectral images process each pixel independently (pixel-wise processing) and do not take into account the spatial information of the neighborhood. Success in classification depends solely on the ability of the classifier to discriminate among pixel vectors. Thus, the classifier discriminates the pixels of the image based only on the spectral features captured by the hyperspectral sensor. One possible framework for spectral classification that we will use in this thesis is given in Figure 3.1. This framework is based on the spectral classification scheme proposed by Landgrebe et al. [95], which is widely used nowadays as the basis for hyperspectral image classification.

Considering that the spectral features are often redundant, Feature Extraction (FE) and Feature Selection (FS) are usually performed as a preprocessing step. FE is designed to remove the redundancy introduced by the spectral correlation between bands [146], while the main characteristics in the spectral domain are retained. One consequence is that the spectral dimensionality is reduced. Different techniques have been investigated for extracting the principal features for urban land cover classification of hyperspectral data using SVM-based classifiers [48, 31]. Section 2.2.1 describes the techniques used in the spectral-spatial classification schemes investigated in this work. Principal Component Analysis (PCA) and Independent Component Analysis (ICA) are two of the most widely used unsupervised techniques in remote sensing for reducing the redundancy of the spectral bands. However, the Minimum Noise Fraction (MNF), which extracts the data sorting the components by signal-to-noise ratio, performed better than PCA in the analysis carried out in [48]. This finding emphasizes the idea that removing noise is a good preprocessing option.

Supervised feature extraction techniques such as Discriminant Analysis Feature Extraction (DAFE), Decision Boundary Feature Extraction (DBFE) and Non-parametric Weighted Feature Extraction (NWFE) have shown their effectiveness in classification of hyperspectral

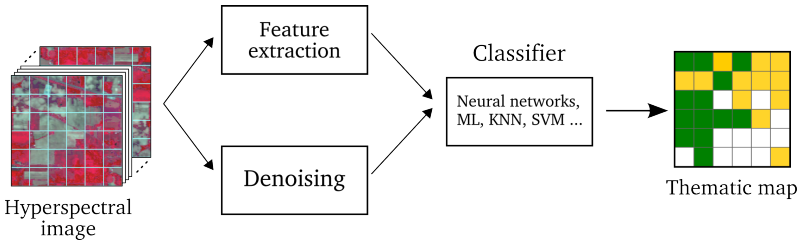


Figure 3.1: Simplified framework for spectral classification schemes incorporating feature reduction and denoising.

data [108, 31, 93, 96]. Kuo and Landgrebe proposed NWFE in [93], as a method for improving DAFE by exploiting different weights on samples close to the decision boundary. The NWFE has been shown to perform better than DAFE and DBFE in classification of hyperspectral data [96].

An automatic extraction of features using wavelet was presented in [86], where the dimensionality of the image is reduced several times, and then reconstructed by wavelets. The best level of decomposition is then determined through the similarity between the original pixel vector and the vector reconstructed using wavelets. In [162] several methods based on wavelet transforms are developed to extract useful features for classification. The Daubechies 3 wavelet is applied to hyperspectral images, and a small subset of wavelet coefficients is then used to extract the effective features for classification. In the work published in [68] wavelet transforms were used to reduce the dimension of the hyperspectral images, owing to its ability to detect the local energy variations better than other transforms.

The acquisition of the image usually introduces undesirable artifacts that may affect the quality of the spectral signature. Therefore, another preprocessing commonly applied are denoising [164, 166] and scatter correction [140]. As illustrated in Figure 3.1, feature extraction and denoising are not mutually exclusive.

Finally, once the preprocessing has been performed, the classification of the hyperspectral image takes place. The result is a thematic map in which each pixel is labelled with a class that identifies the corresponding spectral signature. In Figure 3.1, the result is a classification map where the labels are identified by three different colors. There are a variety of classifiers, such as maximum likelihood, neural networks, decision trees, and kernel-based methods. Melgani and Bruzzone [110] have shown that the SVM classifier is more effective for remote sensing classification than other conventional non-parametric classifiers, in terms of accuracy, computational time, and stability to parameter setting. In addition, the SVM classifier has been

shown to be one of the best classifiers in an exhaustive evaluation of 179 classifier published in [60]. In the spectral-spatial schemes proposed in this thesis, the SVM, previously explained in Section 2.7, is used for classification of hyperspectral images.

3.1.2 Framework for spectral-spatial classification schemes

By using only spectral information in the classification of remote sensing images, we are not taking advantage of the relationship among adjacent pixels. Pixels from homogeneous regions are similar in their spectral response, indicating a relationship between them. It has been clearly stated that the spatial information extracted from the hyperspectral images improves the accuracy of the classification when it is incorporated into the scheme [15, 54, 154, 155, 41, 19, 29, 56, 57].

Figure 3.2 shows one possible spectral-spatial framework for classification of hyperspectral images that will be used as a base for our proposals. The spatial processing is mainly derived from techniques designed for greyscale and most of the commonly used methods are not appropriate for n-dimensional images. In the case of color and multispectral images, the visible (red, green, and blue) and infra-red bands can be processed independently. For hyperspectral images, one way of applying spatial techniques is by reducing the number of spectral features to only a few or even one, for example, by using the first principal component computed by PCA or ICA. Unlike the spectral schemes, the feature reduction stage illustrated in Figure 3.2 is also applied as a first step to reduce the spectral dimensionality in order to extract the spatial information. Therefore, the techniques for reducing the dimensionality of hyperspectral images prior to the pixel-wise classification are also of special interest in the spectral-spatial classification schemes.

Different methods to reduce the number of features on the generation of spectral-spatial classification schemes were investigated in [155, 31]. Multidimensional and vectorial gradient methods, such as the RCMG described in Section 2.2.2, were used in [155] to reduce the number of bands to one. In [31] the research focused on the best techniques for reducing the number of features to create schemes based on mathematical morphology, concluding that other methods than PCA may be more adequate in terms of classification accuracy for creating those schemes.

Additional preprocessing may be applied as a first step, as illustrated in Figure 3.2, for improving hyperspectral image classification [164, 166], for example by denoising the hyperspectral data.

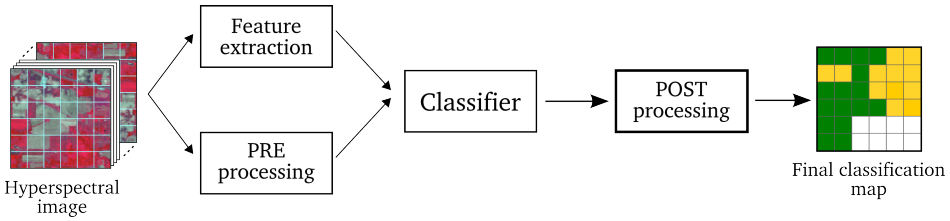


Figure 3.2: Framework of a spectral-spatial classification scheme. Some schemes require a postprocessing for joining the spectral and spatial results.

The spatial information is extracted from the closest neighborhood of a pixel. The closest neighborhood can be defined by a fixed $n \times n$ window, such as the structuring element used in MM. The neighborhood can also be defined by a homogeneous region, named adaptive neighborhood as the number of neighboring pixels depends on the size of the region considered. The regions created in a segmentation map, or the flat zones created by a self-complementary area filter [56] are two examples of adaptive neighborhood.

A method for extracting spatial features based on opening and closing by reconstruction, see Section 2.5.1, was proposed in [124]. In order to apply the morphological approach to hyperspectral images, PCA is first applied on the hyperspectral data, and the most significant PCs are used as base images, creating a new pixel vector of morphological features, known as EMP [15]. The spectral information in the EMP is small compared to the spatial information incorporated by the morphological profiles. Fauvel et. al [54] proposed a scheme for urban land cover classification of hyperspectral images, which extended the EMP by fusing spectral data. The spectral information is incorporated from the original hyperspectral data without any additional processing.

Regarding the segmentation, the spectral-spatial classification scheme presented in [158] combines the results of a pixel-wise SVM classification and a segmentation map. Several segmentation techniques have been proposed for this scheme, such as partitional clustering [154], watershed transform [155, 137] and hierarchical segmentation (HSEG) [160]. Section 2.3 has a summary of these techniques. Clustering algorithms were used in [154], and evolutionary CA were designed and applied in [132] for segmenting hyperspectral images, although in this last case reducing the dimensionality of the data is not required. The idea behind these schemes is to regularize the thematic map produced by the classifier by using the regions of the segmentation map as an adaptive neighborhood.

The classification result by spectral-spatial schemes, as illustrated in the final classification map shown in Figure 3.2, gives rise to greater accuracy and more homogeneous thematic maps as compared to spectral classification schemes. In order to increase the spatial information included in the schemes, a spatial post-regularization (PR) can be applied over the thematic map produced by the classifier [154]. This step that is applied in [154] included as part of the post-processing in Figure 3.2, introduces additional spatial/textural information by considering separately for each pixel the classes that the classifier assigns to the neighbors in a fixed $n \times n$ window. The new class for each pixel is decided by analyzing information of other pixels in a fixed-size neighborhood. This regularization is performed until stability is reached (none of the pixels changes its class) [158]. By repeating the regularization process until stabilization of the class labels, the regions in the final thematic map are homogeneous. Thus, the spatial PR can be applied to the thematic map before a MV or to the final thematic map produced by the spectral-spatial classification scheme.

3.1.3 Data fusion techniques

Different data fusion techniques are used to combine the spectral and the spatial information, and they can be applied before, after and/or within the classification step, as shown in Figure 3.2. The integration of spectral and spatial information can be achieved by introducing spatial information into the kernel function used by the classifiers [29, 28, 56, 5]. This approach exploits the properties of Mercer's kernels [1, 27] to construct composite kernels combining the spatial and the spectral information. The spectral-spatial classification scheme proposed in [5] integrates the spatial information provided by extended morphological profiles using composite feature mappings. In the following sections we described the data fusion techniques, majority vote and stack vectors, which will be used in the two schemes proposed in this thesis.

Fusion via majority vote

The MV is a standard data fusion procedure which combines the spatial information extracted by a segmentation technique with the thematic map produced by a classifier. The thematic map is regularized summing up the votes that identify the spectral class for each pixel within the segmented region they belong to. An example of MV is illustrated in detail in Figure 3.3 for the case of three spectral classes, represented as three colors in Figure 3.3 (a), and a

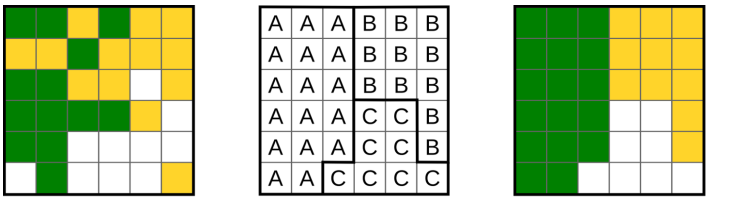


Figure 3.3: Example of majority vote between a classification map with three classes (green, yellow and white) in (a), and a segmentation map with three regions (A, B, and C) in (b). The result of the MV is illustrated in (c).

segmentation map with three regions, namely A, B, C, in Figure 3.3 (b). The results of the majority vote are shown in Figure 3.3 (c).

The schemes that incorporate the spatial information by a segmentation technique combine the spatial and spectral results by a MV after the classification [154, 155, 137, 78]. This data fusion technique will be used in the CA–WSHED–MV scheme proposed in this thesis and described in Section 3.2.

Fusion via stack vector

The stack vector is a commonly used data fusion technique that incorporates the spatial information before the classification.

Let us denote α_k as a pixel vector of the hyperspectral image with k being the number of spectral features, such as $\alpha_k = [x_1, x_2, \ldots, x_k]$, and $\beta_p = [y_1, y_2, \ldots, y_p]$ as a pixel vector of p spatial features. For each pixel of the hyperspectral image, a new stack vector $\tau = [\alpha_k, \beta_p]$ is created prior to the classification. Although the stack vector is a simple data fusion strategy, the classification results have been demonstrated to be good in schemes using morphological profiles, such as the EMP originally proposed by Benediktsson et. al [15], or the scheme proposed in [54] which improved the EMP scheme by fusing spectral and spatial data by stack vectors. The schemes based on Extended Attribute Profile (EAP) [41] and Extended Multi-Attribute Profile (EMAP) [42] also use this technique to combine the data created by the different attribute profiles. In addition, one advantage of this fusion method is that only one classification is performed using as input the stack vector, and no postprocessing is needed to combine any other results.

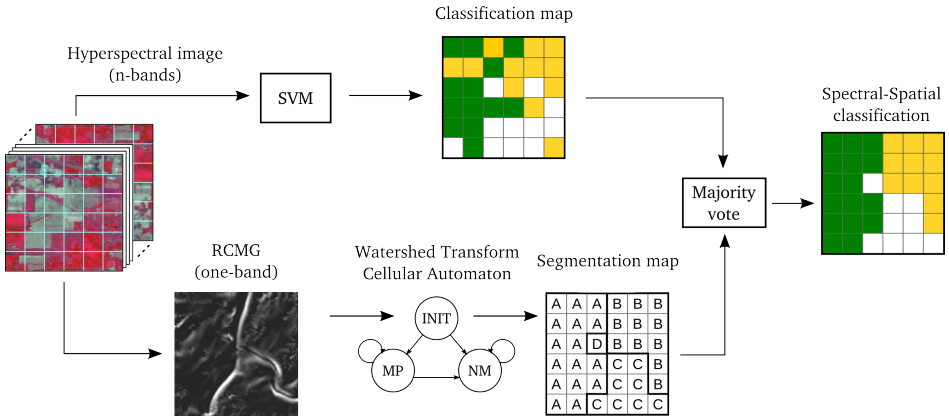


Figure 3.4: Flow-chart of the CA–WSHED–MV classification scheme based on vectorial gradient (RCMG), segmentation (CA-Watershed), classification by SVM and Majority Vote.

This data fusion technique will be used in the WT–EMP scheme proposed in this thesis and described in Section 3.3.

3.2 Scheme based on segmentation: CA–WSHED–MV

In this section we present our CA–WSHED–MV scheme, originally proposed in [155] but incorporating the spatial information by a watershed transform (see Section 2.3.2) based on cellular automata (CA–Watershed), which results in a more efficient algorithm for GPU processing. In addition, the watershed algorithm used in the proposed scheme does not create the so-called watershed lines, and thus it is not necessary to compute a standard vector median for every watershed region unlike the scheme proposed by Tarabalka [155]. The scheme consists of the calculation of a RCMG (see Section 2.2.2) that reduces the dimensionality of the hyperspectral image, followed by the CA–Watershed computation on the RCMG. The CA-Watershed is explained in Section 3.2.2. The classification is carried out by SVM combining the spectral and spatial results with a Majority Vote (MV). Figure 3.4 shows the CA–WSHED–MV scheme flow-chart.

On the one hand, the spectral processing is applied over the hyperspectral image using a SVM classifier that produces a classification map. Each pixel of this map belongs to one class predicted by the SVM. On the other hand, the spatial processing creates a segmentation map of the hyperspectral image. In this map, all of the pixels are labelled according to the

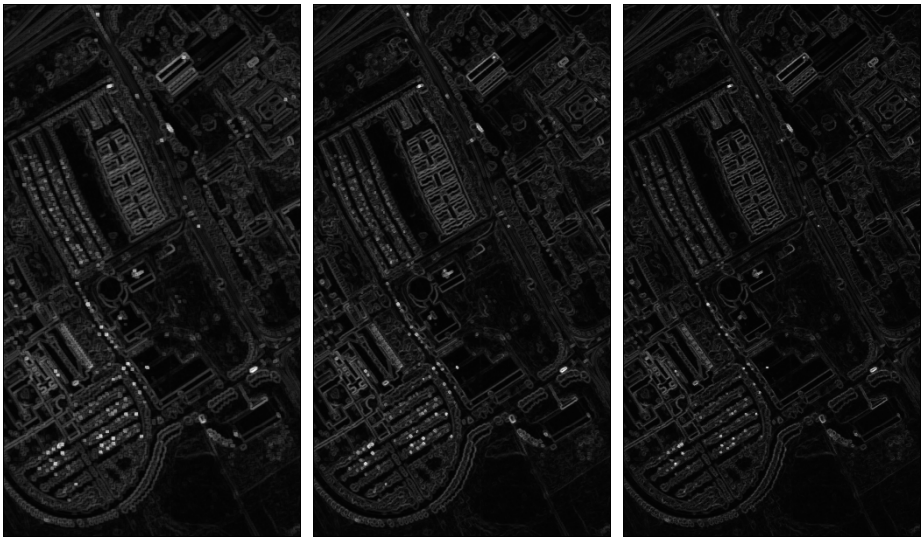
region they belong to. The segmented regions are combined with the classes obtained by the pixel-wise spectral classification using a data fusion via MV, as described in Section 3.1.3.

The following sections describe the stages of the spectral-spatial classification scheme that is illustrated in Figure 3.4.

3.2.1 Robust Color Morphological Gradient (RCMG)

The Robust Color Morphological Gradient, describe in detail in Section 2.2.2, is a vectorial gradient for multichannel images, which is also very robust to image noise [52]. This gradient was developed for color images in [52] and applied to hyperspectral images in [154, 155]. Lower values of robustness, that is the set of pairs of pixel vectors removed given by the parameter s (see Eq. (2.9) in Section 2.2.2) are more appropriated in practice for the RCMG.

Figure 3.5 shows the RCMG applied to the University of Pavia dataset computed using 8-connectivity with s ranging from 0 to 2 (s cannot exceed $\frac{N-1}{2} - 1$ if at least two vectors from the original set are to remain). In the proposed scheme, the RCMG is computed using



(a) (b) (c)

Figure 3.5: RCMG with different robustness applied to the University of Pavia dataset: (a) $s = 0$, (b) $s = 1$, and (c) $s = 2$. The result is normalized between 0 and 255.

8-connectivity with $s = 1$. The result is a one-band gradient image which will be segmented by the CA–Watershed algorithm.

3.2.2 Watershed transform based on cellular automata (CA-Watershed)

Regarding segmentation, Galilée et al. [64] introduced a parallel algorithm-architecture based on asynchronous CA to compute the watershed transform. The asynchronous behavior of this automaton is subject to its implementation; i.e., the CA–Watershed can be synchronously or asynchronously implemented. The 3–state cellular automaton proposed in [64] follows the Hill-Climbing algorithm described in Section 2.3.2. The main advantage of this algorithm is that minima detection, labelling, and climbing the steepest paths are performed simultaneously and locally. We now go on to describe the CA–Watershed algorithm, where each cell of the automaton computes a pixel of the image. The fundamentals of the watershed transform and cellular automata are described in Section 2.3.2 and Section 2.4, respectively.

Figure 3.6 shows the 3–state cellular automaton that computes the watershed transform which is described hereafter. First, the pixels are sequentially labelled. In the initial state, all pixels compute the sets $\mathcal{N}^F(u)$ and $\mathcal{N}^=(u)$ corresponding to an arbitrary lower slope and the neighbors with the same grey value, respectively. If a pixel has no lower slope, $\mathcal{N}^F(u) = \emptyset$, it switches to the minimum or plateau (MP) state and its label is initialized as follows:

$$l(u) = min\,(\,l(v)\,)\,, \tag{3.1}$$

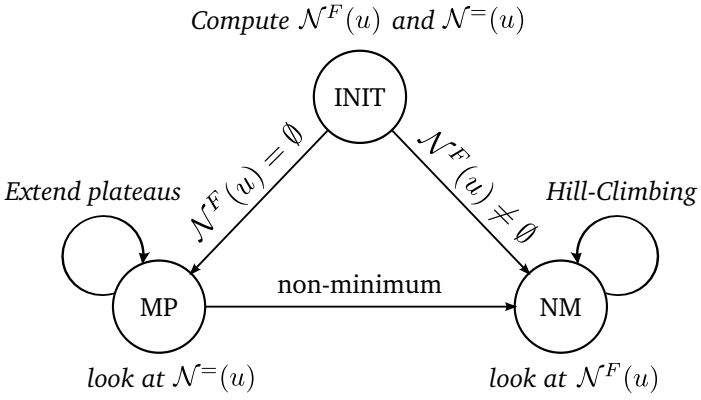


Figure 3.6: 3-state cellular automaton implementing Hill-Climbing algorithm [64].

Algorithm 1 CA–Watershed algorithm [64]

```
1:  procedure CA–WATERSHED(state, f)
2:      switch state do
3:          case INIT :                                          ▷ Initialize automaton
4:              compute N^=(u), N^F(u)
5:              if (N^F(u) = ∅) then
6:                  l(u) ← min_{v∈N^=(u)} (l(v))                 ▷ Eq. (3.1)
7:                  state(u) ← "MP"
8:              else
9:                  f(u) ← f(N^F(u))                             ▷ Eq. (3.2)
10:                 state(u) ← "NM"
11:             end if
12:
13:         case MP :                                            ▷ Minimum or Plateau
14:             for each pixel v ∈ N^=(u) do
15:                 if h(v) < h(u) then
16:                     N^F(u) = {v}                             ▷ Eq. (3.4)
17:                     f(u) ← f(v)
18:                     state(u) ← "NM"
19:                 else
20:                     l(u) ← min(l(u), l(v))                   ▷ Eq. (3.3)
21:                 end if
22:             end for
23:
24:         case NM :                                            ▷ Non Minimum
25:             v = N^F(u)
26:             f(u) ← f(v)                                      ▷ Eq. (3.5)
27:
28:     end switch
29: end procedure
```

where $l(v)$ are the labels of the pixels $v \in \mathcal{N}^{=}(u)$. Otherwise, the state of the pixel switches to non-minimum (NM) and $\mathcal{N}^{F}(u)$ points to the climbing direction. The grey value and the label of the pixel are modified as follows:

$$f(u) = f\left(\mathcal{N}^{F}(u)\right), \tag{3.2}$$

where $f(u)$ is the pair $h(u), l(u)$, $h(u)$ is the grey value of the pixel u and $l(u)$ its label.

Once the pixel has been initialized, the update stage begins. This is an iterative task that processes the MP and NM states. A pixel in MP state waits for data from any neighbor $v \in \mathcal{N}^{=}(u)$, and, depending on the grey value of the neighbor, two cases are considered. If the value of the neighbor is equal to or greater than its current value, the label of the pixel is

updated as:

$$l(u) = min\,(\,l(u), l(v)\,) \quad \text{if } h(v) \geq h(u). \tag{3.3}$$

Otherwise, the pixel belongs to the lower border of the plateau and its state switches to NM as follows:

$$\left.\begin{array}{l} \mathscr{N}^F(u) = v, \\ f(u) = f(v), \\ state = \text{NM}, \end{array}\right\} \quad \text{if } h(v) < h(u). \tag{3.4}$$

On the other hand, a pixel in NM state remains in that state and waits for data from the neighbor $\mathscr{N}^F(u)$, updating its data as follows:

$$f(u) = f(\mathscr{N}^F(u)). \tag{3.5}$$

This algorithm is non-deterministic and may lead to different segmentation results. A formal proof of correctness and convergence towards a watershed segmentation using a mathematical model of data propagation in a graph is presented in [64]. The pseudocode of this algorithm is listed in Algorithm 1 and Figure 3.7 shows a guided example of the evolution

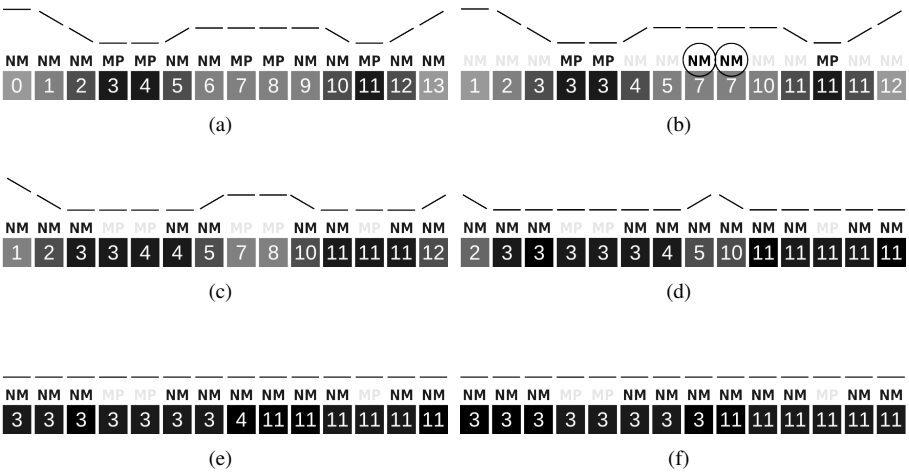


Figure 3.7: CA–Watershed based on Hill-Climbing algorithm. Example of 1D image represented as a terrain with dashed lines and as grey values with the squares: a) pixels are sequentially labelled and states are initialized, b) MP updating stage where two pixels change their state to NM c) to f) the automaton evolves only with the NM rules.

of the CA–Watershed on a 1D image, representing the grey values with the squares and the terrain with dashed lines. The automaton initialization is shown in Figure 3.7(a). The first updating step, Figure 3.7(b) and Figure 3.7(c), illustrates the MP and NM states, respectively. It can be observed in Figure 3.7(b) that two pixels change their state from MP to NM as they belong to the lower border of the inner plateau. In the successive updating steps, Figure 3.7(d) to Figure 3.7(f) the automaton evolves only with the NM rules. The labels are propagated from the minimum climbing up the hills, flooding the terrain.

3.2.3 Results

In this section we conduct an analysis[1]. of the proposed CA–WSHED–MV scheme in terms of Overall Accuracy (OA), Average Accuracy (AA) and the kappa coefficient of agreement (k). These criteria are computed from the confusion matrix.

The RCMG is computed using 8-connectivity and the pairs of pixel vectors removed is $s = 1$, (see Section 2.2.2 and Section 3.2.1 for details on the robustness of this morphological gradient). The CA–Watershed is configured to use 8-connectivity as well.

Given that the classification results depend on the set of training samples and their distribution within the hyperspectral image [39], the results are obtained executing the classification 10 times with different sets of training samples each time. Since the training samples were randomly selected, the results were calculated as the average of the 10 different values obtained for each execution. The hyperspectral remote sensing scenes used in the experiments are two urban areas (Pavia University and Pavia City datasets) taken by the ROSIS-03 hyperspectral sensor and one hyperspectral image of a crop area (Indian Pines dataset) taken by the AVIRIS sensor. These datasets are described in Section 2.9.3, including the number of available samples.

The number of training samples for each scene is chosen as in [155, 121, 54] to compare our scheme on equal terms. In order to obtain a reliable evaluation of the results, the accuracies are calculated excluding the samples used for training.

[1]Part of these results have been published in P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Efficient segmentation of hyperspectral images on commodity GPUs," *in 16th International Conference on Knowledge-Based and Intelligent Information & Engineering System*, vol. 243, pp. 2130–2139, 2012. And P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Computing efficiently spectral-spatial classification of hyperspectral images on commodity gpus," *in Recent Advances in Knowledge-based Paradigms and Applications* (J. W. Tweedale and L. C. Jain, eds.), vol. 234 of Advances in Intelligent Systems and Computing, Ch. 2, pp. 19–42, Springer International Publishing, 2014.

	Pavia University	Pavia City	Indian Pines
C	128	128	1024
γ	0.125	0.125	0.5

Table 3.1: Best parameters (C, γ) for the SVM determined for each hyperspectral image in the range $C = [1, 4, 16, 64, 128, 512, 1024]$, $\gamma = [0.5, 0.25, 0.125, 0.0625]$ by 5–fold cross-validation.

The whole process is computed in CPU and the classification is carried out using the LIBSVM library [35] with the RBF kernel. The parameters (C, γ) for training the SVM are determined for each hyperspectral image in the range $C = [1, 4, 16, 64, 128, 512, 1024]$, $\gamma = [0.5, 0.25, 0.125, 0.0625]$ by 5–fold cross-validation, as explained in Section 2.7. For each one of the hyperspectral images under study, a random training set of the available samples is used to find the best parameters for the SVM, which are summarized in Table 3.1.

The classification results are compared to those obtained by the pixel-wise SVM classifier and other spectral-spatial classification schemes based on segmentation and SVM, the results of which are available in the literature: SVM+EM (PR) [154], SVM+ISODATA [154], SVM+ ISODATA (PR) [154], and HSeg+MV [158]. These schemes are based on clustering and region growing segmentation techniques. The HSeg+MV scheme is based on the HSEG algorithm proposed in [161]. A description of these segmentation techniques was presented in Section 2.3. The schemes marked with PR also include an additional post regularization as described in the previous Section. The segmentation map obtained by these schemes is combined with the thematic map produced by the SVM using the majority voting data fusion. These segmentation techniques are also described in Section 2.3.

For the Pavia City dataset there are no published results of spectral-spatial classification schemes based on segmentation[2], so we have produced additional results for this dataset by using two segmentation algorithms based on clustering. The first classification scheme is based on k-means (k-means–MV) and the second one is based on Quick-Shift [163] (QS–MV). Both schemes combine the spectral and spatial information by majority vote.

University of Pavia dataset

The University of Pavia dataset is a moderately dense urban area, with some buildings and large meadows and high spatial resolution (1.3 m). Nine classes of interest are available for

[2]Partial result for this dataset were found in [21] but using a subset of 900×300 pixels.

	SVM [158]	SVM+EM (PR) [154]	SVM+ISODATA (PR) [154]	Hseg+MV [158]	CA–WSHED–MV
1-Asphalt	84.93	93.45	94.40	94.77	**95.42**
2-Meadows	79.79	93.78	87.45	89.32	**95.51**
3-Gravel	67.16	82.53	61.32	**96.14**	87.01
4-Trees	97.77	99.38	**98.63**	98.08	95.93
5-Metal sheets	99.46	**100**	99.91	99.82	99.72
6-Bare Soil	92.83	97.38	97.88	**99.76**	97.26
7-Bitumen	90.42	94.19	**100**	**100**	98.97
8-Bricks	92.78	98.31	99.02	**99.29**	98.51
9-Shadows	98.11	97.86	97.86	96.48	**100**
OA	81.01	94.64	91.20	93.85	**95.88**
AA	88.25	95.21	92.94	**97.07**	96.48
k	0.758	0.929	0.884	0.919	**0.944**

Table 3.2: Classification results for the CA–WSHED–MV scheme on the University of Pavia dataset and compared to SVM, SVM+EM (PR), SVM+ISODATA (PR), and Hseg+MV. Best results are indicated in bold.

this scene. The number of training samples used in the experiments was taken from [54], as described in Section 2.9.3, Table 2.6. We recall that the number of training samples is not included in the test set.

