DEPARTAMENTO DE TECNOLOGÍA DE LOS COMPUTADORES Y DE LAS COMUNICACIONES

ESCUELA POLITÉCNICA DE CÁCERES

Diseño e implementación de una cadena completa para desmezclado de imágenes hiperespectrales en tarjetas gráficas programables (GPUs)

Tesis Doctoral

Sergio Sánchez Martínez

2013

A mis padres, ***Jesús*** *y* ***Loli****,*

a mi hermano ***Jesús****.*

Agradecimientos

Quiero dar las gracias a una serie de personas sin las cuales la realización del presente trabajo de Tesis Doctoral no hubiera sido posible:

- A Antonio J. Plaza, excelente director, compañero y mejor persona, por darme la oportunidad de trabajar con él en un entorno en el que destacan su atención, su trato personal y su pasión por la investigación.
- Al grupo de investigación "Computación Hiperespectral" (HYPERCOMP) por hacer que el día a día en el laboratorio suponga un verdadero lujo. En especial dar las gracias a Gabri, amigo y compañero de batallas, por su valiosa ayuda y su siempre sabio consejo.
- A Marijose y Kiki, a los que quiero mucho, por cuidarme durante mi estancia en Cáceres y hacer de su casa mi casa.
- A mi familia por su ánimo y apoyo incondicional.

Resumen

La principal contribución del presente trabajo de tesis doctoral viene dada por la propuesta de nuevos algoritmos paralelos para desmezclado de imágenes hiperespectrales en aplicaciones de observación remota de la superficie terrestre mediante sensores aerotransportados o de tipo satélite. Dichos algoritmos se fundamentan en el problema de la mezcla, que permite expresar los píxeles de una imagen hiperespectral como una combinación lineal o no lineal de elementos espectralmente puros (*endmembers*) ponderados por sus correspondientes fracciones de abundancia. Una vez descrita la base teórica del estudio, el presente trabajo de tesis doctoral presenta una serie de nuevos algoritmos paralelos que se han desarrollado con el objetivo de resolver el problema de la mezcla en imágenes hiperespectrales. Dichos algoritmos integran una cadena completa de desmezclado espectral o *spectral unmixing* con las siguientes etapas: 1) estimación automática del número de endmembers en una imagen hiperespectral, 2) identificación automática de dichos endmembers en la imagen hiperespectral, y 3) estimación de la abundancia de cada endmember en cada píxel de la imagen. Tras presentar los nuevos algoritmos paralelos desarrollados con motivo del presente trabajo, realizamos un detallado estudio cuantitativo y comparativo de su precisión en el proceso de desmezclado y su rendimiento computacional en un conjunto de arquitecturas basadas en tarjetas tarjeta gráficas programables de NVidia^{TM} (modelos NVidia^{TM} Tesla C1060 y NVidia^{TM} GeForce 580 GTX). Los resultados experimentales han sido obtenidos utilizando imágenes hiperespectrales obtenidas por el sensor Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) de NASA en el contexto de varias aplicaciones reales de gran relevancia social, consistentes en la detección de los incendios que se propagaron en los días posteriores al atentado terrorista del World Trade Center en Nueva York o en la identificación automática de minerales en la región de Cuprite, Nevada, Estados Unidos. En dichos escenarios, los equipos de NASA y el Instituto Geológico de Estados Unidos (USGS) que participaron en las tareas de extinción y emergencia (en el caso de la imagen World Trade Center) e identificación de minerales (en el caso de la imagen de Cuprite) reconocieron que la disponibilidad de técnicas de desmezclado espectral en tiempo real hubiese facilitado las labores de los equipos que actuaron en dichas zonas, por lo que las técnicas desarrolladas se han desarrollado con el objetivo de permitir la realización de dichas tareas en tiempo real en el futuro. La memoria de tesis concluye con una discusión de las técnicas desarrolladas (incluyendo una serie de recomendaciones sobre su mejor uso en diferentes aplicaciones), con la descripción de las principales conclusiones y líneas futuras derivadas del estudio, y con la bibliografía relacionada, tanto en la literatura general como la generada por el candidato.

Palabras clave: Arquitectura de computadores, análisis hiperespectral, desmezclado espectral, tarjetas gráficas programables (GPUs).

Abstract

The main contribution of the present thesis work is the proposal of several new parallel algoritms for spectral mixture analysis of remotely sensed hyperspectral images obtained from airborne or satellite Earth observation platforms. These algorithms are focused on the identification of the most spectrally pure constituents in a hyperspectral image, and on the characterization of mixed pixels as linear or nonlinear combinations of endmembers weighted by their fractional abundances on a sub-pixel basis. Once the theoretical foundations of the proposed study are described, we provide a detailed presentation of the new parallel algorithms developed as the main contribution of this research work, discussing the different steps followed in their development which comprise the following stages: 1) automatic identification of the number of endmembers in the hyperspectral image; 2) automatic identification of the spectral signatures of such endmembers; and 3) estimation of the fractional abundance of endmembers on a sub-pixel basis. After describing the new parallel algorithms introduced in this work, we develop a comprehensive quantitative and comparative analysis in terms of unmixing accuracy and computational performance using a set of graphics processing unit (GPU)-based architectures, including the NVidia^{TM} Tesla C1060 and the NVidia^{TM} GeForce 580 GTX. The experimental results reported in this work are evaluated in the context of two real applications with great societal impact: the possibility to automatically detect the thermal hot spots of the fires which spread in the World Trade Center area during the days after the terrorist attack of September 11th, 2001, and the possibility to perform real-time mapping of minerals in the Cuprite mining district of Nevada, USA, using hyperspectral data sets collected by NASA's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS). It was acknowledged by some of the organizations involved in the collection of the aforementioned data sets that, if high performance computing infrastructure had been available at the time of these events, the hyperspectral data would have been much more useful. The design of new techniques for this purpose may help the development of such tasks in future events. The thesis document concludes with a detailed discussion on the techniques presented herein (including processing recommendations and best practice), with the drawing of the main conclusions and hints at plausible future research, and with a detailed bibliography on the research area and on the specific contributions of the candidate to the scientific literature devoted to this topic.

Keywords: Computer architecture, hyperspectral imaging, spectral unmixing, graphics processing units (GPUs).

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Motivaciones y objetivos

1.1. Motivaciones

El presente trabajo se ha desarrollado en el marco de las líneas de trabajo del grupo de investigación "Computación Hiperspectral" (HyperComp) del Departamento de Tecnología de Computadores y Comunicaciones de la Universidad de Extremadura, y se centra en el procesamiento eficiente de imágenes hiperespectrales obtenidas mediante sensores de observación remota de la superficie terrestre, instalados en plataformas aerotransportadas o de tipo satélite. Dichas imágenes se basan en la capacidad de los sensores hiperespectrales para medir la radiación reflejada por la superficie terrestre en diferentes longitudes de onda.

Las imágenes hiperspectrales suponen una extensión del concepto de imagen digital, en el sentido de que sus píxeles no están formados por un único valor discreto, sino por un conjunto amplio de valores correspondientes a las diferentes mediciones espectrales. Utilizando estas mediciones podemos obtener gran información sobre las propiedades de los materiales que aparecen en la escena. Algunas de las aplicaciones prácticas en las que tienen relevancia las técnicas de análisis hiperespectral tienen que ver con aplicaciones militares (detección de targets u objetivos); detección y monitorización de fuegos y agentes contaminantes en aguas y atmósfera (agentes químicos o vertidos en aguas); agricultura de precisión; identificación y cuantificación de especies geológicas; análisis y caracterización de la vegetación en ecosistemas terrestres para estudio de fenómenos como el cambio climático global, el impacto del crecimiento urbano y la contaminación en el medio ambiente, y un largo etcétera.

Una de las técnicas más ampliamente utilizadas en este tipo de aplicaciones es el desmezclado espectral. Debido a la resolución espacial disponible en los sensores de observación terrestre, la mayor parte de los píxeles registrados contienen mezclas a nivel sub-píxel de diferentes sustancias puras. Las técnicas de desmezclado permiten expresar los píxeles de una imagen hiperespectral como una combinación lineal o no lineal de elementos espectralmente puros (denominados *endmembers*) ponderados por sus correspondientes fracciones de abundancia. Uno de los principales problemas que se plantean a la hora de extraer información útil a partir de imágenes hiperespectrales de la

superficie terrestre es la elevada dimensionalidad de dichas imágenes, unida al problema de la mezcla espectral que se produce a nivel sub-píxel, y que requiere algoritmos computacionalmente costosos. Generalmente, el tiempo de respuesta en aplicaciones de desmezclado espectral de imágenes hiperespectrales ha de ser razonable, y los resultados normalmente deben obtenerse en un corto periodo de tiempo, por ejemplo, para detectar un incendio y poder aplicar los mecanismos de respuesta pertinentes. En este sentido, la posibilidad de obtener técnicas de análisis computacionalmente eficientes para procesar grandes cantidades de datos hiperespectrales resulta de gran interés en numerosas aplicaciones de gran impacto social, tales como detección y monitorización de incendios, estudios medioambientales, detección de agentes contaminantes en aguas y atmósfera, etc.

Las imágenes hiperespectrales se caracterizan por requerimientos extremos en cuanto a espacio de almacenamiento, ancho de banda de transmisión y velocidad de procesamiento. Estos requerimientos deben traducirse en algoritmos de análisis susceptibles de ser ejecutados en paralelo. En este sentido, la mayoría de los algoritmos de análisis hiperespectral de última generación (incluyendo algoritmos para desmezclado espectral) son altamente susceptibles de ser paralelizados, debido a sus escasas restricciones secuenciales. Así una de las posibilidades más interesantes en la actualidad es la obtención de técnicas de procesamiento a bordo, capaces de ofrecer una respuesta en tiempo casi real.

En el presente trabajo se han desarrollado nuevas técnicas de análisis hiperespectral específicamente diseñadas para su ejecución eficiente sobre arquitecturas paralelas, y se han evaluado cuantitativamente tanto en lo referente a su precisión en el análisis como a su rendimiento computacional al implementarse en tarjetas gráficas programables (GPUs). El principal motivo de la elección de este tipo de arquitectura frente a otras posibilidades descritas en la literatura es el hecho de que presenta algunas ventajas competitivas con respecto a otro tipo de arquitecturas que se han empleado en los últimos años para procesar imágenes hiperespectrales, incluyendo *clusters* de tipo Beowulf, sistemas heterogéneos, y sistemas multi-core. Estos sistemas son, en general, caros y difíciles de adaptar a escenarios de procesamiento a bordo, en los que los componentes de peso reducido y bajo consumo son muy importantes para reducir el coste de la misión.

Dirigidas por las cada vez más exigentes demandas de la industria del videojuego, las GPUs han pasado de ser caras unidades de propósito específico a convertirse en sistemas programables altamente paralelos y susceptibles de ser utilizados en aplicaciones de análisis hiperespectral. Aunque sus capacidades en cuanto a programación todavía plantean retos importantes, las GPUs actuales son suficientemente generales para realizar cómputo más allá del dominio de la representación de gráficos, permitiendo su adaptación a problemas científicos de índole genérica y, en lo que a la presente memoria concierne, en aplicaciones de análisis hiperespectral. En este sentido, la aparición de las GPUs ha supuesto un nuevo desarrollo apasionante en aplicaciones de observación remota de la tierra a bordo de satélites y aviones tanto tripulados como no tripulados, estos últimos denominados *unmanned aerial vehicles* (UAVs). Hoy en día, es posible utilizar UAVs para realizar vuelos de bajo coste sobre zonas específicas y es posible acoplar dispositivos

especializados de procesamiento a bordo (como las GPUs) para realizar el procesamiento de los datos obtenidos por sensores hiperespectrales instalados en los UAVs en tiempo real. Esto supone una gran revolución en el ámbito del procesamiento hiperespectral, que ha pasado de ser específico de grandes corporaciones y agencias internacionales como NASA o la Agencia Europea del Espacio (ESA), a convertirse en una tecnología altamente asequible y de fácil implantación en pequeñas y medianas empresas.

1.2. Objetivos

El principal objetivo del presente trabajo de tesis doctoral ha consistido en el desarrollo de nuevas técnicas paralelas de análisis de imágenes hiperespectrales, eficientes en términos computacionales y orientadas a la resolución precisa del problema de la mezcla espectral que tiene lugar en dichas imágenes. Para ello, se han desarrollado varias cadenas de desmeclado o *unmixing*, las cuales han sido implementadas eficientemente utilizando GPUs de NVidia^{TM} y la arquitectura *compute unified device architecture* (CUDA) como plataforma de desarrollo y validadas en el contexto de diferentes aplicaciones. Esto ha permitido analizar el rendimiento de las técnicas desarrolladas en diferentes contextos y elaborar recomendaciones de uso en diferentes aplicaciones relacionadas con el análisis de imágenes hiperespectrales. La consecución de este objetivo general se ha llevado a cabo abordando una serie de objetivos específicos, los cuales se enumeran a continuación:

- Estudiar en detalle las técnicas para desmezclado espectral utilizadas actualmente en el ámbito del análisis hiperespectral, evaluando sus ventajas e inconvenientes, así como los requerimientos computacionales de las mismas para conseguir un alto grado de precisión en su funcionamiento.

- Implementar de forma eficiente una serie de algoritmos para identificar de forma automática el número de referencias espectrales puras (*endmembers*) en imágenes hiperespectrales, incluyendo métodos estándar como *virtual dimensionality* (VD) y *hyperspectral subspace identification with minimum error* (HySime). Desarrollar nuevas versiones paralelas de los algoritmos VD y HySime, utilizando la arquitectura CUDA para facilitar la implementación eficiente de dichos algoritmos en arquitecturas de tipo GPU.

- Implementar de forma eficiente una serie de algoritmos para identificar de forma automática las firmas espectrales de los *endmembers* en imágenes hiperespectrales, incluyendo algoritmos clásicos como *pixel purity index* (PPI), *iterative error analysis* (IEA) y N-FINDR, el cual requiere un proceso de reducción dimensional previa mediante el método *principal component analysis* (PCA). Desarrollar nuevas versiones paralelas para ser ejecutadas en arquitecturas de tipo GPU de los algoritmos de identificación de *endmembers* PPI, IEA y N-FINDR (así como una nueva versión paralela del método PCA).

- Implementar de forma eficiente una serie de algoritmos para estimar la abundancia a nivel sub-píxel de los *endmembers* en imágenes hiperespectrales, incluyendo métodos como *unconstrained least lquare* (UCLS) *non-negative constrained least square* (NCLS), dependiendo de las restricciones aplicadas en el proceso de estimación de abundancias. Desarrollar nuevas versiones paralelas de los algoritmos UCLS e NCLS para su ejecución en arquitecturas de tipo GPU.

- Realizar un exhaustivo estudio comparativo (en términos de precisión y rendimiento computacional) de las nuevas técnicas paralelas de análisis hiperespectral desarrolladas en el contexto de aplicaciones reales. Específicamente, se considerarán dos aplicaciones reales: la primera está relacionada con la detección de incendios en una imagen obtenida por el sensor *airborne visible infra-red imaging spectrometer* (AVIRIS) de *NASA Jet Propulsion Laboratory* sobre la zona del World Trade Center en la ciudad de Nueva York, días después del atentado terrorista del 11 de Septiembre de 2001. La segunda se está relacionada con la extracción de minerales, es una imagen obtenida también con el sensor AVIRIS sobre la región minera de Cuprite, Nevada, Estados Unidos.

- Evaluar la viabilidad de implementar diferentes cadenas completas de desmezclado espectral para analizar datos hiperespectrales en arquitecturas GPU, de cara a su explotación en misiones reales de observación remota de la tierra. Las cadenas completas están compuestas por los siguientes pasos: 1) estimación automática del número de *endmembers* en una imagen hiperespectral, 2) reducción dimensional (opcional), 3) identificación automática de los *endmembers* presentes en la imagen hiperespectral, y 4) estimación de la abundancia de cada *endmember* en cada píxel de la imagen.

- Discutir de forma detallada cada una de las cadenas completas de desmezclado espectral, evaluadas en el presente trabajo en el contexto de diferentes aplicaciones reales, y proporcionar recomendaciones de uso acerca de dichas cadenas. Conviene destacar que dichas cadenas completas de desmezclado espectral permiten analizar de forma no supervisada la información contenida por una imagen hiperespectral de forma eficiente, sin necesidad de información *a priori* y proporcionando información muy precisa acerca de los materiales espectralmente diferenciables presentes en la escena y su distribución espacial en la misma, con precisión a nivel sub-píxel. Dichas características son altamente deseables y están propiciando la rápida implantación de las técnicas de desmezclado espectral ó *spectral unmixing* como la alternativa más sencilla, eficiente y práctica para analizar de forma rápida la información contenida en imágenes hiperespectrales de la superficie terrestre.

1.3. Organización del documento

A continuación se describen brevemente los diferentes capítulos en los cuales se ha estructurado el presente documento.

1. **Motivaciones y objetivos.** En este capítulo se describen las principales motivaciones y objetivos de el presente trabajo de tesis doctoral. También se describe brevemente la estructura del trabajo y sus principales aportaciones.

2. **Análisis hiperespectral.** En este capítulo se exponen los antecedentes y el estado del arte relativo al análisis de imágenes hiperespectrales en general y a las técnicas de desmezclado espectral ó *unmixing* en particular. De forma más específica, el capítulo describe el concepto de imagen hiperespectral, así como aspectos relacionados con el problema de la mezcla espectral y las diferentes soluciones existentes en la literatura para abordar este problema. El capítulo concluye con una justificación de la necesidad de paralelismo a la hora de analizar imágenes hiperespectrales y una descripción de las contribuciones previas en este campo.

3. **Procesadores gráficos GPUs.** En este capítulo se describen las características fundamentales de las arquitecturas GPUs utilizadas en el presente trabajo de tesis doctoral para acelerar el rendimiento computacional de diferentes cadenas completas de desmezclado espectral ó *spectral unmixing*. En particular, se destaca principalmente la arquitectura CUDA de NVidia^{TM} y las potenciales ventajas resultantes de la utilización de GPUs en el ámbito del análisis de imágenes hiperespectrales.

4. **Implementaciones paralelas.** En este capítulo se detallan las principales contribuciones novedosas del presente trabajo de tesis doctoral, consistentes en la implementación de varias cadenas completas de desmezclado espectral (a partir de la implementación de cada uno de los componentes que las integran) utilizando arquitecturas GPUs. Conviene destacar que las implementaciones desarrolladas en el presente trabajo de tesis doctoral suponen una importante contribución al estado del arte relativo al procesamiento eficiente de imágenes hiperespectrales, dado que dichas implementaciones son totalmente novedosas en este campo.