Table 3.2 shows the name of the classes and gives the classification accuracies obtained by the pixel-wise SVM classifier, and different segmentation schemes based on clustering and post-regularization SVM+EM (PR) and SVM+ISODATA (PR), and based on Hierarchical Image Segmentation Hseg+MV. Best results are indicated in bold. Figure 3.8 shows from left to right, the thematic map produced by the pixel-wise SVM classifier, the RCMG, the CA–Watershed segmentation map with watershed lines imposed for visualization, and the final classification map produced for the CA–WSHED–MV scheme.

As can be seen from Table 3.2, the spectral-spatial classification schemes have higher accuracies as compared to the results obtained by the pixel-wise SVM. The best OA is achieved by the proposed CA–WSHED–MV scheme. This scheme improves the OA in 14.87% and the AA in 8.23% as compared to the classification by the pixel-wise SVM. The Hseg+MV scheme has the best AA, showing a good consistency in the class-specific results, which were improved for almost all the classes, except for the *shadows* class. However, the proposed scheme improved all the class-specific accuracies, including the *shadows* class up to 100%. This is an outstanding result because the accuracies in the sixth column in Table 3.2, corresponding to the proposed scheme, are calculated as the average over 10 different classification with different sets of training and test samples.

Pavia City dataset

The Pavia City dataset is a dense urban area, with many buildings and a large river in the top-right corner of the scene. The nine classes of interest, which are available for this dataset are shown in Table 3.3. The number of samples used in the experiments is detailed in Section 2.9.3, Table 2.6. Table 3.3 shows the classification accuracies obtained by the pixel-wise SVM classifier. Other schemes based in segmentation applied to this dataset in the same experimental conditions were not found in the literature. K. Bernard et al. [21] published the results of their spectral-spatial classification scheme based on a stochastic minimum spanning forest approach. Their scheme produces a set of segmentation maps generating random markers from the SVM thematic map, which are aggregated by a MV. However, the Pavia City dataset used in [21] is a subset of 900×300 pixels, so it is not included in the analysis because the number of test samples would not be the same. In order to obtain a reliable evaluation of the proposed scheme, we have added the results of a partitional clustering approach by k-means (k-means–MV) generated by us.

We have also used a Quick Shift (QS) segmentation technique [163] to create a second clustering-based scheme (QS–MV) for comparison. These schemes are similar to the one proposed by Tarabalka et al. [154] but without post-regularization. Figure 3.9 shows from left to right the thematic map produced by the pixel-wise SVM classifier, the RCMG, the CA–Watershed segmentation map with watershed lines imposed for visualization, and the final classification map produced for the proposed CA–WSHED–MV scheme.

	SVM [58]	k-means–MV	QS–MV	CA–WSHED–MV
1-Water	99.1	99.99	99.97	**100**
2-Trees	90.8	92.84	**96.29**	95.84
3-Meadow	97.4	63.25	96.89	**98.33**
4-Brick	87.5	97.49	**98.65**	98.47
5-Bare Soil	94.6	94.49	**94.79**	94.76
6-Asphalt	96.4	97.85	94.59	**99.09**
7-Bitumen	96.5	92.65	**96.69**	95.22
8-Tile	99.5	98.88	99.36	**99.80**
9-Shadows	100	95.53	94.58	**100**
OA	98.1	97.93	98.76	**99.20**
AA	95.1	92.56	96.84	**97.94**
k	0.97	0.970	0.982	0.988

Table 3.3: Classification results for the CA–WSHED–MV scheme on the Pavia City dataset and compared to SVM, k-means–MV and QS–MV. Best results are indicated in bold.

Table 3.3 gives the classification accuracies obtained by the pixel-wise SVM classifier, and the segmentation schemes based on k-means and QS, as well as the proposed scheme. Results for the pixel-wise SVM classifier are taken from [58]. As can be seen from Table 3.3, all the spectral-spatial classification accuracies are higher as compared to the SVM pixel-wise classifier, despite the OA obtained as basis for comparison is fairly high (98.1%). The best OA is achieved when using the spectral-spatial classification scheme based on segmentation by the CA–Watershed algorithm. With the proposed scheme, the OA is improved by 1.1% and the AA is improved by 2.87% compared to the pixel-wise SVM classification. The k-means–MV scheme obtains worse results mainly because the class-specific accuracy for the class *meadows* is only of 63.25, which is 35.15% less than the base for comparison. The QS–MV shows a good regularity in the class specific accuracies with an AA of 96.84% but the CA–WSHED–MV scheme is more regular among the different classes (AA = 97.94%).

Indian Pines dataset

The Indian Pines dataset has a low spatial resolution (20 m/pixel) and similar agricultural and forested regions. Sixteen classes of interest are available in the reference map. The total number of samples is presented in Section 2.9.3, Table 2.7. The number of samples used for training the SVM is randomly taken from the reference map as 10% of the available data as in [154, 58].

Table 3.4 gives the classification accuracies obtained by the pixel-wise SVM classifier, the segmentation scheme based on ISODATA [154] and the segmentation scheme based on hierarchical image segmentation (HSeg+MV) [58]. Best results are indicated in bold. Figure 3.8 shows from left to right the thematic map produced by the pixel-wise SVM classifier, the RCMG, the CA–Watershed segmentation map with watershed lines imposed for visualization, and the final classification map produced for the CA–WSHED–MV scheme.

The Hseg+MV scheme obtained the best OA (90.8%), AA (94.0%) and k coefficient (0.90%) for this scene, as shown in Table 3.4. The proposed scheme has the next highest AA (91.20%) showing also good regularity in the class specific results. The SVM+ISODATA scheme (with and without PR) shows a less uniform result in the CS accuracies, leading to a lower AA as compared to the Hseg+MV and CA–WSHED–MV schemes. This is because the *alfalfa* class and the *oats* class are poorly classified by the SVM+ISODATA scheme. Moreover, the post-regularization (fourth column in Table 3.4) reduces the accuracy of these two classes even further.

	SVM [154]	SVM+ISODATA [154]	SVM+ISODATA (PR) [154]	Hseg+MV [58]	CA–WSHED–MV
1-Alfalfa	32.65	12.24	6.12	92.3	**95.83**
2-Corn-notill	74.59	79.32	80.48	**90.5**	85.22
3-Corn-mintill	64.58	84.95	**88.02**	83.0	86.14
4-Corn	58.77	75.83	85.31	**95.7**	93.90
5-Grass/pasture	87.05	93.08	93.75	94.4	**96.26**
6-Grass-trees	92.72	94.80	96.88	**97.6**	97.27
7-Grass/mowed	29.17	91.67	91.67	**100**	**100**
8-Hay-windrowed	96.37	97.51	97.51	**99.5**	99.38
9-Oats	22.22	16.67	11.11	**100**	65.55
10-Soybean-notill	69.76	83.85	84.19	**92.1**	90.43
11-Soybean-mintill	79.21	93.16	**95.77**	84.1	75.28
12-Soybean-clean	75.41	85.17	89.87	**95.4**	95.19
13-Wheat	90.58	93.19	98.43	98.2	**99.57**
14-Woods	91.07	97.17	97.85	**98.6**	93.23
15-Bld-Grs-Trs-Drs	65.50	79.53	85.38	82.1	**92.28**
16-Stone-Steel	84.88	86.05	87.21	**100**	93.64
OA	78.76	88.53	90.64	**90.8**	87.95
AA	69.66	79.01	80.60	**94.0**	91.20
k	0.757	0.869	0.893	**0.90**	0.864

Table 3.4: Classification results for the CA–WSHED–MV scheme on the Indian Pines dataset and compared to SVM, SVM+ISODATA, SVM+ISODATA (PR), and Hseg+MV. Best results are indicated in bold.

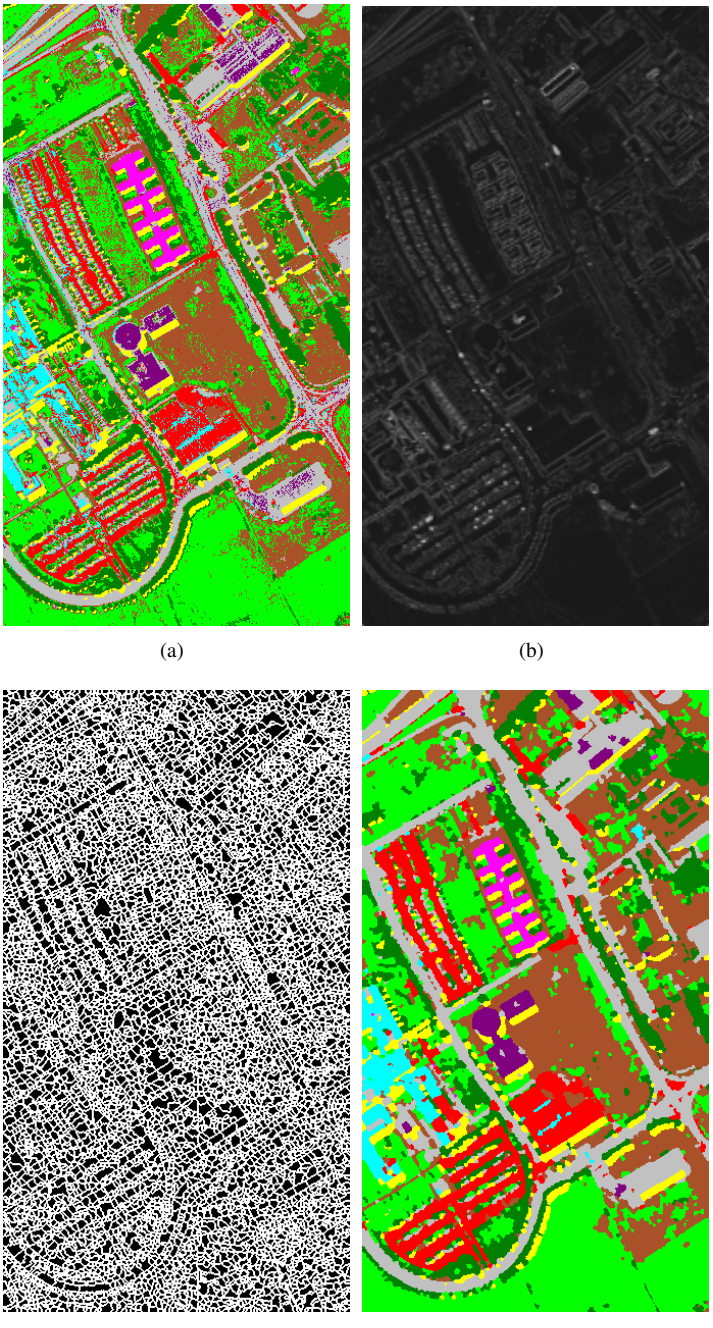
3.2.4 Final discussion

In this section we have described and analyzed the proposed spectral-spatial n-dimensional image classification scheme based on segmentation and Majority Vote (MV): CA–WSHED–MV. The scheme reduces the dimensionality of the hyperspectral image computing a Robust Color Morphological Gradient (RCMG), and then incorporates the spatial information by a watershed transform based on cellular automata (CA-Watershed). The classification was carried out by SVM combining the spectral and spatial results with a MV between the thematic map produced by the SVM and the segmentation map created by the CA–Watershed. This scheme was originally proposed in [155] but the watershed algorithm used in our scheme has the advantage that it does not create the watershed lines, eliminating the need of a post-processing of the segmentation map to assign a region to the pixel in the watershed lines.

The classification accuracy of the CA–WSHED–MV scheme was proven, producing good results on two urban areas acquired by the ROSIS-03 sensor (University of Pavia and Pavia City datasets), and one mixed agricultural and forested scene (Indian Pines) from the AVIRIS sensor. The classification accuracies were compared to spectral-spatial classification schemes

found in the literature based on the same framework (segmentation + majority vote): SVM+EM, SVM+EM (PR), SVM+ISODATA, SVM+ISODATA (PR) and Hseg+MV. In addition, other segmentation techniques such as k-means and quick-shift (QS) were used to obtain a reliable evaluation of the proposed scheme in the cases where the same spectral-spatial framework was not found in the literature for comparison.

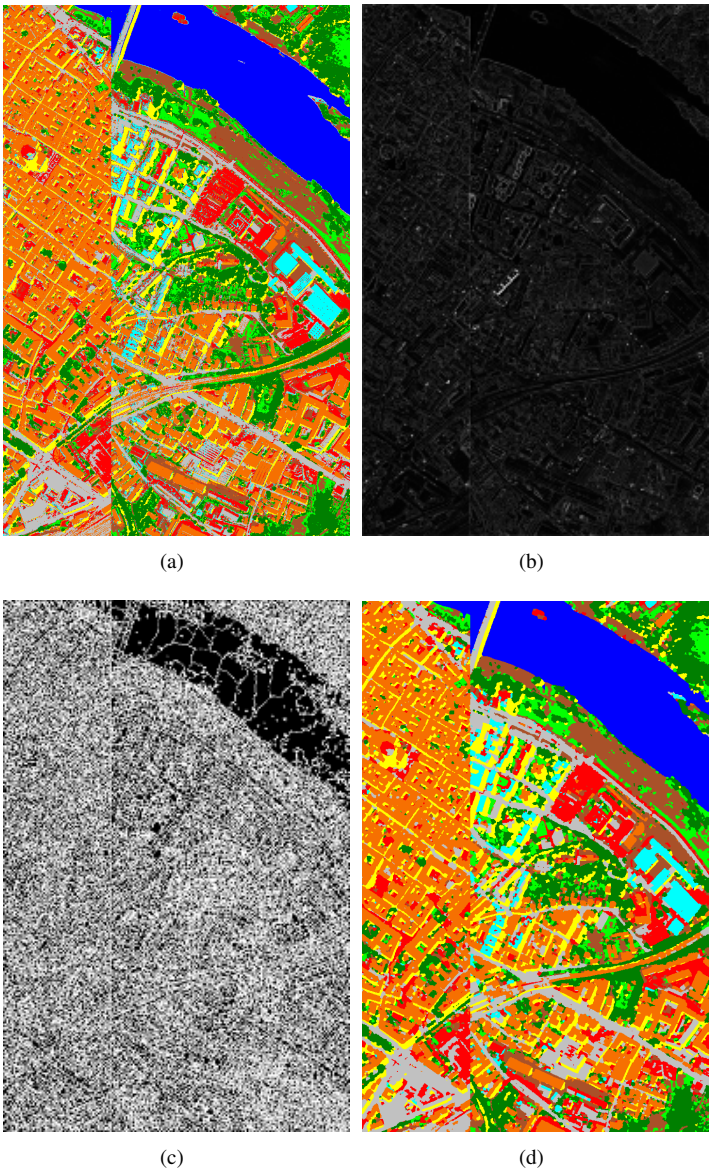
Experimental results have shown that the proposed scheme based on CA–Watershed improves the classification accuracies and performs better than other segmentation-based schemes in urban areas. By including spatial information from the closest neighbors, through a fixed or an adaptive post-regularization, small spatial structures may disappear being assimilated by larger structures [155]. However, the CA–WSHED–MV scheme has shown to be robust to this drawback, showing a good consistency in the class-specific accuracies. In the thematic maps produced by the proposed scheme, see Figure 3.8(d), Figure 3.9(d), and Figure 3.10(d), the spatial structures are more homogeneous as compared to the SVM pixel-wise thematic map.



(a) (b)

(c) (d)

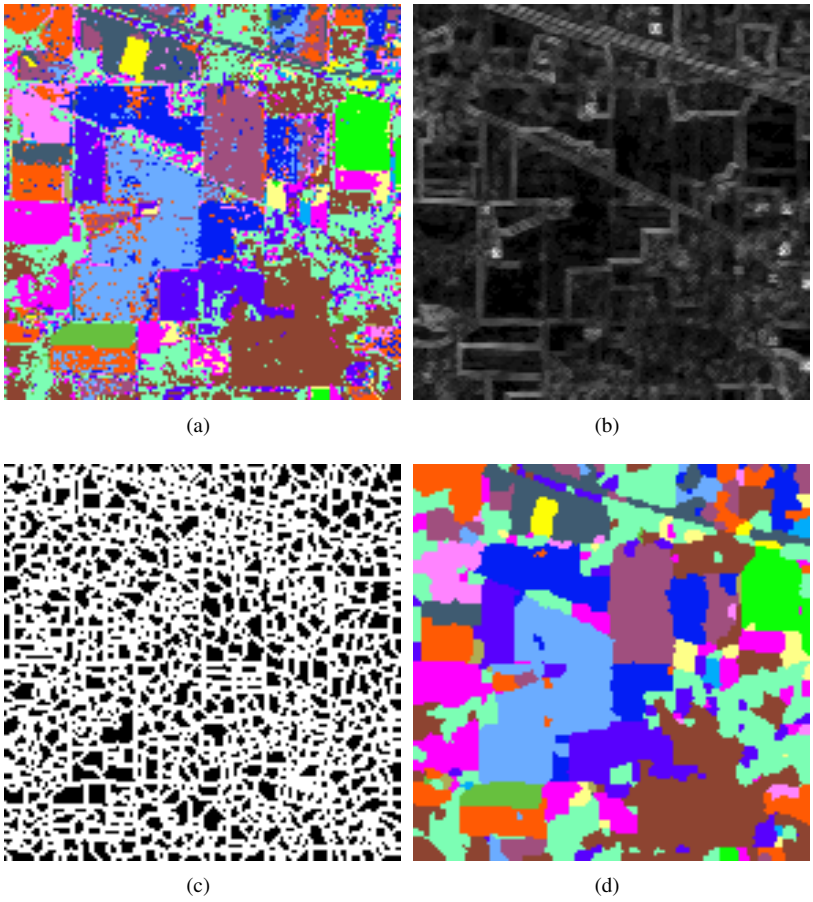
Figure 3.8: CA–WSHED–MV on the University of Pavia dataset: (a) SVM classification map, (b) RCMG, (c) CA–Watershed segmentation, (d) final classification by majority vote.



(a) (b)

(c) (d)

Figure 3.9: CA–WSHED–MV on the Pavia City dataset: (a) SVM classification map, (b) RCMG, (c) CA–Watershed segmentation, (d) final classification by majority vote.



(a) (b)

(c) (d)

Figure 3.10: CA–WSHED–MV on the Indian Pines dataset: (a) SVM classification map, (b) RCMG, (c) CA–Watershed segmentation, (d) final classification by majority vote.

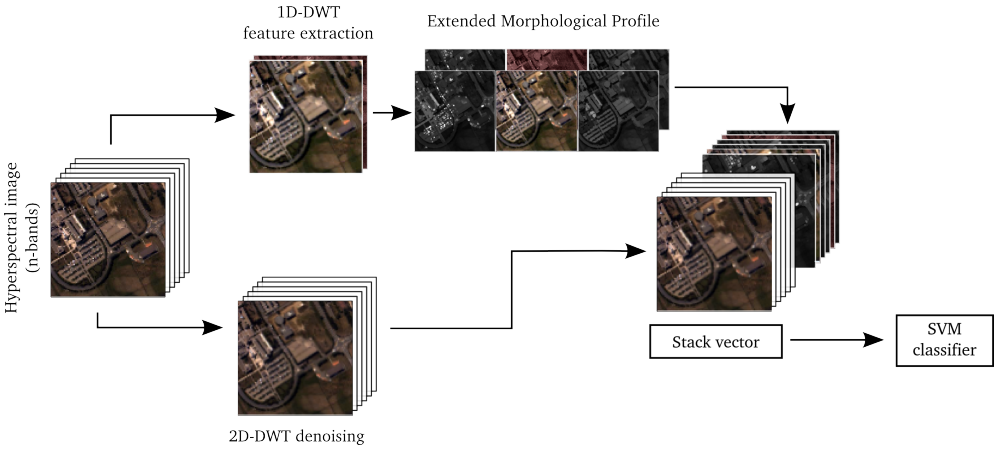


Figure 3.11: Flow-chart of the WT–EMP classification scheme based on feature extraction and denoising by wavelets, MM, and classification by SVM.

3.3 Scheme based on morphological profiles: WT–EMP

In this section we present a new proposal for efficient spectral-spatial hyperspectral image classification, based on wavelets and Mathematical Morphology (MM): WT–EMP scheme from now on. This second scheme extracts spatial information using morphological profiles and creates a new vector of features prior to the classification. This scheme is designed with the efficiency focused on producing good classification results in terms of accuracy, as well as an efficient computation on commodity hardware, such as a GPU.

Morphological transformations have been proposed to use spatial information in remote sensing classification [124, 15, 122, 125, 17]. In [124] the Morphological Profile (MP) was introduced and used in classification of panchromatic urban data. The MP, summarized below in Section 3.3.2, is defined as a set of n openings and n closings by reconstruction using a structuring element of growing size, creating a profile of $2n+1$ images including the original data. In order to apply the profile to multi-dimensional data, Palmason et al. [122] suggested creating the profile from the first principal component extracted from the hyperspectral data by DAFE and DBFE (see Section 2.2.1). The most two significant principal components were used in [15] as base images creating the so called EMP (a profile based on more than one image). This scheme mostly use spatial information extracted by the use of morphological operations. Fauvel et. at [54] proposed a scheme for urban land cover classification of hyper-

spectral images improving the EMP scheme by fusing spectral and spatial data. The EMP was created from PCA and was combined with the original spectral features of the hyperspectral image in a new, and even more extended, vector of features.

The WT–EMP scheme proposed in this work illustrated in Figure 3.11, combines the EMP with the spectral information but creating each morphological profile from wavelet coefficients. In addition, the hyperspectral image is denoised, also using wavelets, before creating the new vector of features via stack vectors, as described in Section 3.1.3. The main novelty of the proposed scheme is that the EMP is created from the wavelet coefficients obtained by a 1D-DWT in the spectral domain, and not from DAFE and DBFE [122, 15], PCA [54], ICA [121] and other FE methods proposed in [31].

The following sections describe the stages of this spectral-spatial classification scheme.

3.3.1 1D-DWT feature extraction

The first contribution of the proposed scheme is that the EMP is created from the wavelet coefficients obtained by a 1D-DWT (see Section 2.6 for a description of the wavelet transform). The spectral processing carried out in this work starts with a reduction in the number of bands of the hyperspectral image.

In this work, the Cohen-Daubechies-Feauveau 9/7 wavelet (CDF97) [44] has been used by applying a wavelet decomposition several times to reduce the dimensionality in the spectral domain of the image. The CDF97 coefficients used by the 1D-DWT are shown in Table 3.5. The effect is a reduction in the number of components of the pixel vector, (i.e., in the number of spectral bands of the image).

From the remaining band-coefficients of the hyperspectral image, the Extended Morphological Profile is created. The number of levels of decomposition of the original data depends on the desired size of the EMP. In the proposed scheme, the reduction is performed to a fixed level of decomposition to always produce 4-band coefficients. The objective is to create an EMP of a fixed size, as in [15, 54] where the EMP is created from the first PCs. The number

h_0	0.0378284555	h_3	0.3774028556	h_6	-0.1106244044
h_1	-0.0238494650	h_4	0.8526986790	h_7	-0.0238494650
h_2	-0.1106244044	h_5	0.3774028556	h_8	0.0378284555

Table 3.5: CDF97 low-pass filter coefficients used by the 1D-DWT [44].

of PCs is usually 3 or 4, corresponding to a certain amount of the cumulative variance (usually the 95% or 99%).

3.3.2 EMP from wavelet coefficients

Be W the transformed data by wavelets and be W_i the set of wavelet coefficients in the i-th band, the wavelet-based MP is defined as a set of openings and closings by reconstruction (see Section 2.5.1) using a structuring element (SE) of growing size:

$$\mathrm{MP}^{(n)}(W_i) = \{\gamma_r^{(n)}(W_i), \ldots, \gamma_r^{(1)}(W_i), W_i, \phi_r^{(1)}(W_i), \ldots, \phi_r^{(n)}(W_i)\},$$

where n is the number of morphological operations applied to the set W_i. The MP also includes the transformed wavelet data and results in a profile of $2n+1$ components.

The EMP is created from m MPs as follows:

$$\mathrm{EMP}_m^{(n)}(W) = \{\mathrm{MP}^{(n)}(W_1), \mathrm{MP}^{(n)}(W_2), \ldots, \mathrm{MP}^{(n)}(W_m)\},$$

where m is the number of band-coefficients created in the 1D-DWT step.

3.3.3 2D-DWT denoising

The second contribution of the proposed scheme is a denoising preprocessing in the spatial domain. The preprocessing performed is a wavelet shrinkage based on the separable 2D-DWT described in Section 2.6, applied to each band of the hyperspectral image. The hyperspectral image is denoised by soft thresholding before creating the new vector of features with the EMP, with the aim of removing the noise introduced by the sensor or by atmospheric phenomena.

We have used the Double-Density DWT presented in [148] which outperforms the standard 2D-DWT in terms of denoising. The Double-Density DWT is based on a set of three filters (one low-pass filter and two distinct high-pass filters) for perfect reconstruction. The filters are given in Table 3.6.

For image denoising, four levels of wavelet decomposition are applied to each hyperspectral band. Soft thresholding to the wavelet coefficients is then performed through all the subbands of details. By applying an inverse wavelet transform (IWT) after soft thresholding, the hyperspectral bands are reconstructed.

h	$g1$	$g2$
0.14301535070442	-0.01850334430500	-0.04603639605741
0.51743439976158	-0.06694572860103	-0.16656124565526
0.63958409200212	-0.07389654873135	0.00312998080994
0.24429938448107	0.00042268944277	0.67756935957555
-0.07549266151999	0.58114390323763	-0.46810169867282
-0.05462700305610	-0.42222097104302	0

Table 3.6: Set of filters used by the 2D-DWT for denoising [148].

3.3.4 Results

In this section the WT–EMP scheme proposed for efficient spectral-spatial classification of hyperspectral images is evaluated in terms of Overall Accuracy (OA), Average Accuracy (AA) and k coefficient, which are computed from the confusion matrix[3].

The scheme is created by applying the Cohen-Daubechies-Feauveau $9/7$ wavelet (CDF97) wavelet decomposition several times in the spectral domain. The number of levels of decomposition is performed to a fixed level to produce always 4-band coefficients. From the remaining 4-band coefficients, the EMP is created applying 4 openings and 4 closings by reconstruction, using a disk of radius of increasing size $r \in [1,3,5,7]$. Thus, the size of the EMP is 36 $(2n+1 \times 4$ band-coefficients$)$, independently of the size of the hyperspectral data. See Section 3.3.2 for more details. For image denoising, four levels of the Double-Density DWT are applied to the hyperspectral data as described in Section 3.3.3. The best threshold is found experimentally for each dataset. The denoised data and the EMP are combined via stack vectors (see Section 3.1.3). The whole process is computed in MATLAB[4].

The hyperspectral remote sensing scenes used in the experiments are two urban areas (Pavia University and Pavia City datasets) taken by the ROSIS-03 hyperspectral sensor and one hyperspectral image of a crop area (Indian Pines dataset) taken by the AVIRIS sensor. The number of training samples for each scene are selected as in [54, 41] to compare our scheme on equal terms (see Section 2.9.3, Table 2.6 and 2.7). In order to obtain a reliable evaluation of the results, the accuracies are calculated excluding the samples used for training.

[3]Part of these results have been published in P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Spectral-spatial classification of hyperspectral images using wavelets and extended morphological profiles," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 7, no. 4, pp. 1177–1185, 2014.

[4]The Double-Density DWT software used in these experiments is available at [147].

	Pavia University	Pavia City	Indian Pines
C	128	128	1024
γ	0.125	0.125	0.5

Table 3.7: Best parameters (C, γ) for the SVM determined for each hyperspectral image in the range $C = [1, 4, 16, 64, 128, 512, 1024]$, $\gamma = [0.5, 0.25, 0.125, 0.0625]$ by 5–fold cross-validation.

In this analysis the results are obtained executing the classification 100 times with different sets of training samples each time. Since the training samples were randomly selected, the results were calculated as the average of the 100 different values obtained for each execution. The classification is carried out using the LIBSVM library [35] with the RBF kernel. The parameters (C, γ) for training the SVM are determined by 5–fold cross-validation and are shown in Table 3.7.

We also conduct a standalone analysis of the different stages of the proposed scheme to ascertain whether the EMP created using wavelets and the contribution of the denoising as a preprocessing stage are efficient. The stages are named:

1. WT–EMP† is the stage considering only the EMP created from wavelet coefficients as input to the SVM classifier, as indicated in the upper branch in Figure 3.11.

2. WT–EMP‡ is the stage considering only the Double-Density DWT applied to the hyperspectral data, see bottom branch in Figure 3.11.

In addition, we have evaluated a variant of the proposed scheme, hereinafter WT–EMP*, by creating the stack vector from the original data without denoising and the EMP created from wavelet coefficients. It is similar to the scheme proposed by Fauvel et al. [54]. The different stages of the proposed scheme are also evaluated separately in presence of noise.

The classification results are compared to those obtained by the pixel-wise SVM classifier and other spectral-spatial classification schemes, which results are available in the literature for the datasets used in these experiments: EMP-PCA [58], EMP-ICA [121], Spec-EMP [54], WSHED+MV [58], HSeg+MV [58], DB-DB [108], EMD-DWT [68], KPCA$_p$ [108] and NW-NW [108]. A brief description of these schemes is given in the following sections where the mentioned schemes are included for comparison.

Pavia University dataset

The University of Pavia is a moderately dense urban area with nine classes of interest available in the reference map. This dataset has 103 spectral bands.

For this scene, the spectral dimensionality is reduced 6 times by the 1D-DWT, producing a EMP of 36 features. It has been experimentally found that the best parameters for denoising this hyperspectral image by the Double-Density DWT [148] was by using a threshold of 4. The new pixel vector is stacked, resulting in 139 features which are used to classify the image by the SVM. Table 3.8 gives the classification accuracies obtained by the pixel-wise SVM classifier, EMP-PCA, Spec-EMP, DB-DB, EMD-DWT, WT–EMP† stage, WT–EMP*, as well as the proposed WT–EMP scheme. In the following, these schemes are briefly described:

– EMP-PCA [58] is a spectral-spatial classification scheme based on EMPs where the morphological profiles are created from the first principal components extracted by PCA.

– Spec-EMP [54] is created as the EMP-PCA scheme and stacks the original spectral information within the EMP.

– DB-DB [108] is a spectral-spatial classification scheme based on EMAPs where each attribute profile is created from the first components extracted by DBFE. The attributes considered in this scheme are the area and the standard deviation. The EAP$_a$ created from the area and the EAP$_s$ created from the standard deviation are concatenated via stack vectors to create the EMAP, which is reduced a second time by DBFE.

– EMD-DWT [68] uses spatial and spectral features combining Empirical Mode Decomposition (EMD) with wavelets, in order to generate the smallest set of features that leads to the best classification accuracy. Thus, data fusion is implicitly performed by a sequence of techniques applied in the spatial (EMD) and the spectral (DWT) domain.

The best results in Table 3.8 are indicated in bold. All results are obtained with the number of training and test samples described in Section 2.9.3, except for the EMD-DWT scheme that considers 1% more of training samples, as reported in their work.

By considering only the EMP created from the wavelet coefficients (fourth column in Table 3.8) the improvement of the OA is 12.89%, reaching 93.9%, as compared to the pixel-wise SVM. The WT–EMP† stage as compared to the EMP-PCA improves the OA in 14.8%. All classes except the *bare soil* class are improved in this stage when compared to the SVM classifier, unlike the EMP-PCA, where neither the *gravel* class and *shadows* class are improved (see Table 3.8). The EMP from 1D-DWT (WT-EMP†) performs better than the EMP from PCA (EMP-PCA) in terms of classification accuracy for this dataset.

	SVM [54]	EMP-PCA [58]	WT-EMP†	DB-DB [114]	spec-EMP [54]	WT-EMP*	EMD-DWT [68]	WT-EMP
1-Asphalt	84.5	94.5	97.6	95.43	95.3	98.0	—	**98.8**
2-Meadows	66.2	72.8	91.6	95.88	73.5	97.2	—	**98.8**
3-Gravel	72.0	53.2	95.7	**100**	65.9	96.2	—	98.8
4-Trees	98.0	98.9	98.9	90.94	**99.2**	99.0	—	**99.2**
5-Metal sheets	99.5	99.5	99.7	**100**	99.5	99.7	—	99.9
6-Bare Soil	93.1	58.1	89.9	98.41	84.1	96.3	—	**98.5**
7-Bitumen	91.2	96.1	96.9	**99.18**	97.2	97.5	—	99.0
8-Bricks	92.3	95.3	96.1	**98.65**	96.1	96.7	—	98.0
9-Shadows	96.6	91.2	**99.9**	**99.96**	93.5	**99.9**	—	**99.9**
OA	79.5	79.1	93.9	97.89	83.5	97.4	**99.0**	98.8
AA	88.1	84.3	96.3	97.60	89.4	97.8	—	**99.0**
κ	0.74	0.73	0.91	0.972	0.79	0.96	0.97	**0.98**

Table 3.8: Classification results for the WT–EMP scheme on the University of Pavia scene compared to SVM, EMP-PCA, EMAP, spec-EMP, EMD-DWT. Note that in EMD-DWT 1% more training samples are used in the classification. The best accuracies are indicated in bold. The † indicates that the EMP is used as the only input to the SVM and * that the stacked vector is built from the original data and the EMP.

(a) (b) (c)

Figure 3.12: WT–EMP on the University of Pavia dataset: (a) true color representation, (b) SVM classification map, and (c) WT–EMP classification map.

The results of the DB-DB, which uses different attributes to create an extended multi-attribute profile, are higher, outperforming the aforementioned schemes. Two classes, *gravel* and *metal sheets*, have a class-specific accuracy of 100%. However, the accuracy of the *trees* class falls 7% as compared to the pixel-wise SVM classifier.

By incorporating the original spectral data into the EMP via stack vectors (seventh column in the table), all the class-specific accuracies are improved. This is the same approach as the one proposed in [54] (Spec-EMP) but using 1D-DWT instead of PCA to compute the EMP. The WT-EMP* improves the OA 3.5% and the AA 1.5% as compared to the WT-EMP† stage. These results are better not only over the image as a whole, but also over each class-specific accuracy. This indicates that the classification using the EMP created from the features extracted by wavelets performs better when the spectral information is incorporated to the scheme.

The OA of 97.4% obtained by the WT-EMP* approach is high, but there is room for improvement by applying the 2D-DWT denoising to each hyperspectral band. If the original spectral data if denoised before combining the results with the EMP (ninth column in Ta-

ble 3.8), the best result in terms of class-specific accuracy, AA (99.0%) and k coefficient (0.98) are obtained. These result correspond to the proposed scheme. In the WT–EMP scheme, all the class-specific accuracies are among the highest, particularly for the *asphalt*, *meadows*, *trees* and *bare soil* classes, where some of the other schemes presented a number of difficulties in improving those classes. Compared to the EMD-DWT scheme, the OA values are close in both schemes (0.2% higher for the EMD-DWT scheme) with the advantage that the WT-EMP is computationally less costly.

Figure 3.12 shows from left to right, the true color representation, the SVM classification map, and the WT–EMP classification map.

Pavia City dataset

The Pavia City dataset is a dense urban area with 102 spectral bands and nine classes of interest. For this image, a threshold of 4 for denoising the hyperspectral is used, and the spectral dimensionality is reduced 5 times by the 1D-DWT. Thus, the size of the pixel vectors is 138 (123 spectral bands + 36 features from the EMP).

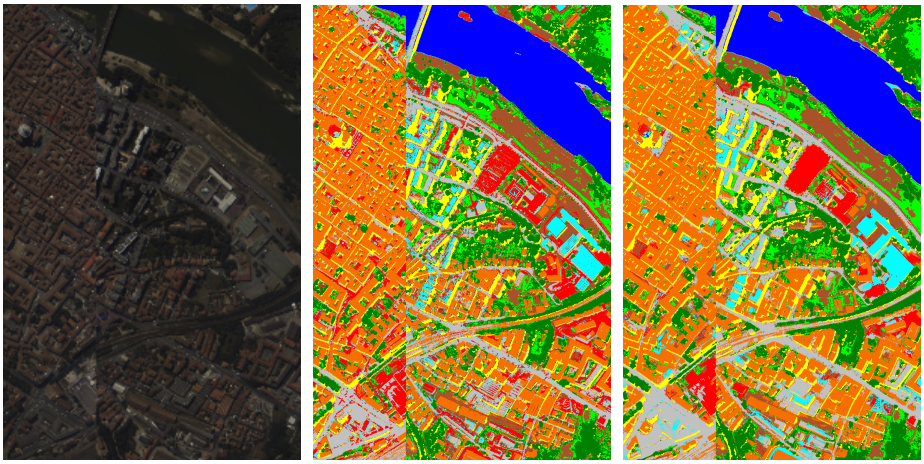
The number of training samples used in the experiments is described in Section 2.9.3, Table 2.6. The classification accuracies obtained by the propose scheme, the EMP-ICA [121], the Spec-EMP [54], as well as WT–EMP† stage and the WT–EMP* approach are given in Table 3.9. The accuracy of the pixel-wise SVM is included as a basis for comparison. The EMP-ICA is a spectral-spatial classification scheme based on EMPs where the morphological profiles are created from the first principal components extracted by ICA. In the Spec-EMP scheme, the morphological profiles are created from PCA and then they are stacked to the original spectral information.

It can be observed in the table that the base classification accuracy is already fairly high (97.7%) using only the spectral information. The most significant improvements are achieved by the WT–EMP scheme. Comparing the third and fourth columns (two similar approaches), 7 from the 9 classes are better classified with the WT–EMP† stage as compared to the EMP-ICA scheme. The same happens when analyzing the results obtained by the WT–EMP* approach, where all the class specific accuracies are improved. This indicates that the EMP created from the wavelet coefficients is working properly and it is the factor that contributes most to the scheme.

The best OA (99.7%), AA (99.3%), and k coefficient (0.99) are obtained for the proposed scheme, when the denoised data are stacked with the EMP created from wavelets. Figure

	SVM [54]	EMP-ICA [121]	WT-EMP†	spec-EMP [54]	WT-EMP*	WT-EMP
1-Water	98.3	99.5	**99.9**	98.6	**99.9**	**99.9**
2-Trees	91.2	91.7	95.7	93.5	**97.8**	97.4
3-Meadow	96.8	85.3	98.0	95.9	98.5	**99.1**
4-Bricks	88.4	99.2	99.7	98.8	99.7	**99.8**
5-B. Soil	97.0	98.4	99.8	99.4	99.8	**99.9**
6-Asphalt	96.3	98.6	98.7	98.4	99.0	**99.5**
7-Bitumen	96.0	98.1	97.1	98.2	98.3	**98.5**
8-Tiles	99.4	99.8	99.8	99.8	99.8	**99.9**
9-Shadows	99.9	99.2	99.9	99.9	99.9	**100**
OA	97.7	98.8	99.5	**99.7**	**99.7**	**99.7**
AA	95.6	96.6	98.8	98.1	99.2	**99.3**
κ	0.96	–	**0.99**	0.98	**0.99**	**0.99**

Table 3.9: Classification results for the WT-EMP scheme on the Pavia City compared to SVM, EMP-ICA, and spec-EMP. The best accuracies are indicated in bold. The † indicates that the EMP is used as the only input to the SVM and * that the stacked vector is built from the original data and the EMP.



(a) (b) (c)

Figure 3.13: WT–EMP on the Pavia City dataset: (a) true color representation, (b) spectral classification map with SVM, (c) WT-EMP classification map.

3.13 shows from left to right the true color representation, the spectral classification map with SVM and the WT-EMP classification map.

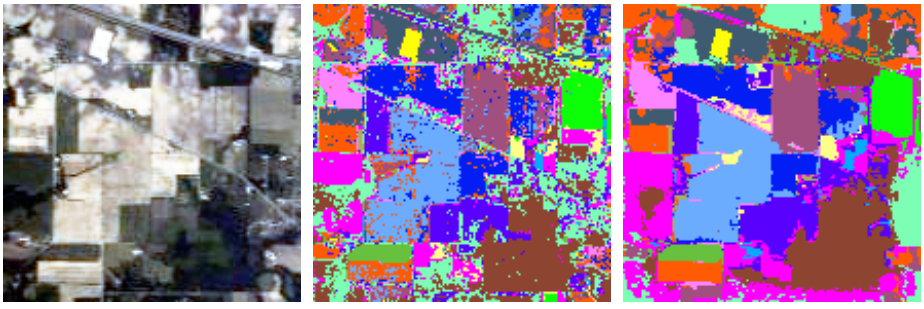
Indian Pines dataset

The third scenario used in the experiments is the Indian Pines dataset, with the sixteen classes extracted from its reference map. This dataset has 220 spectral bands. A threshold of 0.01 for denoising the hyperspectral data obtained the best classification accuracy in this image. The spectral dimensionality is reduced 6 times by the 1D-DWT, creating stack vectors of 256 features (220 spectral bands + 36 features from the EMP). The number of samples used for training the SVM is the same as in [58, 108]. The training samples are randomly taken from the reference map, described in Section 2.9.3.

Table 3.10 gives the classification accuracies obtained by the pixel-wise SVM classifier, by the WT–EMP scheme, and two MM-based schemes, KPCA$_p$ and NW-NW [108]. The best results are indicated in bold.

The MM-based schemes are based on EMAPs, creating the profile using the area and standard deviation attributes. The morphological profiles are created on the principal components extracted by kernel-PCA (KPCA$_p$) and NWFE (NW-NW) feature extraction techniques, respectively. The first has 325 features that are used as input to the SVM classifier, and the NW-NW uses a feature extraction technique a second time to reduce the EMAP to 30 features.

Two segmentation-based schemes, WSHED+MV and HSeg+MV, are also included for a broader comparison. The results of these two schemes are taken from [58] and not from [155], as the results presented by Fauvel were produced in [58] by the same number of training samples used in this experiment.



(a) (b) (c)

Figure 3.14: WT–EMP on the Indian Pines dataset: (a) true color representation, (b) spectral classification map with SVM, (c) WT-EMP classification map.

	SVM [58]	WSHED+MV [58]	Hseg+MV [58]	KPCA$_p$ [108]	NW-NW [108]	WT–EMP
1-Alfalfa	74.4	**94.9**	92.3	94.87	94.87	91.5
2-Corn-notill	78.2	**94.2**	90.5	71.41	90.24	83.7
3-Corn-mintill	69.6	78.1	83.0	92.73	**98.85**	85.7
4-Corn	91.9	88.6	95.7	96.20	**97.28**	94.6
5-Grass-pasture	92.2	95.1	94.4	93.96	**95.52**	92.3
6-Grass-trees	91.7	98.8	97.6	98.42	**99.56**	97.2
7-Grass-pasture-mowed	100	**100**	**100**	90.91	**100**	**100**
8-Hay-windrowed	97.7	99.5	99.5	98.63	**99.54**	99.1
9-Oats	100	**100**	**100**	**100**	**100**	**100**
10-Soybean-notill	82.0	**96.3**	92.1	87.39	86.27	84.4
11-Soybean-mintill	58.0	68.8	84.1	87.84	**94.58**	74.9
12-Soybean-clean	87.9	90.8	95.4	**96.35**	93.61	84.6
13-Wheat	98.8	**99.4**	98.2	99.38	99.38	99.2
14-Woods	93.0	**99.7**	98.6	95.90	92.36	87.7
15-Bldg-Grass-Trees-Drives	61.5	69.4	82.1	96.36	**99.09**	94.3
16-Stone-Steel-Towers	97.8	95.6	**100**	**100**	**100**	96.9
OA	78.2	86.6	90.8	90.20	**94.2**	86.6
AA	86.9	91.6	94.0	**96.64**	96.3	92.3
κ	0.75	0.85	0.90	0.888	**0.93**	0.848

Table 3.10: Classification results for the WT–EMP scheme on Indian Pines and compared to SVM, WSHED+MV, HSeg+MV, KPCA$_p$, NW-NW. The best accuracies are indicated in bold.

As can be seen from Table 3.10, the best OA (94.2%) and k value (0.93) are achieved by the NW-NW scheme, and the best AA (96.64%) for the KPCA_p approach, both based on attribute profiles. Our proposed scheme gives an OA similar to the WSHED+MV scheme (86.6%), showing difficulties in classifying an image with low spatial resolution. However, the CS of the *grass-pasture-mowed* and *oats* classes are among the best, with a value of 100%. Even though, the EMP is considered an effective approach for combining spectral and spatial information [58, 108], the EMAP gives more accurate results in classifying the Indian Pines dataset.

Figure 3.14, shows from left to right, the true color representation of the Indian Pines scene, the spectral thematic map with SVM, and the WT-EMP thematic map.

Experimental results in presence of noise

This section analyzes how each one of the stages of the proposed scheme works separately in the presence of noise. The two stages are WT–EMP‡, in which the Double-Density DWT is applied to the hyperspectral data, and the WT–EMP† stage, which only considers the EMP created from the wavelet coefficients as input to the SVM classifier. The experiments were carried out with the same configuration as described at the beginning of Section 3.3.4.

The University of Pavia dataset was corrupted by additive white Gaussian noise (AWGN) [98] with a peak signal-to-noise ratio (PSNR) of 10, 16 and 20 dB.

	SVM	WT–EMP‡	WT-EMP†	WT-EMP
1-Asphalt	56.5	90.2	89.4	97.6
2-Meadows	40.2	99.1	85.8	97.8
3-Gravel	15.9	99.6	83.9	98.4
4-Trees	78.5	80.5	91.9	96.4
5-Metal	91.9	99.8	99.5	99.6
6-Bare Soil	33.7	99.9	90.8	99.2
7-Bitumen	16.3	99.9	89.5	98.9
8-Bricks	49.2	98.2	85.8	97.4
9-Shadows	90.8	67.2	98.6	99.0
OA	45.9	96.0	88.0	97.3
AA	52.6	92.7	90.6	98.3
κ	0.33	0.94	0.84	0.97

Table 3.11: Classification results for WT–EMP in presence of noise for a PSNR of 10 dB. University of Pavia dataset.

PSNR	SVM	WT–EMP‡	WT-EMP†	WT-EMP
10 dB	45.9	96.0	88.0	97.3
16 dB	55.5	94.4	93.5	99.0
20 dB	60.0	93.1	94.3	99.2

Table 3.12: Overall accuracy for WT–EMP in presence of noise for a PSNR of 10, 16 and 20 dB. University of Pavia dataset.

The classification results are shown in Table 3.11 for the case of 10 dB in terms of OA, CS, AA and k coefficient. The first thing to note in the results using the image with noise is that the standalone spectral SVM classification obtained a very low accuracy, particularly for the *gravel* and *bitumen* classes. These low values indicate that the image has been considerably corrupted by the AWGN. Therefore, the PSNR of 10 dB is a reasonable ratio of noise to test the proposed scheme.

On the one hand, the WT–EMP‡ stage gives an OA of 96.0%, which is double the accuracy obtained by the SVM. The *gravel* and *bitumen* classes are greatly improved, up to 83.6% in the case of the second mentioned class. The good results obtained in this stage indicate the effectiveness of the filters used for the 2D-DWT image denoising [148]. On the other hand, the classification results for the WT-EMP† also improved respect to the SVM classification. The EMP created from wavelet coefficients is able to improve the OA up to 42% reaching to a total of 88.0%. However, it can be observed that the class-specific accuracy of classes *meadows*, *gravel* and *bitumen* are among the worst classified. The EMP shows some robustness in the presence of noise although there is still margin for improvement. In fact, the contribution of the two stages, when they are joined via stack vectors, leads to the best results in OA (97.3%), AA (98.3%), k coefficient (0.97), as well as all the class-specific accuracies, as can be observed in Table 3.11 (fifth column). The OA of 97.3% is close to the OA obtained by the WT–EMP over the original image of the University of Pavia, that is, without additional noise.

Table 3.12 shows a summary of the OA results for the three PSNR ratios used in the experiment. The same behavior is observed. While the SVM classification is affected by the presence of noise, the proposed scheme is more robust.

3.3.5 Final discussion

In this section we have presented a new proposal for efficient spectral-spatial hyperspectral image classification based on wavelets and Mathematical Morphology (MM). The proposed

WT–EMP scheme was designed with the efficiency focused on producing good classification results in terms of accuracy, as well as an efficient computation on commodity hardware, such as a GPU. This scheme extracts spatial information by creating an EMP from wavelet coefficients. In addition, the hyperspectral image is denoised, also using wavelets, before creating the new vector of features by data fusion via stack vectors. The main novelty of the proposed scheme is that the EMP is created from the wavelet coefficients obtained by a 1D-DWT in the spectral domain, instead of with other commonly used techniques such as PCA, ICA, DAFE or NWFE. The main benefits of this scheme are the good classification results obtained by the SVM classifier and the low computational requirements.

The classification results produced by the WT–EMP scheme were evaluated in terms of accuracy on three well-known scenes: two were acquired by the ROSIS-03 sensor (Pavia University and Pavia City datasets), and the third scene (Indian Pines) by the AVIRIS sensor. By comparing the classification accuracies to other spectral-spatial classification schemes based on MM and segmentation: EMP-PCA, EMP-ICA, Spec-EMP, WSHED+MV, HSeg+MV, DB-DB, EMD-DWT, KPCA$_p$ and NW-NW, the proposed scheme proved its efficiency, particularly in urban areas.

In addition, the different stages of the proposed scheme were evaluated separately in the presence of noise. Experimental results have shown that the scheme produces best classification accuracies when the contributions of both stages are combined.

3.4 Conclusions

In this chapter we have presented our two proposals for efficient spectral-spatial n-dimensional image classification, one based on segmentation (CA–WSHED–MV), and the second based on Mathematical Morphology (MM) and wavelets (WT–EMP). The general framework of the spectral-spatial classification schemes, as well as the common data fusion strategies for joining the spectral and the spatial information of such schemes, were reviewed in this chapter.

The CA–WSHED–MV scheme incorporates the spatial information employing a watershed transform based on cellular automata, and combines them with the spectral results of the classifier with a majority vote. The WT–EMP scheme extracts spatial information using morphological profiles and creates a new vector of features prior to classification. This scheme

was designed with the aim of producing good classification results in terms of accuracy and efficient computation on commodity hardware, such as a GPU.

The classification accuracy of both schemes was evaluated on real hyperspectral images taken by the ROSIS-03 and the AVIRIS sensor, and the results thereof were compared to classification schemes based on segmentation, MM and wavelets. Experimental results have shown that the proposed schemes produce good classification accuracies in urban areas, and perform better than other segmentation-based and MM-based schemes, but a number of problems were encountered classifying the crop area in the Indian Pines dataset. The OA for Indian Pines was 87.9% and 86.6% for the first and the second scheme, respectively.

The CA–WSHED–MV scheme obtained good classification results in terms of accuracy on the urban scenes of Pavia University (OA 95.88%) and Pavia City (OA 99.20%). The WT–EMP proposal was even better on the aforementioned datasets with an OA of 98.9% and 99.7%, respectively.

CHAPTER 4

TECHNIQUES AND STRATEGIES FOR EFFICIENT GPU COMPUTING

4.1 Introduction

In this chapter we present the techniques and strategies applied in this work for efficiently computing the proposed schemes on GPU. First, we describe general strategies for parallel processing related to data partitioning, data movement and data packing. We then go on to highlight the challenges of GPU computing.

We present the Block–Asynchronous (BA) strategy proposed in this thesis which can be easily adapted to solve different problems, such as the watershed transform, Connected Component Labelling (CCL), and Mathematical Morphology (MM) operations that require iterative computation. This method reduces the number of global synchronization points, allowing efficient exploitation of the memory hierarchy of the GPU.

The BA approach is applied to compute the asynchronous CA–Watershed used in the CA–WSHED–MV scheme, and to compute opening and closing by reconstruction used in the WT–EMP scheme. The BA can be also applied for CCL in greyscale attribute filtering, which is an advanced morphological operation used in schemes based on attribute profiles.

The strategies for efficient multi-class SVM classification on GPU are also described in this chapter. The techniques are independently analyzed, comparing the execution times to efficient CPU implementations, parallel multi-threaded CPU implementations using OpenMP and to GPU proposals found in the literature.

4.1.1 Parallel patterns

A number of parallel problems can be thought of as patterns [40]: loop-based, fork/join, tiling/grids, divide and conquer, and split and merge parallel patterns.

Loop-based patterns are found in problems where instructions are executed sequentially. If there are no inter-loop dependencies (ideal case), and one iteration can be executed independently from the other, the loop-based pattern is one of the easiest to parallelize. The iterations are split among the available processors. The OpenMP fork-join model and the Single Instruction, Multiple Thread (SIMT) parallel programming model of CUDA adapt perfectly to this parallel pattern. The distribution of the work should be based on the threads available in the CPU in the case of OpenMP, and based on the number of Streaming Multiprocessor (SM) multiplied by the maximum number of blocks each SM can support in the case of CUDA [40].

The fork-join parallel programming model of OpenMP, described in Section 2.8.1, is another parallel pattern. This is used in serial programming where the work can be distributed to different processors. The tasks are executed in parallel but they do not have to be the same for each processor.

Most parallel problems make use of the tiling/grids pattern, where the problem is broken into smaller parts that are concurrently and locally solved. CUDA provides a grid model of one, two and three dimensions that is used to map the data of the problem on different blocks of threads. By way of example, Figure 4.1(a) shows a grid of 16×16 threads divided into blocks of 4×4 threads. In some cases, the local computation also involves neighboring data. Therefore, to solve a problem in these cases, such as the cellular automata evolution, we must extend the block with a border, resulting in a block with an apron, as illustrated in Figure 4.1(b). Thus, the border of one region overlaps the adjacent regions [131]. Memory coalescing techniques are often used in conjunction with tiling techniques to allow CUDA devices to exploit their performance potential by efficiently using the global memory bandwidth [90].

The divide and conquer pattern is also used for breaking down large problems into smaller tasks that can be individually computed. Taking together all the computed tasks, the final solution is obtained. This is the typical pattern used on recursive algorithms, which have been supported in CUDA since the Fermi architecture. However, each recursive call stacks data into the global memory of the GPU, which is much slower than using registers. Most recursive algorithms can be rewritten as iterative processes; hence, using iterative solutions where possible will generally perform much better and run on a wider range of GPUs [40]. If we divide the problem in smaller tasks only once and are able to directly compute each task,

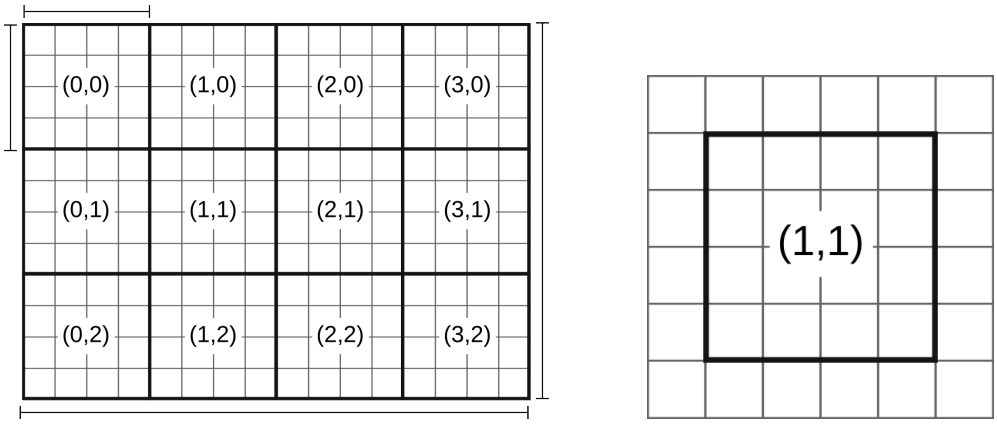


Figure 4.1: Example of grid/tiling model in CUDA: (a) grid of 16×16 threads divided into blocks (tiles) of 4×4 threads, and (b) a block extended with a border (apron) of size one.

the divide and conquer pattern is similar to the tiling/grids pattern [90]. For a more complete and detailed description of each parallel pattern, see [40, 90].

The split and merge pattern [85] is an extension of the divide and conquer parallel pattern used in problems where the tasks cannot be divided into independent computations. Accordingly, tasks are first split and partially computed in parallel and then, in a second step, the results are merged. This technique was applied in the context of general purpose computation on GPUs in [6].

The tiling/grids pattern is used in most of the CUDA implementations developed in this thesis. The Block–Asynchronous strategy, the RCMG, the 1D-DWT, the 2D-DWT and the multi-class SVM classification implementations are based on this parallel pattern. In particular, the 2D-DWT is based in the divide and conquer parallel pattern, with only one division of the problem into smaller tasks. Some tasks such as soft-thresholding, and intermediate kernels created for processing data at pixel level use a loop-based pattern. OpenMP implementations are also based on this pattern.

4.1.2 Data packing

With the objective of minimizing the data transfers between the CPU and the GPU, data can be packed into one, four or eight bytes. For example, eight Boolean values can be packed into

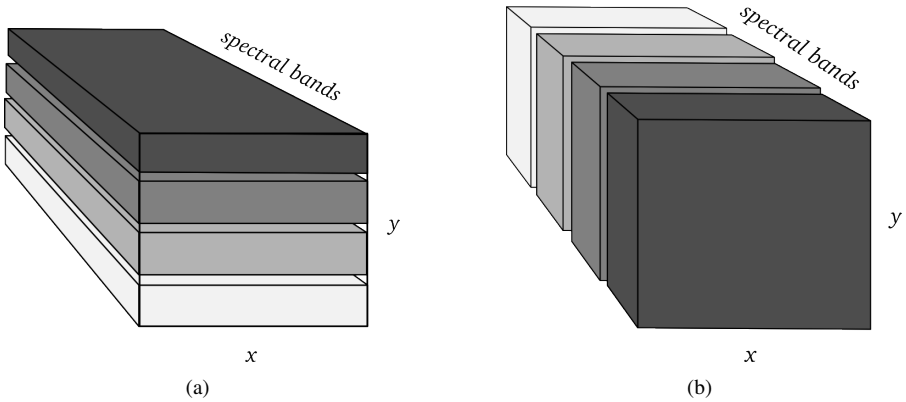


Figure 4.2: Hyperspectral data partitioning techniques: (a) spatial-domain partitioning, (b) spectral-domain partitioning.

one byte, one value per bit, and the RGB values of a color image can be packed into three bytes. Thus, by packing data the number of global memory accesses is reduced when data are required in a kernel. In addition, if data from neighboring pixels are also required, the global memory accesses are even more reduced. This technique is used in the CA–Watershed based on Block–Asynchronous computation.

4.1.3 Spectral and spatial partitioning

In hyperspectral imaging, two types of data partitioning techniques can be exploited: one in the spectral-domain and other in the spatial-domain [127]. These two data partitioned strategies are shown in Figure 4.2 where the $[x, y]$ front plane in each figure corresponds to the first band of the image. In the spatial-domain partitioning, the pixel vectors are kept as a whole, as shown in Figure 4.2(a). In the spectral-domain partitioning, the image is subdivided into slices comprising contiguous spectral bands, as shown in Figure 4.2(b). The data partitioning approach strongly depends on the processing techniques applied to analyze the hyperspectral image. For example, if all the spectral features of a pixel are required at once to make a computation, as in the case of unmixing processing, the spatial-domain partitioning is desired, as each pixel vector can be assigned to a different block of threads. However, if the computation can be individually done in each band, as in the case of morphological operations,

the spectral-domain partitioning can take advantage of the fine-grained level of parallelism of CUDA.

In this work, different hyperspectral data partitioning strategies and thread block arrangements are studied in order to effectively exploit the memory and computing capabilities of the GPU architecture.

4.1.4 Challenges of GPU computing

In order to maximize the performance on the GPU, different aspects must be taken into consideration. This section highlights some key GPU performance issues that can be grouped in three main rules for GPGPU programming: 1) minimizing data transfer between CPU and GPU, 2) giving the GPU enough work to do, and 3) focusing on data reuse within the GPU [53].

Most of the important issues are related to an efficient use of the memory hierarchy. Data movement from the CPU to the GPU global memory is through the PCIe bus, which is very slow (8GB/s) in comparison to the peak bandwidth of the global memory (250 GB/s) and the shared memory (2.5 TB/s). Therefore, data should be copied to the GPU once and be reused as much as possible.

As memory operations are executed per warp, it is important to known how data are going to be accessed, which mainly depends on how we implement the algorithm. By aligning accesses to consecutive memory locations in global memory, we ensure coalesced accesses (see CUDA parallel programming model in Section 2.8.2), minimizing load/store operations into the fewest possible memory transactions. Data in constant memory can be broadcast if the same value is accessed by all the threads within the warp. And texture memory has special features, such as filtering, and a dedicated cache memory which can improve performance when threads access values in some regular spatial neighborhood, for example in a 3×3 window. Therefore, data may need to be rearranged or distributed in a different way in order to obtain maximum performance of the GPU memory.