5. **Resultados.** En este capítulo se describen en detalle los resultados obtenidos en la validación experimental de los algoritmos e implementaciones paralelas propuestas. En primer lugar, se describen las imágenes hiperespectrales reales en la que se ha centrado el estudio. Para finalizar el capítulo se realiza un estudio del rendimiento obtenido comparando las versiones serie (optimizadas) y paralelas de los métodos, haciendo especial énfasis en la posibilidad de obtener resultados en tiempo real a partir de la aplicación de las cadenas completas de desmezclado espectral desarrolladas.

6. **Conclusiones y líneas futuras.** En este capítulo se resumen las principales aportaciones del presente trabajo de tesis doctoral, conclusiones extraídas y se presentan las diferentes líneas futuras de trabajo que han surgido a partir del estudio realizado.

7. **Bibliografía.** Este capítulo enumera las referencias bibliográficas utilizadas que guardan relación con los nuevos desarrollos planteados en la presente memoria.

8. **Publicaciones.** Finalmente, en este capítulo se enumeran las diferentes publicaciones conseguidas por el candidato. Dichas publicaciones comprenden un total de 6 publicaciones en revistas indexadas en el *journal citation reports* (JCR), 3 publicaciones JCR enviadas (en proceso de revisión), 2 capítulos de libro (difusión internacional), 18 publicaciones en congresos internacionales revisados por pares, 1 publicación en congreso nacional y 2 publicaciones online (disponibles en NVidia™ CUDA Zone, dedicada a la difusión internacional de los avances científicos más destacados conseguidos utilizando la arquitectura CUDA de NVidia™). Esto hace un total de 31 publicaciones (30 con carácter internacional, todas ellas revisadas por pares) directamente relacionadas con el presente trabajo de tesis doctoral.

Capítulo 2

Análisis hiperespectral

Este capítulo se organiza de la siguiente forma. En primer lugar, describimos el concepto de imagen hiperespectral, detallando las particularidades y características propias de este tipo de imágenes de alta dimensionalidad. A continuación, describimos algunas características del sensor hiperespectral utilizado en este trabajo. Seguidamente, se muestra una visión general de las técnicas de desmezclado espectral disponibles en la actualidad, con particular énfasis en el modelo lineal de mezcla utilizado en el presente trabajo para abordar el problema de la caracterización sub-píxel de una imagen hiperespectral a partir de la identificación de los píxeles espectralmente más puros en la misma. El capítulo concluye con una justificación de la necesidad de paralelismo a la hora de implementar de forma eficiente algoritmos para desmezclado espectral de imágenes hiperespectrales, con vistas a mejorar la explotación de este tipo de técnicas en aplicaciones reales.

2.1. Imágenes hiperespectrales: el problema de la mezcla

El asentamiento de la tecnología hiperespectral en aplicaciones de observación remota de la tierra ha dado como resultado el desarrollo de instrumentos de medida de muy elevada resolución en los dominios espacial y espectral [24, 43]. Los sensores hiperespectrales adquieren imágenes digitales en una gran cantidad de canales espectrales muy cercanos entre sí, obteniendo, para cada porción de la escena o píxel, una firma espectral característica de cada material [60, 94]. El resultado de la toma de datos por parte de un sensor hiperespectral sobre una determinada escena puede ser representado en forma de cubo de datos, con dos dimensiones para representar la ubicación espacial (x, y) de un píxel, y una tercera dimensión que representa la singularidad espectral de cada píxel en diferentes longitudes de onda [8].

La Fig. 2.1 muestra la estructura de una imagen hiperespectral donde el eje X es el indicador de las líneas, el eje Y es el indicador de las muestras y el eje Z es el número de bandas, es decir, la longitud de onda de esa banda (canal). Como puede apreciarse en la Fig. 2.1, el resultado de la toma de datos por parte de un sensor

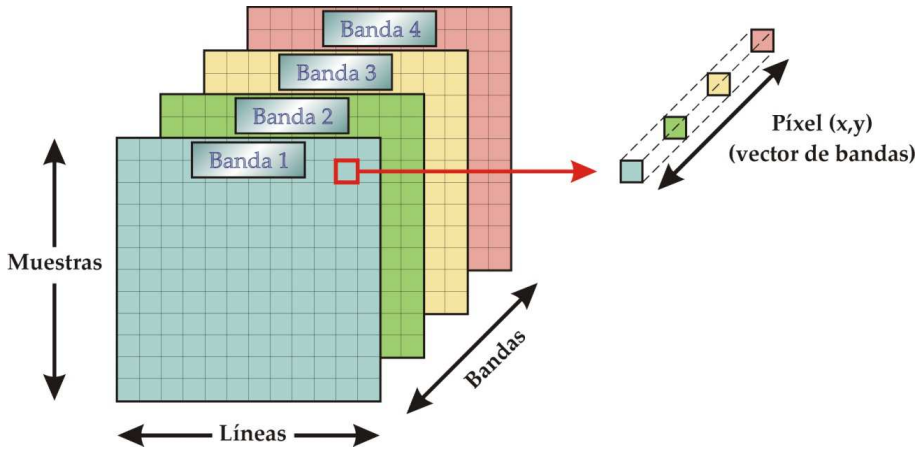

Figura 2.1: Concepto de imagen hiperespectral.

hiperespectral sobre una determinada escena puede ser representado en forma de cubo de datos, con dos dimensiones para representar la ubicación espacial (x, y) de un píxel, y una tercera dimensión que representa la singularidad espectral de cada píxel en diferentes longitudes de onda. En concreto, la capacidad de observación de los sensores denominados hiperespectrales permite la obtención de una *firma espectral* detallada para cada píxel de la imagen, dada por los valores de reflectancia adquiridos por el sensor en diferentes longitudes de onda, lo cual permite una caracterización muy precisa de la superficie de nuestro planeta.

Como ejemplo ilustrativo, la Fig. 2.2 muestra el procedimiento de toma de datos por parte del sensor hiperespectral *airborne visible infra-red imaging spectrometer* (AVIRIS) [29], desarrollado por *NASA Jet Propulsion Laboratory*[1], el cual cubre el rango de longitudes de onda entre 0.4 y 2.5 micrómetros (μm) utilizando 224 canales y resolución espectral de aproximadamente 10 nm. Como puede apreciarse en la figura, el píxel (vector) en una determinada localización espacial (x, y) puede interpretarse como una firma espectral característica del material observado en dicha localización.

Conviene destacar que, en este tipo de imágenes obtenidas a partir de un sensor de observación remota instalado en una plataforma aerotransportada [29] o de tipo satélite [41], es habitual la existencia de mezclas a nivel sub-píxel debidos a la resolución espacial disponible y a otros fenómenos, como el de la mezcla a nivel íntimo o de partículas [38]. El fenómeno de la mezcla es bien conocido desde comienzos de la década de los 80 [36, 97, 1, 23]. En particular, en el sensor AVIRIS (aerotransportado) la resolución espacial es de 20 metros por píxel, lo cual nos da una idea de la gran cantidad de materiales diferentes que pueden encontrarse presentes en una localización tan amplia. Por ejemplo, en una zona urbana dicha localización puede estar compuesta por materiales como carreteras, edificios, vegetación, etc. En imágenes obtenidas a partir de sensores instalados en satélites, la resolución espacial puede decrementarse aún más debido al objetivo de dichos satélites de cubrir amplias zonas de la superficie terrestre. Por motivos

[1] http://aviris.jpl.nasa.gov

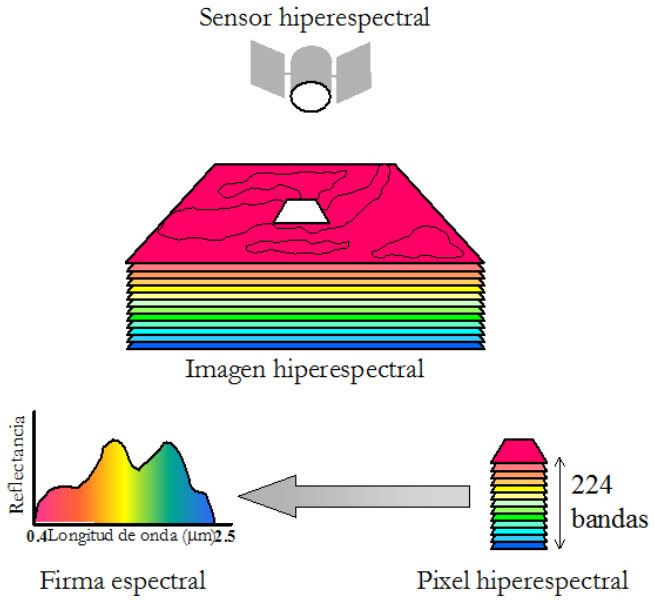

Figura 2.2: Adquisición de una imagen hiperespectral por el sensor AVIRIS.

Tabla 2.1: Misiones hiperespectrales de observación remota de la tierra.

	EO-1 Hyperion*	**Prisma†**	**EnMAP‡**	**HyspIRI§**
País de origen	Estados Unidos	Italia	Alemania	Estados Unidos
Resolución espacial (por píxel)	30 metros	5-30 metros	30 metros	60 metros
Resolución temporal (días)	16	3/7	4	18
Rango espectral (μm)	0.4-2.5	0.4-2.5	0.42-2.45	0.38-2.5
Resolución espectral (nm)	10	10	6.5-10	10
Cobertura por imagen (km)	7.7	30	30	120
Cobertura de la tierra	Parcial	Total	Total	Total
Lanzamiento	2000	2016	2014	2018
Tiempo de vida estimado	10 años	≈ 6 años	≈ 6 años	≈ 6 años

*http://eo1.gsfc.nasa.gov †http://www.asi.it/en/flash_en/observing/prisma ‡http://www.enmap.org §http://hyspiri.jpl.nasa.gov

ilustrativos, la Tabla 2.1 muestra un resumen de las nuevas misiones hiperespectrales de tipo satélite que se encuentran en fase desarrollo. Como puede apreciarse en la Tabla 2.1, la resolución espacial de los nuevos sensores hiperespectrales permite anticipar que seguirán existiendo píxeles mezcla en las imágenes adquiridas a partir de dichos sensores.

Por todo lo anteriormente expuesto, a grandes rasgos podemos encontrar dos tipos de píxeles en imágenes hiperespectrales: puros y mezclas [63]. Se puede definir un píxel mezcla como aquel en el que cohabitan diferentes materiales. Este tipo de píxeles son los que constituyen la mayor parte de la imagen hiperespectral, en parte, debido a que este fenómeno es independiente de la escala considerada [5]. Por motivos ilustrativos, la Fig. 2.3 muestra un ejemplo del proceso de adquisición (a nivel macroscópico) de píxeles puros y mezcla en una imagen hiperespectral. Como puede observarse, algunos materiales como el agua pueden considerarse macroscópicamente puros, aunque estos materiales están

sujetos a variabilidad espectral debida a diferentes condiciones de iluminación, ángulo de observación, etc. [58]. Por otra parte, otros materiales en la escena dan lugar a píxeles mezcla con motivo de la resolución espacial insuficiente, o bien debido a efectos de mezcla íntima [5].

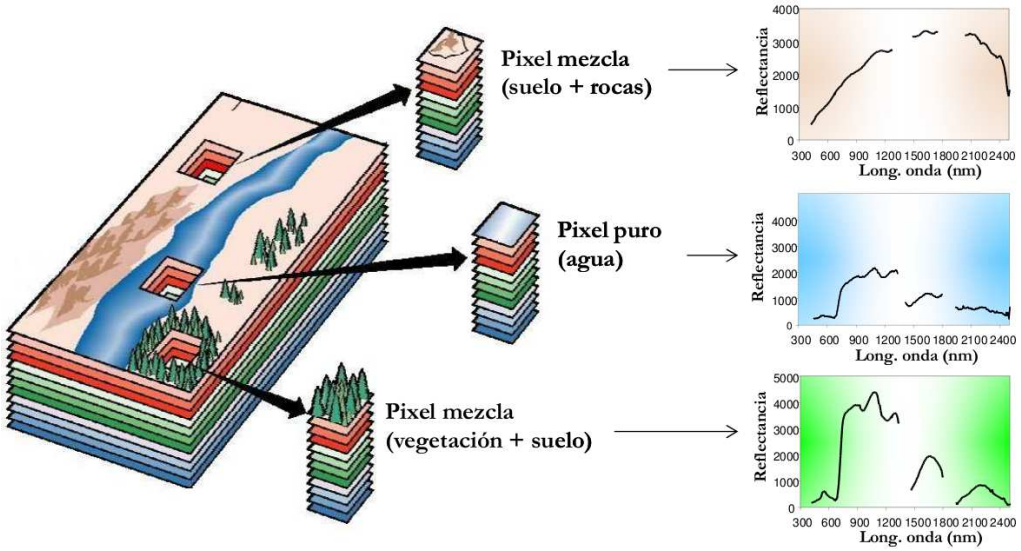

Figura 2.3: Tipos de píxeles en imágenes hiperespectrales.

A continuación destacamos las técnicas más destacadas existentes en la literatura para resolver el problema de la mezcla. Comenzamos describiendo dicho problema en mayor detalle, y analizando las diferentes aproximaciones (lineal y no lineal) para resolverlo. A continuación nos centramos en el modelo lineal de mezcla como la aproximación más genérica y ampliamente utilizada en la literatura para resolver dicho problema, y comentamos diferentes estrategias para identificar el número de *endmembers* en la escena, las firmas espectrales de dichos *endmembers*, y su abundancia en cada píxel de la imagen. Dichas estrategias darán lugar a cadenas completas de desmezclado espectral que permiten extraer información a partir de una determinada imagen de forma completamente no supervisada y con precisión sub-píxel.

2.2. Técnicas para resolver el problema de la mezcla

El desmezclado espectral es un proceso en el que se representa cada píxel de una imagen hiperespectral como una combinación de diferentes firmas espectrales, llamados *endmembers*, y un conjunto de *fracciones de abundancia* que determinan la proporción de cada uno de los *endmembers* en cada píxel de la imagen [40, 63, 3]. En términos generales, los *endmembers* representan materiales espectralmente puros presentes en la imagen, y las abundancias representan la distribución espacial de los mismos, es decir, la presencia de cada endmember en los diferentes píxeles (la mayoría de ellos mezcla) que constituyen la imagen hiperespectral. En esta definición hay algunos matices.

- Primeramente, el concepto de material puro puede ser subjetivo y depende del problema considerado. Por ejemplo, supongamos una imagen hiperespectral en la que aparece un suelo de baldosas, cemento entre las baldosas y dos tipos de plantas que crecen a través de las grietas de las baldosas. Podríamos suponer entonces que tenemos cuatro *endmembers*. Sin embargo, si el área cubierta por el cemento es muy pequeña, es posible que no queramos considerar un *endmember* para el cemento y sólo queramos considerar un *endmember* para el conjunto baldosas/cemento. Todo dependerá de si tenemos una necesidad directa de medir la proporción de cemento presente en la escena. Si nuestra necesidad es medir la proporción de cemento en la escena, seguramente no nos interese diferenciar entre los dos tipos de plantas ya que pueden tener firmas espectrales similares. Por otro lado, supongamos que un tipo de plantas es una especie invasiva que es necesario eliminar. En este caso querríamos tener dos *endmembers* para cada una de los tipos de plantas considerados, para poder diferenciar entre la especie invasiva y la no invasiva. Además, podríamos estar interesados en medir el porcentaje de clorofila presente en la escena. Viendo los diferentes contextos y necesidades en el estudio de una imagen hiperespectral queda más claro que la definición de *endmember* puede depender de la aplicación y es necesario adaptarla a la variabilidad espectral que puede estar presente en la escena.

- El segundo matiz está relacionado con las fracciones de abundancia. La mayoría de los investigadores coinciden en que una abundancia representa el porcentaje de material asociado con un *endmember* presente en la escena y representado por un píxel. Hapke [31] establece que las abundancias en una mezcla lineal representan el área relativa del *endmember* correspondiente en una región de la imagen. Sin embargo, en un caso de mezcla no lineal, la situación no es tan sencilla. En este caso, la radiación reflejada normalmente no es una función lineal. Un objeto pequeño que refleja poca radiación puede predominar en un píxel sobre un objeto oscuro, lo cual supone una estimación imprecisa de la cantidad de material presente en un píxel aunque la estimación de la contribución de cada material a la radiación medida sea precisa. A pesar de todo, es importante destacar que el gran número de aplicaciones de la tecnología hiperespectral a problemas de desmezclado en las últimas décadas indican que los modelos existentes para describir el fenómeno de la mezcla espectral son válidos [63].

Los algoritmos de desmezclado están basados en los diferentes modelos de mezcla. Los modelos de mezcla se pueden caracterizar como lineales y no lineales [46, 40, 3]. La mezcla lineal ocurre cuando la escala de la mezcla es macroscópica, es decir, suponemos que la radiación incidente interactúa de forma separada con cada uno de los materiales que residen a nivel sub-píxel [30, 10], de forma que la respuesta medida por el sensor se puede expresar como una combinación lineal de una serie de firmas espectrales puras junto con sus correspondientes fracciones de abundancias. En este caso la mezcla se produce en el propio sensor, y se debe principalmente a la insuficiente resolución espacial del mismo para separar los diferentes materiales espectralmente puros. La Fig. 2.4 describe

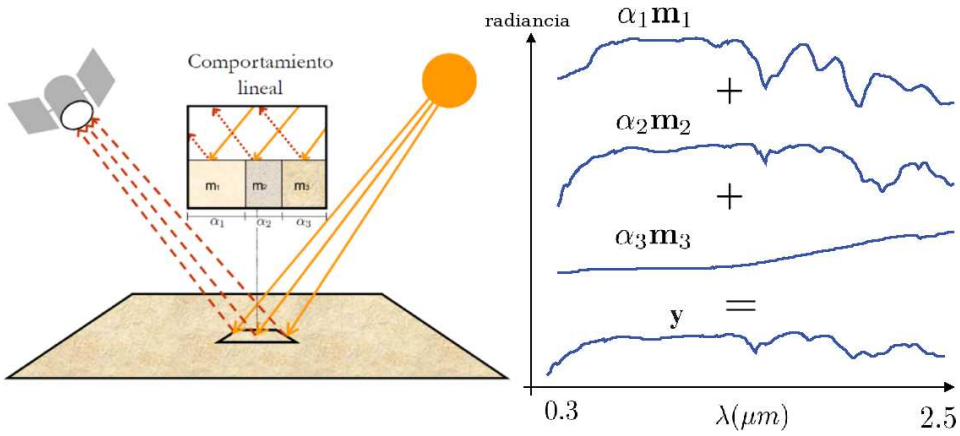

Figura 2.4: Escenario de mezcla lineal.

el modelo lineal de mezcla de forma gráfica: podemos ver como el sensor mide la radiación reflejada por tres *endmembers*, denominados $\mathbf{m}_1$, $\mathbf{m}_2$ y $\mathbf{m}_3$. El espectro medido por el sensor será una media ponderada de las firmas espectrales de los tres materiales puros, donde la cantidad relativa de cada material presente en ese píxel se representa por una serie de valores escalares correspondientes a las fracciones de abundancia (en el ejemplo, α_1, α_2 y α_3).