Data packing, as described in Section 4.1.2, can be used to minimize data transfers, and different data partitioning strategies, see Section 4.1.3, as well as thread block arrangements can improve the performance of the GPU.

The size of a block of threads should be a multiple of the warp size (32). In most of the cases, it is often necessary to perform different kernel launch configurations (threads per block and blocks per grid) to find the best one that maximizes the device utilization [118].

The limit in the hardware resources required to execute a kernel can be achieved by the size of the block, as described in Section 2.9.1. The occupancy, which can be expressed as the number of concurrent threads per SM can be used to measure the performance on the GPU. The amount of occupancy required to reach the maximum performance depends on the code. If the code is limited by memory accesses, the highest number of concurrent blocks per SM is desired in order to hide the latency of those accesses.

Instructions are also executed per warp, so the 32 threads within the warp take the same path in the code. In conditional control flows, such as if/else statements, if at least one thread within the warp takes a different path, we have a warp divergence in the code, which potentially affect performance [118]. Hence, care must be taken to write a coherent control flow code. In some cases, sequential code must be rewritten to expose sufficient parallelism to the GPU, and arithmetic instructions reordered and split for balancing the workload among memory accesses and computation.

Regarding data reuse within the GPU, if data accesses have sufficient locality, redundant accesses to global memory must be minimized by exploiting the shared and the cache memories. The main use for the shared memory is to reuse data within a block and share data among the threads of the same block. This memory is managed by the programmer, and the effective use of this memory can lead to speedups of $7\times$ compared to global memory implementations [40]. This is a significant challenge when programming for the GPU as data in this on-chip memory are only shared for the threads of the same block. Sharing data among all the threads must be through the global memory, coming into play the aforementioned key GPU performance issues related to an efficient use of the memory hierarchy.

Atomic memory operations may be necessary to avoid race conditions in applications where multiple threads need to access the same memory space simultaneously for reading and writing data, and there is no other way of synchronizing. CUDA provides atomic operations for ensuring that all concurrent updates to the same memory location can be performed atomically, so that all threads can see those updates. However, the atomic operations can degrade the performance if many threads try to carry out an atomic operation on a small number of memory locations.

A extensive range of skills are needed to be an effective parallel programmer [90]. These skills can be grouped into computer architecture, programming models and compilers, algorithm techniques and domain knowledge. The CUDA C programming guide [119] and the

CUDA C best practices guide [118] have a comprehensive manual and numerous tips and key issues for increasing the computational throughput of NVIDIA GPUs.

4.2 Block–Asynchronous strategy

In this section we present the Block–Asynchronous (BA) strategy proposed in this thesis for efficient computation of problems which require iterative computation, such as the Cellular Automata (CA). This method reduces the number of points of global synchronization allowing efficient exploitation of the memory hierarchy of the GPU. By BA computation, we mean updating a group of values an unbounded number of times without a global synchronization. Thus, each region is updated asynchronously with respect to other regions. For example, a cellular automata can be partitioned into different groups of cells which can be updated independently and locally. This strategy perfectly matches the tiling/grids parallel pattern described in the preceding section. This model is shown in Fig. 4.3 for a 6×6 cellular automaton. Each block (tile) of 3×3 cells is updated an unbounded number of times (intra-block updates) but the values outside the block, corresponding to the apron, are kept constant; i.e., equal to their values at the beginning of the stage. The entire grid is updated after a global synchronization, so data are read at the block boundaries (inter-block updates), which allows the propagation of data across the blocks. The BA strategy is also adequate for multicore architectures.

This strategy can easily be adapted to solve different algorithms, such as the asynchronous cellular automaton to compute the watershed transform on GPU [139, 135, 138]. Advanced MM operations can also benefit from the BA approach, for example opening and closing by reconstruction and greyscale attribute filtering, as it will be shown in the next sections.

The remainder of this section is organized as follows. Section 4.2.1 present the CA–Watershed algorithm based on Block–Asynchronous computation and its extension to 3D for volumes processing. In Section 4.2.2 we present a Block–Asynchronous algorithm for computing opening and closing by reconstruction (BAR) on GPU [134]. In Section 4.2.3 we describe a new proposal for greyscale area opening and closing on GPU which can be extended to other attributes. Finally, the results are discussed in Section 4.2.4.

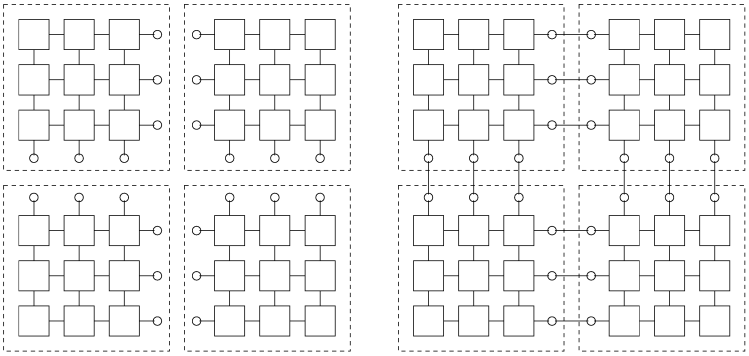


Figure 4.3: Example of Block–Asynchronous strategy, mapping on 6×6 CA (4-connectivity). Intra-block updating (left) and inter-block updating (right).

4.2.1 CA–Watershed based on Block–Asynchronous computation on GPU

The Watershed Transform based on cellular automata (CA–Watershed) was described in Section 3.2.2. The asynchronous behaviour of this automaton, introduced by Galilée et al. [64], is subject to its implementation. In the following, we describe first the synchronous implementation that can be executed in CPU using OpenMP, as well as on GPU. The idea is giving the foundations for understanding the asynchronous behaviour of the automaton. Then, the GPU Block–Asynchronous computation of the same algorithm is described. The algorithm is non-deterministic and may introduce artifacts into the border of the segmented regions. Therefore, we propose an artifacts-free block–asynchronous GPU implementation that produces the correct results by correcting the data propagation speed among the blocks using wavefront techniques [112]. These implementations follow the parallel tiling/grids pattern in which the grid of cells of the automaton is partitioned into regular regions that are assigned to blocks of threads of the GPU.

Finally, due the nature of the CA, the implementation can be easily extended to three dimensions in order to process a 3D volume.

Block–Synchronous computation on GPU

The CA–Watershed synchronous implementation has two kernels: one for initializing and another for updating the automaton. The pseudo-code presented in Figure 4.5 shows the

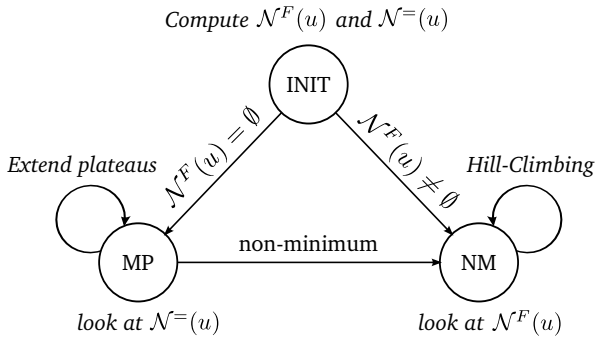


Figure 4.4: Three-state cellular automaton implementing Hill-Climbing algorithm [64].

block-synchronous computation on GPU. The kernels executed on GPU are placed between <> symbols. The pseudocode also includes the Tex, GM and SM acronyms to indicate kernels executed with data on texture, global and shared memory, respectively. Figure 4.4 in this page recalls the 3–state cellular automaton described in Section 3.2.2 to compute the watershed transform. The $\mathcal{N}^F(u)$ and the set of neighbors with the same grey value, $\mathcal{N}^=(u)$, are computed for each pixel in the first kernel (line 2 in the pseudocode). In the second kernel (line 4), the updates flood each region with a representative label in an iterative process executed by the CPU with a global synchronization at each step (line 5). These kernels are configured to work in rectangular thread blocks with a thread operating on one pixel.

With the first kernel, the automaton is initialized according to $l(u) = min\,(\,l(v)\,)$ and $f(u) = f\left(\mathcal{N}^F(u)\right)$, corresponding to Eq. (3.1) and Eq. (3.2), respectively (see page 68). The grey values are read from texture memory, as this read-only memory speeds up the ac-

Input: one band image *I* on the global memory of the GPU
Output: segmentation map

1: copy input data *I* from global to texture memory
2: <initialize cellular automaton> ▷ (Tex and GM)
3: **while** cellular automaton is not stable **do**
4: <synchronous updating of the automaton> ▷ (GM)
5: global synchronization among blocks
6: **end while**

Tex: texture memory, GM: global memory.

Figure 4.5: Pseudocode for CA–Watershed synchronous implementation on GPU.

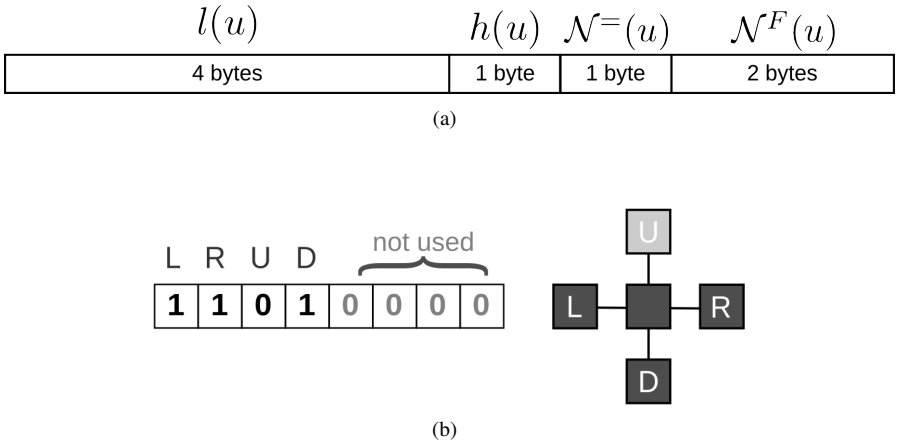


Figure 4.6: Example of data packing. (a) Data of the automaton packed into 64 bits, (b) structure of the variable $\mathcal{N}^{=}(u)$ using 4-connectivity packed in 1 byte.

cesses to data when they present high spatial locality. Once all data have been initialized, they are packed into 64 bits before being transferred to global memory, with the objective of increasing the data locality and reducing number of global memory accesses. The minimum amount of data required are 4 bytes for $l(u)$, 1 byte for $h(u)$, 1 byte for $\mathcal{N}^{=}(u)$ and 2 bytes for $\mathcal{N}^{F}(u)$. Fig. 4.6(a) shows an example of data packing for one pixel. The set $\mathcal{N}^{=}(u)$ is compressed in 1 byte using 1 bit for each neighbor (L, R, U, D in Fig. 4.6(b)) where "1" means a neighbor with the same grey level and "0" means a neighbor with a different one. The four least significant bits are ignored but they may be used to connect up to 8 neighbors. It is not necessary to store the state of each pixel as it can be deduced from $\mathcal{N}^{F}(u)$ (see Figure 4.4). If the lower slope is empty, the state is MP; otherwise, the state is NM.

At the end of the initialization stage the state of each pixel, a cell of the automaton, has switched to NM or MP. In order to update all the pixels synchronously, we use two 64-bit buffers, one input buffer for reading data and one output buffer for writing the results.

The updating stage has been implemented through a loop executed by the CPU (lines 3–6 in the pseudocode of the Figure 4.5), which calls a CUDA kernel at each step (line 4). There is one global synchronization per step, as indicated in line 5, which makes the computation wait until the GPU finishes its current work [119]. The input and output buffers are swapped before the next iteration. Only one flag needs to be moved to the CPU at each inter-block iteration

Input: cellular automaton packed in 64 bits
Output: cellular automaton updated

1: load and unpack data from global memory to registers ▷ input buffer
2: updates image and labels according to Eqs. (3.3)–(3.5)
3: pack results in 64 bits from registers and store data in global memory ▷ output buffer

Figure 4.7: Pseudocode for CA–Watershed synchronous CUDA kernel executed in global memory.

indicating whether the automaton must be further processed. The kernel for the synchronous updating of the automaton is shown in Figure 4.7.

In each call to the kernel, data are read from the input buffer in global memory and are unpacked and automatically stored in registers (line 1). The pixels are updated once (line 2) as described by Eq. (3.3) and Eq. (3.4) if their state is MP, and by Eq. (3.5) if their state is NM. These equations are described in Section 3.2.2. Finally, the resulting data are packed and stored in the output buffer. The update ends when all regions have been flooded.

Block–Asynchronous computation on GPU

The CA–Watershed based on block-asynchronous computation has the advantage of reusing information within a block, unlike the synchronous one, efficiently exploiting the shared and cache memories of the GPU. The storage requirements are the same as for the block-synchronous implementation. The updating stage has been adapted to perform in shared memory as many updates inside a region as possible (intra-block updates), before performing a synchronization among thread blocks (inter-block updates). Each region is synchronously updated (i.e. all cells within a region are updated at each iteration), while the regions themselves are asynchronously updated (an update of the entire grid is performed at certain selected steps). Hence, this is a hybrid iterative process that includes block-asynchronous and block-synchronous updates. The pseudocode in Figure 4.8 and Figure 4.9 show the inter-block updates (lines 3–6) and the intra-block updates (lines 2–5), respectively. The kernels is placed between the $<$ and $>$ symbols.

The inter-block updating loop is executed on the CPU and calls the asynchronous updating kernel that is executed on the GPU. In this kernel, for each block, once data are loaded in shared memory, the pixels are modified according to Eqs. (3.3) – (3.5) in an iterative intra-block process within each region of the image. Threads within a block are synchronized

Input: one band image I on the global memory of the GPU
Output: segmentation map

```
1: copy input data I from global to texture memory
2: <initialize cellular automaton>                          ▷ (Tex and GM)
3: while cellular automaton is not stable do                ▷ inter–block updating
4:     <asynchronous updating of the automaton>             ▷ (SM)
5:     synchronization among blocks                         ▷ global synchronization
6: end while
```

Tex: texture memory, GM: global memory, SM: shared memory

Figure 4.8: Pseudocode for Asynchronous CA–Watershed implementation on GPU.

Input: cellular automaton packed in 64 bits
Output: cellular automaton updated

```
1: load and unpack data from global memory to shared memory                ▷ input buffer

2: while cellular automaton is not stable within a block do                ▷ intra–block updating
3:     updates image and labels according to Eqs. (3.3)–(3.5)              ▷ (SM)
4:     synchronize threads within the block                                ▷ local synchronization
5: end while

6: pack results in 64 bits from shared memory and store data in global memory   ▷ output buffer
```

Figure 4.9: Pseudocode for Asynchronous CA–Watershed CUDA kernel executed in shared memory.

locally at each step of the intra-block process, so data updated within a block can be reused from the shared memory, which is much faster than the global memory space (see Section 4.1.4).

The intra-block updating ends when no new modifications are made with the available data within the region. Then the data in shared memory are packed and stored in global memory. The updating stage ends when all regions have been flooded.

In order to update the pixels at the edge of the block in this block-asynchronous implementation, the shared memory allocated for each region must be extended with a border of size one. This extra shared memory space corresponds to the apron illustrated in Figure 4.1(b) that is required in tiling/grid parallel patterns when the local computation also involves neighboring data. Thus, the apron of one region overlaps the adjacent regions. Threads on the edge of the block have to do extra work loading the data of the border.

The block-asynchronous algorithm avoids points of global synchronization that are costly in execution time, and efficiently exploits the shared memory, which has lower access times than the global memory.

Artifacts-free block–asynchronous computation on GPU

The block–asynchronous CA–Watershed implementation obtains a correct segmentation according to the watershed segmentation definition. Thus, when non-minimum plateaus exist in the image, the algorithm gives a correct segmentation; nevertheless, the watershed lines may not match the geodesic distance properly. When the data propagation speed is similar for all the cells, the algorithm may give a good approximation of the watershed lines. However, when computed by regions, it presents the problem of data propagation at the region boundaries, which causes artifacts as shown in the example in Figure 4.10. Initially data are propagated within the region during the intra-block updating, and later this is performed at the region boundaries during the inter-block updating. The different speed of data propagation in the intra-block and inter-block updates result in improperly placed watershed lines. This situation is visually observed as small irregularities in the watershed lines, as illustrated in Figure 4.10.

As a consequence of the asynchronous computation by blocks some undesirable artifacts arises and the quality of the segmentation is slightly affected. The asynchronous behavior of the CA–Watershed implementation is fixed by correcting the data propagation speed among the blocks. The artifact–free block-asynchronous algorithm is based on the application of a technique known as wavefront [112] increasing the quality of the watershed lines obtained. The wavefront starts from the lower border of a plateau (see Section 2.3.2) and iteratively

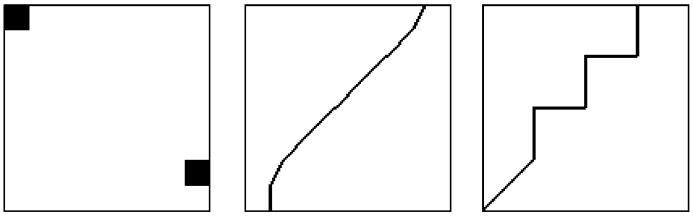


Figure 4.10: An image of size 128×128 pixels (left), the correct watershed line obtained using 4-connectivity (middle), and the artifacts produced by the asynchronous computation using blocks of cells of size 32×32 (right).

floods the flat zones. The pixels within the plateau swept by the wave will set their distance to the border to the number of iterations that the wave needs to reach them. The distance is initially set to one and is incremented each iteration. The original CA–Watershed algorithm described in Section 3.2.2 is modified as shown in Algorithm 2, where the modifications are described below.

The origin of this problem can be shown by introducing a variable which measures the propagation distance of data from the slopes of the image, as shown in Figure 4.11. The region boundaries delay the data propagation. However, this problem can be solved by data correction from the information provided by the inter-block updating process and the measured distances. Accordingly, a procedure for performing this is required.

In order to correct the artifacts produced by the asynchronous computation, we propose incorporating the wavefront technique into the algorithm, in a similar way as how it was introduced by Moga et. al [112]. It is necessary to define a distance variable, $d(u)$, which is initiated as,

$$d(u) = \begin{cases} \infty & \text{if } N^F(u) = \emptyset \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

i.e., the pixels in the neighborhood of a lower border of a plateau are assigned a distance of 0. This modification is introduced in the `INIT` state, lines 8 and 12 in Algorithm 2.

Then an iterative process for updating the automaton starts. The distances of pixels in the "MP" state are computed by increasing the distance from a neighbor by one, see line 21 in Algorithm 2. Thus, at the same time as the data are propagated over the plateau, the distances are computed by the pixels. However, the distances obtained during the intra-block

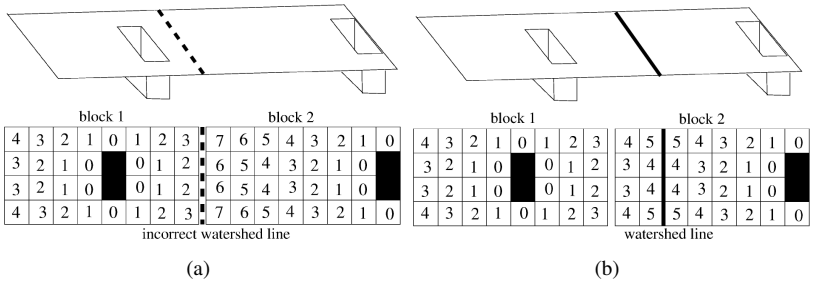


Figure 4.11: An image of 4×16 pixels displayed as a topographical relief (top) and the map of distances when the watershed is computed using asynchronous blocks (a), and the map of corrected distances (b).

Algorithm 2 Artifacts–free CA–Watershed algorithm

```
1:  procedure CA–WATERSHED(state, f, d)
2:      switch state do
3:          case INIT :                                          ▷ Initialize automaton
4:              compute N^=(u), N^F(u)
5:              if (N^F(u) = ∅) then
6:                  l(u) ← min_{v∈N^=(u)} (l(v))                 ▷ Eq. (3.1)
7:                  state(u) ← "MP"
8:                  d(u) = ∞                                     ▷ Eq. (4.1)
9:              else
10:                 f(u) ← f(N^F(u))                             ▷ Eq. (3.2)
11:                 state(u) ← "NM"
12:                 d(u) = 0                                     ▷ Eq. (4.1)
13:             end if
14:
15:         case MP :                                            ▷ Minimum or Plateau
16:             for each pixel v ∈ N^=(u) do
17:                 if h(v) < h(u) then
18:                     N^F(u) = {v}                             ▷ Eq. (3.4)
19:                     f(u) ← f(v)
20:                     state(u) ← "NM"
21:                     d(u) = d(v) + 1
22:                 else
23:                     l(u) ← min(l(u), l(v))                   ▷ Eq. (3.3)
24:                 end if
25:             end for
26:
27:         case NM :                                            ▷ Non Minimum
28:             v = N^F(u)
29:             f(u) ← f(v)                                      ▷ Eq. (3.5)
30:             for each pixel v ∈ N^=(u) do
31:                 if (d(u) − d(v) > 1 then
32:                     l(u) ← l(v)
33:                     N^F(u) = {v}
34:                     d(u) = d(v) + 1
35:                 end if
36:             end for
37:
38:     end switch
39: end procedure
```

updating may take incorrect values, as shown in Figure 4.11(a). If a pixel is switched to the "NM" state, its distance and its label might need to be corrected, as the pixel could be part

of a non-minimum plateau which should be split between two basins. The decision is taken by comparing the variable d of the pixel to those of its neighbors, and in the event that the difference is greater than one, it must be corrected. This modification is introduced from line 30 to line 36 in Algorithm 2.

This technique provides watershed lines that match the geodesic distance properly. In Figure 4.11(b), two slopes are symmetrically placed at both sides of a non-minimum plateau that are separated by an even number of pixels; hence, in this case only one solution is possible.

Extension to three dimensions

As explained in Section 2.4 the cells of an automaton may be arranged in three dimensions in order to process a 3D volume. In this case the connectivity of the automaton needs to be adapted to connect a cell to its surrounding neighbors, which may range from six (connecting a cell to its left, right, top, bottom, forward and backward neighbors) up to a maximum of twenty six neighbors.

The block–asynchronous computation explained above can be easily adapted to process a 3D volume of data, especially when only a low number of neighbors is considered. The new 3D implementation also consists of two kernels which are configured to work in 3D blocks of threads, with each one operating on a different 3D region in the volume. In the first stage the input data are read from global memory and the cells are initialized by accessing to their neighbors. The hybrid iterative process performs the inter-block and intra-block updates in the second stage. The same memory requirements per pixel as for the 2D artifacts-free asynchronous algorithm can be applied in this case. If we consider a 6-neighbor connectivity, the data required in global memory per each voxel can be compressed into 8 bytes, including one byte that stores the connectivity.

The shared memory allocated for each 3D region is extended with a border of size one in each dimension, so the borders with the adjacent regions are overlapped in the same way as in the 2D case. So, compared to the 2D algorithm, the shared memory requirements are much higher.

4.2.2 Opening and closing by reconstruction on GPU

The opening by reconstruction of an image I is defined as the reconstruction by dilation of I from the erosion with a SE of size n of I, (see Section 2.5). By definition, this is an iterative process that can easily be implemented, even in parallel for multi-core architectures [169],

Algorithm 3 Sequential reconstruction algorithm

```
1: procedure SR(I, J)
2:     repeat
3:         for each pixel p in forward scanning do                    ▷ forward scan
4:             J(p) = min(max{J(q), q ∈ N_G^+(p) ∪ {p}}, I(p))
5:         end for
6:         for each pixel p in backward scanning do                   ▷ backward scan
7:             J(p) = min(max{J(q), q ∈ N_G^-(p) ∪ {p}}, I(p))
8:         end for
9:     until stability
10: end procedure
```

although it requires a high number of iterations to reach stability. For single-core computers a Sequential Reconstruction (SR) algorithm, based on forward and backward scanning reduces the number of iterations as compared to the formal definition. Nevertheless, the most efficient reconstruction algorithms for single-core rely on the definition of a smart scanning of the image and are implemented by hierarchical First-In, First-Out (FIFO), or priority queues [145], such as the Hybrid Reconstruction Algorithm (HRA) that does not yield multiple iterations, thus, is the best trade-off for CPU [169].

The SR algorithm is summarized in Algorithm 3 where $N_G^+(p)$ and $N_G^-(p)$ are the backward (left, left-up, up, right-up) and forward (right, left-down, down, right-down) neighborhood of the pixel p, respectively.

In this thesis we propose a morphological reconstruction algorithm based on the block–asynchronous strategy, named BAR, and unlike the SR algorithm, multiple scans are performed in both directions at the same time. Therefore, the data can be reused in shared

```
Input: mask image I and marker image J
Output: morphological reconstruction

1: copy input data I and J from CPU to the global memory of the GPU
2: repeat                                                  ▷ inter-block updating
3:     <multiple updates of the marker image>              ▷ block–asynchronous reconstruction (SM)
4:     global synchronization among blocks
5: until stability
```

SM: shared memory

Figure 4.12: Pseudocode for the block–asynchronous reconstruction (BAR) algorithm on GPU.

Input: mask image I and marker image J on the global memory of the GPU
Output: updated marker image J

```
1: load data from global memory to shared memory
2: repeat                                              ▷ intra-block updating
3:     update J according to Eq. (4.2)                 ▷ (SM)
4:     synchronize threads within the block            ▷ local synchronization
5: until stability within the block
6: store data from shared memory to global memory
```

Figure 4.13: Pseudocode for the block–asynchronous reconstruction (BAR) CUDA kernel executed in shared memory.

memory. As the morphological reconstruction requires neighboring data, its implementation requires an apron of size one in order to update the pixels at the edge of the block.

The GPU proposal for the BAR algorithm is shown in Figure 4.12 and Figure 4.13. In the pseudocode of Figure 4.12, an iterative process (lines 2–5) executes the morphological reconstruction until stability is reached. Each iteration performs one call to the reconstruction kernel, placed between <> symbols at line 3, and a global synchronization among blocks (line 4). Threads are configured in two dimensional blocks and threads within the block load data from global to shared memory.

The kernel shown in Figure 4.13 consists of a loop (lines 2–5) where the updating is performed in shared memory (line 3) requiring only synchronization among the threads inside the block (line 4). The SM acronym indicates that the computation takes place in shared memory. The reconstruction operation is calculated according to the following equation:

$$J(p) = \min(\max\{J(q), q \in N_G(p) \cup \{p\}\}, I(p)), \tag{4.2}$$

where J is the marker image, I is the mask image and $N_G(p) = \{N_G^+(p) \cup N_G^-(p)\}$ the complete neighborhood of pixel p. The forward and backward scans are joined in one step, as defined in (4.2).

As this implementation is based on the block–asynchronous strategy, it also makes use of the tiling/grids parallel pattern.

4.2.3 Greyscale attribute filtering on GPU

Attribute filtering are connected operators that process an image according to a criterion [25]. The formal definition described in Section 2.5.2 can be extended to greyscale images through

threshold decomposition [76]. In the threshold decomposition, the greyscale image is decomposed into a set of binary sets. The filtering is applied to each threshold set and the results are subsequently stacked to reconstruct the greyscale image.

The number of iterations to compute an attribute filtering based on threshold decomposition is generally fairly high. However, it is the simplest parallel algorithm for greyscale area filtering (opening and closing) [169, 172] as each threshold level can be assigned to a different processing unit. By combining the results into a final greyscale image, many attribute filters can be computed.

The max-tree introduced by Salembier et al. [144] is a versatile data structure for connected set operators. The root of the tree represents the binary connected components obtained by thresholding the image at level zero, and the nodes corresponding to the components with the highest intensity are the leaves. The second level in the max-tree has the binary connected components by thresholding the image at level two, and so on. Each node besides the current grey level points to its parent [172].

Figure 4.14 shows an example of a greyscale attribute opening on a image with five connected components and the corresponding max-tree representation. The grey level is indicated by the subscript, and the *i*-th component within a level is indicated by the superscript. In the example given in Figure 4.14, the C_2^1 component is removed, the C_3^1 component also being pruned. Filtering the image using this structure is very easy, for example by pruning the branches and all of its descendant nodes at the level where the criterion for the connected component is false. This is known as the max filtering rule. Salembier [144] described four different rules for filtering the tree. The most appropriate rule depends mainly on the application. In this work we have used the max rule. We refer the reader to [144] for a description of the different rules. Attributes as area can be included within the node structure but other attributes require auxiliary data structures.

The main difficulties for an efficient parallel implementation of greyscale attribute filtering lies in the hierarchically representation of the connected components and the labelling of the components (see Figure 4.14), as they are global but non-separable structures; hence, they do not fit into any standard parallelization strategy employed in image analysis [172]. Wilkinson et. al [172] parallelized the max-tree construction on shared memory parallel architectures by dividing the image into chunks which are assigned to different processors. The sub-trees are created in each processor using the Tarjan's union-find algorithm [159]. The final max-tree

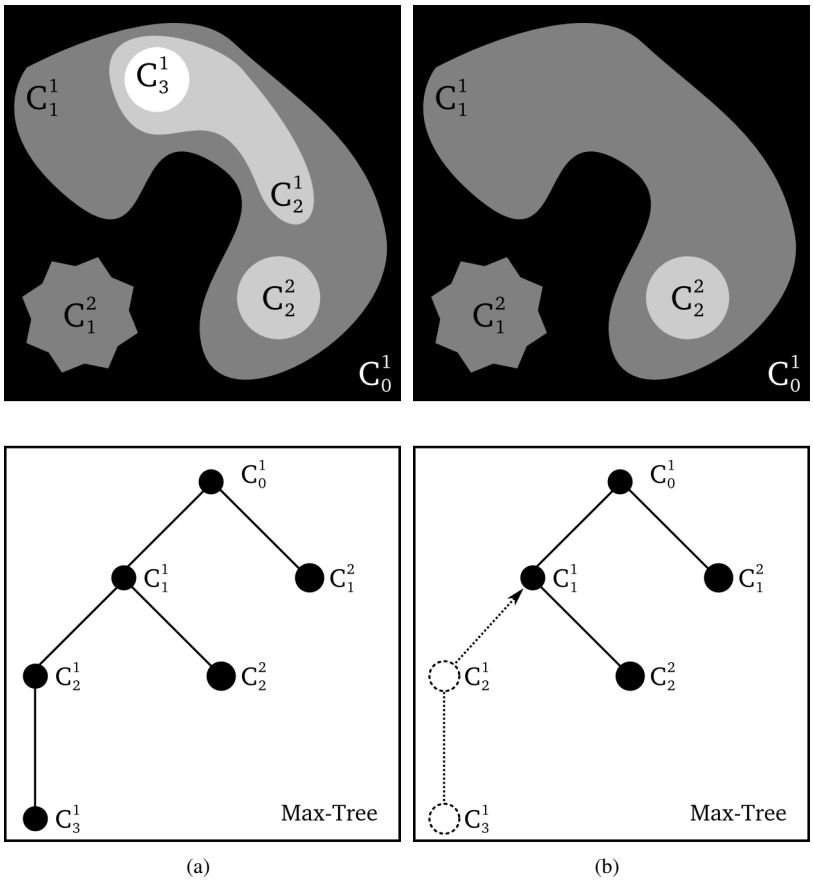


Figure 4.14: Example of a greyscale attribute opening. (a) Greyscale image with five connected components (the grey level is indicated by the subscript, and the *i*-th component within a level is indicated by the superscript) and the corresponding max-tree, (b) result of the attribute opening on the greyscale image and the pruning of the max-tree.

can be constructed by merging the different sub-trees using the unique labels assigned by the union-find algorithm.

Direct implementations without building the max-tree are also possible based on hierarchical FIFO and priority queues [170, 43, 97]. In [43] the filtering and the flooding for labelling the components are combined by passing the max-tree construction. It is considered a direct approach only for area filtering. The work presented in [97] is based on [43] but it can

Input: greyscale image I
Output: attribute opening

1: copy input data I from CPU to the global memory of the GPU
2: <labelling the connected components of I at each grey level> ▷ BA labelling (SM)
3: **for** (each grey level $h \in$ histogram(I)) **do**
4: <merge the regions of the connected components at level h > ▷ (GM)
5: <calculate the attribute for each region> ▷ (GM)
6: <perform attribute filtering> ▷ (GM)
7: **end for**

GM states for computation in global memory and SM in shared memory.

Figure 4.15: Pseudocode for the greyscale attribute opening on GPU.

be applied to attributes other than area. One drawback of these approaches that do not create the max-tree is that the pruning rule must be known a priori. The grey value assigned to the final image after removing a component cannot be retrieved by traversing the tree as it is not constructed.

In this thesis we propose a GPU implementation for greyscale attribute openings and closings without using queues, through threshold decomposition and merging of connected components. This proposal is a member of the group of direct implementations that simulate the max-tree [170, 43, 97].

The pseudocode in Figure 4.15 shows the workflow for the greyscale attribute opening on GPU proposed in this thesis. The kernels executed on GPU are placed between <> symbols. The area closing can be computed with the same algorithm but using the complement of the image.

The connected component labelling (line 2) is performed by using the block–asynchronous strategy proposed in this thesis. All the connected components of the image are labelled at once. First, each pixel is given a unique label that identifies its position within the image in a row-major order. Then, an iterative process propagates the minimum label between all the connected neighbors. This is similar to the block–asynchronous computation described on page 108. The updating stage is adapted to perform in shared memory as many intra-block updates as possible, before performing the inter-block synchronization.

Second, the image is processed through threshold decomposition (lines 3–7). At each grey level h, from the highest to the lower intensity, the connected components at grey level h, which have been already labelled in the last iteration, are merged if they have a common

border (line 4). This iterative process simulates the max-tree from the leaves to the root and only keep in memory the nodes at the current level h. Once the components have been merged, the attribute for each one can be computed (line 5) in a new kernel.

Finally, based on the value of the attribute, a component is filtered at the current level h (line 6) if the criterion for that component is false. We have used the filtering max rule that prunes the branches from the leaves up to the first node that needs to be preserved [144]. This proposal could be used to compute other attributes than the area of a region, simply by modifying the kernel at line 5.

4.2.4 Results

In this section we present the performance results for the Block–Asynchronous strategy applied to the asynchronous cellular automaton to compute the watershed transform, the opening and closing by reconstruction and the area attribute filtering.

Experimental setup

The different implementations based on the Block–Asynchronous (BA) strategy have been evaluated on the Intel quad-core i7-860 microprocessor. The GPUs used in the experiments are the GTX 580 and the GTX TITAN based on the Fermi and Kepler architectures, respectively. The CUDA code has been compiled under Linux using the nvcc compiler with the CUDA toolkit 4.2 (GTX 580) and 5.5 (GTX TITAN). The hardware, the compute capability of the graphic cards and the images used in these tests are described in Section 2.9.1.

The reference codes for comparison are optimized OpenMP parallel implementations for the CA–Watershed, the Fast Hybrid Reconstruction (HRA) algorithm [169] for the opening and closing by reconstruction, and an efficient implementation based on the max-tree (min-tree) [42] for attribute filtering. These algorithms will be described in the subsection where the comparisons are made. The performance results analyzed are expressed in terms of execution times and speedups. The execution times were obtained as the average of 20 executions. In all the tests, a connectivity of four pixels is used.

The datasets used in the experiments are two images (Lena and CT Scan Head) and the BrainWeb volume. The datasets are described in Section 2.9.3. The two images used are representative cases of processing small (Lena) and large (CT Scan Head and BrainWeb) plateaus, respectively. The processing of large plateaus (regions of uniform grey values) makes it necessary to propagate the labels through large regions of the image. This requires

Size	512×512	1024×1024	2048×2048
Transfer time	0.0022s	0.0082s	0.0321s
Size	$45 \times 54 \times 45$	$90 \times 108 \times 90$	$181 \times 217 \times 181$
Transfer time	0.0013s	0.0073s	0.0581s

Table 4.1: CPU–GPU data transfer times for 2D and 3D images at different sizes.

more computation time than processing small plateaus. Thus the selected images represent two very different cases regarding to computational cost of data propagation among blocks. The CPU–GPU data transfers are carried out at the beginning and are shown in seconds in Table 4.1. This time is the same independently of the application where the BA approach is used.

CA–Watershed based on Block–Asynchronous computation on GPU

In this section we present the results for the following GPU implementations of the watershed transform based on cellular automata: block-synchronous, block-asynchronous and artifacts-free block-asynchronous implementations[1]. First, we have checked the correctness of the asynchronous CA–Watershed implementation by comparing the number of segmented regions obtained by the GPU algorithms to the number of regions obtained by a sequential watershed algorithm over the images.

Table 4.2 shows the number of regions created by the watershed transform, which is the same for the CPU and GPU implementations. The difference between the number of regions at different resolutions is due to the process of scaling the image. The CT Scan Head and the BrainWeb datasets present large plateaus and therefore the number of regions is lower (but larger in size) than for the Lena image.

In the following, the performance is analyzed in terms of occupancy, execution times and speedup.

– Analysis of the GPU Parameters: The initialization and updating kernels used on the GPU implementation, see the pseudocode presented in Figure 4.7 and Figure 4.9, have been

[1] Part of these results have been published in P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2638–2655, 2013.

Size	512×512	1024×1024	2048×2048
Lena	24958	25139	28521
CT Scan Head	6221	7300	13381
Size	$45 \times 54 \times 45$	$90 \times 108 \times 90$	$181 \times 217 \times 181$
BrainWeb	1115	5669	14348

Table 4.2: Number of regions generated by the watershed transform.

analyzed according to the resources available on the GTX 580 (Fermi architecture). This GPU has 16 SMs with the following limits per SM: 1536 threads, 8 active blocks, 32768 registers of 32 bits and 64 KB of on-chip memory that can be configured as a shared memory of 16 KB and 48 KB for the L1 cache or vice versa (see Table 2.3 for a full description of the GPU). Table 4.3 shows the maximum number of active blocks based on the block size and the on-chip memory configuration. For the *ca_watershed_asynchronous* kernel, the reason for selecting the L1 configuration with the remaining 16 KB being for the shared memory is that the maximum number of active blocks is given by the limit of 1536 threads per SM. As described in Section 4.2, 8 bytes per pixel are required for data packing. For a block with 16×16 threads, the shared memory required would be 2048 bytes but as the shared memory has been extended with an apron of size one, each block needs $18 \times 18 \times 8$ bytes of this on-chip memory, i.e. 2.5 KB per block. Therefore, considering a maximum of 6 blocks (1536 threads) per SM, a total of 15 KB of shared memory per SM are used.

For the case of the artifacts-free asynchronous proposal (*ca_watershed_asynchronous-free* in Table 4.3), the number of simultaneously active blocks per SM is reduced to four. In this case the limiting factor is the number of 32768 registers available per SM. The proposal requires 26 registers per thread, which gives a total of $16 \times 16 \times 26 = 6656$ registers per block,

Kernel		16×16	32×16	32×32	$8 \times 8 \times 4$
ca_watershed_asynchronous	L1	6	3	1	3
ca_watershed_asynchronous-free	L1	4	2	1	2
ca_watershed_asynchronous	Sh	6	3	1	6
ca_watershed_asynchronous-free	Sh	6	3	1	6

Table 4.3: Number of active blocks per SM for the different kernels based on the block size and the shared memory requirements. L1 indicates that 48 KB are used for the L1 memory and 16 KB for the shared memory. Sh states for the opposite configuration. Analysis for CA–Watershed implementations.

so there are enough registers in each SM for only 4 blocks. Regarding the shared memory use in the artifacts-free kernel, each pixel requires 4 extra bytes to manage the geodesic distance properly, (see Algorithm 2), so $18 \times 18 \times (8+4) = 3888$ bytes per block are required. By using the Sh configuration (48 KB for shared memory), the limit in the number of active blocks is given by the block size.

– Performance analysis: The OpenMP implementation is based on the block-synchronous approach (see pseudocode on Figure 4.5) and it uses 4 threads scheduling the work statically among the threads by a loop construct distributing the iterations into 4 chunks of the same size (one per thread), in order to evenly distribute the workload among the threads and to achieve a high locality in the data accesses. The need to access data outside the region assigned to each thread is not a problem in the OpenMP implementation, as all the threads access the same memory space. The algorithm includes an implicit synchronization barrier at each step of the updating.

Table 4.4 gives the performance results obtained in the GTX 580. The execution times for the GPU proposals in this table include the CPU–GPU data transfer time. In all the tests, the CA–Watershed based on block-asynchronous computation obtains high speedups for all the image sizes. As shown in Table 4.4, the speedups also scale well with the size of the image; i.e. from 9.0× to 13.6× for the block–synchronous implementation for the Lena image up to 11.7× to 22.2× with the block-asynchronous proposal. When the image size increases, so does the amount of computational work, the hundreds of available threads are better exploited. The 3D GPU proposals obtain speedups for all the volume sizes, and the speedup values increase with the volume size as the computational load also increases. The performance results for the block-asynchronous proposals are always better than for the synchronous implementation; approximately twice as good. The performance results for both, the block-asynchronous and the artifacts-free block-asynchronous approaches are very similar.

We focus now the test on the 2D images at a resolution of 2048×2048 pixels as the behavior of processing large plateaus is better appreciated in this case. When the image presents large plateaus the computational cost of the watershed transform increases as the labels must be propagated through large regions of the image. Comparing the execution times of the block-synchronous and block-asynchronous approaches (see Table 4.4), a speedup of 1.6x is obtained for the image of Lena while the speedup increases up to 4.3x for the CT Scan image. The improvement of the block-asynchronous proposal versus the synchronous imple-

Lena (2D)	512×512	1024×1024	2048×2048
OpenMP (4 threads)	0.0351s	0.1990s	1.2452s
GPU Synchronous	0.0039s (9.0×)	0.0188s (10.6×)	0.0916s (13.6×)
GPU Asynchronous	**0.0030s** (11.7×)	**0.0131s** (15.2×)	**0.0562s** (22.2×)
GPU Artifacts-Free Async.	0.0034s (10.3×)	0.0158s (12.6×)	0.0736s (16.9×)
CT Scan (2D)	512×512	1024×1024	2048×2048
OpenMP (4 threads)	0.4941s	2.8793s	15.0919s
GPU Synchronous	0.0305s (16.2×)	0.1436s (20.1×)	0.6992s (21.6×)
GPU Asynchronous	**0.0093s** (53.1×)	**0.0381s** (75.6×)	**0.1628s** (92.7×)
GPU Artifacts-Free Async.	0.0126s (39.2×)	0.0522s (55.2×)	0.2353s (64.1×)
BrainWeb (3D)	$45 \times 54 \times 45$	$90 \times 108 \times 90$	$181 \times 217 \times 181$
OpenMP (4 threads)	0.1337s	2.2611s	37.7378s
GPU Synchronous	0.0084s (15.9×)	0.0820s (27.6×)	1.1227s (33.6×)
GPU Asynchronous	**0.0044s** (30.4×)	**0.0451s** (50.2×)	**0.5907s** (63.9×)
GPU Artifacts-Free Async.	0.0050s (26.5×)	0.0540s (41.8×)	0.7304s (51.7×)

Table 4.4: Performance results including data transfer times (speedup in brackets). Best results in bold.

mentation is better for the second image; although, as shown in Table 4.4, processing large plateaus takes more time: 0.1628s for the CT Scan image while the Lena image only requires 0.0562s. This is because for the block-asynchronous proposal the intra-block updating allows the labels to propagate faster among regions, especially in images with large plateaus. If a region is entirely within a plateau, the labels have to be propagated from side to side of that region. In this situation, only one inter-block update and w intra-block updates are needed, where w is the width of the region. The synchronous implementation would need w inter-block updates, with the consequent penalty for transferring data from and to global memory at each step, with each one of those steps corresponding with a global synchronization. The block-asynchronous approach reduces the number of synchronizations among thread blocks and increases data reuse thanks to the inter- and intra-block updating scheme.

The decrease in the number of synchronizations for the block-asynchronous proposals is illustrated in Table 4.5, where the number of inter-block and intra-block updates are summarized for the synchronous and the block-asynchronous implementations and the test images. Only the values for the block-asynchronous implementation are shown, as the numbers are the same for the artifacts-free proposal. For the block-synchronous implementations (on CPU and on GPU) only inter-block updates take place in the sense that after each update of all the

Lena	inter-block	intra-block (min.)	intra-block (max.)	intra-block (avg.)
GPU Synchronous	114	—	—	—
GPU Asynchronous	16	22	195	108.5
CT Scan Head	**inter-block**	**intra-block (min.)**	**intra-block (max.)**	**intra-block (avg.)**
GPU Synchronous	1156	—	—	—
GPU Asynchronous	76	88	1248	668

Table 4.5: Number of updates per pixel for the block-synchronous and block-asynchronous implementations for the 2048×2048 images.

pixels of the image one global synchronization operation is required. Observing, for example, the values for the CT Scan image in the table, the number of inter-block updates (i.e. the number of global synchronizations required) is 1156 for the synchronous implementation. For the block-asynchronous cases the number of inter-block updates decreases to 76 and the total number of asynchronous intra-block updates per block summing up all the iterations ranges from 88 to 1248, depending on the block, with 668 being the average value over all the blocks. Hence, the number of updates per pixel is 1156, with the same number of corresponding global synchronizations for the synchronous implementation, and an average of 668 local synchronizations with only 76 global synchronizations for the block-asynchronous algorithms. In the case of the Lena image, a similar decrease is observed.

– Comparison to other works: Proposals of different watershed algorithms on the GPU using shaders [88] and CUDA [171, 91, 81] have been presented in the last few years. In [171] a new algorithm is presented based on the introduction of a chromatic function for establishing the order in which the voxels are processed. The experiments are carried out over volume data sets, obtaining maximum speedups of $7\times$ on a GTX295 when compared to the sequential proposal of the algorithm, even when large volume data sets of up to $600 \times 600 \times 600$ are considered. In our case, the largest volume considered was $181 \times 217 \times 181$, 9 times smaller, achieving speedup values of $63.9\times$ on a GTX 580. Taking into account that the speedups of our block-asynchronous proposals increase with the volume size, as shown in Table 4.4, and that our experiments have proved that the block-asynchronous proposals scale by a factor of $2\times$ to $4\times$ between the GTX 295 and the GTX 580 GPUs [139], we can conclude that our proposals outperform the results in [171].

The algorithm presented in [91] is inspired by the drop of water paradigm and performs a component labelling and a path compression approach [74]. Its results are compared to a GPU synchronous algorithm for the watershed based on a CA [88], outperforming it. Given that the experiments in [91] are performed on an older GPU than in our case, we have executed them on our GTX 580, obtaining similar speedup results to the obtained with our block-asynchronous proposal of the CA-watershed described in this work.

Opening and closing by reconstruction on GPU

The opening and closing by reconstruction GPU implementation based on the block-asynchronous strategy, see Section 4.2.2, has been evaluated on the GTX TITAN (Kepler architecture)[2]. The block size is configured with 32×8 threads. Each block requires only 340 bytes of shared memory, so the on-chip memory is configured with 48 KB for L1 cache. The reference codes for comparison are the Fast Hybrid Reconstruction (HRA) algorithm in CPU [169], and the GPU Sequential Reconstruction (SR_GPU)[3] algorithm proposed in [87]. The performance results analyzed are expressed in terms of execution times and speedups. The speedups are calculated with respect to the HRA (CPU) and the SR_GPU algorithms. The execution times were obtained as the average of 20 executions. The datasets used in the experiments are the Lena and the CT Scan Head images.

[2]Part of these results have been published in P. Quesada-Barriuso, F. Argüello, D. B. Heras, and J. A. Benediktsson, "Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. PP, no. 99, pp. 1–9, 2015 (published online, print edition pending).

[3]My thanks to Prof. Karas for sharing his implementation.

Lena	512×512	1024×1024	2048×2048
HRA (CPU)	0.1100s (1.0×)	0.1244s (1.0×)	0.1723s (1.0×)
SR_GPU (GPU)	0.0168s (6.5×)	0.0610s (2.0×)	0.2109s (0.8×)
BAR (GPU)	**0.0027s** (40.1×)	**0.0112s** (11.1×)	**0.0485s** (3.5×)
CT Scan Head	512×512	1024×1024	2048×2048
HRA (CPU)	0.1086s (1.0×)	0.1243s (1.0×)	0.1541s (1.0×)
SR_GPU (GPU)	0.0194s (5.6×)	0.0679s (1.8×)	0.2097s (0.7×)
BAR (GPU)	**0.0026s** (41.7×)	**0.0113s** (11.0×)	**0.0388s** (3.9×)

Table 4.6: Block–asynchronous morphological reconstruction results including data transfer times (speedup in brackets). Best results in bold.

For this experiment we have created a marker image J for each dataset I as $J(p) = max\{I(p) - h, 0\}$, that is known as the hmax transform. A value $h = 10$ was used to create the initial marker.

The results for the three algorithms are shown in Table 4.6. Both, SR_GPU and BAR algorithms include the data transfer from CPU to GPU memory. It can be observed that the GPU proposals outperform the HRA algorithm. However, the SR_GPU algorithm cannot beat the HRA algorithm with the images of 2048×2048 pixels. It can be observed that the speedup decreases by increasing the size of the images. The reason is that the HRA algorithm is optimized on CPU to process only those pixels that need to be reconstructed, while the SR_-GPU and the BAR algorithms are based on a raster scanning of all the pixels of the image.

The proposed algorithm offers a better performance in all cases. By performing multiple scans in shared memory at each iteration, this morphological reconstruction on GPU efficiently exploits the shared memory through the block-asynchronous updating process. The best speedup is $41.7\times$ obtained on the CT Scan Head (512×512 pixels).

Greyscale attribute filtering on GPU

This section presents the results obtained by the GPU greyscale attribute opening (closing) algorithm, described in Section 4.2.3. We have evaluated the performance on the GTX TITAN (Kepler architecture). The available resources for this GPU are described in Section 2.9.1. The reference code for comparison is an efficient implementation based on the max-tree (min-tree) presented in [42] for classification of hyperspectral images by using extended attribute profiles (EAPs). The University of Pavia dataset, described in Section 2.9.3, is used in the test. The first principal component extracted by PCA from this dataset is used to create an EAP_a based on area and an EAP_d based in the diagonal of the bounding box. The thresholds (λ) for creating each profile are $\lambda \in \{36, 49, 169, 361, 625, 1369\}$ for the area and $\lambda \in \{10, 25, 50, 100, 150, 250\}$ for the diagonal. These values were extracted from [41].

	Max-tree / min-tree (CPU) [42]	Opening GPU	Closing GPU	TOTAL GPU
EAP_a	0.9450	0.0825	0.1200	0.2025 (4.6×)
EAP_d	1.0331	0.0941	0.1335	0.2276 (4.5×)

Table 4.7: Performance results for the greyscale attribute filtering (opening and closing) including data transfer times for the first PCA of the hyperspectral image of the University of Pavia.

Kernel	Area opening	Diagonal opening
BA labelling	0.00054s (0.65%)	0.00090s (0.96%)
Threshold greylevel	0.00405s (4.92%)	0.00332s (3.53%)
Merge regions	0.04129s (50.15%)	0.04515s (48.05%)
Calculate attribute	0.01921s (23.33%)	0.02344s (24.94%)
Filtering	0.00932s (11.32%)	0.01415s (15.06%)
Others	0.00816s (9.63%)	0.00704s (7.46%)
Total	0.08257s (100%)	0.09400s (100%)

Table 4.8: Execution time breakdown for the greyscale attribute opening (area and diagonal) on GPU applied to the first principal component of the University of Pavia. Execution time in seconds (percentage time in brackets).

The performance results are analyzed in terms of execution times and speedups. The execution times were obtained as the average of 20 executions. The block size is configured as 32×8 threads and the on-chip memory as 48 KB of L1 cache for all the kernels (see Section 4.2.3 for more details).

Table 4.7 shows the execution time and speed of the proposed GPU algorithm for greyscale attribute opening (closing). Note that the max-tree (min-tree) proposed in [42] was designed for computing EAPs. So, for a comparison in equal terms, we have executed our algorithm first for area (diagonal) opening, and then for area (diagonal) closing. The fifth column in the table is the sum of both execution times.

The execution time is reduced from 0.9450 seconds to 0.2025 seconds ($4.6\times$) for the extended profile based on area (EAP_a). The results for the EAP_d are almost the same. We can observe in Table 4.7 that the attribute opening is $1.4\times$ faster than the attribute closing. The execution time breakdown for the greyscale area (diagonal) opening is presented in Table 4.8. It can be observed that half of the time is used for merging the regions. The kernel for computing the area takes 23.33% of the overall time (24.94% for the diagonal of the bounding box), and the filtering takes 11.32% for the area and 15.06% for the diagonal. The row labelled as *Others* in Table 4.8 includes the data transfer time from CPU to GPU and the loop control flags on CPU, lines 3–7 in the pseudocode for the greyscale attribute filtering on GPU described in Section 4.2.3.

4.3 Multi-class SVM classification

In this section we present our GPU multi-class implementation for n-dimensional image classification. Different implementations of SVM on GPU are available in the literature [32, 79, 8, 99]. Catanzaro et al. [32] developed a binary (one-against-one) SVM for training and classification on GPU (gpuSVM) using CUDA. They approached the SVM classification problem by making use of MapReduce computations [45], as well as CUDA Basic Linear Algebra Subroutines (CUBLAS) to perform matrix multiplications. The GPU-accelerated LIBSVM implemented in [8] is based on the original LIBSVM library with the same functionality and interface. The results published in [8] deal only with the training phase of the SVM, solving the multi-class problem separately and do not produce final classification maps. Herrero-Lopez [79] extended the GPU binary problem proposed in [32] to multi-class problems on GPU using the one-against-all (OAA) approach. A survey of GPU accelerated SVMs has recently been published in [103].

The reason for having our own SVM implementation stems from the need to classify hyperspectral images in real-time. The reference maps in remote sensing usually have more than two classes (multi-class problems) with a small number of training samples. As described in Section 2.7, if the training phase uses a small number of training samples, the GPU can not be fully exploited [137]. Hence, we focus our efforts in the classification stage, adhering to the One-Against-One (OAO) approach obtaining better results than the previous methods as it will be shown in the next section. We have implemented the SVM classification stage on GPU (called GPUSVM), which keeps the input data in global memory to compute all the pairs of OAO SVM classifiers in order to solve the multi-class problem, and reuse data in shared memory to find the final class assigned to each pixel. Thus, data transfer between the host and the device in minimized.

4.3.1 SVM Implementation on GPU

The pseudocode for the SVM classification phase proposed in this work is illustrated in Figure 4.16. First, data are copied to the GPU global memory and they are properly aligned to obtain coalesced accesses. The test data $\mathbf{X}$ are arranged in column-major order (one pixel vector per column) while the support vectors **SVs** are arranged in row-major order.

A loop in the host (lines 2–3) executes the RBF kernel T times, with T the number of binary classifiers necessary to solve the multi-class problem using the OAO approach. At

Input: test data **X** and support vectors **SVs**
Output: classification map

1: copy input data from CPU to the global memory of the GPU
2: **for** $i \in 1,2,\ldots,T$ **do**
3: <compute RBF kernel> ▷ One-Against-One strategy (GM)
4: **end for**
5: <sums the score calculated by the T classifiers> ▷ winner takes all (SM)

GM states for computation in global memory and SM in shared memory.

Figure 4.16: Pseudocode for the multi-class SVM classification on GPU (GPUSVM).

Input: test data **X** and support vectors **SVs**
Output: classification map

1: **for** each support vector $\mathbf{x_i} \in$ **SVs do**
2: compute the discriminant function $D(x) = \alpha_i y_i \exp\left(-\gamma||\mathbf{x}_i - \mathbf{x}||^2\right) + b$
3: **end for**
4: store result in global memory

Figure 4.17: Pseudocode for the RBF CUDA kernel executed in global memory.

the end of the loop the results of the T classifiers are kept in the global memory. The RBF kernel, illustrated in Figure 4.17, is executed in global memory and computes the discriminant function (2.25), see Section 2.7. Each thread computes the distance $||\mathbf{x_i} - \mathbf{x}||^2$ of a pixel vector $\mathbf{x} \in \mathbf{X}$ with all the support vectors $\mathbf{x}_i \in$ **SVs**. As the threads access the same support vectors, the global memory accesses are broadcast to the threads of the same block, and are also reused among different blocks by data caching.

The final decision in the one-against.-one strategy is taken on the basis of the winner-takes-all rule [110], which corresponds to the following kernel (line 5 in Figure 4.16). For each class, this kernel computes a score function $S(\mathbf{x})$ which is the sum of the favorable and unfavorable votes calculated by the T classifiers. In this kernel each thread processes one pixel and counts the number of positive and negative votes among the binary classifiers. The decision $S(\mathbf{x})$ for each pixel vector is computed and stored in shared memory and used in the same kernel to find the winner by a MV process. The voting corresponding to one pixel is computed also by one thread. The final prediction is stored back in global memory.

This implementation of the SVM computes the multi-class problem in a single execution. Other implementations, such as [32], consider only the standard two-class SVM problem and require more time to compute a multi-class problem as they need to execute the two-class classification T times independently.

4.3.2 Results

The multi-class SVM classification on GPU (GPUSVM) has been evaluated on the Intel quad-core i7-860 microprocessor. The GPU used in the experiments is the GTX 680 based on the Kepler architecture. The CUDA code has been compiled under Linux using the nvcc compiler with the CUDA toolkit 5.0. The LIBSVM library is used as a base for comparison. Among the GPU implementations found in the literature we have compared our implementation with those that provide the source code (or executable), perform the classification stage, and produce a final classification map. These criteria correspond to the gpuSVM [32] implementation.

The n-dimensional datasets used in the experiments are University of Pavia, Pavia City, Indian Pines, Salinas Valley and Hekla Volcano. These datasets are described in Section 2.9.3.

The performance results are expressed in terms of execution times and speedups. The execution times are obtained as the average of 10 executions. The tests are carried out as in [32], excluding the file I/O time for both, the CPU and GPU, but including CPU-GPU data transfer times in the GPU implementation.

Table 4.9 gives the performance results obtained on the GTX 680. The reference codes for comparison are a LIBSVM (single core), an optimized OpenMP (4 threads) parallel implementation (OMPSVM), and the gpuSVM [32]. The OpenMP implementation uses 4 threads and distributes the work statically among the threads by a loop construct. The speedup of our OMPSVM classification ranges from 2.1× to 4.3× as compared to the LIBSVM. The

	LIBSVM	OMPSVM	gpuSVM	GPUSVM
Pavia University	101.6520s	23.2992s (4.3×)	14.0497s (7.2×)	**3.2355s** (31.4×)
Pavia City	247.9579s	70.6478s (3.5×)	–	**8.2909s** (29.9×)
Indian Pines	4.8343s	2.5463s (1.9×)	40.9321s (0.12×)	**0.5224s** (9.2×)
Salinas Valley	89.1554s	42.4208s (2.1×)	47.7323s (2.3×)	**3.4980s** (25.4×)
Hekla Volcano	46.4376s	22.2023s (2.1×)	23.5852s (1.9×)	**2.7723s** (16.7×)

Table 4.9: Results for multi-class SVM classification on GPU (speedup in brackets). Best results in bold.

gpuSVM was not designed for multi-class problems but performs better than the LIBSVM in three of the four datasets with a speedup of 7.2× for the Pavia University dataset. The GPUSVM proposed in this thesis outperforms the other implementations and speed ups the execution times for all the datasets. The best result is obtained classifying the University of Pavia, with an execution time of 3.2355s.

4.4 Conclusions

In this chapter we have presented the techniques and strategies developed in this thesis for efficiently computing the proposed schemes on GPU. General strategies for parallel processing, such as adequate data partitioning, data movement and data packing have been described and some key GPU performance issues have been expounded.

The Block–Asynchronous (BA) strategy proposed in this work was adapted to solve a watershed transform based on CA, to compute morphological reconstruction and to perform Connected Component Labelling (CCL). The asynchronous computation by blocks (intra-block updates) reduces the number of points of global synchronization (inter-block updates) allowing efficient exploitation of the GPU's memory hierarchy. The CA–Watershed will be used in the CA–WSHED–MV scheme, and the morphological reconstruction in the WT–EMP scheme. The BA strategy was also applied for connected component labelling in a new proposal for greyscale attribute filtering, which is a technique used in spectral-spatial classification schemes based on Extended Attribute Profiles (EAPs) and Extended Multi-Attribute Profiles (EMAPs).