Por otra parte, los modelos de mezcla no lineal tienen presentes las interacciones físicas en la luz reflejada por los diferentes materiales de la escena. Debido a estas interacciones los modelos de mezcla no lineal pueden basarse en diferentes principios:

- **Mezcla no lineal clásica**. En este caso, el modelo asume que la luz reflejada desde uno o más objetos vuelve a reflejarse en otros objetos y a continuación es medida por el sensor. Este efecto se denomina dispersión múltiple de la luz ó *multiple scattering* [40].

- **Mezcla multinivel**. En este caso, la mezcla no lineal se modela como una secuencia infinita de productos de reflectancias [5].

- **Mezcla microscópica o íntima**. En este caso, se asume que los materiales están mezclados de forma homogénea [31]. En este caso, las interacciones consisten en que los fotones emitidos por moléculas de un material son absorbidos por las moléculas de otro material, que a su vez emite más fotones. Esta situación ocurre independientemente de cuál sea la resolución espacial del sensor. Por tanto, aumentar la resolución espacial del sensor no siempre resuelve el problema de la mezcla.

La Fig. 2.5 ilustra gráficamente dos de los escenarios de mezcla no lineal comentados anteriormente. En la parte izquierda de la Fig. 2.5 se ilustra el modelo basado en mezcla íntima, en el que las partículas de los diferentes materiales aparecen entremezcladas. En este caso, la función de mezcla no lineal f depende no solamente de las abundancias α (que en este caso vienen dadas por la densidad de los materiales y de parámetros propios

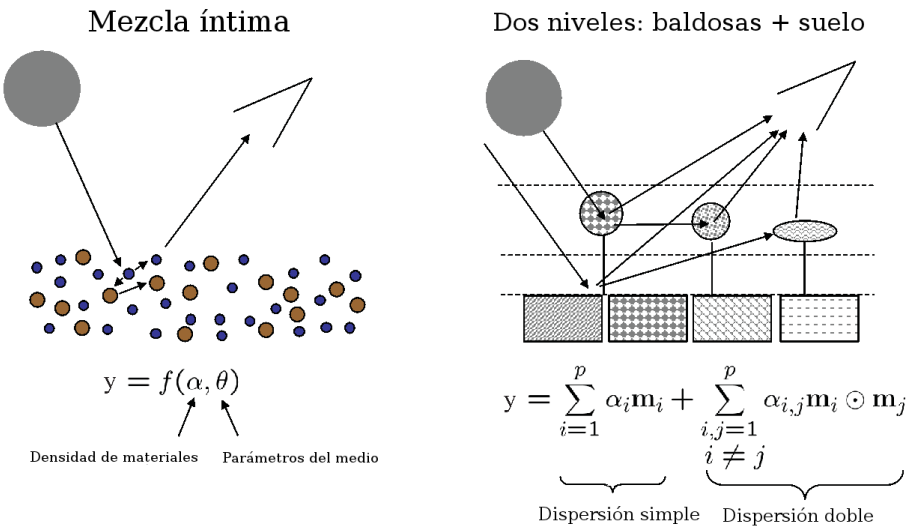

Figura 2.5: Dos escenarios de mezcla no lineal: mezcla íntima (izquierda) y mezcla multinivel (derecha).

del medio (objeto observado) que aparecen representados por un parámetro θ. Por tanto, la falta de información acerca del parámetro θ imposibilita la estimación de la función de mezcla f. Esta es una situación muy habitual en escenarios reales, en los que muchas veces no se dispone de información *a priori* acerca de los objetos observados. Por otra parte, la parte derecha de la Fig. 2.5 ilustra una situación de mezcla no lineal multinivel en la que se producen múltiples interacciones en la dispersión de la luz en los diferentes estratos. En este caso, a los efectos de dispersión simple entre los *endmembers* $\{\mathbf{m}_i\}_{i=1}^{p}$ se añaden efectos de dispersión doble que dificultan la caracterización de la interacción entre los diferentes *endmembers* que participan en la mezcla, haciendo de nuevo que resulte muy difícil el proceso de estimación de sus abundancias a no ser que se disponga de información detallada acerca del medio.

Dada la dificultad de modelar correctamente los efectos de mezcla no lineal en ausencia de información *a priori* detallada acerca de las características del medio, la tendencia en la literatura reciente sobre desmezclado espectral [3] ha sido suponer que el modelo lineal ofrece una aproximación razonable para caracterizar mezclas que, posteriormente, puede refinarse en el caso de que exista información suficiente acerca de las propiedades del medio. En un escenario genérico, el modelo lineal resulta ser una aproximación aceptable al fenómeno de la mezcla caracterizado por su simplicidad y generalidad. En lo sucesivo, nos centramos en dicho modelo lineal no sin reconocer que el modelo no lineal está sujeto a numerosos desarrollos futuros que, sin duda, cristalizarán en aproximaciones más robustas basadas en las propiedades físicas de los objetos observados.

2.3. Modelo lineal de mezcla

El modelo lineal de mezcla asume que la radiación solar incidente interactúa de forma lineal con los diferentes componentes espectralmente puros ó *endmembers* que componen un determinado píxel mezcla. En este caso, la radiación total reflejada por un píxel mezcla se puede descomponer de forma proporcional a la abundancia de cada uno de los *endmembers* en el píxel. Suponiendo que las interacciones entre los diferentes materiales son despreciables y que la superficie observada se encuentra bien particionada espacialmente, como se muestra en la Fig. 2.4, entonces la firma espectral medida por el sensor puede aproximarse mediante una combinación lineal de las firmas espectrales de los *endmembers* en la proporción de sus correspondientes fracciones de abundancias [1, 46, 39, 40]. Sea $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$ una imagen hiperespectral con n píxeles de l bandas o canales espectrales cada uno. Si denotamos un determinado píxel de la imagen como $\mathbf{y}_j$, dicho píxel puede expresarse mediante una combinación lineal del conjunto de *endmembers* de la imagen $\{\mathbf{m}_i\}_{i=1}^{p}$, ponderados por sus correspondientes fracciones de abundancia, de la siguiente forma:

$$\mathbf{y}_j = \sum_{i=1}^{p} \alpha_i \mathbf{m}_i + \mathbf{n}, \tag{2.1}$$

donde $\alpha_i \geq 0$ denota la abundancia del *endmember* $\mathbf{m}_i$, $\mathbf{n}$ representa un factor de ruido, y p denota el número de *endmembers*. La fracción de abundancia α_i en un píxel concreto, tal y como su nombre indica, representa la fracción de área ocupada por el *endmember* $\mathbf{m}_i$ en dicho píxel. En realidad las fracciones de abundancia deben estar sujetas a las siguientes restricciones:

$$\begin{array}{ll} \text{Restricción de no negatividad:} & \alpha_i \geq 0,\ i = 1, \ldots, p \\ \text{Restricción de suma unitaria:} & \sum_{i=1}^{p} \alpha_i = 1; \end{array} \tag{2.2}$$

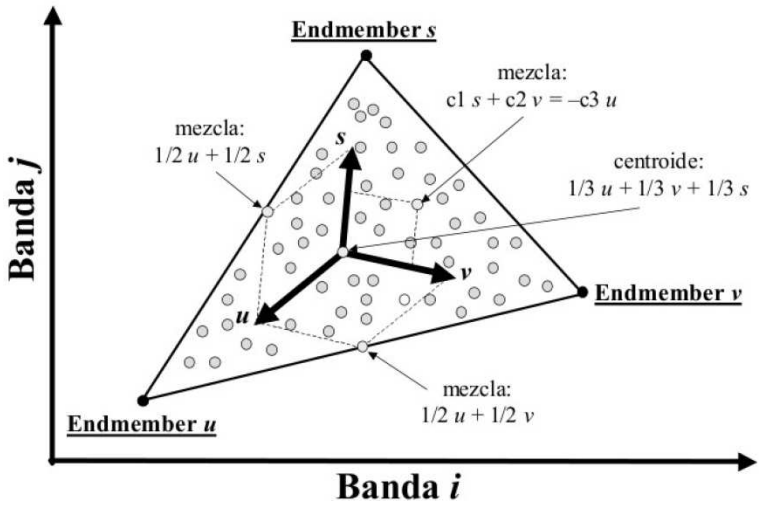

Figura 2.6: Interpretación gráfica de el modelo lineal de mezcla.

El modelo lineal de mezcla puede interpretarse de forma gráfica utilizando un diagrama de dispersión entre dos bandas poco correlacionadas de la imagen, tal y como se muestra en la Fig. 2.6. En la misma, puede apreciarse que todos los puntos de la imagen quedan englobados dentro del triángulo formado por los tres puntos más extremos (elementos espectralmente más puros). Los vectores asociados a dichos puntos constituyen un nuevo sistema de coordenadas con origen en el centroide de la nube de puntos, de forma que cualquier punto de la imagen puede expresarse como combinación lineal de los puntos más extremos, siendo estos puntos los mejores candidatos para ser seleccionados como *endmembers*. El paso clave a la hora de aplicar el modelo lineal de mezcla consiste por tanto en identificar de forma correcta los elementos extremos de la nube de puntos l-dimensional.

A la hora de abordar el proceso de desmezclado de una imagen hiperespectral mediante el modelo lineal de mezcla suelen realizarse diferentes etapas que aparecen resumidas en la Fig. 2.7 [3]. A continuación se describe brevemente cada una de estas etapas las cuales, en conjunto, determinan una cadena completa de desmezclado espectral que toma como punto de partida la imagen original $\mathbf{Y}$ y devuelve como resultado un conjunto de p firmas espectrales o *endmmebers* junto con sus correspondientes abundancias en cada píxel de la imagen. Dado que los valores de abundancia de cada *endmember* en cada píxel pueden representarse en forma de mapa, normalmente se habla de *mapas de abundancia* para referirse a los valores de abundancia estimados para cada *endmember* en la totalidad de la imagen.

1. **Estimación del número de *endmembers***. En esta etapa se realiza un proceso de estimación del número de *endmembers* presentes en la imagen hiperespectral $\mathbf{Y}$ con el objetivo de establecer cuántas firmas espectrales puras deben identificarse en etapas posteriores.

2. **Reducción dimensional**. Normalmente el número de bandas l de la imagen hiperespectral $\mathbf{Y}$ es mucho mayor que el número de *endmembers* ó materiales espectralmente distintos, p, presentes en la imagen. Esto hace que la imagen hiperespectral pueda representarse eficientemente en un subespacio de dimensionalidad inferior a l. La identificación de este subespacio nos permite trabajar de forma más efectiva con la imagen en términos de almacenamiento y de complejidad computacional. Por otra parte, como se muestra en la Fig. 2.6, para formar el *simplex* que engloba a los píxeles de la imagen hiperespectrales necesitamos un número de *endmembers* superior en una unidad a la dimensionalidad intrínseca de los datos, entendida como la dimensionalidad del subespacio en el que los datos altamente dimensionales residen en realidad [3].

3. **Identificación de los *endmembers***. Esta fase consiste en la identificación de las firmas espectrales de los *endmembers* en la escena. Dicho proceso puede abordarse desde diferentes puntos de vista. Por ejemplo, los métodos que explotan una aproximación geométrica al problema aprovechan el hecho de que todos los píxeles mezcla deben encontrarse en el interior del *simplex* o cono positivo definido por los *endmembers* (ver Fig. 2.6) y, por tanto, favorecen la identificación de los píxeles que

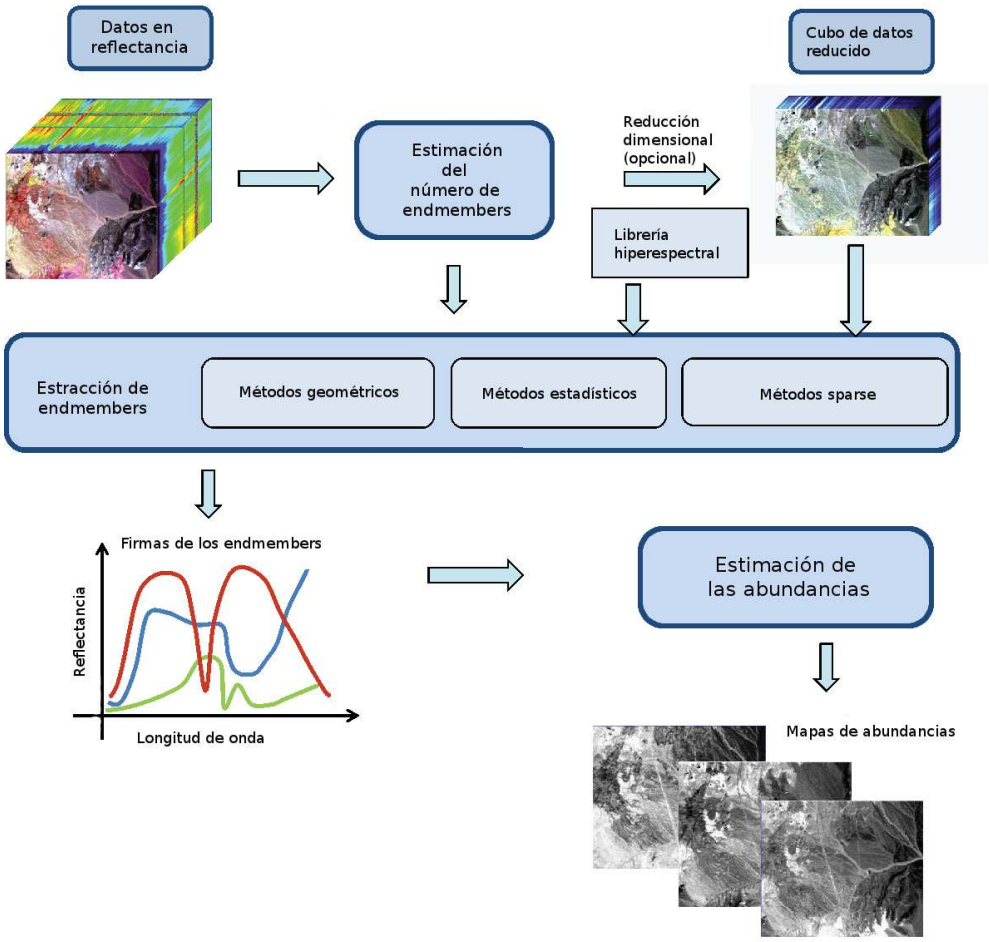

Figura 2.7: Cadena completa para el desmezclado de una imagen hiperespectral.

se encuentran en los vértices del *simplex* como *endmembers*. Sin duda, los métodos geométricos suponen la aproximación más importante en la literatura al problema de identificación de *endmembers*, pero existen otras aproximaciones al problema tales como los métodos estadísticos, entre los que destacan aquellos que incluyen información espacial en el proceso de identificación de *endmembers* [64, 78, 106, 48]. Por otra parte, los métodos *sparse* se caracterizan por utilizan librerías espectrales para abordar el problema del desmezclado utilizando firmas espectrales reales no necesariamente presentes en la imagen [35]. Dentro de las aproximaciones geométricas, existen también técnicas que no asumen la presencia de píxeles puros en la imagen [16, 49, 45, 34]. A pesar de que las aproximaciones anteriormente mencionadas pueden ofrecer ventajas competitivas en determinadas circunstancias, los métodos que no asumen la presencia de píxeles puros normalmente intentan derivar *endmembers virtuales* capaces de englobar a todos los píxeles de la imagen. Esto puede dar lugar a la generación de *endmembers* con firmas espectrales

no necesariamente asociadas a materiales existentes en la naturaleza, con los problemas que ello conlleva desde el punto de vista de la interpretación física de dichos *endmembers*. Por otra parte, los métodos *sparse* basados en librerías espectrales evitan el problema de identificar *endmembers* en la imagen ya que asumen que la librería espectral (con un gran número de entradas) contiene las firmas espectrales de dichos *endmembers*, pero deben hacer frente al hecho de que las firmas espectrales en la librería normalmente se adquieren en condiciones óptimas en laboratorio, mientras que la toma de datos por parte de un sensor hiperespectral está sujeta a condiciones atmosféricas variables, las cuales hacen necesario un proceso de corrección atmosférica óptimo para poder utilizar los *endmembers* de la librería espectral para desmezclar la imagen (adquirida en condiciones muy diferentes). Finalmente, la identificación de *endmembers* directamente a partir de píxeles de la imagen presenta la ventaja de que los *endmembers* se adquieren en las mismas condiciones que la imagen, pero pueden presentar problemas si la imagen hiperespectral no contiene suficientes píxeles puros para los diferentes objetos presentes en la escena. En lo sucesivo, nos centramos en los métodos de identificación de *endmembers* basados en aproximaciones geométricas que asumen la presencia de píxeles puros en la imagen asumiendo las posibles limitaciones de dichos métodos [3].

4. **Estimación de las abundancias**. Una vez extraídos los *endmembers*, la última fase del proceso de desmezclado espectral lineal consiste en representar cada uno de los píxeles de la imagen hiperespectral como una combinación lineal de los píxeles espectralmente puros, esto es, determinar la proporción en la que cada *endmember* contribuye a la mezcla que se produce en cada píxel de la imagen. En el caso de que un píxel sea considerado puro, la presencia de un determinado *endmember* será totalmente dominante con respecto al resto de *endmembers*. Una vez estimada esta información podemos confeccionar los denominados mapas de abundancia, que consisten en una serie de mapas (uno por cada *endmember*) en la que el valor de un píxel representa la proporción en la que el *endmember* asociado a ese mapa está presente en el píxel medido por el sensor. Los valores de las abundancias calculados en esta fase pueden estar o no sujetos a las restricciones de suma unitaria y no negatividad referidas en la ecuación (2.2). Veremos en lo sucesivo cómo algunos métodos de estimación de abundancias realizan al mismo tiempo el proceso de identificación de *endmembers*.

A continuación se describen diferentes aproximaciones clásicas para cada una de las etapas anteriormente descritas, las cuales constituirán el principal objeto de las implementaciones paralelas desarrolladas en el presente estudio. El capítulo finaliza con una justificación de la necesidad de paralelismo a la hora de implementar de forma eficiente cada una de las técnicas que a continuación se relacionan, lo cual constituye la principal contribución innovadora del presente trabajo.