The different applications based on the Block–Asynchronous (BA) strategy were independently analyzed. The block-asynchronous CA–Watershed had the best results with a speedup of 92.7× for a 2048×2048 image of a computed tomography scan of a human head (CT Scan Head), as compared to a parallel multi-threaded CPU implementation using OpenMP. The artifacts-free block–asynchronous implementation, which produces the correct watershed lines, obtained a speedup of 64.1× for the same image. The performance of the block-asynchronous reconstruction (BAR) was compared to a Fast Hybrid Reconstruction (HRA) algorithm [169], and the GPU Sequential Reconstruction (SR_GPU) algorithm proposed in [87], with speedups of 41.7× and 5.6×, respectively, for the 512×512 CT Scan Head image. The BA strategy was included in the greyscale attribute filtering to compute EAPs based on area and diagonal of the bounding box proposed in this thesis. The EAP was

compared to an efficient implementation based on the max-tree (min-tree) presented in [42]. The speedup was 4.6× for the area-based extended profile created for the first PCA of the hyperspectral image of the University of Pavia.

In this chapter we have also presented our GPU multi-class implementation for classification of n-dimensional images (GPUSVM), and compared it with the facto LIBSVM library, a multi-threaded implementation using OpenMP, and the gpuSVM proposed in [32]. The speedup obtained by the GPUSVM was 9.2× (Indian Pines), 16.7× (Hekla Volcano), and 31.4× (University of Pavia) as compared to LIBSVM. We found that our implementation is robust with large datasets, such as the Pavia City, which could not be processed by the gpuSVM, as well as datasets with many classes, such as Indian Pines and Salinas Valley. The CUDA implementations described in this chapter will be used in Chapter 5 with the objective of developing efficient schemes for spectral-spatial n-dimensional image classification.

CHAPTER 5

EFFICIENT IMPLEMENTATION OF SPECTRAL-SPATIAL CLASSIFICATION SCHEMES ON GPU

5.1 Introduction

In this chapter we present the CUDA implementations of the spectral–spatial classification schemes proposed in Section 3.2 and Section 3.3, corresponding to the CA–WSHED–MV and WT–EMP schemes, respectively. The techniques and strategies developed in Chapter 4 will be applied to these schemes.

The CA–WSHED–MV scheme, described in Section 3.2, is based on segmentation and incorporates the spatial information by a watershed transform based on cellular automata (CA–Watershed). The scheme reduces the dimensionality of the hyperspectral image to one by the computation of a Robust Color Morphological Gradient, (see Section 2.2.2) before applying the CA–Watershed algorithm, (see Section 4.2.1). We recall that the CA–Watershed algorithm is based on the block-asynchronous strategy proposed in this thesis and described in Section 4.2. The classification is carried out by our GPUSVM implementation (see Section 4.3.1) combining the spectral and spatial results with a Majority Vote (MV).

The WT–EMP scheme, described in Section 3.3, is based on wavelets and Mathematical Morphology (MM). The kernels have been specially adapted to optimize the use of the GPU resources. The local computation of the wavelet and the morphological operations used in

this scheme match the computing model of the GPU. Therefore, the scheme can be efficiently implemented on this architecture.

The performance is measured in terms of execution time and speedup on real hyperspectral images over CPU multi-threaded implementations using commodity hardware. We analyze the best parameters and the hardware resources that usually limit the occupancy on the GPU. The remainder of this chapter is organized as follows. In Section 5.2 the GPU implementation details of the different stages of the first scheme, named CA–WSHED–GPU are described. Section 5.3 describes the implementations stages of the second scheme, named WT–EMP–GPU. These sections include results on real hyperspectral images on different low-cost computing infrastructures, such as the GTX 680 and GTX TITAN GPUs, showing the real-time execution achieved on CUDA-capable devices. Final conclusions are given in Section 5.4.

5.2 CA–WSHED–GPU

In this section we present the CA–WSHED–GPU implementation. This scheme was described in Section 3.2 and has the following stages:

1. RCMG: this stage computes the Robust Color Morphological Gradient resulting in a one band gradient image as described in Section 3.2.1.

2. CA-Watershed: it is the watershed transform based on cellular automata. As described in Section 3.2.2 the asynchronous behavior of this automaton depends on its implementation. In the scheme we use the artifacts-free block–asynchronous implementation (see Algorithm 2). This stage creates the segmentation map.

3. SVM: it is the classification of the hyperspectral image on GPU (GPUSVM). This stage creates the classification map.

4. MV: it is the data fusion strategy used to join the spectral and spatial information by combining the classification and segmentation maps. The majority vote was described in Section 3.1.3.

Regarding the main GPU optimization strategies applied, the different stages are concatenated in a pipeline processing that minimizes the data transfers between the host and the device and maximizes the computational throughput. We paid special attention to minimizing

data transfer between CPU and GPU, as well as reusing data within the GPU by exploiting the shared memory and cache hierarchy of the architecture.

In order to give the GPU enough work to do, the hyperspectral image is divided into regions that are distributed among the thread blocks. The regions will be one, two or three dimensional depending on the executed stage, enabling all the threads to perform useful work, and therefore exploiting the thousands of threads available on the GPU.

Since data partitioning strongly depends on the processing techniques applied (see Section 4.1.3), different hyperspectral data partitioning strategies and thread block arrangements are studied in order to effectively exploit the memory and computing capabilities of the GPU architecture.

We now go on to describe the implementations details of each stage.

5.2.1 RCMG on GPU

The Robust Color Morphological Gradient computes the vectorial gradient of a n-dimensional image based on the distance between pixel vectors and resulting in a one band gradient image, (see Section 2.2.2).

From a processing point of view, two different algorithms have been implemented to compute the RCMG. The first one is based on the spatial-domain partitioning within a block, as illustrated in Figure 5.1(a), and the second algorithm is based on the spectral-domain partitioning within a block, Figure 5.1(b). Each shaded rectangle in this figure represents a block of $X \times Y$ threads. Thus, in the spatial partitioning, the pixel vectors are kept as a whole, while the pixel vectors are subdivided into slices made up of contiguous spectral bands in the spectral-partitioning. In both cases, data are stored in global memory so that consecutive threads access consecutive global memory locations (coalescent accesses).

In these algorithms, the processing of one pixel requires data from the neighboring pixels. So, in order to compute the RCMG, we must extend the block with a border, resulting in a block with an apron of size one, as explained in Section 4.1.1.

The pseudocode introduced in Figure 5.2 shows the RCMG kernel that was described in Section 3.2.1. The gradient calculation is divided into three steps. First, threads within a block load data from global to shared memory, including the extended border of size one (line 2). Second, the threads of the same block cooperate to calculate the distances of the set χ (lines 3–5). Third, the RCMG is computed (lines 8–10) based on the distances calculated in the

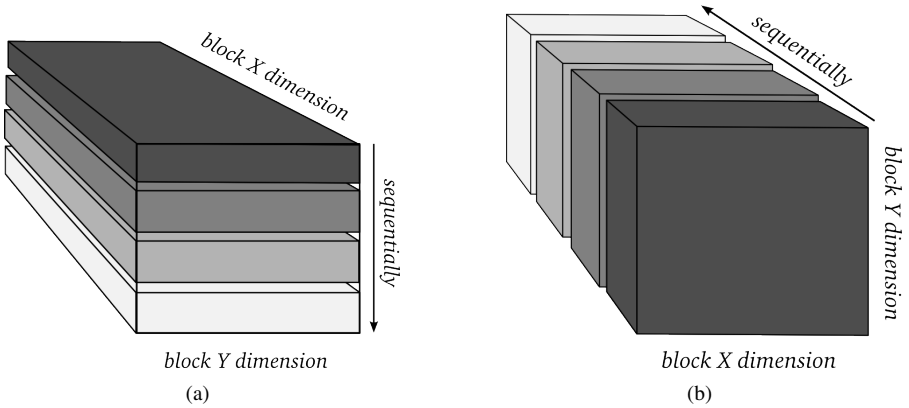


Figure 5.1: Block configuration for spatial (a) and spectral (b) partitioning.

previous step. This kernel makes use of the shared memory at each step, so data are reused within the block.

Spatial partitioning algorithm for RCMG

In this implementation, threads in the X dimension load different components of the same pixel vector simultaneously into the shared memory. In each block, the thread `t1` loads one feature of one pixel vector of the image, the thread `t2` the second feature of the same pixel vector of the image, and the thread `tk` the k-th feature with k the number of hyperspectral bands of the image.

With the data of a pixel vector in shared memory, each thread computes a partial result $(\mathbf{x}_k^i - \mathbf{x}_k^j$, with $i, j \in \chi$ as described in (2.8). First, threads in the X dimension cooperate in a parallel reduction [73] within the block for computing the CMG. Half of the threads work in the reduction, and the number of active threads is halved at each iteration as the reduction proceeds. Second, one thread per pixel vector finds the pair of pixels that generated the maximum distance and computes the RCMG with the remaining distances. It should be noted that the distances are stored in shared memory, and therefore are available for all threads within the block. Finally, in the third step, the RCMG is written in global memory, resulting in one band gradient image which is kept in the global memory of the GPU.

Input: hyperspectral dataset $\mathbf{X}$
Output: one band image

1: **for** each band k of $\mathbf{X}$ **do**
2: load band k in shared memory ▷ step 1 (SM)
3: **for** each pixel x in band k **do** ▷ step 2 (SM)
4: compute and accumulate the corresponding term $\mathbf{x}_k^i - \mathbf{x}_k^j$ in the Euclidean distance $\mathbf{D}_{i,j} | i, j \in \chi$, with χ the set of neighbors of pixel x.
5: **end for**
6: synchronize thread within the block
7: **end for**
8: compute $\mathrm{CMG}(\mathbf{X}) = \max_{i,j \in \chi} \mathbf{D}_{i,j}$
9: find the pair of pixels $R_s = (i, j) | \mathbf{D}_{i,j} = \mathrm{CMG}(\mathbf{X})$
10: compute $\mathrm{RCMG}(\mathbf{X}) = \max_{i,j \in \chi - R_s} \{\mathbf{D}_{i,j}\}$ ▷ step 3 (SM)
11: store $\mathrm{RCMG}(\mathbf{X})$ to global memory

SM states for computation in shared memory.

Figure 5.2: Pseudocode for the RCMG CUDA kernel executed in shared memory (spectral-domain partitioning)

Spectral partitioning algorithm for RCMG

In the spectral partitioning implementation, each thread processes all the spectral components of a pixel vector in a loop through all the hyperspectral bands. The pseudocode shown in Figure 5.2 corresponds to the spectral partitioning algorithm. At each iteration k, all the threads load data in shared memory corresponding to the k-th band, computing the partial results $(\mathbf{x}_k^i - \mathbf{x}_k^j)$ for each pair of neighbors i, j. At the end of the loop, all the distances for each pixel are available in shared memory. To compute the CMG, each thread finds the maximum of the distances of its set χ, (step two in Figure 5.2), and the corresponding pair of pixels which generated that maximum (line 9). Having identified the pixel vectors that are furthest apart, each thread computes the RCMG with the remaining distances, (step three in Figure 5.2), and writes the result back to global memory, which is the last step of the algorithm. This implementation requires less shared memory that the previous one owing to the sequential scanning in the spectral domain.

RCMG		128×4	32×4	32×8	32×16
Spatial Partitioning	L1	(∅)	na	na	na
Spectral Partitioning	L1	na	1	(∅)	(∅)
Spatial Partitioning	Sh	**1**	na	na	na
Spectral Partitioning	Sh	na	**4**	2	1

Table 5.1: Number of active blocks per SMX for the spatial-domain and spectral-domain partitioning, based on the block size and the shared memory requirements. L1 indicates that 48 KB are used for the L1 memory and 16 KB for the shared memory. Sh states for the opposite configuration. Results for Pavia University dataset. The best occupancy is indicated in bold.

Performance analysis for RCMG on GPU

In order to include the best implementation of the RCMG in the CA–WSHED–GPU scheme, we have first evaluated the performance on the Intel quad-core i7-860 microprocessor and the GTX 680 GPU based on the Kepler architecture (compute capability 3.0). The RCMG is compared to a parallel multi-threaded CPU implementation using OpenMP. The OpenMP implementation uses 4 threads and it is based on the spectral-domain partitioning approach. The work is scheduled statically among the threads, through a loop construct. The datasets used in this analysis are the Pavia University which has 103 spectral bands and the Salinas Valley dataset with 220 spectral bands. Details of these datasets are described in Section 2.9.3.

Different block configurations were tested and finally the spectral partitioning RCMG implementation was configured with blocks of 32×4 threads. For the spatial partitioning RCMG implementation, 128×4 threads per block and 256×2 threads per block were considered. Each block in the spatial partitioning approach processes a region of 4×4 pixel vectors for the first case and a region of 2×2 pixel vectors for the second one. Thus, each thread in a block processes 4 or 2 pixel vectors in this implementation.

Table 5.1 shows the maximum number of active blocks for the each implementation using double precision arithmetic. The text "na" states that the block configuration was not available for that implementation, and (∅) that insufficient resources are available for that configuration. For example, the spatial partitioning RCMG if configured with 128×4 threads per block requires 42240 bytes of shared memory per block, in order to compute the distances of each pixel vector in shared memory. Thus, using the L1 configuration with 16 KB of shared memory is not enough to execute at least one block in the SMX. The operations are performed

Part of these results have been in [137, 136].

Spatial-domain Partitioning	OpenMP (4 threads)	GTX 680 simple	GTX 680 double
University of Pavia	0.1702s	0.0537s (2.8×)	0.1317s (1.3×)
Salinas Valley	0.1959s	0.0638s (3.1×)	∅
Spectral-domain Partitioning	**OMP (4 threads)**	**GTX 680 simple**	**GTX 680 double**
University of Pavia	0.1702s	**0.0085s** (17.8×)	**0.0231s** (7.3×)
Salinas Valley	0.1959s	**0.0092s** (21.3×)	**0.0272s** (6.4×)

Table 5.2: Performance results for the spatial and spectral partitioning algorithms for the RCMG computing. Execution times in seconds. Speedups based on the OpenMP implementation using four threads. Best results in bold.

in double precision, which requires twice the amount of memory than simple precision arithmetic.

The computation of the distances between all pairs of pixel vectors requires a lot of shared memory. By using 4–connectivity, we have 10 pairs of combinations in the set χ (see Section 3.2.1), and 36 pairs in the case of 8–connectivity. The highest number of concurrent blocks is always achieved with the Sh configuration, i.e., using 48 KB of shared memory.

Table 5.2 shows the execution time and the speedup using 4-connectivity. The best results are for the spectral partitioning RCMG with speedups of 17.8× and 21.3× operating in simple precision. The shared memory requirements for the spectral partitioning RCMG are 5.7 KB (simple) and 11.4 KB (double) per block, while the spatial partitioning RCMG requires up to 20.6 KB (simple) and 41.2 KB (double). Thus, more blocks per SMX are concurrently executed in the spectral approach which leads to a better speedup. The same applies to the case of double precision arithmetic.

By performing the calculations in double precision arithmetic, which is the case for the spectral-spatial classification scheme, we observe that the spatial partitioning approach, where the pixel vectors are kept as a whole, does not work for the Salinas hyperspectral image. The reason is that the shared memory requirements for a block size of 256×2 threads rises to 76.3 KB. Thus, there is insufficient shared memory in the SXMs to execute one block of threads. However, as the computation can be done individually in each band, the spectral-domain partitioning can be used for computing the RCMG in GPU. The spectral partitioning RCMG obtains speedups of 6.5× for the University of Pavia and 7.2× for the Salinas valley image,

comparing the execution time in double precision arithmetic with the OpenMP implementation.

5.2.2 Artifacts-free CA-Watershed on GPU

The artifacts-free CA–Watershed algorithm creates the segmentation map from the gradient image produced by the RCMG. The implementation of this algorithm is based on the Block–Asynchronous (BA) strategy proposed in this thesis. A comprehensive explanation of the implementation is given in Section 4.2.1. This stage creates the segmentation map which is kept in the GPU global memory. We recall that the regions in the segmentation map are labelled with unique labels identifying pixels of each region, as illustrated in Figure 5.3. As a region can be assigned to different blocks, all the pixels belonging to the same segmented region must be connected. When connecting regions of pixels, one pixel is used as the root of the region and identifies uniquely all the pixel of the same region. If the labels for identifying a region are assigned properly, for example by identifying the root by its position in the image, as illustrated in Figure 5.3(a), the pixels of each segmented region will be implicitly connected as shown in Figure 5.3(b).

5.2.3 Majority vote on GPU

The MV is a data fusion strategy used to join the spectral and spatial information (See Section 3.1.3). The thematic map created by the SVM classifier is regularized by summing up the votes that identify the spectral class for each pixel within the region they belong to in the

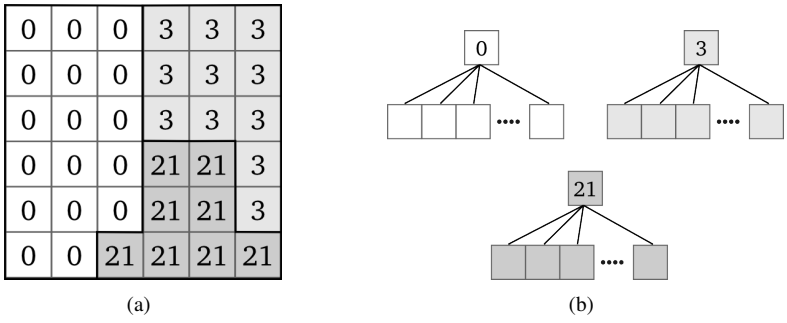


Figure 5.3: An example of an image segmented into three regions identified by its position in the image (a), and the representation of the connected components created from the root of the regions (b).

Input: thematic map and segmentation map
Output: final classification map (spectral-spatial data fusion)

1: <count number of segmented regions> ▷ (GM)
2: <count number of pixels of each class in each region> ▷ voting step (GM)
3: <obtain the winner class for each region> ▷ winner step (GM)
4: <update the pixels inside each region to the winner class> ▷ updating step (GM)

GM states for computation in global memory.

Figure 5.4: Pseudocode for the majority vote implementation on GPU.

segmentation map. By using the segmentation map created by the CA–Watershed algorithm (pixels of each segmented region are already connected), the MV can be projected in the GPU in three kernels, as illustrated in the pseudocode in Figure 5.4: count the number of votes within each region (line 2), find the winner of the voting (line 3) and updating the final classification map (line 4). The kernels calls are marked between the $<$ and $>$ symbols. Each kernel is configured to work in one dimensional blocks of threads.

As the number of segmented regions is unknown a priori, a direct implementation would be to allocate in global memory data structures of a large enough size to compute as many regions as pixels in the image. This would be the worst case where double the size of the image is required. With the aim of saving memory resources, the number of regions generated by the CA–Watershed algorithm are calculated prior to the voting step by an auxiliary kernel (line 1 in Figure 5.4). This kernel counts the number of regions in global memory using atomic operations. Once the number of regions is known, a two-dimensional data structure is defined in global memory with the number of watershed regions and the number of spectral classes being the dimensions of the structure.

In the voting kernel (line 2), each thread operates on one pixel of the image and adds one vote to the class within its region. One majority vote per watershed region is performed. As two or more threads can vote in the same region to the same class with no predictable order, the voting is performed by atomic operations in global memory.

In the winner kernel (line 3) each thread operates on the information collected for one region of the segmentation map. Thus, for this kernel one block per segmented region is configured to be launched on the GPU. Each thread finds the class with the maximum number of votes (winning class) and saves its class identifier in global memory.

Finally, the last kernel (line 4) produces the final spectral–spatial classification map updating each pixel with the winner class obtained in the previous kernel.

5.2.4 Results

This section presents the results obtained for the CA–WSHED–GPU scheme based on its execution time and speedup. The scheme is configured to use 4-connectivity. The RCMG used is based on the spectral-domain partitioning approach. The segmentation is based on the artifacts-free CA–Watershed algorithm. The best parameters for the SVM are based on the analysis of the CA–WSHED–MV scheme given in Section 3.2. The reference implementations for comparison are an optimized OpenMP (4 threads) parallel implementation of the RCMG, CA–Watershed, MV and SVM.

Experimental setup

The CA–WSHED–GPU implementation has been evaluated on the Intel quad-core i7-860 microprocessor. The GPUs used in the experiments are the GTX 680 and the GTX TITAN based on the Fermi and Kepler architectures, respectively. The CUDA code has been compiled under Linux using the nvcc compiler with the CUDA toolkit. The experiments are executed under Linux using the gcc compiler version 4.6.3 for the OpenMP implementations, and the nvcc compiler version 5.0 (GTX 680) and 5.5 (GTX TITAN) for the case of the CUDA implementations, respectively, with full optimization flags (-O3) in both cases. The OpenMP implementation uses 4 threads scheduling the work statically among the threads by a loop construct distributing the iterations into 4 chunks of the same size (one per thread), in order to evenly distribute the workload among the threads and to achieve a high locality in the data accesses. Each execution time is calculated as the average of 10 executions.

The datasets used are the Pavia University acquired by the ROSIS-03 sensor and the Salinas Valley dataset adjured by the AVIRIS sensor. There is a full description of these datasets in Section 2.9.3. Table 5.3 summarizes the spatial and spectral dimensions and the size in MBs of these images.

Part of these results have been published in P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Computing efficiently spectral-spatial classification of hyperspectral images on commodity gpus," *in Recent Advances in Knowledge-based Paradigms and Applications* (J. W. Tweedale and L. C. Jain, eds.), vol. 234 of Advances in Intelligent Systems and Computing, Ch. 2, pp. 19–42, Springer International Publishing, 2014.

Dataset name	Dimensions	Size (MB)	Source
Pavia University	$610 \times 340 \times 103$	162.9	ROSIS-03
Salinas Valley	$512 \times 217 \times 224$	189.9	AVIRIS

Table 5.3: Datasets used in the CA–WSHED–GPU analysis. There is a full description of these datasets in Section 2.9.3.

Analysis of the GPU parameters

Table 5.4 gives the block size used in each one of the stages of the CA–WSHED–GPU Scheme, as well as the on-chip memory configuration. The analysis of the GPU parameters for the RCMG, CA–Watershed and SVM implementations was given in Section 5.2.1, Section 4.2, and Section 4.3 but the configurations are included in this table to facilitate the analysis. The text "na" in the table indicates that the block configuration was not available for the kernel, and (∅) that insufficient resources are available for that configuration. For example, our SVM Implementation on GPU is optimized by using 128×1 threads per block.

The RCMG is limited by the amount of shared memory and is configured with 32×4 threads per block. The on-chip memory is configured with 48 KB for shared memory. The kernel computing the CA–Watershed is limited by the maximum number of concurrent blocks per SMX (16) using blocks of 32×4 threads, and by the maximum number of active warps per SMX (64) using blocks of 32×8 and 32×16 threads. So, in the CA–Watershed kernel any of the three block sizes reaches maximum occupancy. However, by using a rectangular

		128×1	32×4	32×8	32×16
RCMG (Spectral Partitioning)	L1	na	1	(∅)	(∅)
CA–Watershed	L1	na	7	4	4
Majority Vote	L1	**16**	na	na	na
SVM (RBF kernel)	L1	**16**	na	na	na
RCMG (Spectral Partitioning)	Sh	na	**4**	2	1
CA–Watershed	Sh	na	**16**	8	4
Majority Vote	Sh	16	na	na	na
SVM (RBF kernel)	Sh	16	na	na	na

Table 5.4: Number of active blocks per SMX for the different kernels based on the block size and the shared memory requirements. L1 indicates that 48 KB are used for the L1 memory and 16 KB for the shared memory. Sh states for the opposite configuration. Results for Pavia image. The best occupancy is indicated in bold.

	University of Pavia			Salinas Valley		
	OMP (4 threads)	GTX 680		OMP (4 threads)	GTX 680	
HtoD	–	0.0801s	(-)	–	0.1078s	(-)
RCMG	0.1702s	0.0231s	(7.3×)	0.1752s	0.0272s	(6.4×)
CA–Watershed	0.0307s	0.0014s	(21.9×)	0.0194s	0.0008s	(24.2×)
MV	0.0027s	0.0003s	(9.0×)	0.0020s	0.0002s	(10.0×)
SVM	23.2992s	3.1554s	(7.3×)	42.4208s	3.3902s	(12.5×)
TOTAL	23.5028s	3.2603s	(7.2×)	42.6174s	3.5662s	(11.9×)

Table 5.5: Performance results on the GTX 680. Execution times in seconds. Speedup in brackets.

block, with the longest dimension being the one along which data is stored in global memory, the data of the border is packed in the minimum number of cache lines. In this way the overhead associated to global memory accesses is minimized [10]. The MV does not use shared memory and the SVM requires a small amount of this type of memory, so both kernels are configured with 128×1 threads per block. This configuration maximizes the use of the L1 cache and each SMX is fully exploited with 16 blocks simultaneously active.

Execution times

In the following, we analyze the results of the CA–WSHED–GPU scheme over the hyperspectral images of the University of Pavia the valley of Salinas. Table 5.5 and Table 5.6 present the execution times and speedups obtained in the GTX 680 and GTX TITAN for each dataset.

The CA–WSHED–GPU scheme is first evaluated on the GTX 680. The total execution time (Table 5.5) are 3.2603 seconds and 3.5662 seconds for the Pavia City and Salinas Valley datasets, respectively. The speedups are $7.2\times$ and $11.9\times$ compared to the OpenMP implementation. By looking at the different stages, the CA–Watershed achieved speedups of $21.9\times$ and $24.2\times$. As explained in Section 4.2.1, the asynchronous CA–Watershed presents the advantage of reusing information within each thread block, efficiently exploiting the shared and cache memories of the GPU. Therefore, it achieves better results when the image has large plateaus, such as the Salinas Valley image. Regarding the majority vote, the reference time in CPU is very low (0.0027 and 0.0020 seconds), However, the GPU implementation obtained speedups of $9.0\times$ and $10.0\times$ for the datasets used in the test. The time shown in Table 5.5 for the MV includes the auxiliary kernel for counting the watershed regions.

	University of Pavia			Salinas Valley		
	OMP (4 threads)	GTX TITAN		OMP (4 threads)	GTX TITAN	
HtoD	–	0.0801s	(-)	–	0.1078s	(-)
RCMG	0.1702s	0.0159s	(10.7×)	0.1752s	0.0181s	(9.6×)
CA–Watershed	0.0307s	0.0012s	(25.6×)	0.0194s	0.0008s	(24.2×)
MV	0.0027s	0.0003s	(9.0×)	0.0020s	0.0003s	(6.7×)
SVM	23.2992s	2.0593s	(11.3×)	42.4208s	2.6584s	(15.9×)
TOTAL	23.5028s	2.0767s	(11.3×)	42.6174s	2.6776s	(15.9×)

Table 5.6: Performance results on the GTX TITAN. Execution times in seconds. Speedup in brackets.

The CA–WSHED–GPU scheme is also evaluated on the GTX TITAN. The execution times and the speedups are given in Table 5.6. The results are similar to those obtained on the GTX 680, with a significant difference in the RCMG obtaining higher speedups, 10.7× (University of Pavia) and 9.6× (Salinas Valley), as compared to the OpenMP implementation. The GTX TITAN has 15 SMXs, 1.8× more than the GTX 680, and 1536 KB of L2 cache, that is 3× the size of L2 cache in the GTX 680 (see Section 2.3 for more details). The increase in the hardware resources improves execution times.

The total execution time is below the 3 seconds for both datasets which is close for real-time applications.

5.2.5 Final discussion

The CA–WSHED-MV scheme, described in Section 3.2, has been efficiently implemented using CUDA, exploiting the memory hierarchy and the thousands of threads available in the GPU architecture. The different stages of the scheme have been concatenated to minimize the data transfers between the CPU and the GPU and to maximize the computational throughput. Different hyperspectral data partitioning strategies and thread block arrangements were studied in order to have a larger number of blocks being concurrently executed. The performance in terms of execution times and speedups was evaluated on a GTX 680 and a GTX TITAN, and compared to a multi-threaded OpenMP implementation. The execution time for the whole classification process was better in the GTX TITAN and it was below 3.0 seconds, with a speedup of 15.9× for the Salinas Valley dataset. This is close to (but not within) real-time performance for the AVIRIS sensors as the cross-track line scan time, a push-broom

instrument, is 8.3 ms (to collect 512 full pixel vectors) [127]. Therefore, the limit can be established at 1.8 seconds for an image of 512×217 pixels, such as the Salinas Valley, to fully achieve real-time performance. Although these are promising results for on-board processing of hyperspectral information, other spectral-spatial classification schemes more suitable for the efficient projection thereof on GPU should be investigated.

5.3 WT–EMP–GPU

In this section, the WT-EMP scheme proposed in Section 3.3 is mapped to the GPU using CUDA to achieve real-time classification. The following listing summarizes the different stages of this scheme:

1. 1D-DWT feature extraction: this stage computes the CDF97 wavelet several times in the spectral domain to reduce the dimensionality of the hyperspectral image.

2. EMP: it is the extended morphological profile created from wavelet coefficients. This stage creates a morphological pixel vector with the spatial information.

3. 2D-DWT Denoising: in this stage, the hyperspectral image is denoised by soft thresholding. We have used the Double-Density DWT [148].

4. Fusion via Stack Vector: in this stage, the denoised pixel vector and the morphological pixel vector are joined.

5. SVM: it is the classification of the hyperspectral image on GPU. This stage uses as input the new stacked vector and creates the classification map.

The proposed 1D-DWT implementation is adapted to compute thousands of transformations (pixel vectors in the spectral domain) in parallel. An asynchronous reconstruction algorithm, based on the hybrid iterative updating process described in Section 4.2 is used to compute the EMP. In the case of the 2D-DWT, the GPU implementations [62, 6] cannot manage more than two filters. Therefore, a new implementation is required to manage the three filters that are used in the denoised step.

The kernels have been specially adapted to optimize the use of the GPU resources, as well as the memory hierarchy. By using techniques that match the computing model of the GPU architecture, such as the local computation of the DWT and the morphological operations, the scheme can be efficiently implemented on the GPU.

Input: hyperspectral dataset **X**
Output: spectral-spatial classification map

1: copy **X** to global memory

2: <compute 1D-DWT on each pixel vector of **X** > ▷ 1D-DWT feature extraction (SM)

3: **for** each band-coefficient W_i in the transformed data by wavelets **do**
4: <create a morphological profile for W_i > ▷ EMP (SM)
5: **end for**

6: <compute 2D-DWT denoising on each band of **X** > ▷ 2D-DWT denoising (SM)

7: <data fusion via stack vector> ▷ (GM)

8: <classify the stack vector by SVM> ▷ (GM and SM)

GM states for computation in global memory and SM states for computation in shared memory.