2.3.1. Estimación del número de *endmembers*

La estimación del número de *endmembers* constituye una etapa fundamental para el proceso de desmezclado de una imagen hiperespectral. Esto se debe a que el número de *endmembers* es un parámetro crucial en etapas posteriores de la cadena, y servirá como entrada al proceso de identificación de *endmembers* y también al proceso de estimación de abundancias. A pesar de la gran importancia de esta etapa, en la literatura no existen numerosas aproximaciones automáticas para estimar el número de *endmembers*. En el presente apartado describimos las dos aproximaciones más ampliamente utilizadas: *virtual dimensionality* (VD) [9], que permite cierta flexibilidad en la estimación al disponer de un parámetro adicional de entrada que permite controlar la sensibilidad del método, y *hyperspectral signal identification with minimum error* (HySime) [9], que ofrece características avanzadas a la hora de modelar el ruido presente en la imagen hiperespectral. A continuación se describen estas dos aproximaciones al problema.

2.3.1.1. *Virtual Dimensionality* (VD)

Este método se basa en el principio de que, si una determinada señal o *endmember* está presente en la imagen hiperespectral, dicho *endmember* debe ser representativo en una banda espectral concreta en la cual su identificación debe ser más sencilla [9]. Dada una imagen hiperespectral $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$ cuyas dimensiones son n píxeles por l bandas, el primer paso de VD es calcular los autovalores para cada banda de las matrices de correlación $\mathbf{R}_{l\times l}$ y de covarianza $\mathbf{K}_{l\times l}$ referidos como $\widehat{\lambda}_i$ y λ_i siendo $i \in \{1, \cdots, l\}$. Una vez calculados los autovalores se realiza un *test* o prueba para cada banda espectral, en el que se analizan dos hipótesis: $H_0 : \widehat{\lambda}_i - \lambda_i = 0$ (la cual indica la ausencia de u determinado *endmember* en la i-ésima banda espectral), y $H_1 : \widehat{\lambda}_i - \lambda_i > 0$ (la cual indica la presencia de un determinado (*endmember*) en la i-ésima banda espectral). Para estimar la dimensionalidad de la imagen, el método VD utiliza un detector de Neyman-Pearson [8] que estima cuántas veces falla la prueba para cada una de las l bandas espectrales de la imagen $\mathbf{Y}$, utilizando como parámetro de entrada un valor de probabilidad de falsa alarma P_F que determina la sensibilidad del método al comprobar las dos hipótesis anteriormente referidas. La correspondiente dimensionalidad se define como el número veces que el test falla, lo que indica el número de señales (*endmembers*), p, presentes en los datos. Los autores del método recomiendan utilizar valores de P_F entre 10^{-1} y 10^{-5} [9].

2.3.1.2. *Hyperspectral signal identification by minimum error* (HySime)

El método HySime [2] consta de dos partes claramente diferenciadas. En la primera parte se realiza una estimación del ruido en la imagen hiperespectral $\mathbf{Y}$, obteniendo una matriz $\widehat{\xi}$ de tamaño $n \times l$ con la estimación del ruido presente en la imagen (recordamos que n es el número de píxeles y l el número de bandas de la imagen hiperespectral). El Algoritmo 1 muestra en detalle los pasos a seguir para llevar a cabo la fase de estimación del ruido. En dicho algoritmo, $[\mathbf{R}']_{\alpha_i,\alpha_i}$ denota la matriz resultante de eliminar la i-ésima fila y la i-ésima columna de $\mathbf{R}'$. Por su parte, $[\mathbf{R}']_{i,\alpha_i}$ denota la i-ésima fila de

$\mathbf{R}'$ y $[\mathbf{R}']_{\alpha_i,i}$ denota la i-ésima columna de $\mathbf{R}'$. Los pasos 2 y 3 del Algoritmo 1 realizan el cálculo de la matriz $\widehat{\mathbf{R}} := (\mathbf{Z}^T\mathbf{Z})$ y su inversa, respectivamente. Los pasos 5 y 6 del Algoritmo 1 estiman respectivamente el vector de regresión $\widehat{\beta}_i$ y el ruido $\widehat{\xi}$ para cada banda $i \in \{1, \ldots, l\}$.

Algorithm 1 Estimación del ruido

1: **Entrada**: $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$
2: $\mathbf{Z} = \mathbf{Y}^T$, $\widehat{\mathbf{R}} := (\mathbf{Z}^T\mathbf{Z})$
3: $\mathbf{R}' = \widehat{\mathbf{R}}^{-1}$
4: **for** i=1 to l **do**
5: $\quad \widehat{\beta}_i = ([\mathbf{R}']_{\alpha_i,\alpha_i} - [\mathbf{R}']_{\alpha_i,i}[\mathbf{R}']_{i,\alpha_i}/[\mathbf{R}']_{i,i})[\widehat{\mathbf{R}}]_{\alpha_i,i}$
$\quad \{\alpha_i = 1, ..., \text{i-1}, \text{i+1}, ..., l\}$
6: $\quad \widehat{\xi}_i = \mathbf{z}_i - \mathbf{Z}_{\alpha_i}\widehat{\beta}_i$
7: **end for**
8: **Salida**: $\widehat{\xi}$ $\{n \times l\}$

La segunda parte del método HySime realiza una estimación del subespacio en el que se incluyen los datos hiperespectrales. Esta parte comienza con la estimación de las matrices de correlación de la señal $\widehat{\mathbf{R}}_s$ y del ruido $\widehat{\mathbf{R}}_n$. A continuación se selecciona el subconjunto de autovectores que mejor representa el subespacio de señales, teniendo en cuenta un criterio basado en el mínimo error cuadrático medio. La aplicación de este criterio lleva a la minimización de una función objetivo formada por dos términos: el primero se corresponde con la potencia de la proyección de la señal, mientras que el segundo término se corresponde con la potencia de la proyección del ruido. Realizando la minimización de esta función se puede estimar el número de *endmembers* p presentes en el subespacio de señales $\widehat{\mathbf{S}}$.

El Algoritmo 2 muestra en detalle los pasos que se siguen en la segunda etapa del algoritmo HySime. Las entradas a este paso son los vectores espectrales observados $[\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$, es decir los píxeles de la imagen $\mathbf{Y}$, y la matriz de correlación $\widehat{\mathbf{R}}_y$. El paso 2 del Algoritmo 2 estima la matriz de correlación del ruido $\widehat{\mathbf{R}}_n$. El paso 3 del Algoritmo 2 estima la matriz de correlación de la señal $\widehat{\mathbf{R}}_s$. Los pasos 4 y 5 del Algoritmo 2 calculan los autovectores de la matriz de correlación de señal y los términos δ_i basados en formas cuadráticas. Los pasos 6 y 7 del Algoritmo 2 llevan a cabo la función de minimización para estimar el número de *endmembers* p. Finalmente el paso 8 del Algoritmo 2 almacena el subespacio de señal para p y $\widehat{\pi}$.

Por razones ilustrativas, la Fig. 2.8 muestra un ejemplo sencillo en el que se muestra de forma gráfica el funcionamiento de HySime en un ejemplo sencillo basado en una imagen con tres bandas. En la Fig. 2.8(a) podemos ver como HySime selecciona un hiperplano donde queda incluido el subespacio de señales $\widehat{\mathbf{S}}$. En la Fig. 2.8(b) se representa la dimensionalidad del subespacio estimado, es decir el número de *endmembers*, p.

2.3.2. Reducción dimensional

La reducción dimensional es un paso opcionalmente utilizado por algunos algoritmos de identificación de *endmembers* con objeto de trabajar en un subespacio (como el

Algorithm 2 Estimación del subespacio

1: **Entradas**: $\mathbf{Y} \equiv [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$, $\widehat{\mathbf{R}}_y \equiv (\mathbf{Y}\mathbf{Y}^T)/n$
2: $\widehat{\mathbf{R}}_n = 1/n \sum_i (\widehat{\xi}_i \widehat{\xi}_i^T)$
3: $\widehat{\mathbf{R}}_s = 1/n \sum_i ((\mathbf{y}_i - \widehat{\xi}_i)(\mathbf{y}_i - \widehat{\xi}_i^T))$ estimación de $\widehat{\mathbf{R}}_s\}$
4: $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_l]$ $\{\mathbf{v}_i$ son autovectores de $\widehat{\mathbf{R}}_s\}$
5: $\delta = [\delta_1, ..., \delta_l]$
6: $(\widehat{\delta}, \widehat{\pi}) :=$ sort(δ)
{ordenar δ_i en orden ascendente; guardar la permutación $\widehat{\pi}$}
7: $p =$número de términos $\widehat{\delta}_i < 0$
8: **Salida**: $\widehat{\mathbf{X}} = \langle [\widehat{\mathbf{m}}_{i1}, \cdots, \widehat{\mathbf{m}}_{ip}] \rangle$

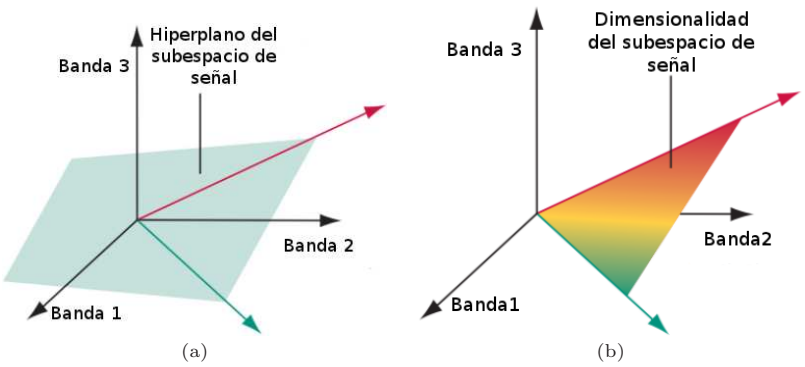

Figura 2.8: Representación gráfica del funcionamiento del algoritmo HySime.

determinado por los algoritmos VD y HySime descritos en el anterior apartado) y reducir así la carga computacional de los mismos mediante la eliminación de ruido y datos redundantes en la imagen. Una de las transformaciones más ampliamente utilizadas en análisis hiperespectral para llevar a cabo este proceso es el análisis de componentes principales ó *principal component analysis* (PCA) [76]. En el presente apartado describimos la técnica PCA y una versión simplificada denominada simple PCA (SPCA) susceptible de ser paralelizada de forma más eficiente.

2.3.2.1. *Principal component analysis* (PCA)

La PCA [37] es una técnica estadística utilizada para reducir la dimensionalidad de un conjunto de datos. Intuitivamente, la técnica sirve para hallar las causas de la variabilidad de un conjunto de datos y así ordenar los datos según su importancia. A nivel técnico, la PCA busca las proyección que mejor representan a los datos en términos de la varianza de los mismos. Para ello, es necesario calcular la descomposición en autovalores de la matriz de covarianza, normalmente tras centrar los datos en la media de cada observación (en nuestro caso, los píxeles l-dimensionales que componen la imagen hiperespectral). Este proceso suele realizarse utilizando la descomposición en valores singulares ó *singular value decomposition* (SVD) [76], que supone un proceso costoso en términos computacionales, sobre todo cuando se aplica a imágenes hiperespectrales de gran dimensionalidad. La Fig. 2.9 muestra una sencilla representación gráfica que ilustra la utilización de la PCA

como una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual dirección con mayor varianza del conjunto de datos es capturada en el primer eje (primera componente principal), la segunda dirección con mayor varianza define el segundo eje, y así sucesivamente. La expresión utilizada para calcular la PCA de un conjunto de datos es:

$$\mathbf{K} = \mathbf{V}\Lambda\mathbf{V}^{-1}, \tag{2.3}$$

donde donde $\mathbf{K}$ es la matriz de covarianza de la imagen hiperespectral $\mathbf{Y}$, $\mathbf{V} = [\mathbf{v}_1\mathbf{v}_2, \ldots, \mathbf{v}_l]$ es una matriz ortogonal cuyas columnas son los autovectores de $\mathbf{K}$, l es el número total de bandas espectrales y Λ es una matriz diagonal que contiene los autovalores de $\mathbf{K}$. La proyección de la imagen hiperespectral $\mathbf{Y}$ sobre los autovectores $\mathbf{V}$ proporciona como resultado las componentes principales de $\mathbf{Y}$. Los autovalores en Λ contienen la relevancia o peso de las componentes principales de los datos resultantes. Eligiendo sólo los autovectores correspondientes a los p autovalores más grandes, podemos reducir la dimensionalidad de los datos originales al número de *endmembers* de la imagen manteniendo una gran cantidad de información (varianza) en los datos. En análisis hiperespectral esto se traduce en la posibilidad de trabajar en un espacio reducido de manera que la cantidad de información contenida en el subespacio es similar a la contenida en el espacio original. Por ejemplo, en este contexto podemos extraer los mismos *endmembers* que en la imagen original pero a partir de una cantidad o volumen de datos mucho menor.

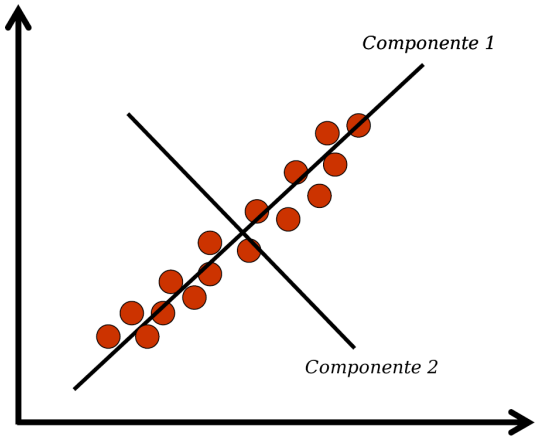

Figura 2.9: Representación gráfica de la PCA.

La Fig. 2.10 muestra una representación gráfica del efecto de aplicar la transformación PCA a una imagen hiperespectral estándar. Como podemos ver, tras aplicar la PCA y representar gráficamente las primeras 20 componentes se observa intuitivamente que la cantidad de información contenida en las componentes va decreciendo, de forma que las últimas componentes aportan menos información y están dominadas por una mayor cantidad de ruido. Intuitivamente, podríamos reducir la dimensionalidad de la imagen hiperespectrales eliminando las componentes con mayor ruido y reteniendo las que tienen mayor información o varianza, que son precisamente las primeras en la ordenación que

proporciona la PCA. En resumen, la PCA ofrece una transformación intuitiva que ordena los datos en componentes con información progresivamente decreciente que permite obtener una representación reducida de los datos originales en un subespacio optimizado según la distribución de los datos. Conviene destacar que la técnica PCA se ha utilizado como base para desarrollar otras técnicas de reducción dimensional como la técnica de la fracción mínima de ruido ó *minimum noise fraction* (MNF) [28] y análisis de componentes independientes ó *independent component analysis* (ICA) [51]. A continuación describimos una versión simplificada de la PCA denominada SPCA y susceptible de ser paralelizada de forma más eficiente.

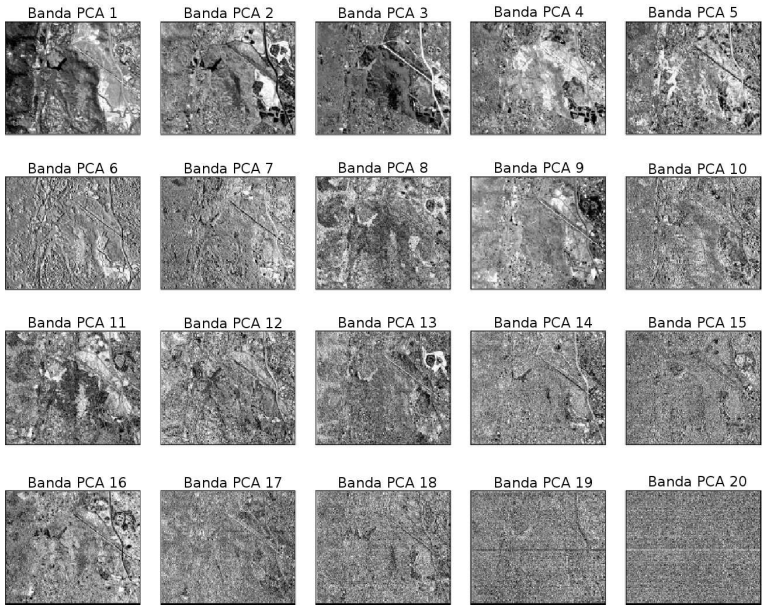

Figura 2.10: Ejemplo de aplicación de la transformada PCA sobre una imagen hiperespectral.

2.3.2.2. *Simple principal component analysis* (SPCA)

El método SPCA es una variante de la PCA que no requiere de la utilización de la transformación SVD para calcular los autovalores y autovectores de los datos. Más concretamente, la SPCA permite la rápida obtención de los p primeros autovectores y además es muy susceptible de ser paralelizada. Existen varios algoritmos para extraer los autovalores y los autovectores además de SVD, incluyendo la descomposición QR [54], el algoritmo *multiple relatively robust representations* (MRRR) [55], el algoritmo Jacobi-Davidson [96] o el algoritmo de Arnoldi [79]. Hay que tener en cuenta que, en nuestro caso, solamente necesitamos los $p << l$ autovalores más significativos (y sus autovectores asociados). Aprovechando esta particularidad, podemos utilizar el algoritmo *power iteration* [25] para extraer los autovalores (y autovectores asociados) más significativos definiendo en primer lugar un vector aleatorio $\tilde{\mathbf{q}}$ como $\mathbf{q} = \mathbf{V}\tilde{\mathbf{q}}$. Entonces

tenemos:

$$\mathbf{K}^k\mathbf{q} = \mathbf{K}^k\mathbf{V}\tilde{\mathbf{q}} = \mathbf{V}\Lambda^k\mathbf{V}^{-1}\mathbf{V}\tilde{\mathbf{q}} = \\ = \sum_{j=1}^{n}\mathbf{v}_j\lambda_j^k\tilde{q}_j = \lambda_1^k\sum_{j=1}^{n}\mathbf{v}_j\left(\frac{\lambda_j}{\lambda_1}\right)^k\tilde{q}_j. \tag{2.4}$$

Si la matriz $\mathbf{K}$ (covarianza de la imagen hiperespectral $\mathbf{Y}$) tiene un autovalor dominante, por ejemplo $|\lambda_1| > |\lambda_j| \quad \forall j \neq 1$, entonces como $k \to \infty$ el término $(\lambda_j/\lambda_1)^k \to 0 \quad \forall j \neq 1$. Por tanto, la ecuación 2.4 convergerá al autovalor dominante a medida que k aumente. Como la magnitud de λ_1^k también se incrementa con k, es necesario normalizar el vector que se está tratando en cada iteración. La ecuación (2.5) define el comportamiento del algoritmo *power iteration* anteriormente mencionado:

$$\mathbf{q}^{(k+1)} = \frac{\mathbf{K}^k\mathbf{q}^{(0)}}{\|\mathbf{K}^k\mathbf{q}^{(0)}\|} = \frac{\mathbf{K}\mathbf{q}^{(k)}}{\|\mathbf{K}\mathbf{q}^{(k)}\|}. \tag{2.5}$$

En particular, se pueden realizar iteraciones simultáneas [15] aplicando el algoritmo *power iteration* en la ecuación (2.5) a varios autovectores de forma simultánea. A pesar de que este método es menos eficiente en términos computacionales que otras aproximaciones para calcular la PCA [54, 55, 96, 79], se trata ciertamente de un método muy regular y presenta paralelismo a nivel de datos, lo cual permite su implementación en arquitecturas masivamente paralelas como GPUs. Debido a que el algoritmo *power iteration* converge sólo al autovalor dominante, aplicarlo continuamente puede proporcionar siempre el mismo resultado. Para evitar esta situación, el resultado debe ser decorrelacionado. Esto se lleva a cabo restando a la matriz la influencia del autovector más grande (operación denominada deflación), que se puede obtener de forma simultánea para todos los autovalores (y autovectores asociados) que están siendo calculados. Si la deflación se realiza en el espacio de datos original $\mathbf{Y}$, dicha operación lleva asociada un aumento de la dimensionalidad del problema, ya que se necesita crear una sub-matriz $\mathbf{Y}_i$ para cada autovalor/autovector calculado. Sin embargo, esta operación de deflación puede también llevarse a cabo en el espacio de datos dado por los autovectores (la matriz $\mathbf{V}_{dfl}$), que se obtiene al multiplicar la matriz $\mathbf{V}$ por una matriz de deflación Δ. Como estamos restando a cada autovector la influencia de los autovectores anteriores (más significativos), la matriz Δ es triangular superior. Cada elemento distinto de cero δ_{ij} de la matriz Δ se obtiene conforme a la expresión (2.6) de la siguiente forma:

$$\delta_{ii} = 1$$
$$\delta_{ij} = -\mathbf{v}_i^T\mathbf{v}_j - \sum_{k=i+1}^{j-1}\delta_{ik}\mathbf{v}_k^T\mathbf{v}_j, \quad \text{para} \quad j > i \tag{2.6}$$

Llegados a este punto, es importante destacar que aunque la construcción de la matriz Δ se hace de forma recursiva, cada fila es completamente independiente de las demás y se puede calcular en paralelo. El proceso completo se muestra en el Algoritmo 3, en el que Φ se corresponde con la operación *power iteration*, y Ψ es la corrección introducida por el proceso de deflación. La función triu() obtiene la parte superior de una matriz triangular.

La convergencia se consigue cuando la dirección de los autovectores se mantiene invariante entre una iteración a otra. Mediante el proceso basado en iteraciones simultaneas se calculan los p autovectores más significativos de forma directa, sin necesidad de reducir la matriz de datos o de calcular los autovectores más pequeños.

Algorithm 3 Iteraciones simultáneas

1: $\mathbf{K} = covarianza(\mathbf{Y})$
2: **while** no convergencia **do**
3: construir matriz Δ
4: $\mathbf{V}_{dfl} = \mathbf{V}\Delta$
5: $\Phi = \mathbf{K}\,\mathbf{V}_{dfl}$
6: $\Psi = \mathbf{V}\,\text{triu}(\mathbf{V}_{dfl}^T\Phi, -1)$
7: $\mathbf{V} = \Phi - \Psi$
8: $\mathbf{V} = \mathbf{V}/norm(\mathbf{V}) = \left[\frac{\mathbf{v}_1}{norm(\mathbf{v}_1)}\ \frac{\mathbf{v}_2}{norm(\mathbf{v}_2)}\cdots\frac{\mathbf{v}_d}{norm(\mathbf{v}_d)}\right]$
9: **end while**

2.3.3. Identificación de endmembers

En esta sección describimos una serie de algoritmos utilizados para la extracción de firmas espectrales puras o *endmembers* a partir de una imagen hiperespectral. Como hemos comentado anteriormente, en la literatura existen múltiples aproximaciones al problema [3]. A continuación describimos dos aproximaciones clásicas que asumen la presencia de píxeles puros en la escena: PPI y N-FINDR, quizá los dos algoritmos de identificación de *endmembers* más populares en la literatura.

2.3.3.1. *Pixel purity index* (PPI)

El algoritmo PPI [4] es uno de los algoritmos más populares en análisis hiperespectral debido a su temprana incorporación al software ENVI durante la década de los 90. El algoritmo se basa en un concepto muy sencillo: la proyección sucesiva de todos los píxeles de la imagen hiperespectral $\mathbf{Y}$ sobre un conjunto de vectores aleatorios l-dimensionales denominados *skewers*, que se utilizan para particionar el espacio de búsqueda dado por los píxeles l-dimensionales de la imagen hiperespectral original, identificando aquellos píxeles extremos en cada proyección (ver Fig. 2.11) e incrementando un contador de pureza asociado a cada píxel. A medida que el número de *skewers* crece, también crece la precisión del algoritmo a la hora de identificar los píxeles extremos (se puede decir que el algoritmo converge asintóticamente). Una vez localizados los píxeles más extremos de la imagen hiperespectral, se aplica a continuación un proceso supervisado (basado en una herramienta denominada visualizador l-dimensional disponible en el software ENVI) para localizar los puntos espectralmente más puros de la imagen de forma supervisada, utilizando un valor umbral que establece cuántas veces ha de ser seleccionado un determinado píxel durante el proceso para ser considerado como *endmember* candidato. Por este motivo, el algoritmo PPI se ha utilizado también ampliamente en tareas de preprocesado, con objeto de eliminar aquellos píxeles no suficientemente puros con carácter previo a la aplicación de otro algoritmo automático

de extracción de *endmembers* capaz de proporcionar las firmas espectrales finales de los *endmembers* de forma automática como es el caso de N-FINDR.

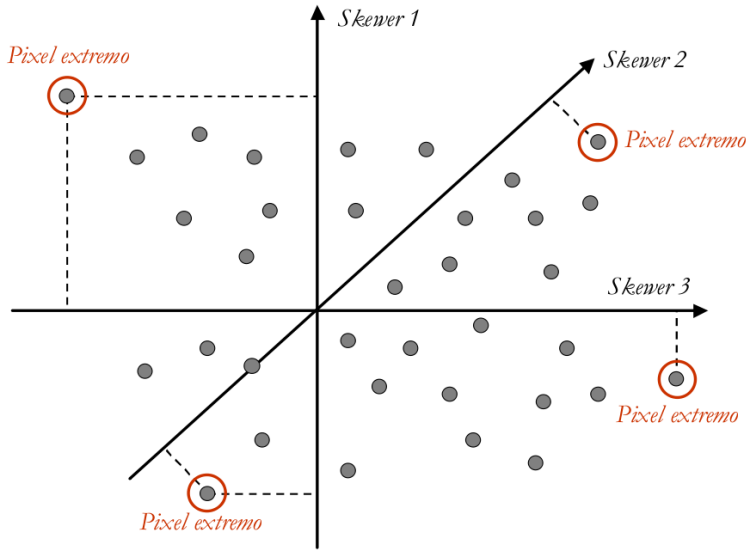

Figura 2.11: Representación gráfica del algoritmo PPI en un espacio de dos dimensiones.

A continuación se describe paso a paso el algoritmo PPI. Las entradas del algoritmo son la imagen hiperespectral, $\mathbf{Y}$ y el número de *skewers*, k. La salida del algoritmo es una imagen de pureza en la que cada píxel de la imagen lleva un contador asociado que indica su índice de pureza, es decir, el número de veces que ha sido seleccionado como extremo al proyectar los n píxeles de la imagen sobre los k *skewers* generados de forma aleatoria. A partir de dicha imagen de pureza se puede aplicar un umbral t_v para retener únicamente los píxeles más puros de la imagen, los cuales pueden ser utilizados como candidatos por otro algoritmo automático de extracción de *endmembers* o bien servir como entrada a la herramienta de visualización l-dimensional de ENVI para seleccionar un conjunto final de *endmembers* de manera supervisada. A continuación describimos los pasos que el algoritmo aplica para obtener la imagen de pureza:

1. *Generación de skewers.* Producir un conjunto de k vectores aleatorios *skewers* $\{\mathbf{skewer}_j\}_{j=1}^{k}$ que serán utilizados para identificar los píxeles extremos de la nube de puntos dada por los píxeles (vectores l-dimensionales) de la imagen hiperespectral original $\mathbf{Y}$.

2. *Proyecciones de skewers.* Para cada $\mathbf{skewer}_j$, todos los píxeles $\{\mathbf{y}_i\}_{i=1}^{n}$ de la imagen original $\mathbf{Y}$ se proyectan sobre el $\mathbf{skewer}_j$ calculando el producto escalar $|\mathbf{y}_i \cdot \mathbf{skewer}_j|$ para cada combinación píxel-*skewer*. En cada proyección, se calculan los píxeles extremos (máximo y mínimo) y se forma un conjunto de extremos para el $\mathbf{skewer}_j$ que se denota como $S_{extrema}(\mathbf{skewer}_j)$. A pesar del hecho de que diferentes $\mathbf{skewer}_j$ pueden generar diferentes $S_{extrema}(\mathbf{skewer}_j)$, es muy probable que los mismos píxeles extremos aparezcan en diferentes conjuntos

$S_{extrema}(\mathbf{skewer}_j)$. Para tener en cuenta esta situación, definimos la siguiente función $I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{y}_i)$ para denotar la pertenencia de un píxel $\mathbf{y}_i$ de la imagen hiperespectral $\mathbf{Y}$ al conjunto de píxeles extremos seleccionados utilizando un determinado $\mathbf{skewer}_j$, lo cual denotamos como $S_{extrema}(\mathbf{skewer}_j)$, de la siguiente manera:

$$I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{y}_i) = \begin{Bmatrix} 1 & \text{si} & \mathbf{y}_i \in S_{extrema}(\mathbf{skewer}_j) \\ 0 & \text{si} & \mathbf{y}_i \notin S_{extrema}(\mathbf{skewer}_j) \end{Bmatrix} \tag{2.7}$$

3. Cálculo de la imagen PPI. Utilizando la función definida en la expresión (2.7), calculamos el índice de pureza del píxel $\mathbf{y}_i$ (es decir, el número de veces que ha sido seleccionado como extremo en el paso anterior) mediante la siguiente expresión:

$$N_{PPI}(\mathbf{y}_i) = \sum_{j=1}^{k} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{y}_i) \tag{2.8}$$

Opcionalmente, a continuación se puede aplicar una fase de selección de candidatos en la que los píxeles $\{\mathbf{y}_i\}_{i=1}^n$ cuyo valor de $N_{PPI}(\mathbf{y}_i)$ esté por debajo de un valor umbral t_v son eliminados como candidatos. Llegados a este punto, se puede utilizar un proceso interactivo para seleccionar los *endmembers* finales a partir de dichos candidatos, o bien puede aplicarse otro algoritmo de extracción de *endmembers* al conjunto de candidatos para identificar un conjunto de p *endmembers*. Hoy en día el algoritmo PPI se utiliza principalmente para seleccionar un conjunto de *endmembers* candidatos, pero siempre es preciso aplicar un proceso posterior de selección de *endmembers*, ya sea supervisado o no. A continuación describimos un método automático capaz de identificar p endmembers a partir del conjunto de píxeles de la imagen hiperespectral original $\mathbf{Y}$.

2.3.3.2. N-FINDR

El algoritmo N-FINDR [105, 104] utiliza una técnica basada identificar los *endmembers* como los vértices del *simplex* de mayor volumen que puede formarse con el conjunto de píxeles de la imagen hiperespectral $\mathbf{Y}$. Dado que la estimación del volumen se realiza a partir del cálculo del determinante de la matriz cuadrada formada por los *endmembers* cuyo volumen deseamos estimar, y teniendo en cuenta que en una imagen hiperespectral real el número de *endmembers* suele ser inferior al número de bandas, es decir $p << l$, el método N-FINDR no trabaja con toda la imagen hiperespectral $\mathbf{Y}$ sino que aplica un proceso de reducción dimensional previo mediante la técnica PCA anteriormente descrita o bien mediante la técnica MNF. Teniendo en cuenta esta consideración, el único parámetro de entrada del algoritmo N-FINDR, aparte de la imagen hiperespectral $\mathbf{Y}$ a procesar, es el número de *endmembers* que se desean identificar, p. A continuación se describe el algoritmo N-FINDR paso a paso.

1. *Reducción dimensional.* Aplicar un proceso de reducción dimensional (por ejemplo, la transformación PCA) para reducir la dimensionalidad de la imagen hiperespectral original $\mathbf{Y}$ de l a $p-1$, siendo p el número de *endmembers* a extraer por el algoritmo.

2. *Inicialización.* Sea $\mathbf{M}^{(0)} = \{\mathbf{m}_1^{(0)}, \mathbf{m}_2^{(0)}, \cdots, \mathbf{m}_p^{(0)}\}$ un conjunto de *endmembers* inicial obtenidos mediante un proceso de selección aleatoria de p píxeles de la imagen hiperespectral original $\mathbf{Y}$.

3. *Cálculo de volumen.* En la iteración $k \geq 0$, el algoritmo N-FINDR calcula el volumen definido por el conjunto actual de *endmembers* de la siguiente forma:

$$V(\mathbf{m}_1^{(k)}, \mathbf{m}_2^{(k)}, \cdots, \mathbf{m}_p^{(k)}) = \frac{\left| det \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{m}_1^{(k)} & \mathbf{m}_2^{(k)} & \cdots & \mathbf{m}_p^{(k)} \end{bmatrix} \right|}{(p-1)!} \tag{2.9}$$

4. *Reemplazamiento.* Para cada píxel $\mathbf{y}_i$ de la imagen hiperespectral $\mathbf{Y}$, recalculamos el volumen probando el píxel en todas las p posiciones correspondientes a los diferentes *endmembers*, es decir, primero calculamos $V(\mathbf{y}_i, \mathbf{m}_2^{(k)}, \cdots, \mathbf{m}_p^{(k)})$, luego calculamos $V(\mathbf{m}_1^{(k)}, \mathbf{y}_i, \cdots, \mathbf{m}_p^{(k)})$, y así sucesivamente, hasta calcular finalmente $V(\mathbf{m}_1^{(k)}, \mathbf{m}_2^{(k)}, \cdots, \mathbf{y}_i)$. Si ninguno de los p volúmenes recalculados es mayor que el volumen original $V(\mathbf{m}_1^{(k)}, \mathbf{m}_2^{(k)}, \cdots, \mathbf{m}_p^{(k)})$, entonces no se reemplaza ningún *endmember*. En caso contrario, se retiene la combinación con mayor volumen. Supongamos que el *endmember* que falta en la combinación que resulta en mayor volumen se denota como $\mathbf{m}_j^{(k+1)}$. En este caso, se obtiene un nuevo conjunto de *endmembers* haciendo que $\mathbf{m}_j^{(k+1)} = \mathbf{y}_i$ y $\mathbf{m}_i^{(k+1)} = \mathbf{m}_i^{(k)}$ para $i \neq j$. El proceso de reemplazamiento se repite de forma iterativa con todos los píxeles de la imagen original, realizando todas las iteraciones que sean necesarias, hasta que no se produzcan más reemplazamientos. En este caso, la combinación final da lugar al conjunto de endmembers definitivo $\mathbf{M} = \{\mathbf{m}_i\}_{i=1}^p$.

La Fig. 2.12 muestra una representación gráfica del método N-FINDR donde puede apreciarse cómo, partiendo de un conjunto de píxeles aleatorios, el volumen del *simplex* va aumentando en las sucesivas iteraciones del algoritmo hasta llegar a una situación en la que los píxeles que forman los vértices del *simplex* son los *endmembers* finales determinados por el algoritmo.

2.3.4. Estimación de abundancias

La última etapa aplicada por las técnicas de desmezclado basadas en el modelo lineal de mezcla es la estimación de las abundancias, es decir, la estimación de la proporción de cada uno de los *endmembers* identificados, $\mathbf{M} = \{\mathbf{m}_i\}_{i=1}^p$, en cada píxel de la imagen hiperespectral $\mathbf{Y}$. Como resultado de este proceso, se obtiene un mapa de abundancia asociado con cada *endmember*, por lo que tenemos tantos mapas de abundancias como *endmembers* extraídos, es decir, p. Como se indicó en la ecuación (2.2), en la estimación de abundancias se pueden aplicar las restricciones de no negatividad y suma unitaria. En la literatura reciente [35], han surgido críticas a la condición de suma unitaria dado que la identificación de un conjunto adecuado de *endmembers* debería satisfacer dicha condición de forma automática. Por su parte, la condición de no negatividad se ha venido aplicando de forma natural para evitar la estimación de abundancias

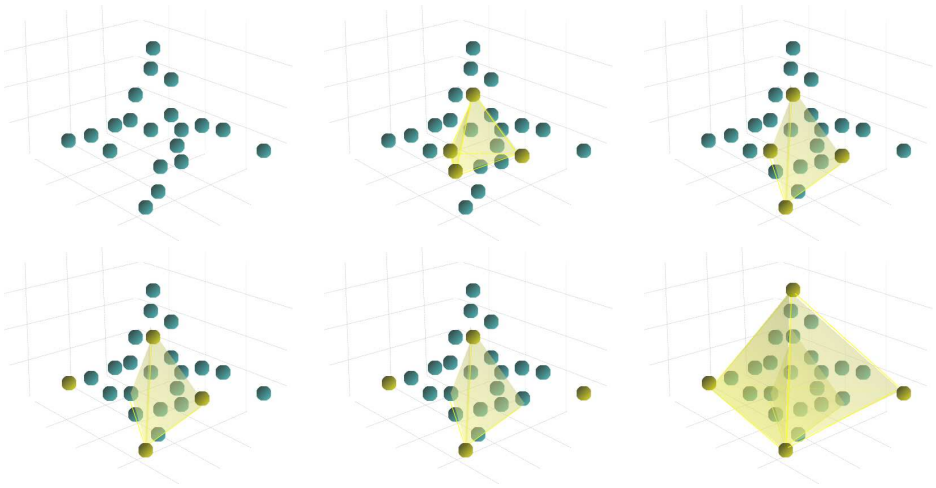

Figura 2.12: Representación gráfica del funcionamiento del algoritmo N-FINDR en un ejemplo sencillo en tres dimensiones.

negativas y, por tanto, sin sentido físico. Aunque existen métodos que imponen ambas restricciones en el proceso de estimación de abundancias, tales como *fully constrained least squares* (FCLS) [33], dichos métodos son computacionalmente costosos y a veces llevan a soluciones erróneas en el caso de que los *endmembers* no hayan sido identificados de forma correcta. Por estos motivos, en la actualidad las soluciones más populares para la estimación de abundancias se basan en procesos en los que no se imponen ninguna de las dos restricciones anteriormente comentadas, o bien en los que se impone únicamente la condición de no negatividad. A continuación presentamos dos soluciones ampliamente utilizadas en la literatura: el método *unconstrained least squares* (UCLS), que no impone ninguna de las dos restricciones, y el método *non-negative constrained least squares* (NCLS), que únicamente impone la restricción de no negatividad en el proceso de estimación de abundancias. El apartado finaliza con una descripción del método *iterative error analysis* (IEA), que integra las etapas de identificación de *endmembers* y estimación de abundancias (sin restricciones) ya que utiliza el resultado de la etapa de desmezclado para identificar nuevos *endmembers*, realizando de este modo las dos etapas (identificación de *endmembers* y estimación de abundancias) de forma conjunta.