Figure 5.5: Pseudocode for the WT–EMP–GPU implementation.

The pseudocode in Figure 5.5 shows the WT–EMP–GPU workflow. Each step has been implemented in different kernels, marked between $<$ and $>$ symbols in the pseudocode. The stages and the kernels are described in the following sections.

5.3.1 1D-DWT feature extraction on GPU

The 1D-DWT is applied to each pixel vector in the spectral domain using the Cohen-Daubechies-Feauveau 9/7 wavelet (CDF97) with nine coefficients for the low-pass filter. Only the approximation coefficients are used at the next level of decomposition, while the detail coefficients are discarded. It should be note that in this case there is no need to compute the inverse 1D-DWT. The CDF97 coefficients used by the 1D-DWT are shown in Table 3.5.

The GPU 1D-DWT proposed in this work is adapted to compute thousands of transformations of the pixel vectors in parallel. The hyperspectral image is copied to the global memory of the GPU as a matrix with one pixel vector per row. Threads are configured in one dimensional blocks of size the multiple of 32 nearest to the number of hyperspectral bands; i.e., 128×1 or 256×1. With these configurations we can optimize the number of active blocks that are executed on each SM (see Section 4.1.4). The analysis of the GPU parameters will be discussed in more detail in Section 5.3.4 on page 154.

Input: hyperspectral dataset **X**
Output: band-coefficients **W**

```
1: load pixel vector in shared memory
2: for each level of decomposition do
3:     padding the last elements of the shared memory array
4:     synchronize threads within the block                    ▷ local synchronization
5:     compute circular convolution according to Eq. (2.18a)   ▷ (SM)
6:     synchronize threads within the block                    ▷ local synchronization
7: end for
8: write the output to global memory
```

SM states for computation in shared memory.

Figure 5.6: Pseudocode for the 1D-DWT CUDA kernel executed in shared memory

In the kernel shown in Figure 5.6, threads within the block load data from the global to the shared memory (line 1), where the convolution for the 1D-DWT takes place (line 5). Extra data are loaded with a circular padding in shared memory (line 3) for the convolution of the last elements. The kernel performs all the spectral decompositions (lines 2–7) in shared memory. Thus, each decomposition is temporarily stored in shared memory. Threads are synchronized within the block before the next level of decomposition, so data can be safely shared among them. Finally, the band-coefficients **W** are copied to global memory. Data are rearranged as a hyperspectral cube before the next step.

5.3.2 Asynchronous reconstruction algorithm applied to EMP on GPU

The EMP, see Section 3.3.2, is the set of morphological profiles created through opening and closing by reconstruction for each coefficient-band resulting from the previous step.

In order to compute the EMP using a SE of increasing size (up to 7 in the WT-EMP scheme), the shared memory allocated for the block must be extended with a border of size 7. As explained in Section 4.1.1 some operations require information of the neighboring pixels. In this case, the neighborhood is defined by the radius of the structuring element.

The basic morphological operations (erode / dilate) have been carried out by the NVIDIA Performance Primitives (NPP) library. The NPP has a collection of GPU-accelerated image and video processing functions that deliver up to 10× faster performance than comparable CPU-only implementations [119]. However, this library does not include advanced morphological operations such as opening and closing by reconstruction.

Input: band-coefficients **W**
Output: extended morphological profile (EMP)

```
1:  for each band W_i do
2:      for each r ∈ 1,3,5,7 do
3:          <erode W_i with a structuring element of radius r >          ▷ NPP library
4:          <block–asynchronous reconstruction>                          ▷ opening by BAR (SM)

5:          <complement the input data W_i >                             ▷ (GM)
6:          <dilate W_i with a structuring element of radius r >         ▷ NPP library
7:          <complement the output of the dilatei >                      ▷ (GM)
8:          <block–asynchronous reconstruction>                          ▷ closing by BAR (SM)

9:          <complement the output of the reconstruction>                ▷ (GM)
10:     end for
11: end for
```

GM: global memory, SM: shared memory, NPP: NVIDIA Performance Primitive library.

Figure 5.7: Pseudocode for the EMP implementation on GPU.

The opening (closing) by reconstruction is implemented in this thesis using the block–asynchronous reconstruction (BAR) algorithm described in Section 4.2.2. We recall that the concepts regarding the mathematical morphology are described in 2.5.1. So, in this section we focus on the steps required to create the EMP. The pseudocode in Figure 5.7 shows the steps for computing the EMP on GPU, which describes in detail the lines 3–5 of the pseudocode detailed in Figure 5.5.

The input data W (band-coefficients) are already in the global memory of the device. For each band-coefficient W_i a morphological profile is calculated. The EMP creates 4 openings by reconstruction (line 4 in Figure 5.7) and 4 closings by reconstruction (line 7 in Figure 5.7). The closing is computed with the same BAR algorithm but using the complement of the image (line 5 and 8). The BAR algorithm performs multiple iterations on GPU reusing the data in shared memory as explained in Section 4.2.2 . The results are kept in the global memory of the GPU.

5.3.3 2D-DWT denoising adapted to three filters on GPU

The 2D-DWT denoising applied in this scheme uses the set of filters for perfect reconstruction presented in [148]. Three filters are used: one low-pass filter h and two high-pass filters $g1$ and

$g2$, shown in Table 3.6. In order to implement this wavelet on GPU, the techniques described in Section 2.6 must be adapted.

Implementation

This implementation uses two kernels for the forward DWT. First, the 1D-DWT by rows is applied to the original image to produce three temporal subbands (one for each filter) L, H1, H2, as shown in Figure 5.8(a). Second, the 1D-DWT by columns applied to the temporal subbands produces nine new subbands corresponding to the LL, LH1, LH2, H1L, H1H1, H1H2, H2L, H2H1, and H2H2 subbands, as shown in Figure 5.8(b).

Both kernels apply three convolutions (see Section 2.6) between one pixel and each filter in the same kernel call, so data are reused among the filter convolutions. Threads within the block read two data items in shared memory. In the transformation by rows, the last threads of each block load additional extra data from global memory to shared memory, as illustrated in Figure 5.9(a). To achieve this, the shared memory used per block is extended with four values, corresponding to the data necessary to compute all the convolution in shared memory. The grey squares correspond to the extra data allocated in shared memory. Each thread within the block computes the convolution of one pixel of the image with each filter and save the results in global memory.

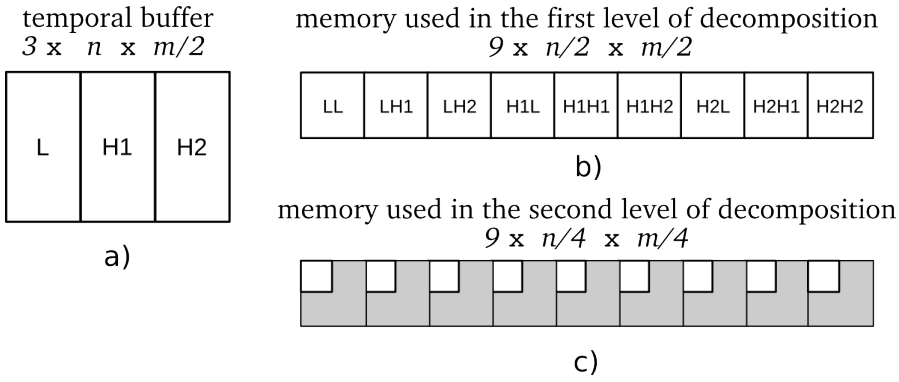


Figure 5.8: 2D-DWT global memory requirements (indicated with a white background), for an image of $n \times m$ pixels, using a set of three filters and two levels of wavelet transform. (a) Temporal buffer for separable wavelet transform, (b) and (c) memory used in the first and the second level of decomposition, respectively.

In the transformation by columns, the threads of the first rows perform the convolution in shared memory, whereas the last threads perform the convolution directly with the data read from the global memory. The reason is that the shared memory requirements are higher in the 1D-DWT by columns and increasing even more the shared memory use for all the threads would reduce the number of active blocks per SMX, decreasing the performance. Figure 5.10(a) shows the data read from global to shared memory. The 1D-DWT by columns requires almost twice the shared memory used in the 1D-DWT by rows (see Figure 5.9(a)).

The access pattern in global memory is coalesced, i.e., consecutive threads access to consecutive memory locations. In addition, if the number of threads within the block in the row dimension is multiple of 32, the GPU can execute the same instruction on different data without any divergence in the code (see challenges of GPU computing in Section 4.1.4).

After performing L decomposition levels, a soft-thresholding is applied to the detail coefficients at each level, corresponding to the last eight subbands in Figure 5.8(b) and Figure 5.8(c). The kernel is executed for each hyperspectral band after the forward 2D-DWT. Each thread computes the soft-thresholding directly in global memory. After soft-thresholding, an inverse 2D-DWT is applied first by columns and then by rows to reconstruct the denoised image.

Memory requirements

The first level of decomposition needs space for nine subbands of $n/2 \times m/2$ pixels for an image of $n \times m$ pixels, as shown in Figure 5.8b. The second level needs space for nine subbands of $n/4 \times m/4$ pixels, as shown in Figure 5.8(c), and so on. In order to simplify the

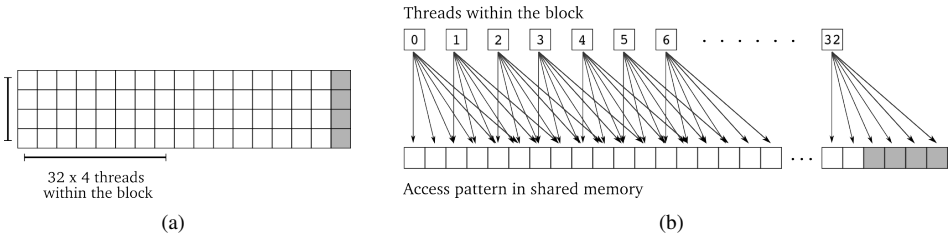


Figure 5.9: 1D-DWT by rows kernel: data read from global to shared memory by a block of 32×4 threads (a), and shared memory access pattern for each thread within the block (b). Each square in (a) represents 4×1 data items in the shared memory. The grey squares correspond to the data allocated in the extended shared memory used in the kernel.

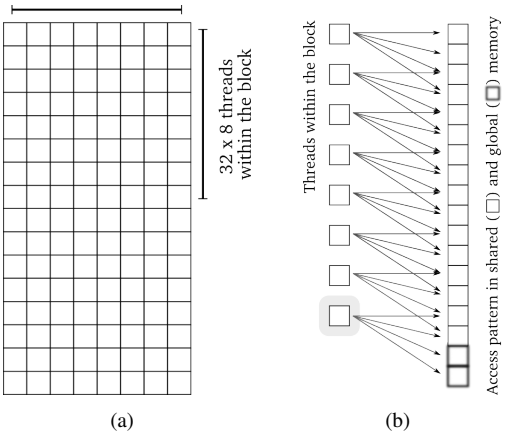


Figure 5.10: 1D-DWT by columns kernel: data read from global to shared memory by a block of 32×8 threads (a), and shared memory access pattern for each thread within the block (b). Each square in (a) represents 4×1 data items in the shared memory.

memory access pattern and keep the data aligned in memory, $9 \times n/2 \times m/2 \times L$ subbands are allocated at the beginning of the process, where L is the number of levels of decomposition applied to the image. The memory that is allocated but not used in the second and successive decompositions is, for the case of a 610×340 image, less than 1% of the total amount of memory available in a GPU with 1 GB of global memory. This memory space is shaded in grey in Figure 5.8c.

A temporal buffer of $3 \times n \times m/2$, see Figure 5.8a, is used for the results produced by the 1D-DWT by rows. All the memory allocated is reused for each hyperspectral band of the original image where the soft-thresholding is applied. The drawback of this 2D-DWT in terms of memory requirements, compared to the case of two filters, is that the subbands cannot be stored in one quarter of the original image as nine bands are generated instead of the four bands of the general case. Therefore, extra global memory and shared memory have been allocated to improve the memory access pattern and keep the data memory aligned.

Dataset name	Dimensions	Size (MB)	Source
Pavia University	$610 \times 340 \times 103$	162.9	ROSIS-03
Hekla Volcano	$500 \times 600 \times 157$	402.5	AVIRIS

Table 5.7: Datasets used in the WT–EMP–GPU analysis. There is a full description of these datasets in Section 2.9.3.

5.3.4 Results

This section presents the results obtained for the WT–EMP–GPU implementation based on its execution time and speedup compared to an efficient multi-threaded OpenMP implementation.

Experiment Setup

The WT–EMP–GPU implementation has been evaluated on the GTX TITAN (Kepler architecture). The reference code in CPU is an optimized OpenMP parallel implementations of the same scheme considering 4 threads. The CUDA code has been compiled under Linux using the nvcc compiler with CUDA 5.5 with full optimization flags (-O3) in both cases. Each execution time is calculated as the average of 10 executions.

Different configurations of the number of threads per block are studied, and based on the best configuration, the performance of the scheme is analyzed for real-time hyperspectral image classification over two remote sensing images: the Pavia University dataset acquired by the ROSIS-03 sensor and the Hekla Volcano dataset acquired by the AVIRIS sensor. The spectral and spatial characteristics of the images are described in Section 2.9.3 and summarized in Table 5.7.

Analysis of the GPU parameters

The GPU used in the experiments is based on the Kepler architecture, with compute capability 3.5. The GTX TITAN has 14 SMXs with the following limits per SMX: 2048 threads, 16 active blocks, 65536 registers of 32 bits and 64 KB of on-chip memory that can be configured as a shared memory of 16 KB and 48 KB for the L1 cache or the opposite configuration.

Part of these results have been published in P. Quesada-Barriuso, F. Argüello, D. B. Heras, and J. A. Benediktsson, "Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. PP, no. 99, pp. 1–9, 2015 (published online, print edition pending).

		128×1	32×4	32×8	32×16
1D-DWT	L1	**16**	na	na	na
2D-DWT (rows)	L1	na	7	3	1
2D-DWT (columns)	L1	na	na	4	2
Soft-thresholding	L1	**16**	**16**	8	4
EMP (reconstruction)	L1	na	**16**	8	4
SVM (RBF kernel)	L1	**16**	na	na	na
1D-DWT	Sh	**16**	na	na	na
2D-DWT (rows)	Sh	na	**16**	8	4
2D-DWT (columns)	Sh	na	na	**8**	4
Soft-thresholding	Sh	**16**	**16**	8	4
EMP (reconstruction)	Sh	na	**16**	8	4
SVM (RBF kernel)	Sh	16	na	na	na

Table 5.8: Number of active blocks per SMX for the different kernels based on the block size and the shared memory requirements. L1 indicates that 48 KB are used for the L1 memory and 16 KB for the shared memory. Sh states for the opposite configuration. Results for Pavia image. The best occupancy is indicated in bold.

The different kernels used in the GPU implementation have been analyzed according to these resources. The number of registers used per thread in the different kernels was always lower than the maximum number of registers available per thread.

Table 5.8 shows the maximum number of active blocks based on the block size and the on-chip memory configuration. The text "na" in the table indicates that the block configuration was not available for that kernel. For example, on the one hand, the 2D-DWT by columns requires a minimum of 8 threads in the second dimension in order to compute the convolution in shared memory. On the other hand, for the 1D-DWT and the SVM kernels the block size is adapted to the multiple of 32 nearest to the size of the pixel vector (128 for the image of Pavia). For these kernels there are no resources which limit the number of active blocks. Therefore, the on-chip memory is configured to maximize the L1 cache size.

From Table 5.8 it can be observed that the 2D-DWT (by rows) can only be executed by 16 active blocks, the maximum possible for the GTX TITAN, when the GPU is configured to use 48 KB of shared memory. The reason is that the kernel requires 2176 bytes of shared memory per block. Therefore, in order to have 16 active blocks per SMX, 34816 bytes of shared memory are required for this kernel.

The kernel computing the 2D-DWT (by columns) is limited by the amount of shared memory and the maximum number of threads per SMX. This kernel uses 4 KB of shared

memory. In the Sh configuration there is sufficient shared memory for 12 blocks. However, as the block size is 32×8, that is 256 threads per block, the limit of 2048 threads per SMX is reached before the limit of shared memory, resulting in a maximum number 8 active blocks.

The EMP is based on the block-asynchronous reconstruction (BAR) algorithm. This implementation requires only 204 bytes of shared memory, so the on-chip memory is configure with 48 KB of L1 cache. Any block size configuration in the BAR algorithm will reach the limit in the number of threads per block or the number of active block per SMX before the limit in the shared memory.

Performance Analysis

Table 5.9 and Table 5.10 present the execution times and speedups obtained in the GTX TITAN for each dataset, respectively. The speedups are based on the OpenMP implementation using four threads.

The first thing to note is that the GPU total execution time is below 2.5 s. This result represents a speedup of 8.2× and 6.3× for the University of Pavia and the Hekla images, respectively, including the data transfer from the CPU to the GPU memory. Regarding the 1D-DWT, speedup values of 28.7× and 30× are obtained for each one of the images. These improvements are mainly obtained through the use of the shared memory. All the 1D-DWT levels of decomposition are applied in the same kernel call. Therefore, data are loaded once from global memory to shared memory and reused at each level of the wavelet decomposition. The speedups obtained for the 2D-DWT implementation, 3.4× and 3.9×, respectively, for each image, are within the expected values for a separable 2D-DWT convolution

University of Pavia	OpenMP (4 threads)	GTX TITAN	Speedup GTX TITAN
HtoD	—	0.071	—
1D-DWT	0.259	0.009	28.7×
EMP	0.714	0.104	6.9×
2D-DWT	0.748	0.221	3.4×
SVM	12.611	1.344	9.4×
TOTAL	14.342	1.749	8.2×

Table 5.9: Performance results for the WT–EMP–GPU scheme on the GTX TITAN. Execution times in seconds. Speedups based on the OpenMP implementation using four threads.

Hekla Volcano	OpenMP (4 threads)	GTX TITAN	Speedup GTX TITAN
HtoD	—	0.141	—
1D-DWT	0.630	0.021	30.0×
EMP	0.766	0.100	7.6×
2D-DWT	1.644	0.421	3.9×
SVM	12.515	1.755	7.1×
TOTAL	15.555	2.438	6.3×

Table 5.10: Performance results for the WT–EMP–GPU scheme on the GTX TITAN. Execution times in seconds. Speedups based on the OpenMP implementation using four threads.

by rows and columns. Similar speedups were obtained in [17]-[19] by comparing OpenMP and GPU implementations. Note that in Table 5.9 and Table 5.10 , the time required for the soft-thresholding is also included in the 2D-DWT step. The EMP obtained speedups of 6.9× and 7.6×, mainly due to the block-asynchronous implementation. The performance of this step is based on the block–asynchronous reconstruction (BAR) algorithm studied in detail in Section 4.2.2.

The execution time of the SVM classification takes approximately 76% of the overall time. This is the most computationally expensive step of the scheme and it has been speeded up 9.4× and 7.1×.

5.3.5 Final discussion

In this section the spectral-spatial classification scheme based on wavelets and mathematical morphology has been specifically adapted for efficient GPU computation (WT–EMP–GPU). The different steps of the scheme, 1D-DWT feature extraction, EMP, and 2D-DWT denoising, were specifically designed to exploit the hardware available in these architectures. The proposed 1D-DWT implementation was adapted to compute thousands of transformations in parallel. The different levels of the wavelet decomposition were applied in the same kernel reusing the data of the shared memory. In the case of the 2D-DWT, the implementation was designed with the purpose of managing the three filters that are used in the denoised step, and the number of times that data were loaded within the kernel was minimized by computing the three filters in the same kernel and by loading larger blocks of data.

The proposed BAR algorithm, which performs intra-block updates (asynchronous updates that reuse shared memory) and inter-block updates (that require global synchronization operations), was successfully applied to the EMP.

The scheme was tested on two real hyperspectral images, captured by the ROSIS-03, and the AVIRIS hyperspectral sensors. The execution time was speeded up 8.2× in the classification of the Pavia University dataset compared to an OpenMP implementation, and 6.3× in the Hekla Volcano, and it was below 2 and 2.5 seconds, respectively.

5.4 Conclusions

In this chapter we have presented our two proposals for efficient spectral-spatial n-dimensional image classification on low-cost computing infrastructures, such as the multi-threaded CPUs and many-core GPUs shipped in personal computers. The techniques and strategies investigated in Chapter 4 were successfully applied to the proposed schemes: CA–WSHED–GPU and WT–EMP–GPU. Both schemes were efficiently implemented using OpenMP and CUDA, including the classification stage by SVM.

The performance was evaluated in terms of execution times and speedups on a GTX 680 and a GTX TITAN, and compared to a multi-threaded OpenMP implementation executed in an Intel quad-core i7-860 CPU. The optimal GPU parameters were analyzed and different thread block arrangements were studied to maximize the number of active blocks. The schemes were tested on three real hyperspectral images: The University of Pavia, the Salinas Valley and the Hekla Volcano.

The execution time was below 3 seconds for all the datasets used in the experiments. For the CA–WSHED–GPU scheme, the execution time was 2.0767 seconds on the GTX TITAN, with a speedup of 11.3× for the University of Pavia image. For the WT–EMP–GPU scheme, the execution time dropped to 1.749 seconds for the same dataset.

By optimizing the kernels to efficiently exploit the GPU resources, the second proposal achieved real-time classification of the considered hyperspectral images. These results showed that the GPU is an adequate computing platform for on-board processing of hyperspectral information.

Conclusions

The efficient spectral-spatial classification of n-dimensional images in real time using low-cost computing infrastructures was addressed in this thesis. The focus was on designing and developing new schemes by producing good classification results in terms of accuracy, and developing techniques that allow their efficient computation in commodity hardware. In the following, we summarize the main contributions of this thesis and explain how the objectives have been met.

1. We have studied a general framework for spectral-spatial classification schemes, as well as the common data fusion strategies for joining the spectral and the spatial information. By understanding the basis of this framework, we have proposed two new spectral-spatial classification schemes based on segmentation and Mathematical Morphology (MM), taking into account their efficient computation on GPU. The classification was always carried out by a Support Vector Machine (SVM), a widely used classifier in remote sensing.

2. We have proposed one scheme, named CA–WSHED–MV, which incorporates the spatial information by a watershed transform based on cellular automata, and combines the spectral results of the classifier by a majority vote. The dimensionality of the hyperspectral image was reduced to one band by a vectorial gradient (RCMG) and then, a segmentation map was created from this one band image by the watershed transform. The spectral classification was carried out by the SVM. The segmentation map and the thematic map were combined by a majority vote. The main novelty of this scheme was introduced by the watershed algorithm based on cellular automata (CA–Watershed) for two major reasons: 1) the implementation does not create the so-called watershed lines, unlike other schemes based on the same segmentation algorithm, and thus there is no

need to compute a standard vector median to include those pixels in the voting; 2) the concept of parallelism is implicit in cellular automata and matches the computing model of the GPU, multi-core and many-core systems. Therefore, this step will be more efficient on commodity hardware.

3. We have proposed a second scheme, named WT–EMP, designed with the efficiency focused on producing good classification results, as well as an efficient computation on commodity hardware. This scheme was based on wavelets and MM. The Mathematical Morphology was used for creating an Extended Morphological Profile (EMP), and the wavelet transforms were used for feature extraction and for image denoising. The main contribution of the proposed scheme was that the EMP was created from the wavelet coefficients obtained by a 1D-DWT applied in the spectral domain, and not from other techniques for feature extraction. A second contribution was denoising the hyperspectral image prior data fusion via stack vectors, which increased the robustness on the scheme regarding the classification accuracies. The main benefits of this scheme were the good classification results and the low computational requirements.

4. We have developed techniques and analyzed strategies for efficient CPU and GPU computing.

   – We proposed a Block–Asynchronous (BA) strategy where the problem is partitioned into blocks of threads, which run independently from each other an unbounded number of times without a global synchronization. The asynchronous computation by blocks (intra-block updates) reduces the number of points of global synchronization (inter-block updates) allowing efficient exploitation of the memory hierarchy of the GPU. This approach follows a tiling/grid parallel pattern in which data are partitioned into regular blocks of threads. This strategy perfectly matches the asynchronous evolution of cellular automata and was adapted to solve various problems that take part in the schemes proposed in this thesis:

   – *CA–Watershed*: the asynchronous cellular automaton to compute the watershed transform used in the CA–WSHED–GPU scheme was based on the BA strategy. However, the algorithm introduced artifacts into the border of the segmented regions, so we proposed an artifacts-free block–asynchronous implementation, which produced the correct results by correcting the data propagation speed among the blocks using wavefront techniques. The CA–Watershed reuses information within a block, efficiently exploit-

ing the shared and cache memories of the GPU. In order to solve the problem of accessing neighboring data, the shared memory was extended with a apron of size one. Finally, due the nature of the CA and the local computation by blocks performed by the BA strategy, the CA–Watershed was extended to three dimensions in order to process a 3D data.

– *Block-Asynchronous Reconstruction (BAR)*: the morphological reconstruction was used in the WT–EMP–GPU scheme to create the morphological profiles. The BAR algorithm is also based on the BA strategy, where multiple scans are performed in both directions at the same time within each block, in order to propagate the reconstruction as much as possible with the available data. Therefore, the data are reused in shared memory. The implementation required an apron of size one in order to access neighboring data.

– *Attribute filtering*: We proposed a GPU implementation for attribute openings and closings through threshold decomposition and merging of connected components. The BA strategy was included in this implementation for Connected Component Labelling (CCL). Attribute filtering can be applied to create an Extended Attribute Profile for land-cover spectral-spatial classification schemes.

– We adapted different strategies for efficient projection of 1D-DWT and 2D-DWT on GPU. The proposed 1D-DWT implementation was adapted to compute thousands of transformations in parallel, corresponding to the pixel vectors of the hyperspectral image. The different levels of the wavelet decomposition were applied in the same kernel reusing the data of the shared memory. In the case of the 2D-DWT, the implementation was designed with the purpose of managing the three filters that were used in the denoising step. The number of times that data were loaded within the kernel was minimized by computing the three filters in the same kernel and by loading larger blocks of data.

– We also presented a GPU multi-class implementation for SVM classification on GPU. The reason for having our own SVM implementation was due to the need of classifying hyperspectral images in real-time. We found that our implementation was more robust with large datasets than other GPU proposals found in the literature.

5. Regarding the efficient computation on GPU, the proposed schemes were implemented using CUDA, named CA–WSHED–GPU and WT–EMP–GPU. By incorporating the

techniques and strategies studied in this thesis, the second main objective regarding the efficient computation on low-cost computing infrastructures was achieved. The total execution time was below the 3 seconds for the datasets used in these experiments: University of Pavia, Salinas Valley and Hekla Volcano. The WT–EMP–GPU was better in terms of classification accuracy and execution times, achieving an OA of 98.8% in 1.749 seconds on the Pavia University dataset. By optimizing the kernels to efficiently exploit the GPU resources, the proposed implementations achieved real-time classification on a GTX TITAN.

The experiments were carried out over different n-dimensional datasets: 2D images, 3D images and hyperspectral images, including the well-known image of Lena, a computed tomography scan of a human head, a simulated MRI volume from the BrainWeb database [38, 9] and real hyperspectral images (urban and agricultural) taken by the Reflective Optics System Imaging Spectrometer (ROSIS-03) and the Airborne Visible-infrared Imaging Spectrometer (AVIRIS) sensors.

On the one hand, we have compared our schemes to other spectral-spatial classification schemes in the same conditions, regarding the size of the scenes, number of training samples, as well as the classifier used. On the other hand, the comparison regarding the execution time was always done by multithreaded OpenMP implementations developed in this thesis for a fair comparison, and by GPU proposals found in the literature.

We have used different commodity hardware. A CPU with an Intel quad-core i7-860 microprocessor, and three NVIDIA GPUs with different architectures: a GTX 580 (Fermi), a GTX 680 and a GTX TITAN (Kepler).

In view of the results presented in this thesis, we have achieved the main objectives proposed at the outset of this dissertation, and we can conclude that the GPU is an adequate computing platform for on-board processing of hyperspectral information.

Future trends in remote sensing on commodity hardware may pass through computation for low-cost and low-power consumption applications. The new architectures designed by NVIDIA for embedded applications, such as the Tegra X1 mobile processor based on ARM and shipped with a Maxwell GPU, present new challenges in HPC and constitute an interesting field for future research in remote sensing applications for on-board processing on drones, or real-time guidance for medical surgery based on hyperspectral analysis.

Bibliography

[1] Shigeo Abe. *Support vector machines for pattern classification*. Springer London, 2005.

[2] Susumu Adachi, Ferdinand Peper, and Jia Lee. Computation by asynchronously updating cellular automata. *Journal of Statistical Physics*, 114(1-2):261–289, 2004.

[3] Matthew Allen Lee. *Exploiting spatial and spectral information in hyperdimensional imagery*. PhD thesis, Mississippi State University, 2012.

[4] Hartwig Anzt, Stanimire Tomov, Jack Dongarra, and Vincent Heuveline. A block-asynchronous relaxation method for graphics processing units. *Journal of Parallel and Distributed Computing*, 73(12):1613 – 1626, 2013. Heterogeneity in Parallel and Distributed Computing.

[5] Francisco Argüello and Dora B. Heras. Elm-based spectral?spatial classification of hyperspectral images using extended morphological profiles and composite feature mappings. *International Journal of Remote Sensing*, 36(2):645–664, 2015.

[6] Francisco Argüello, Dora B. Heras, M. Bóo, and Julián Lamas-Rodríguez. The split-and-merge method in general purpose computation on gpus. *Parallel Computing*, 38(6–7):277–288, 2012.

[7] J. Astola, P. Haavisto, and Y. Neuvo. Vector median filters. *Proceedings of the IEEE*, 78(4):678–689, Apr 1990.

[8] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris. GPU acceleration for support vector machines. In *Procs. 12th Inter. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2011), Delft, Netherlands*, April 2011.

[9] Berengere Aubert-Broche, M. Griffin, G.B. Pike, A.C. Evans, and D.L. Collins. Twenty new digital brain phantoms for creation of validation image data bases. *Medical Imaging, IEEE Transactions on*, 25(11):1410–1416, Nov 2006.