2.3.4.1. *Unconstrained least squares* (UCLS)

El problema de estimación de abundancias puede verse como un problema de optimización, en el que se pretende minimizar el error en la reconstrucción de la imagen hiperespectral original $\mathbf{Y}$ utilizando el conjunto de *endmembers* identificados $\mathbf{M} = \{\mathbf{m}_i\}_{i=1}^{p}$ y el modelo lineal de mezcla. El objetivo es encontrar el vector de abundancias $\mathbf{x}_i$ que mejor reconstruye cada píxel $\mathbf{y}_i$ de la imagen, siendo $\mathbf{x}_i$ un vector p-dimensional (fracciones de abundancia de los p *endmembers* en el píxel) e $\mathbf{y}_i$ un vector l-dimensional. Para ello, normalmente se utiliza la distancia de mínimos cuadrados entre el píxel $\mathbf{y}_i$ y su versión reconstruida mediante el modelo lineal del mezcla, $\hat{\mathbf{y}}_i$ de la siguiente

forma:

$$\mathrm{LS}(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \|\mathbf{y}_i - \mathbf{M}\mathbf{x}_i\|_2^2 \tag{2.10}$$

Para resolver la expresión (2.10), el método UCLS utiliza la siguiente expresión:

$$\hat{\mathbf{x}}_i = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{m}^T\mathbf{y}_i, \tag{2.11}$$

donde $\mathbf{M}$ es la matriz de *endmembers*. Al no aplicar ninguna restricción, este método es rápido y sencillo de implementar, pero puede dar lugar a abundancias negativas sin sentido físico. Uno de los métodos más populares en la literatura para resolver este problema es NCLS, descrito en el siguiente subapartado.

2.3.4.2. *Non-negative constrained least squares* (NCLS)

El algoritmo NCLS [103] es un método de estimación de abundancias que satisface la restricción de no negatividad, es decir, todas las abundancias estimadas por el algoritmo son mayores o iguales a cero. El algoritmo recibe como parámetro de entrada la imagen hiperespectral original $\mathbf{Y}$, un valor umbral de error ε y un valor de parada ρ. Los pasos que aplica el algoritmo pueden resumirse en las siguientes tres etapas:

1. *Inicialización*. En primer lugar, inicializamos un vector de abundancias estimado $\hat{\mathbf{x}}_i$ para cada píxel $\mathbf{y}_i$ de la imagen hiperespectral $\mathbf{Y}$. Es importante utilizar valores positivos en dicho proceso de inicialización, ya que de esta forma se mantendrán como valores positivos durante el proceso.

2. *Proceso iterativo*. En este proceso tiene lugar el cálculo de las abundancias. El proceso iterativo se efectúa mientras que el error en la estimación sea superior a ε, o bien mientras que se cumpla la condición de parada. Para calcular el error en la estimación de abundancias en el píxel $\mathbf{y}_i$, en primer lugar estimamos la abundancia en la iteración actual, $k+1$, utilizando la siguiente expresión:

$$\hat{\mathbf{x}}_i^{k+1} = \hat{\mathbf{x}}_i^k \left(\frac{\mathbf{M}^T \cdot \mathbf{y}_i}{\mathbf{M}^T\mathbf{M} \cdot \hat{\mathbf{x}}_i^k} \right) \tag{2.12}$$

   Como puede apreciarse en la expresión (2.12), la estimación de la abundancia en la iteración $k+1$ depende de la estimación de la abundancia en la iteración k. El error en una determinada iteración $k+1$ se calcula mediante la expresión (2.10) de la siguiente forma: $\left\|\mathbf{y}_i - \mathbf{M}\mathbf{x}_i^{k+1}\right\|_2^2$. Si dicho error es superior a ε el proceso iterativo finaliza. En caso contrario dicho proceso continúa hasta que se cumpla la condición de parada, que viene dada por la siguiente regla:

$$\frac{\left\|\hat{\mathbf{x}}^{k+1} - \hat{\mathbf{x}}^k\right\|}{\left\|\hat{\mathbf{x}}^k\right\|} \leq \rho \tag{2.13}$$

Como puede apreciarse en la descripción algorítmica anteriormente mostrada, el algoritmo NCLS se detiene cuando el error calculado es menor que ε o cuando las

abundancias calculadas en una iteración son muy similares a las calculadas en la iteración anterior (es decir, cuando $\hat{\mathbf{x}}^{k+1}$ es muy similar a $\hat{\mathbf{x}}^{k}$), viniendo el umbral de similaridad dado por ρ. La convergencia del algoritmo NCLS ha sido demostrada en [103]. La principal ventaja de este algoritmo es que resulta fácil de implementar (y de paralelizar) debido a su carácter regular e iterativo. Como principal desventaja tenemos que la complejidad computacional es bastante elevada. Por este motivo, la versión paralela desarrollada en el presente trabajo pretende acelerar este algoritmo de cara a su explotación a nivel práctico en la comunidad de análisis hiperespectral.

2.3.4.3. *Iterative error analysis* (IEA)

Para terminar esta sección dedicada a la presentación de métodos clásicos de estimación de abundancias en el presente subapartado describimos el algoritmo IEA [52], que utiliza el proceso de estimación de abudancias para identificar, de forma simultánea, los *endmembers* de la imagen hiperespectral. En otras palabras, el algoritmo IEA integra las fases de identificación de *endmembers* y estimación de abundancias. El algoritmo utiliza como parámetro de entrada la imagen hiperespectral $\mathbf{Y}$ y el número de *endmembers*, p. A continuación realiza un proceso iterativo que puede resumirse en los siguientes pasos:

1. *Inicialización.* En primer lugar, el algoritmo calcula el vector promedio $\bar{\mathbf{Y}}$ de la imagen hiperespectral original $\mathbf{Y}$ mediante la siguiente expresión:

$$\bar{\mathbf{Y}} = \frac{1}{n \times l} \sum_{i=1}^{n} \mathbf{y}_i, \tag{2.14}$$

   donde n denota el número de píxeles de la imagen hiperespectral

2. *Cálculo del endmember inicial.* El primer *endmember* $\mathbf{m}_1$ se calcula de la siguiente manera: primero se obtiene una versión reconstruida $\hat{\mathbf{Y}}$ de la imagen hiperespectral original $\mathbf{Y}$ realizando el desmezclado de $\mathbf{Y}$ usando $\bar{\mathbf{Y}}$ como único endmember. El método de desmezclado que el algoritmo IEA utiliza es UCLS. El resultado de esta operación es una estimación de abundancias $\mathbf{x}_i$ para cada píxel $\mathbf{y}_i$ en $\mathbf{Y}$. A continuación, se aplica la expresión (2.10) para aproximar cada píxel $\mathbf{y}_i$ mediante la combinación lineal $\hat{y}_i = \mathbf{M}\mathbf{x}_i$. Este proceso se repite para todos los píxeles de la imagen, seleccionando el píxel con mayor error según la expresión (2.10) como el primer *endmember* $\mathbf{m}_1$ incluyéndolo en el conjunto de *endmembers* $\mathbf{M} = \{\mathbf{m}_1\}$.

3. *Proceso iterativo.* Durante este proceso se calcula un nuevo *endmember* en cada iteración $2 \leq k \leq p$ realizando el desmezclado de cada píxel $\mathbf{y}_i$ y utilizando el conjunto de endmembers actual $\mathbf{M}$. El resultado de esta operación es, de nuevo, una estimación de abundancias $\mathbf{x}_i$ para cada píxel $\mathbf{y}_i$, donde $k \leq p$ es el número de *endmembers* extraídos hasta el momento. Ahora podemos de nuevo seleccionar el píxel con mayor error de reconstrucción asociado como el k-ésimo **endmember**, $\mathbf{m}_k$, e incluirlo en el conjunto de *endmembers* $\mathbf{M} = \{\mathbf{m}_1 \dots \mathbf{m}_k\}$. El proceso iterativo finaliza cuando $k = p$. En este caso, se obtiene como salida del algoritmo

el conjunto final de endmembers $\mathbf{M} = \{\mathbf{m}_1 \dots \mathbf{m}_p\}$ junto con sus correspondientes abundancias para cada píxel de la imagen hiperespectral $\mathbf{Y}$, es decir, el algoritmo IEA proporciona no solamente los *endmembers* sino también los correspondientes mapas de abundancia, integrando los pasos de identificación de *endmembers* y estimación de abundancias en un mismo algoritmo.

Para concluir este apartado, la Fig. 2.13 muestra la relación entre las diferentes técnicas descritas en el presente apartado. Como podemos ver en la figura, se pueden formar diferentes cadenas de procesamiento combinando los métodos descritos para estimación del número de *endmembers* (VD y HySime), reducción dimensional (PCA y SPCA), identificación de *endmembers* (N-FINDR) y estimación de abundancias (UCLS e NCLS). Con estos métodos, se puede formar un total de 8 posibles cadenas de desmezclado no supervisadas que toman como único parámetro de entrada la imagen hiperespectral $\mathbf{Y}$, produciendo como resultado sus correspondientes *endmembers* y mapas de abundancia. Conviene destacar que la Fig. 2.13 no incluye el método PPI, que suele utilizarse como un pre-procesado espectral para eliminar los píxeles no suficientemente puros antes de iniciar el proceso, ni tampoco el método IEA que integra las fases de identificación de *endmembers* y estimación de abundancias en una sola etapa.

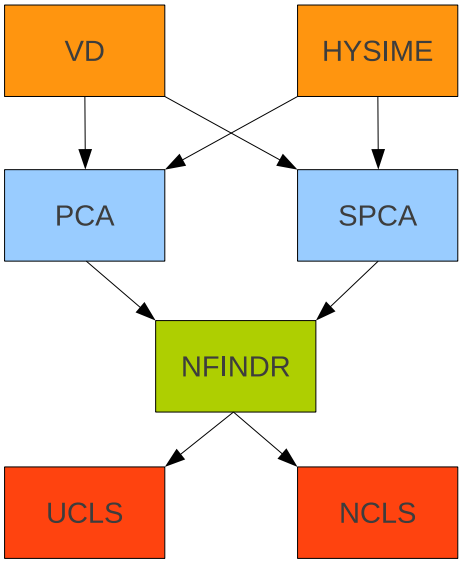

Figura 2.13: Combinación de los métodos descritos en el presente apartado para formar cadenas completas (no supervisadas) de desmezclado espectral lineal de datos hiperespectrales.

2.4. Necesidad de paralelismo en análisis hiperespectral

El proceso de captación de datos hiperespectrales por parte de sensores remotos de observación de la tierra produce cantidades enormes de datos de gran dimensionalidad

que deben ser almacenados y tratados de forma eficiente. La falta de arquitecturas consolidadas para el tratamiento eficiente de imágenes hiperespectrales ha dado lugar a una situación actual en la que, a pesar de que diferentes instituciones como NASA o la agencia Europea del espacio (ESA) obtienen varios Terabytes de datos hiperespectrales cada día, se estima que una parte significativa de dichos datos no son nunca utilizados o procesados, sino meramente almacenados en una base de datos y muchas veces reservados para futuro uso que, en ocasiones, no se materializa debido a los enormes requerimientos computacionales para almacenar, procesar y distribuir dichas imágenes de forma eficiente.

Con el auge de las arquitecturas de computación paralela de bajo coste en problemas de cálculo científico, en la actualidad existe una comunidad investigadora muy activa orientada al desarrollo de algoritmos para el procesamiento eficiente de datos hiperespectrales. Desde un punto de vista computacional, estos algoritmos presentan paralelismo en múltiples niveles: a través de los vectores que constituyen los píxeles (paralelismo de grano grueso a nivel de píxel), a través de la información espectral (paralelismo de grano fino a nivel espectral), e incluso a nivel de tareas ejecutadas por diferentes algoritmos como las técnicas extracción de endmembers y desmezclado que constituyen el principal interés de la presente memoria. Teniendo en cuenta que los accesos a datos en los algoritmos de análisis hiperespectral son normalmente regulares y altamente predecibles, en la literatura reciente se ha demostrado que muchos de estos algoritmos se pueden mapear fácilmente sobre sistemas paralelos tales como *clusters* de computadores, sistemas heterogéneos y dispositivos hardware especializados para procesamiento a bordo.

A continuación, describimos brevemente los avances más recientes en cuanto al procesamiento eficiente de imágenes hiperespectrales utilizando arquitecturas de computación de altas prestaciones. Llegados a este punto, es preciso destacar que las técnicas de procesamiento a bordo son altamente deseables en aplicaciones que requieren una respuesta en tiempo real, tales como aquellas orientadas a la detección de agentes contaminantes en aguas y atmósfera, detección y seguimiento de incendios forestales, etc. [61]. En este sentido, las técnicas de computación paralela han sido ampliamente utilizadas en la literatura reciente para llevar a cabo tareas de procesamiento de imágenes hiperespectrales, facilitando la obtención de tiempos de respuesta muy reducidos en dichas aplicaciones mediante arquitecturas de bajo coste [73].

Siguiendo la evolución histórica en las investigaciones llevadas a cabo en este campo, en la presente sección ofrecemos una visión del estado del arte en cuanto a la aplicación de técnicas de computación de altas prestaciones aplicada a problemas relacionados con el análisis hiperespectral, comenzando con los primeros avances centrados en computación en *clusters* de computadores personales, pasando por nuevos desarrollos basados en computación paralela heterogénea, y concluyendo con avances recientes en hardware especializado para el procesamiento de imágenes hiperespectrales a bordo de sensores de observación remota [66].

2.4.1. Computación *cluster* aplicada a análisis hiperespectral

La idea de aplicar *clusters* de computadores personales [7] para satisfacer los requerimientos computacionales en aplicaciones de análisis hiperespectral surgió a principios de la década de los 90, y originalmente se aplicó a problemas de procesamiento de imágenes multiespectrales con un número limitado (decenas) de bandas [42]. A medida que los sensores de observación remota de la tierra fueron incrementando su resolución espectral, la computación *cluster* aplicada a este tipo de problemas se centró en el procesamiento de imágenes hiperespectrales de mayor dimensionalidad [17].

El primer intento documentado de aplicar técnicas de computación *cluster* a problemas de análisis hiperespectral data de mediados de los 90, periodo en el que un equipo de investigadores del centro *Goddard Space Flight Center* (GSFC) de la NASA diseñó el primer *cluster* específicamente orientado a resolver problemas relacionados con la observación remota de la tierra. Dicho *cluster* consistía en 16 PCs idénticos, cuyas CPUs poseían una frecuencia de reloj de 100MHz, interconectados mediante 2 hub Ethernet configurados en modo *bonding*, de forma que las dos redes de comunicación existentes funcionaban como una sola red al doble de velocidad. Con la evolución de la aplicaciones de procesamiento de datos hiperespectrales, en 1996 el *California Institute of Technology* (CalTech) diseñó un nuevo *cluster* de tipo Beowulf orientado al procesamiento de datos hiperespectrales, demostrando un rendimiento de un Gigaflop en una aplicación de análisis hiperespectral, siendo la primera vez que un *cluster* de estas características mostraba su potencial en este contexto.

De forma general, conviene destacar que hasta 1997 los *clusters* utilizados en aplicaciones de observación remota de la tierra eran esencialmente prototipos de ingeniería, es decir, los *clusters* eran generalmente construidos por los mismos investigadores que iban a hacer uso de ellos. Sin embargo, a finales de 1997 se inició un nuevo proyecto en el centro GSFC para construir un *cluster* ideado para ser explotado por una comunidad de investigadores entre los que no necesariamente se encontraban los diseñadores del mismo. Este proyecto se denominó *Highly Parallel Virtual Environment* (HIVE) [21] y su filosofía consistió en disponer de estaciones de trabajo distribuidas en diferentes lugares y un gran número de nodos de cómputo (el núcleo) concentrados en un único espacio. Aunque el sistema HIVE original sólo tenía una estación de trabajo, varios usuarios podían acceder a él desde sus propias estaciones de trabajo a través de internet. HIVE fue el primer *cluster* que superó un rendimiento sostenido de 10 Gigaflops procesando datos hiperespectrales (entre otras aplicaciones). Más adelante, una evolución de HIVE se utilizó de forma específica en aplicaciones de análisis hiperespectral. Este sistema, conocido como Thunderhead, es un *cluster* homogéneo con 512 procesadores formado por 256 nodos duales Intel Xeon a 2.4GHz, cada uno con 1 GByte de memoria y 80 GBytes de disco. El rendimiento pico de Thunderhead es de 2457.6 Gigaflops. Además de los 512 procesadores, Thunderhead cuenta con otros nodos conectados mediante una red Myrinet. Este *cluster* ha sido utilizado en multitud de estudios de análisis de imágenes hiperespectrales durante los últimos años [60, 73, 62, 102, 65, 74].

Además de los sistemas anteriormente mencionados, NASA y ESA poseen actualmente otras plataformas específicamente dedicadas al tratamiento eficiente de

Cluster Columbia

Cluster Discover

Figura 2.14: Ejemplos de *clusters* de computadores utilizados en análisis hiperespectral.

imágenes hiperespectrales. Un ejemplo de ello es el supercomputador Columbia[2] ubicado en el *Ames Research Center* de la NASA. Está compuesto por 10.240 CPUs SGI Altix Intel Itanium 2, 20 TBytes de memoria e interconexiones heterogéneas de tipo Infiniband y Gigabit Ethernet. Otro ejemplo es el sistema Discover del centro GSFC, con un total de 15.000 procesadores y red de comunicación de tipo Infiniband, dedicado a la predicción de los cambios climáticos a corto, medio y largo plazo mediante el uso de imágenes hiperespectrales. A modo ilustrativo, la Fig. 2.14 pretende mostrar el aspecto físico de estos dos *clusters* (Columbia y Discover) utilizados en análisis hiperespectral. Fuera del entorno de NASA y ESA, otro supercomputador que ha sido utilizado esporádicamente para el tratamiento de imágenes hiperespectrales es Marenostrum [74], ubicado en el *Barcelona Supercomputing Center*. Por último, el *High Performance Computing Collaboratory* (HPC^2) en la *Mississippi State University* posee varias instalaciones de supercomputación que se han utilizado en estudios realacionados con análisis de imágenes hiperespectrales. Entre ellos destaca el sistema Maverick[3], compuesto por 192×335 nodos IBM donde cada nodo contiene un procesador dual Xeon a 3.06 GHz y 2.5 GBytes de RAM. Otro sistema ampliamente utilizado en aplicaciones de análisis hiperespectral disponible en HPC^2 es Raptor, que comenzó a utilizarse en 2006, y que consta de 2048 núcleos de procesamiento, con 512 servidores Sunfire X2200 M2 de Sun Microsystems, cada uno de los cuales posee dos procesadores dual-core AMD Opteron 2218 a 2.6 GHz y 8 GB de memoria. Recientemente, el sistema Talon de HPC^2 ha comenzado a utilizarse de forma activa en aplicaciones de análisis hiperespectral. Dicho sistema consta de 3072 núcleos de proceamiento, con 256 nodos iDataPlex de IBM cada uno con dos procesadores Intel Westmere de 6 cores a 2.8 GHz y 24 GB de memoria.

2.4.2. Computación heterogénea en análisis hiperespectral

Con la evolución en el hardware de red e interconexión, las redes de computación heterogéneas capaces de interconectar recursos de cómputo ubicados en localizaciones geográficas dispersas se convirtieron en una herramienta muy popular para la computación distribuida, y pronto este paradigma comenzó a aplicarse en problemas

[2]http://www.nas.nasa.gov/Resources/Systems/columbia.html
[3]http://www.hpc.msstate.edu/computing/maverick

relacionados con análisis de datos de la superficie terrestre en general, y con análisis de datos hiperespectrales en particular [100]. Al contrario que en la computación *cluster*, en la que el número y la ubicación de los nodos de cómputo es conocida *a priori*, en la computación heterogénea distribuida no siempre es posible garantizar la disponibilidad de los recursos de antemano, lo cual ha motivado cambios en las aplicaciones de procesamiento de datos hiperespectrales que utilizan este paradigma. Una evolución del concepto de computación distribuida anteriormente descrito es el concepto de computación Grid, en el que el número y ubicación de los nodos es relativamente dinámico y se conoce únicamente en tiempo de ejecución. La Fig. 2.15 muestra un esquema del concepto de computación Grid, donde un gran número de dispositivos, que pueden ser de diferentes tipos, cooperan en la computación de una tarea sin tener en cuenta su ubicación y utilizando internet para la comunicación. Este concepto se ha aplicado (aunque todavía no de manera muy consolidada) en el contexto de análisis de imágenes hiperespectrales [77]. Por ejemplo. existen estudios orientados a facilitar la accesibilidad (con tolerancia a fallos gracias al diseño especializado del *middleware*), la distribución eficiente y la compartición entre usuarios geográficamente dispersos de grandes colecciones de imágenes hiperespectrales [6, 69].

Figura 2.15: Esquema de computación Grid.

En la literatura reciente existen otros trabajos de investigación orientados al procesamiento distribuido eficiente de imágenes hiperespectrales. Tal vez el ejemplo más simple es el uso de versiones heterogéneas de algoritmos paralelos de procesamiento de datos desarrollados para *clusters* homogéneos, por ejemplo, introduciendo variaciones que doten a dichos algoritmos de la capacidad de modelar la heterogeneidad inherente en una red distribuida, balanceando la carga de forma eficiente contemplando la heterogeneidad de los recursos disponibles [69, 71, 44, 50]. Este marco de trabajo permite portar fácilmente un código paralelo desarrollado para un sistema homogéneo a entornos completamente heterogéneos. Otra propuesta ampliamente utilizada en la literatura ha consistido en la adaptación de algoritmos diseñados para ejecutarse en un único *cluster* a entornos distribuidos *multi-cluster*, formados por varios *clusters* homogéneos interconectados entre sí a través de una red de tipo *wide area network* (WAN) [47]. Por otra parte, algunas aplicaciones han explorado la posibilidad de utilizar componentes *software* para construir algoritmos eficientes para el procesamiento

de imágenes hiperespectrales. Por ejemplo, existen desarrollos orientados a realizar el proceso de corrección atmosférica de la imagen hiperespectral utilizando componentes [6]. Dichos componentes ofrecen la posibilidad de encapsular gran parte de la complejidad existente en las tareas de procesamiento al tratar los algoritmos como si fuesen *cajas negras*, mostrando únicamente la interfaz de cada componente (por ejemplo, los parámetros de entrada y salida de cada módulo) al resto de componentes. La programación orientada a componentes puede ofrecer nuevas perspectivas en cuanto a compartición eficiente de algoritmos de análisis hiperespectral.

2.4.3. Hardware especializado para análisis hiperespectral

Durante los últimos años se han realizado numerosos esfuerzos para incorporar el procesamiento de imágenes hiperespectrales a bordo del sensor, dados los requerimientos computacionales de determinadas aplicaciones críticas que requieren una respuesta en tiempo real [59, 89, 85, 83]. Un buen ejemplo es el seguimiento de un incendio, que evoluciona de forma muy rápida y que requiere una velocidad de procesamiento de datos equiparable a la velocidad del proceso de adquisición de los mismos, permitiendo procesar los datos a medida que se adquieren por parte del sensor a bordo. La posibilidad de procesar datos hiperespectrales a bordo también introduce numerosas ventajas desde el punto de vista de reducir los requerimientos de ancho de banda del enlace de comunicación con la estación de procesamiento de datos en tierra, tanto para un eventual acondicionamiento y pre-procesado de la imagen, como para la realización de una selección inteligente de los datos hiperespectrales que se van a transmitir a tierra (por ejemplo, seleccionando las bandas espectrales más relevantes en un contexto de aplicación determinado) [19]. En particular, la compresión de datos hiperespectrales a bordo resulta de gran interés, ya sea sin pérdida o bien con pérdida selectiva de información [18, 91, 92, 70]. Por otra parte, algunas misiones de análisis hiperespectral se encuentran orientadas a la exploración planetaria (por ejemplo, de la superficie de Marte [53]), y en este caso la toma de decisiones automática a bordo resulta indispensable con vistas a evitar comunicaciones que pueden resultar lentas y muy costosas.

Llegados a este punto, conviene destacar que las arquitecturas tipo *cluster* y sistemas distribuidos no resultan adecuadas para este tipo de procesamiento a bordo. Los *clusters* son, en general, caros y difíciles de adaptar a escenarios de procesamiento a bordo, en los que los componentes de peso reducido y bajo consumo son muy importantes para reducir el peso (*payload*) del sensor y de la misión en general, así como el coste de la misma.

En la literatura reciente han aparecido numerosos trabajos orientados a la implementación de algoritmos de análisis hiperespectral en dispositivos hardware como *field programmable gate arrays* (FPGAs) y GPUs. Por una parte, las FPGAs ofrecen atractivas características tales como reconfigurabilidad y bajo consumo energético [26, 27], mientras que las GPUs ofrecen gran rendimiento computacional a bajo coste [93, 80, 90, 88, 87]. En el caso de las GPUs, la evolución de estos dispositivos ha venido motivada por las cada vez más exigentes demandas de la industria del video-juego, con la posibilidad de disponer de sistemas programables altamente paralelos a un coste relativamente bajo. Aunque sus capacidades en cuanto a programación todavía plantean

FPGA GPU

Figura 2.16: Aspecto físico de una FPGA y una GPU.

algunos retos, las GPUs actuales son suficientemente generales como para permitir la realización de desarrollos computacionales de propósito general más allá del dominio de la representación de gráficos y del mundo de los video-juegos. Este enfoque se denomina *general purpose* GPU (GPGPU).

La disponibilidad de hardware especializado susceptible de ser utilizado para procesamiento a bordo en misiones de adquisición de datos hiperespectrales limita de forma significativa la explotación de *clusters* y sistemas distribuidos en procesamiento a bordo [59]. Por otra parte, la implantación de un sistema a gran escala requiere una inversión importante en cuanto a mantenimiento, acondicionamiento, etc. Aunque los *clusters* se consideran arquitectura paralelas de bajo coste, el mantenimiento de un sistema de estas características se hace más complejo a medida que aumenta el número de nodos [73]. En cambio, las FPGAs y las GPUs se caracterizan por su bajo peso y tamaño, así como por la capacidad de ofrecer un rendimiento computacional similar al de un *cluster* en determinados contextos. En particular, estudios recientes demuestran que el incremento de prestaciones obtenido al utilizar FPGAs [62, 101, 20, 26, 27] y GPUs [95, 99, 86, 81, 82] en aplicaciones de procesamiento de imágenes hiperespectrales puede llegar a ser del mismo orden de magnitud que el obtenido por un cluster con un número de nodos relativamente alto. En virtud de su reconfigurabilidad [32], las FPGAs ofrecen además la posibilidad de seleccionar de forma adaptativa el algoritmo de análisis hiperespectral a utilizar (de un conjunto o *pool* de algoritmos disponibles) desde una estación de control. Por otra parte, las FPGAs son generalmente más caras que las GPUs. En este sentido, tanto la reconfigurabilidad de las FPGAs por una parte, como el bajo coste y portabilidad de las GPUs por otra, abren nuevas perspectivas innovadoras para el procesamiento en tiempo real de imágenes hiperespectrales.[66] A modo ilustrativo, la Fig. 2.16 muestra el aspecto físico que tienen una FPGA (a la izquierda) y una GPU (a la derecha).

En cualquier caso, a pesar de sus atractivas propiedades, tanto las FPGAs como GPUs aún no se encuentran plenamente incorporadas en misiones de observación de la Tierra. Por ejemplo, las FPGAs todavía no son totalmente eficientes en situaciones en las que es necesario disponer de varios algoritmos implementados en el mismo hardware, caso en el que la reconfigurabilidad requiere que solamente una pequeña parte de los recursos esté disponible para la lógica, mientras que el resto del hardware se utiliza para

aspectos de interconexión y configuración. Esto supone una penalización en términos de velocidad y energía, lo cual es particularmente complicado en FPGAs tolerantes a radiación (imprescindibles para misiones espaciales y sensores de tipo satélite) las cuales, por norma general, cuentan con un número de puertas varios órdenes de magnitud por debajo del número de puertas disponibles en las FPGAs comerciales de última generación [26]. En cuanto a las GPUs, su alto consumo de energía (comparado con las FPGAs) introduce importantes consideraciones desde el punto de vista del *payload* de la misión, aunque ya existen arquitecturas GPU con menor consumo de energía[4]. En los próximos años se esperan grandes avances en el diseño de GPUs de bajo consumo, incluyendo desarrollos orientados a obtener GPUs tolerantes a radiación de cara a su incorporación a sensores diseñados para adquirir imágenes hiperespectrales desde el espacio.

En la literatura reciente se han llevado a cabo estudios relacionados [57, 66, 84, 75, 93, 80] que permiten contrastar el potencial que ofrecen las arquitecturas especializadas en aplicaciones reales de análisis de imágenes hiperespectrales. Otros estudios se han centrado más en el esfuerzo necesario para desarrollar implementaciones capaces de explotar la potencia computacional de dichas arquitecturas en determinados contextos [67, 68]. Desde el punto de vista de la programación de propósito general, una de las principales ventajas que ofrecen las GPUs modernas es que estas plataformas se pueden abstraer, utilizando un modelo de procesamiento de flujos en el que todos los bloques de datos se consideran conjuntos ordenados de datos, de forma que las aplicaciones se construyen encadenando varios núcleos. Los núcleos operan sobre flujos completos, es decir, tomando uno o más flujos como entrada y produciendo uno o más flujos como salida. Este tipo de procesamiento debe satisfacer una restricción principal: la semántica de los flujos no debe depender del orden en el que se generan los elementos de los flujos de salida [56]. Por tanto, el paralelismo a nivel de datos queda expuesto al hardware y los núcleos pueden operar directamente sobre los flujos, sin ningún tipo de sincronización previa. Las GPUs modernas aprovechan las características anteriormente mencionadas para reemplazar los componentes hardware de control de flujo por unidades de proceso adicionales. En este sentido, los algoritmos de análisis hiperespectral pueden beneficiarse de forma significativa de las prestaciones del hardware y de los modelos de programación basados en GPUs, aprovechando características como la disponibilidad de instrucciones rápidas y precisas para funciones trascendentales, así como el reducido tamaño y el relativo bajo coste de estas unidades.

En el próximo capítulo de la presente memoria ofrecemos una visión general de las arquitecturas GPU modernas, con carácter previo a la presentación de las nuevas implementaciones de las técnicas de análisis hiperespectral descritas en el presente capítulo en este tipo de arquitecturas.

[4] http://www.gputech.com

Capítulo 3

Procesadores Gráficos (GPUs)

En este capítulo se ofrece una visión general de los procesadores gráficos (GPUs) destacando principalmente su arquitectura y las ventajas de su utilización en el análisis hiperespectral. También se dedica una parte de este capítulo a la descripción de la plataforma utilizada para la programación de estos dispositivos, en este caso CUDA de $NVidia^{TM}$.

3.1. Conceptos básicos

Una GPU es un dispositivo hardware (conocido comúnmente como tarjeta gráfica programable) que se compone de un conjunto de multiprocesadores que podrían clasificarse dentro de los tipo SIMD (*single instruction multiple data architecture*) [22]. Cada uno de los multiprocesadores que componen la GPU cuenta con los siguientes elementos de memoria (ver Fig. 3.1):

- Un conjunto de registros por procesador.
- Una memoria compartida o caché paralela, que es compartida por todos los procesadores.
- Una caché de constantes y una caché de texturas, ambas de solo lectura.

Antes de describir brevemente la arquitectura de una GPU, vamos a presentar las diferencias fundamentales entre un computador clásico con unidad central de proceso (CPU) y una GPU. Las GPUs han llegado a convertirse en procesadores masivamente paralelos, multihilo y multinúcleo, que consiguen un gran potencial de procesamiento además de un gran ancho de banda de memoria (ver Figs. 3.2 y 3.3). La Fig. 3.2 muestra cómo la nueva gama de GPUs de $NVidia^{TM}$ consigue picos de más de 1500 Gigaflops por segundo, mientras que la gama de CPUs *SandyBridge* apenas llega a 500 Gigaflops. Es destacable cómo, a medida que va pasando el tiempo, la diferencia entre ambas tecnologías crece de forma exponencial a favor de las GPUs, observándose saltos pronunciados en cuanto a su capacidad de procesamiento. Por otra parte, la Fig. 3.3 muestra cómo el ancho de banda que ofrecen las GPUs se incrementa notablemente con

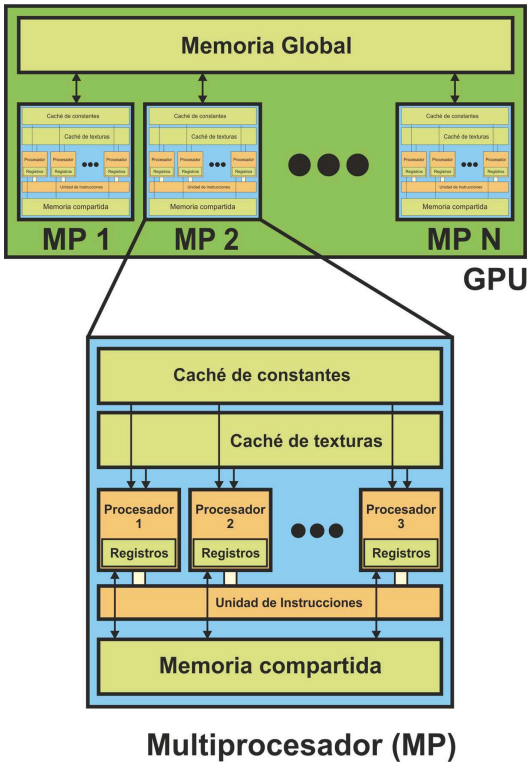

Figura 3.1: Arquitectura hardware de una GPU.

respecto al de las CPUs, progresando de forma exponencial a medida que van apareciendo nuevas generaciones de ambas tecnologías.

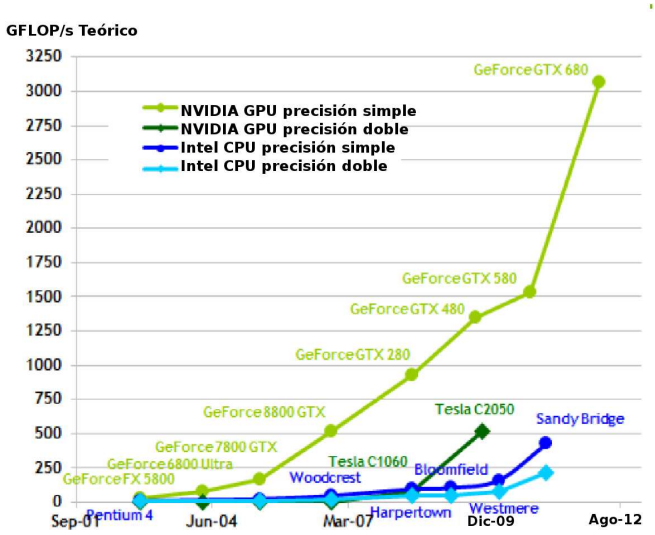

Figura 3.2: Operaciones en coma flotante CPU frente a GPU.

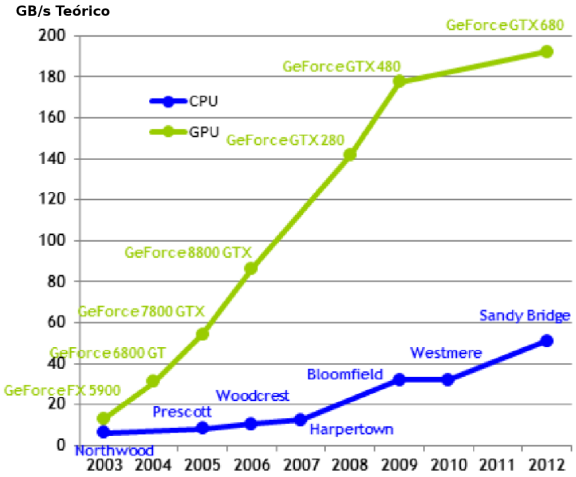

Figura 3.3: Ancho de banda en GB/s de memoria CPU frente a GPU.