[10] James Balasalle, Mario A. Lopez, and Matthew J. Rutherford. Optimizing memory access patterns for cellular automata on GPUs. *GPU Computing Gems Jade Edition*, pages 67–75, 2012.

[11] Geoffrey H Ball and David J Hall. Isodata, a novel method of data analysis and pattern classification. Technical report, DTIC Document, 1965.

[12] John E. Ball and L. M. Bruce. Level set hyperspectral image classification using best band analysis. *Geoscience and Remote Sensing, IEEE Transactions on*, 45(10):3022–3027, 2007.

[13] J.-M. Beaulieu and M. Goldberg. Hierarchy in picture segmentation: a stepwise optimization approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(2):150–163, Feb 1989.

[14] J. A. Benediktsson, J. Chanussot, and W. M. Moon. Very high-resolution remote sensing: Challenges and opportunities [point of view]. *Proceedings of the IEEE*, 100(6):1907–1910, 2012.

[15] J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson. Classification of hyperspectral data from urban areas based on extended morphological profiles. *Geoscience and Remote Sensing, IEEE Transactions on*, 43(3):480–491, 2005.

[16] J.A. Benediktsson and I. Kanellopoulos. Classification of multisource and hyperspectral data based on decision fusion. *Geoscience and Remote Sensing, IEEE Transactions on*, 37(3):1367–1377, May 1999.

[17] J.A. Benediktsson, Martino Pesaresi, and K. Amason. Classification and feature extraction for remote sensing images from urban areas based on morphological transformations. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(9):1940–1949, 2003.

[18] S. Bernabé, S. López, A. Plaza, and R. Sarmiento. Gpu implementation of an automatic target detection and classification algorithm for hyperspectral image analysis. *Geoscience and Remote Sensing Letters, IEEE*, 10(2):221–225, 2013.

[19] S. Bernabe, P. Reddy Marpu, A. Plaza, M. Dalla Mura, and J.A Atli Benediktsson. Spectral-spatial classification of multispectral images using kernel feature space representation. *Geoscience and Remote Sensing Letters, IEEE*, 11(1):288–292, Jan 2014.

[20] Sergio Bernabé, Antonio Plaza, Prashanth Reddy Marpu, and Jon Atli Benediktsson. A new parallel tool for classification of remotely sensed imagery. *Computers & Geosciences*, 46(0):208 – 218, 2012.

[21] K. Bernard, Y. Tarabalka, J. Angulo, J. Chanussot, and J.A. Benediktsson. Spectral-spatial classification of hyperspectral data based on a stochastic minimum spanning forest approach. *Image Processing, IEEE Transactions on*, 21(4):2008–2021, April 2012.

[22] Serge Beucher. The watershed transformation applied to image segmentation. *Scanning Microscopy International*, pages 299–314, 1992.

[23] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Qian Du, P. Gader, and J. Chanussot. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(2):354–379, April 2012.

[24] José M. Bioucas-Dias, Antonio Plaza, Gustavo Camps-Valls, Paul Scheunders, Nasser Nasrabadi, and Jocelyn Chanussot. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geoscience and Remote Sensing Magazine*, 1(2):6–36, 2013.

[25] Edmond J. Breen and Ronald Jones. Attribute openings, thinnings, and granulometries. *Computer Vision and Image Understanding*, 64(3):377 – 389, 1996.

[26] L.M. Bruce, C.H. Koger, and Li Jiang. Dimensionality reduction of hyperspectral data using discrete wavelet transform feature extraction. *Geoscience and Remote Sensing, IEEE Transactions on*, 40(10):2331–2338, 2002.

[27] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[28] G. Camps-Valls, L. Gomez-Chova, J. Munoz-Mari, J.L. Rojo-Alvarez, and M. Martinez-Ramon. Kernel-based framework for multitemporal and multisource

remote sensing data classification and change detection. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(6):1822–1835, June 2008.

[29] G. Camps-Valls, L. Gomez-Chova, J. Munoz-Mari, J. Vila-Frances, and J. Calpe-Maravilla. Composite kernels for hyperspectral image classification. *Geoscience and Remote Sensing Letters, IEEE*, 3(1):93–97, Jan 2006.

[30] Oscar Carrasco, Richard B. Gomez, Arun Chainani, and William E. Roper. Hyperspectral imaging applied to medical diagnoses and food safety. *Proc. SPIE*, 5097:215–221, 2003.

[31] T. Castaings, W. Björn, J. A. Benediktsson, and J. Chanussot. On the influence of feature reduction for the classification of hyperspectral images based on the extended morphological profile. *International Journal of Remote Sensing*, 31(22):5921–5939, 2010.

[32] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 104–111, New York, NY, USA, 2008. ACM.

[33] Robit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[34] Chein-I Chang. *Hyperspectral data processing: algorithm design and analysis*. John Wiley & Sons, 2013.

[35] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[36] E. Christophe, J. Michel, and J. Inglada. Remote sensing processing: From multicore to gpu. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 4(3):643–652, 2011.

[37] L. O. Chua and L. Yang. Cellular neural networks: applications. *Circuits and Systems, IEEE Transactions on*, 35(10):1273–1290, Oct 1988.

[38] Chris A. Cocosco, Vasken Kollokian, Remi K.-S. Kwan, G. Bruce Pike, and Alan C. Evans. Brainweb: Online interface to a 3d mri simulated brain database. *NeuroImage*, 5:425, 1997.

[39] Russell G. Congalton and Kass Green. *Assessing the accuracy of remotely sensed data: principles and practices*. CRC press, 2008.

[40] Shane Cook. *CUDA Programming: a developer's guide to parallel computing with GPUs*. Elsevier, 2013.

[41] M. Dalla Mura, J. A. Benediktsson, B. Waske, and L. Bruzzone. Morphological attribute profiles for the analysis of very high resolution images. *Geoscience and Remote Sensing, IEEE Transactions on*, 48(10):3747–3762, 2010.

[42] Mauro Dalla Mura. *Advanced Techniques based on Mathematical Morphology for the Analysis of Remote Sensing Images*. PhD thesis, University of Trento, University of Iceland, 2011.

[43] J. Darbon and C.B. Akgul. An efficient algorithm for attribute openings and closings. In *Signal Processing Conference, 2005 13th European*, pages 1–4, Sept 2005.

[44] I. Daubechies. *Ten lectures on wavelets*, volume 61. Society for Industrial and Applied Mathematics, 1992.

[45] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[46] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[47] D. L. Donoho. De-noising by soft-thresholding. *Information Theory, IEEE Transactions on*, 41(3):613–627, 1995.

[48] I. Dópido, A. Villa, A. Plaza, and P. Gamba. A comparative assessment of several processing chains for hyperspectral image classification: What features to use? In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 3rd Workshop on, pages 1–4, 2011.

[49] J Dorband, Josephine Palencia, and Udaya Ranawake. Commodity computing clusters at goddard space flight center. *Journal of Space Communication*, 1(3):113–123, 2003.

[50] G. J. Edelman, E. Gaston, T. G. van Leeuwen, P. J. Cullen, and M. C. G. Aalders. Hyperspectral imaging for non-contact analysis of forensic traces. *Forensic Science International*, 223(1?3):28 – 39, 2012.

[51] G. Bard Ermentrout and Leah Edelstein-Keshet. Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 160(1):97 – 133, 1993.

[52] A. N. Evans and X. U. Liu. A morphological gradient approach to color edge detection. *Image Processing, IEEE Transactions on*, 15(6):1454–1463, 2006.

[53] Rob Farber. *CUDA Application Design and Development*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[54] M. Fauvel, J. A. Benediktsson, J. Chanussot, and J. R. Sveinsson. Spectral and spatial classification of hyperspectral data using svms and morphological profiles. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(11):3804–3814, 2008.

[55] M. Fauvel, J. Chanussot, and J.A. Benediktsson. Evaluation of kernels for multiclass classification of hyperspectral remote sensing data. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II, May 2006.

[56] M. Fauvel, J. Chanussot, and J.A. Benediktsson. Adaptive pixel neighborhood definition for the classification of hyperspectral images with support vector machines and composite kernel. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1884–1887, Oct 2008.

[57] M. Fauvel, J. Chanussot, and J.A. Benediktsson. A spatial-spectral kernel-based approach for the classification of remote-sensing images. *Pattern Recognition*, 45(1):381 – 392, 2012.

[58] M. Fauvel, Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton. Advances in spectral-spatial classification of hyperspectral images. *Proceedings of the IEEE*, 101(3):652–675, 2013.

[59] Felix Fernandes. *Directional, Shift-Insensitive, Complex Wavelet Transforms with Controllable Redundancy*. PhD thesis, Rice University, 2002.

[60] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[61] J. E. Fowler and J. T. Rucker. Three-dimensional wavelet-based compression of hyperspectral imagery. In Chein-I Chang, editor, *Hyperspectral Data Exploitation. Theory and Applications*, chapter 14, pages 379–408. John Wiley & Sons, Inc., Hoboken, Hoboken, 2011.

[62] Joaquín Franco, Gregorio Bernabé, Juan Fernández, and Manuel Ujaldón. The 2d wavelet transform on emerging architectures: Gpus and multicores. *Journal of Real-Time Image Processing*, 7(3):145–152, 2012.

[63] K.S. Fu and J.K. Mui. A survey on image segmentation. *Pattern Recognition*, 13(1):3 – 16, 1981.

[64] B. Galilee, F. Mamalet, M. Renaudin, and P.-Y. Coulon. Parallel asynchronous watershed algorithm-architecture. *Parallel and Distributed Systems, IEEE Transactions on*, 18(1):44–56, 2007.

[65] P. Ghamisi, M. Dalla Mura, and J. A. Benediktsson. A survey on spectral–spatial classification techniques based on attribute profiles. *Geoscience and Remote Sensing, IEEE Transactions on*, 53(5):2335–2353, 2015.

[66] Alexander F.H. Goetz, Gregg Vane, Jerry E. Solomon, and Barrett N. Rock. Imaging spectrometry for earth remote sensing. *Science*, 228(4704):1147–1153, 1985.

[67] Carlos González, Sergio Sánchez, Abel Paz, Javier Resano, Daniel Mozos, and Antonio Plaza. Use of fpga or gpu-based architectures for remotely sensed hyperspectral image processing. *Integration, the {VLSI} Journal*, 46(2):89 – 103, 2013.

[68] E.T. Gormus, N. Canagarajah, and A. Achim. Dimensionality reduction of hyperspectral images using empirical mode decompositions and wavelets. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(6):1821–1830, Dec 2012.

[69] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2):50–61, 1995.

[70] A. A. Green, M. Berman, P. Switzer, and M. D. Craig. A transformation for ordering multispectral data in terms of image quality with implications for noise removal. *Geoscience and Remote Sensing, IEEE Transactions on*, 26(1):65–74, Jan 1988.

[71] Robert O. Green, Michael L. Eastwood, Charles M. Sarture, Thomas G. Chrien, Mikael Aronsson, Bruce J. Chippendale, Jessica A. Faust, Betina E. Pavri, Christopher J. Chovit, Manuel Solis, Martin R. Olah, and Orlesa Williams. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris). *Remote Sensing of Environment*, 65(3):227 – 248, 1998.

[72] J. Anthony Gualtieri and Robert F. Cromp. Support vector machines for hyperspectral remote sensing classification. *Proc. SPIE*, 3584:221–232, 1999.

[73] Mark Harris et al. Optimizing parallel reduction in cuda. *NVIDIA Developer Technology*, 2(4), 2007.

[74] K. A. Hawick, A. Leist, and D. P. Playne. Parallel graph component labelling with GPUs and CUDA. *Parallel Computing*, 36(12):655 – 678, 2010.

[75] Kenneth A Hawick, P.D Coddington, and H.A James. Distributed frameworks and parallel algorithms for processing large-scale geographic data. *Parallel Computing*, 29(10):1297 – 1333, 2003. High Performance Computing with geographical data.

[76] H. J. A. M. Heijmans. Theoretical aspects of gray-level morphology. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(6):568–582, Jun 1991.

[77] Dora B. Heras, F. Argüello, J. L. Gómez, J. A. Becerra, and R. J. Duro. Towards real-time hyperspectral image processing, a gp-gpu implementation of target identification. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, volume 1, pages 316–321, 2011.

[78] Dora B. Heras, Francisco Argüello, and Pablo Quesada-Barriuso. Exploring ELM-based spatial-spectral classification of hyperspectral images. *International Journal of Remote Sensing*, 35(2):401–423, 2014.

[79] Sergio Herrero-Lopez, John R. Williams, and Abel Sanchez. Parallel multiclass classification using svms on gpus. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pages 2–11, New York, NY, USA, 2010. ACM.

[80] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, Mar 2002.

[81] Michal Hučko and Miloš Šrámek. Streamed watershed transform on gpu for processing of large volume data. In *Proceedings of the 28th Spring Conference on Computer Graphics*, SCCG '12, pages 137–141, New York, NY, USA, 2012. ACM.

[82] Intel Corporation. Intel 64 and IA-32 architectures software developer?s manual. Volumen 3, system programming guide, part 1, Intel Corporation, 2011.

[83] A.K. Jain, , M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[84] Maarten Jansen. *Noise reduction by wavelet thresholding*, volume 161 of *Lecture notes in statistics*. Springer New York, 2001.

[85] Wenqing Jiang and A. Ortega. Lifting factorization-based discrete wavelet transform architecture design. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(5):651–657, May 2001.

[86] S. Kaewpijit, J. Le Moigne, and T. El-Ghazawi. Automatic reduction of hyperspectral imagery using wavelet spectral analysis. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(4):863–871, 2003.

[87] Pavel Karas. Efficient computation of morphological greyscale reconstruction. In *Sixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 54–61, 2010.

[88] C. Kauffmann and N. Piche. Cellular automaton for ultra-fast watershed transform on gpu. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, Dec 2008.

[89] N. Keshava and J.F. Mustard. Spectral unmixing. *Signal Processing Magazine, IEEE*, 19(1):44–57, Jan 2002.

[90] David B Kirk and W Hwu Wen-mei. *Programming Massively Parallel Processors: a Hands-on Approach*. Elsevier, 2010.

[91] A. Körbes, G. B. Vitor, R. Lotufo, and J.V. Ferreira. Advances on watershed processing on gpu architecture. In Pierre Soille, Martino Pesaresi, and GeorgiosK. Ouzounis, editors, *Mathematical Morphology and Its Applications to Image and Signal Processing*, volume 6671 of *Lecture Notes in Computer Science*, pages 260–271. Springer Berlin Heidelberg, 2011.

[92] F. A. Kruse, J. W. Boardman, and J. F. Huntington. Comparison of airborne hyperspectral data and eo-1 hyperion for mineral mapping. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(6):1388–1400, 2003.

[93] Bor-Chen Kuo and D.A. Landgrebe. A robust classification procedure based on mixture classifiers and nonparametric weighted feature extraction. *Geoscience and Remote Sensing, IEEE Transactions on*, 40(11):2486–2494, Nov 2002.

[94] Julián Lamas-Rodríguez, Pablo Quesada-Barriuso, Francisco Argüello, Dora B. Heras, and M. Bóo. Proyección del método de segmentación del conjunto de nivel en GPU. In *XXIII Jornadas de Paralelismo*, pages 273–278, 2012.

[95] D. Landgrebe. Hyperspectral image data analysis. *Signal Processing Magazine, IEEE*, 19(1):17–28, Jan 2002.

[96] David A Landgrebe. *Signal theory methods in multispectral remote sensing*. John Wiley & Sons, 2003.

[97] David Lesage, Jérôme Darbon, and CeyhunBurak Akgül. An efficient algorithm for connected attribute thinnings and thickenings. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Ara Nefian, Gopi Meenakshisundaram, Valerio Pascucci, Jiri Zara, Jose Molineros, Holger Theisel, and Tom Malzbender, editors, *Advances in Visual Computing*, volume 4292 of *Lecture Notes in Computer Science*, pages 393–404. Springer Berlin Heidelberg, 2006.

[98] D. Letexier and S. Bourennane. Noise removal from hyperspectral images by multidimensional filtering. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(7):2061–2069, Jul 2008.

[99] Qi Li, Raied Salman, Erik Test, Robert Strack, and Vojislav Kecman. Gpusvm: a comprehensive cuda based support vector machine package. *Central European Journal of Computer Science*, 1(4):387–405, 2011.

[100] Haida Liang. Advances in multispectral and hyperspectral imaging for archaeology and art conservation. *Applied Physics A*, 106(2):309–323, 2012.

[101] J. López-Fandiño, P. Quesada-Barriuso, Dora B. Heras, and F. Argüello. Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on commodity gpus. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, PP(99):1–10, 2015.

[102] Guolan Lu and Baowei Fei. Medical hyperspectral imaging: a review. *Journal of Biomedical Optics*, 19(1):010901, 2014.

[103] Yunmei Lu, Yun Zhu, Meng Han, Jing (Selena) He, and Yanqing Zhang. A survey of gpu accelerated svm. In *Proceedings of the 2014 ACM Southeast Regional Conference*, ACM SE '14, pages 15:1–15:7, New York, NY, USA, 2014. ACM.

[104] J. Luitjens and S. Rennich. Cuda warps and occupancy [webminar]. In *GPU Computing Webinars*. NVIDIA Corporation, 2011.

[105] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[106] Stéphane Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[107] D. Manolakis and G. Shaw. Detection algorithms for hyperspectral imaging applications. *Signal Processing Magazine, IEEE*, 19(1):29–43, 2002.

[108] Prashanth Reddy Marpu, Mattia Pedergnana, Mauro Dalla Mura, Stijn Peeters, Jon Atli Benediktsson, and Lorenzo Bruzzone. Classification of hyperspectral data using extended attribute profiles based on supervised and unsupervised feature extraction techniques. *International Journal of Image and Data Fusion*, 3(3):269–298, 2012.

[109] Moreno Marzolla. Fast training of support vector machines on the cell processor. *Neurocomputing*, 74(17):3700 – 3707, 2011.

[110] F. Melgani and L. Bruzzone. Classification of hyperspectral remote sensing images with support vector machines. *Geoscience and Remote Sensing, IEEE Transactions on*, 42(8):1778–1790, Aug 2004.

[111] Fernand Meyer. Topographic distance and watershed lines. *Signal Processing*, 38(1):113 – 125, 1994. Mathematical Morphology and its Applications to Signal Processing.

[112] Alina N. Moga, Bogdan Cramariuc, and Moncef Gabbouj. Parallel watershed transformation algorithms for image segmentation. *Parallel Computing*, 24(14):1981 – 2001, 1998.

[113] Andreas A. Mueller, Andrea Hausold, and Peter Strobl. Hysens-dais/rosis imaging spectrometers at dlr. *Proc. SPIE*, 4545:225–235, 2002.

[114] M. D. Mura, A. Villa, J. A. Benediktsson, J. Chanussot, and L. Bruzzone. Classification of hyperspectral images by using extended morphological attribute profiles and independent component analysis. *Geoscience and Remote Sensing Letters, IEEE*, 8(3):542–546, 2011.

[115] Chrystopher L. Nehaniv. Evolution in asynchronous cellular automata. In *Proceedings of the Eighth International Conference on Artificial Life*, ICAL 2003, pages 65–73, Cambridge, MA, USA, 2003. MIT Press.

[116] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[117] Guillaume Noyel, Jesús Angulo, and Dominique Jeulin. Morphological segmentation of hyperspectral images. *Image Analysis & Stereology*, 26(3):101–109, 2007.

[118] NVIDIA Corporation. CUDA C best practice guide, 2013.

[119] NVIDIA Corporation. CUDA C programming guide, 2013.

[120] OpenMP Architecture Review Board. OpenMP application program interface version 3.1, July 2011.

[121] J. Palmason, J.A. Benediktsson, J.R. Sveinsson, and J. Chanussot. Classification of hyperspectral data from urban areas using morphological preprocessing and

independent component analysis. In *Geoscience and Remote Sensing Symposium, 2005. IGARSS '05. Proceedings. 2005 IEEE International*, volume 1, pages 4 pp.–, 2005.

[122] J.A. Palmason, J.A. Benediktsson, and K. Arnason. Morphological transformations and feature extraction of urban data with high spectral and spatial resolution. In *Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*, volume 1, pages 470–472 vol.1, Jul 2003.

[123] J.S. Pearlman, P.S. Barry, C.C. Segal, J. Shepanski, D. Beiso, and S.L. Carman. Hyperion, a space-based imaging spectrometer. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(6):1160–1173, June 2003.

[124] Martino Pesaresi and J.A. Benediktsson. A new approach for the morphological segmentation of high-resolution satellite imagery. *Geoscience and Remote Sensing, IEEE Transactions on*, 39(2):309–320, Feb 2001.

[125] Martino Pesaresi and Ioannis Kanellopoulos. Detection of urban features using morphological based segmentation and very high resolution remotely sensed data. In Ioannis Kanellopoulos, GraemeG. Wilkinson, and Theo Moons, editors, *Machine Vision and Advanced Image Processing in Remote Sensing*, pages 271–284. Springer Berlin Heidelberg, 1999.

[126] A. Plaza, Qian Du, Yang-Lang Chang, and R.L. King. High performance computing for hyperspectral remote sensing. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 4(3):528–544, Sept 2011.

[127] A. Plaza, J. Plaza, A. Paz, and S. Sánchez. Parallel hyperspectral image and signal processing [applications corner]. *Signal Processing Magazine, IEEE*, 28(3):119–126, May 2011.

[128] Antonio Plaza, Jon Atli Benediktsson, Joseph W. Boardman, Jason Brazile, Lorenzo Bruzzone, Gustavo Camps-Valls, Jocelyn Chanussot, Mathieu Fauvel, Paolo Gamba, Anthony Gualtieri, Mattia Marconcini, James C. Tilton, and Giovanna Trianni. Recent advances in techniques for hyperspectral image processing. *Remote Sensing of Environment*, 113, Supplement 1(0):S110 – S122, 2009. Imaging Spectroscopy Special Issue.

[129] Antonio Plaza, Javier Plaza, and Hugo Vegas. Improving the performance of hyperspectral image and signal processing algorithms using parallel, distributed and specialized hardware-based systems. *J. Signal Process. Syst.*, 61(3):293–315, 2010.

[130] Antonio Plaza, David Valencia, and Javier Plaza. An experimental comparison of parallel algorithms for hyperspectral analysis using heterogeneous and homogeneous networks of workstations. *Parallel Computing*, 34(2):92 – 114, 2008.

[131] Victor Podlozhnyuk. Image Convolution with CUDA. Technical Report June, NVIDIA Corporation, 2007.

[132] Blanca Priego, Daniel Souto, Francisco Bellas, and Richard J. Duro. Hyperspectral image segmentation through evolved cellular automata. *Pattern Recognition Letters*, 34(14):1648 – 1658, 2013. Innovative Knowledge Based Techniques in Pattern Recognition.

[133] P. Quesada-Barriuso, F. Argüello, and Dora. B. Heras. Spectral-spatial classification of hyperspectral images using wavelets and extended morphological profiles. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 7(4):1177–1185, 2014.

[134] P. Quesada-Barriuso, F. Argüello, Dora B. Heras, and J. A. Benediktsson. Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, PP(99):1–9, 2015.

[135] P. Quesada-Barriuso, Dora B. Heras, and Francisco Argüello. Efficient gpu asynchronous implementation of a watershed algorithm based on cellular automata. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 79–86, 2012.

[136] Pablo Quesada-Barriuso, Francisco Argüello, , and Dora B. Heras. Efficient segmentation of hyperspectral images on commodity GPUs. In *16th International Conference on Knowledge-Based and Intelligent Information & Engineering System*, volume 243, pages 2130–2139, 2012.

[137] Pablo Quesada-Barriuso, Francisco Argüello, and Dora B. Heras. Computing efficiently spectral-spatial classification of hyperspectral images on commodity gpus.

In Jeffrey W. Tweedale and Lakhmi C. Jain, editors, *Recent Advances in Knowledge-based Paradigms and Applications*, volume 234 of *Advances in Intelligent Systems and Computing*, pages 19–42. Springer International Publishing, 2014.

[138] Pablo Quesada-Barriuso, Dora B. Heras, and Francisco Argüello. Efficient 2d and 3d watershed on graphics processing unit: block-asynchronous approaches based on cellular automata. *Computers & Electrical Engineering*, 39(8):2638 – 2655, 2013.

[139] Pablo Quesada-Barriuso, Julián Lamas-Rodríguez, Dora B. Heras, and Francisco Argüello. Influencia de las mesetas en la implementación de watershed sobre GPUs. In *XXIII Jornadas de Paralelismo*, pages 249–254, 2012.

[140] Å. Rinnan, F. v. d. Berg, and S. B. Engelsen. Review of the most common pre-processing techniques for near-infrared spectra. *TrAC Trends in Analytical Chemistry*, 28(10):1021–1222, 2009.

[141] Jean-Francois Rivest, Pierre Soille, and Serge Beucher. Morphological gradients. *Journal of Electronic Imaging*, 2(4):326–336, 1993.

[142] Jos B. T. M. Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41(1,2):187–228, 2000.

[143] P. L. Rosin. Training cellular automata for image processing. *Image Processing, IEEE Transactions on*, 15(7):2076–2087, Jul 2006.

[144] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *Image Processing, IEEE Transactions on*, 7(4):555–570, Apr 1998.

[145] P. Salembier and M.H.F. Wilkinson. Connected operators. *Signal Processing Magazine, IEEE*, 26(6):136–157, November 2009.

[146] Robert A. Schowengerdt. *Remote Sensing, Third Edition: Models and Methods for Image Processing*. Academic Press, Inc., Orlando, FL, USA, 2006.

[147] I. W Selesnick. Double-density wavelet software. available for download at http://eeweb.poly.edu/iselesni/DoubleSoftware/.

[148] I. W. Selesnick. The double density dwt. In *Wavelets in Signal and Image Analysis: From Theory to Practice*, chapter 2, pages 36–66. Kluwer Academic Publishers, 2001.

[149] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

[150] Jean Serra and Luc Vincent. An overview of morphological filtering. *Circuits, Systems and Signal Processing*, 11(1):47–108, 1992.

[151] Lindsay I. Smith. A tutorial on principal components analysis. *Cornell University, USA*, 51:52, 2002.

[152] Pierre Soille. *Morphological image analysis: principles and applications*. Springer-Verlag New York, Inc., 2003.

[153] Pierre Soille and Laurent Najman. On morphological hierarchical representations for image processing and spatial data clustering. In Ullrich Köthe, Annick Montanvert, and Pierre Soille, editors, *Applications of Discrete Geometry and Mathematical Morphology*, volume 7346 of *Lecture Notes in Computer Science*, pages 43–67. Springer Berlin Heidelberg, 2012.

[154] Y. Tarabalka, J. A. Benediktsson, and J. Chanussot. Spectral-spatial classification of hyperspectral imagery based on partitional clustering techniques. *Geoscience and Remote Sensing, IEEE Transactions on*, 47(8):2973–2987, 2009.

[155] Y. Tarabalka, J. Chanussot, and J. A. Benediktsson. Segmentation and classification of hyperspectral images using watershed transformation. *Pattern Recognition*, 43(7):2367–2379, 2010.

[156] Y. Tarabalka, J. Chanussot, J. A. Benediktsson, J. Angulo, and M. Fauvel. Segmentation and classification of hyperspectral data using watershed. In *International Geoscience and Remote Sensing Symposium, (IGARSS)*, volume 3, pages III–652–III–655, 2008.

[157] Y. Tarabalka, J.C. Tilton, J.A. Benediktsson, and J. Chanussot. A marker-based approach for the automated selection of a single segmentation from a hierarchical set of image segmentations. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(1):262–272, 2012.

[158] Yuliya Tarabalka, Jon Atli Benediktsson, and Jocelyn Chanussot. Classification of hyperspectral data using support vector machines and adaptive neighborhoods. In *6th EARSEL SIG Imaging Spectroscopy workshop*, Israel, 2009.

[159] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.

[160] J. C. Tilton. Analysis of hierarchically related image segmentations. In *Advances in Techniques for Analysis of Remotely Sensed Data, 2003 IEEE Workshop on*, pages 60–69, 2003.

[161] James C Tilton. Image segmentation by region growing and spectral clustering with natural convergence criterion. In *International geoscience and remote sensing symposium*, volume 4, pages 1766–1768, 1998.

[162] Pai-Hui Hsu1 Yi-Hsing Tseng and Peng Gongg. Spectral feature extraction of hyperspectral images using wavelet transform. *Journal of Photogrammetry and Remote Sensing*, 11(1):93–109, 2006.

[163] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision - ECCV 2008*, volume 5305 of *Lecture Notes in Computer Science*, pages 705–718. Springer Berlin Heidelberg, 2008.

[164] S. Velasco-Forero and V. Manian. Improving hyperspectral image classification using spatial preprocessing. *Geoscience and Remote Sensing Letters, IEEE*, 6(2):297–301, 2009.

[165] M. Vetterli and C. Herley. Wavelets and filter banks: theory and design. *IEEE Transactions on Signal Processing*, 40(9):2207–2232, 1992.

[166] M. Vidal and J. M. Amigo. Pre-processing of hyperspectral images. essential steps before image analysis. *Chemometrics and Intelligent Laboratory Systems*, 117:119–148, 2012.

[167] Anthony J. Viera and Joanne M. Garrett. Understanding interobserver agreement: the kappa statistic. *Fam Med*, 37(5):360–363, 2005.

[168] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(6):583–598, 1991.

[169] Luc Vincent. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. *Image Processing, IEEE Transactions on*, 2(2):176–201, 1993.

[170] Luc Vincent. Morphological area openings and closings for grey-scale images. In Ying-Lie O, Alexander Toet, David Foster, HenkJ.A.M. Heijmans, and Peter Meer, editors, *Shape in Picture*, volume 126 of *NATO ASI Series*, pages 197–208. Springer Berlin Heidelberg, 1994.

[171] B. Wagner, P. Müller, and G. Haase. A parallel watershed-transformation algorithm for the gpu. In *in Proc. of the Workshop on App. of Discrete Geometry and Mathematical Morphology*, pages 111–115, 2010.

[172] M.H.F. Wilkinson, Hui Gao, W.H. Hesselink, J.-E. Jonker, and A. Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1800–1813, Oct 2008.

[173] Tarabalka Yuliya. *Classification of hyperspectral data using spectral-spatial approaches*. PhD thesis, Signal and Image processing. Institut National Polytechnique de Grenoble - INPG, 2010.

List of Figures

List of Tables