La diferencia fundamental entre ambas arquitecturas (CPU frente a GPU) radica en que las GPUs invierten más transistores para el procesamiento de datos que para caché y control de flujo, como puede verse en la Fig. 3.4. Esto significa que una GPU está preparada especialmente para afrontar problemas que pueden resolverse de forma paralela, y necesita muy poco control de flujo ya que estos problemas se basan fundamentalmente en aplicar un mismo programa a cada elemento de un conjunto de datos. Por otra parte, la GPU favorece la implementación paralela de problemas completamente regulares (a menudo denominados *embarrasingly parallel* en la literatura) y con escasas necesidades de sincronización. A continuación describimos las características fundamentales de *NVidia*^{TM} CUDA, la plataforma que hemos utilizado en el presente trabajo para realizar todos los desarrollos propuestos.

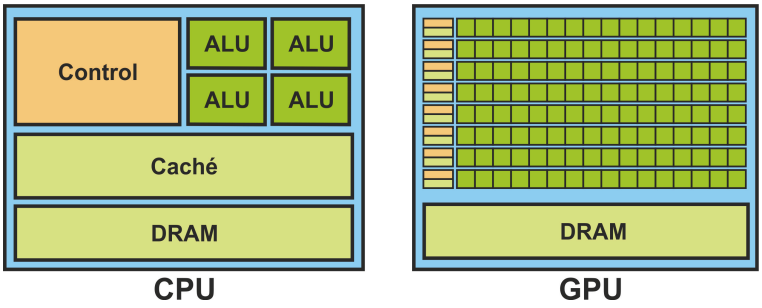

Figura 3.4: Arquitectura CPU frente GPU.

3.2. *NVidia*^{TM} CUDA

NVidia^{TM} CUDA es una plataforma de computación paralela de propósito general que presenta un nuevo modelo de programación paralelo y un conjunto de instrucciones, que combinados permiten utilizar una GPU *NVidia*^{TM} para resolver grandes problemas

complejos de forma más eficiente que si se utilizase una CPU. Una de las características más interesantes de CUDA es que permite utilizar el lenguaje C como lenguaje de programación de alto nivel, del mismo modo que soporta también otros lenguajes como C++, OpenCL, Fortran, etc. CUDA se compone básicamente de un compilador y un conjunto de herramientas de desarrollo creadas por $NVidia^{TM}$ que permiten usar una variación de un lenguaje de programación para codificar algoritmos en una GPU. En lo sucesivo, describimos diferentes aspectos de CUDA que permiten obtener una funcionalidad avanzada de la GPU (como arquitectura hardware y software) permitiendo la utilización de la misma como un dispositivo de procesamiento paralelo capaz de dar soporte como coprocesador avanzado a la CPU.

3.2.1. Paralelismo en $NVidia^{TM}$ CUDA: *grids*, bloques e hilos

CUDA consigue proporcionar paralelismo a través de tres elementos bien estructurados y anidados: *grids*, bloques e hilos. Básicamente, un *grid* se compone de bloques de hilos (ver Fig. 3.5), siendo el hilo la unidad más pequeña e indivisible de paralelismo. Estos elementos son utilizados por las funciones del núcleo de la GPU (en adelante *kernel*) de forma totalmente personalizable por parte del usuario, permitiendo elegir la configuración de paralelismo que sea más adecuada para cada procedimiento. La GPU puede definir un solo *grid* para cada ejecución de una función en el *kernel*, en el cual pueden definirse tantos bloques dentro del grid (e hilos dentro de cada bloque) como se desee. El número de hilos dentro de un bloque es el mismo para una invocación de una función, y puede ser diferente en las siguientes funciones que vayan a ser ejecutadas, simplemente cambiando el valor del número de hilos por bloque.

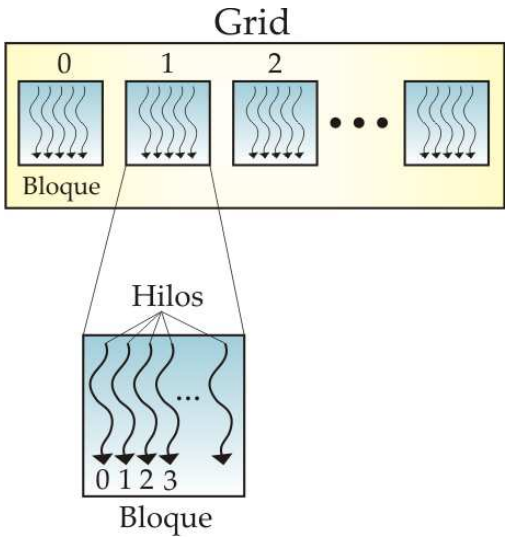

Figura 3.5: Concepto de grid, bloque e hilo en CUDA.

Hay que tener en cuenta que sólo puede ejecutarse una función del *kernel* cada vez, y mientras que no termine esa función no puede invocarse otra. Por tanto es esencial controlar qué hacen los hilos en esa función y asegurarse de tener el mayor número de

hilos ejecutándose a la vez para obtener el mejor rendimiento posible. En este sentido, la diferencia fundamental de estos hilos con respecto a los hilos con los que cuentan algunas CPUs es que los hilos de CUDA son extremadamente ligeros dado que su creación tiene un coste insignificante, y además ofrecen un cambio de contexto instantáneo. CUDA utiliza miles de hilos frente a los multiprocesadores convencionales, que sólo pueden usar unos pocos. En CUDA, todos los hilos dentro de un bloque ejecutan el mismo código, con la salvedad de que cada hilo posee su propio identificador, que puede ser utilizado para realizar parte del control de flujo y para calcular la dirección de memoria de inicio de los datos a procesar. De esta forma, cada hilo ejecuta exactamente las mismas líneas de código pero con diferente información, facilitando de manera notable la realización de cálculos vectoriales y matriciales (ver Fig. 3.6).

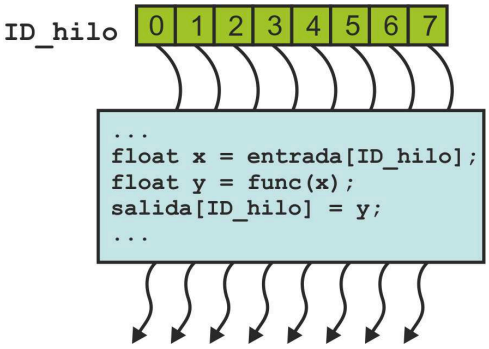

Figura 3.6: Todos los hilos dentro de un bloque CUDA ejecutan el mismo código.

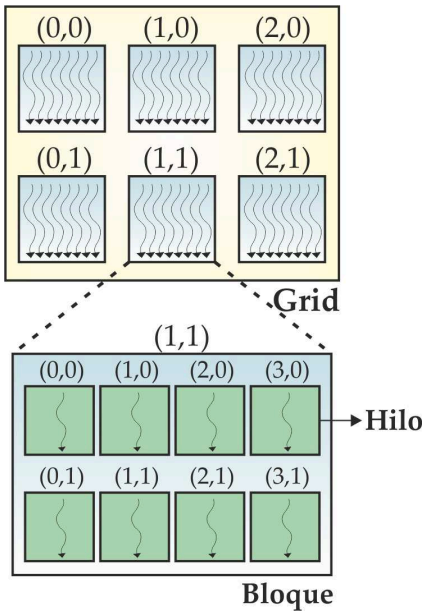

Figura 3.7: Ejemplo de *grid* bidimensional de bloques.

Llegados a este punto, es importante destacar que en la arquitectura CUDA es posible

que los hilos de ejecución (asociados a un mismo bloque) trabajen de forma conjunta, lo cual permite compartir resultados para evitar cálculos redundantes y compartir accesos a memoria, con vistas a aumentar el ancho de banda en dichos accesos. Como valor añadido, CUDA permite definir *grids* y bloques de varias dimensiones que permiten afrontar problemas mucho más complejos de forma sencilla. La Fig. 3.7 muestra un ejemplo de *grid* bidimensional que a su vez cuenta con bloques de dos dimensiones. Otro aspecto importante de la arquitectura CUDA es el hecho de que los hilos dentro de un bloque pueden cooperar entre ellos, compartiendo información a través de la memoria compartida, y sincronizar su ejecución para coordinar los accesos a memoria. A continuación describimos la arquitectura de memoria de CUDA.

3.2.2. Arquitectura de memoria de *NVidia*^{TM} CUDA

En CUDA, los hilos pueden acceder a la información de diferentes formas según la jerarquía de memoria. En primer lugar, como muestra la Fig. 3.8, cada hilo posee su propia memoria privada. Subiendo en la jerarquía, cada bloque de hilos cuenta con su propia memoria compartida, la cual está disponible para todos los hilos que se encuentran en un mismo bloque. El tiempo de vida de la memoria privada de cada hilo y la memoria de un bloque de hilos es el mismo que el del elemento en sí, por lo que, tras terminar su ejecución, ésta es inaccesible. En el nivel más alto de la jerarquía se encuentra la memoria compartida global, a la que todos y cada uno de los hilos pueden acceder. La GPU cuenta además con dos espacios adicionales de memoria de sólo lectura (accesibles por todos los hilos) que son la memoria de constantes y la memoria de texturas, las cuales están optimizadas para utilizarse con tipos de datos específicos. Estas tres memorias (global, de constantes y de texturas) tienen el mismo tiempo de vida que la aplicación que las invoca, perdurando entre diferentes invocaciones de funciones del *kernel* dentro de la misma ejecución de la aplicación. A continuación describimos el modelo de ejecución en la GPU.

3.2.3. Modelo de ejecución en *NVidia*^{TM} CUDA

La arquitectura CUDA se construye sobre la base de un conjunto de multiprocesadores ó *streaming multiprocessors* o (MPs) que componen la GPU (ver Fig. 3.1). Cuando un programa que está siendo ejecutado en la CPU invoca una función del kernel, los bloques del *grid* que invocan la función son enumerados y distribuidos entre los multiprocesadores que estén libres (ver Fig. 3.9). Los hilos de un bloque son ejecutados de forma concurrente en un multiprocesador, y en cuanto un bloque de hilos termina, el multiprocesador queda libre para procesar otro bloque (que estará esperando su ejecución).

Un multiprocesador está compuesto de varios procesadores escalares, así como de una unidad para instrucciones multihilo y memoria compartida. El multiprocesador crea, controla y ejecuta los hilos de forma concurrente sin ningún coste de planificación ó *scheduling* adicional. El multiprocesador cuenta de forma intrínseca con una barrera de sincronización de hilos que puede invocar con una sola instrucción `__syncthreads()`. La rápida sincronización de la barrera, junto con el coste insignificante de creación y el

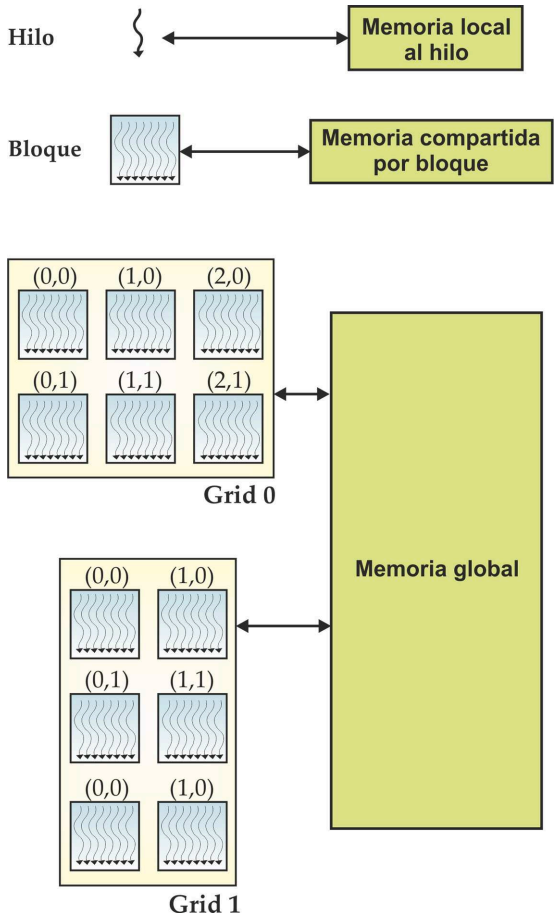

Figura 3.8: Jerarquía de memoria en la GPU según CUDA.

coste cero de *scheduling*, permiten definir un paralelismo de granularidad muy fina. Esto posibilita, por ejemplo, la descomposición de un problema en operaciones elementales, asignando un hilo a cada elemento (como un píxel en una imagen, un voxel en un volumen, o una celda en computación Grid). Para controlar los cientos de hilos que pueden estar en ejecución al mismo tiempo, los multiprocesadores utilizan una nueva arquitectura que *NVidia*^{TM} ha definido como *single instruction, multiple thread* (SIMT). El multiprocesador asocia cada hilo a un procesador escalar y cada *hilo escalar* se ejecuta de forma independiente, con su dirección de memoria y registros.

Un multiprocesador SIMT crea, controla, planifica y ejecuta los hilos en grupos de 32. Estos grupos de 32 hilos paralelos se definen como *warps*. Los hilos que componen un *warp* comienzan su ejecución de forma conjunta en la misma dirección del programa, pero son libres de ramificarse y ejecutarse de forma independiente. Cuando un multiprocesador recibe varios bloques para procesar, los divide en *warps* que son planificados por la unidad SIMT. Esta división en *warps* se lleva a cabo siempre de la misma forma: los hilos de cada *warp* se numeran siguiendo el orden incremental desde el hilo cuyo identificador es cero, el cual se encuentra en el primer *warp*.

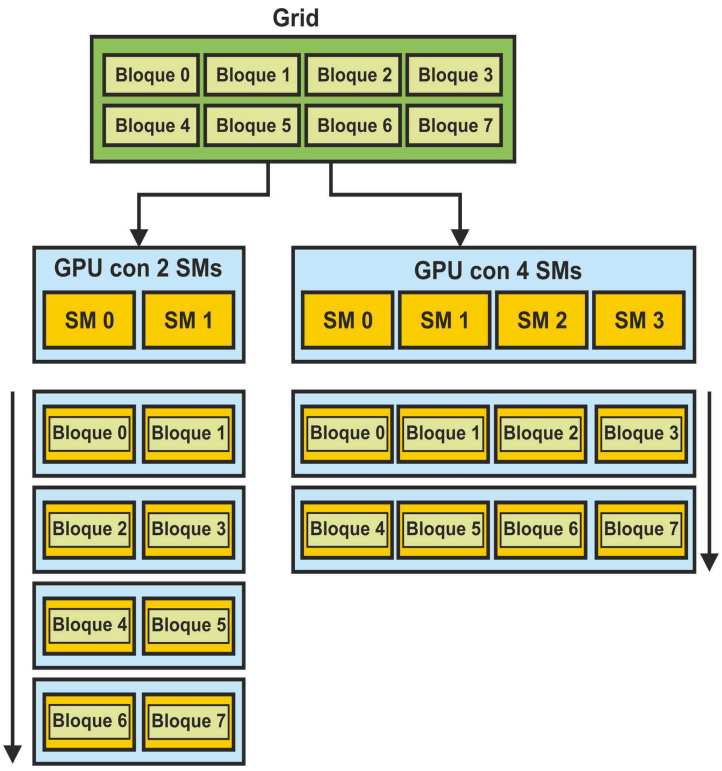

Figura 3.9: Distribución automática de bloques de hilos entre multiprocesadores.

Cuando es necesario ejecutar una instrucción, la unidad SIMT selecciona un *warp* que esté listo para ejecutarse y le entrega la instrucción a los hilos que conforman el *warp*. Un *warp* ejecuta una instrucción cada vez, por lo que se conseguirá la máxima eficiencia cuando los 32 hilos del *warp* trabajen en el mismo sentido. Si los hilos de un *warp* se dividen en diferentes ramas, éste ejecutará de forma secuencial cada rama, deshabilitando los hilos que no sigan el mismo camino que los que esté ejecutando en ese momento, y no convergerán hasta terminar todas las ramas. En este sentido, la arquitectura SIMT es similar a la SIMD en el sentido de que una sola instrucción controla varios elementos de procesamiento. Sin embargo, las instrucciones SIMT pueden especificar la ejecución y la ramificación de un solo hilo. Esto permite a los programadores crear código a nivel de hilo, ya sea para controlar un solo hilo o bien un conjunto de ellos que se encuentran trabajando de forma conjunta.

Por otra parte, el número de bloques que un multiprocesador puede ejecutar a la vez (número de bloques activo por MP) depende de cuántos registros y de cuánta memoria compartida por bloque es necesaria para el *kernel*, teniendo en cuenta que los registros del MP y la memoria compartida se dividen entre todos los hilos de los bloques activos. Si no hay suficientes registros o memoria compartida disponible por multiprocesador para procesar al menos un bloque, el *kernel* no se ejecutará. En cuanto a los *warps* y las escrituras en memoria, si una instrucción no atómica de escritura en memoria es ejecutada por varios hilos de un *warp* en la misma zona de la memoria global o

GPU	Tesla C1060	GTX 580
Nº de GPUs	1	1
Nº de núcleos	240	512
Frecuencia de los núcleos	1.3 Ghz	1.54 Ghz
Memoria total dedicada	4 GB	1536 MB
Frecuencia de la memoria	800 Mhz	2004 Mhz
Interfaz de memoria	512-bit GDDR3	384-bit GDDR5
Ancho de banda de memoria	102 GB/sec	192.4 GB/sec
Consumo máximo	187.8 W	244 W
Interfaz de sistema	PCIE x16 Gen 2	PCIE x16 Gen 2
Solución de disipación térmica	Ventilador / Disipador	Ventilador / Disipador
Entorno de programación	C (CUDA)	C (CUDA)

Tabla 3.1: Especificaciones hardware de las GPUs Tesla C1060 y GTX 580.

Tesla C1060 GTX 580

Figura 3.10: Aspecto físico de las GPUS Tesla C1060 y GTX 580 de Nvidia.

compartida, el número de escrituras (y el orden de las mismas) será indeterminado, pero al menos uno de ellos lo hará. En cambio, si una instrucción atómica que lee, modifica y escribe la misma posición de memoria es ejecutada por varios hilos de un *warp*, cada lectura, modificación y escritura se llevarán a cabo de forma secuencial, pero el orden de ejecución seguirá siendo indeterminado.

3.3. Arquitecturas GPU utilizadas

Las arquitecturas GPU utilizadas en el presente trabajo pertenecen a dos familias diferentes de $NVidia^{TM}$, por un lado tenemos la GPU Tesla C1060 (de la familia *Tesla*), y por otro la GPU GTX 580 (de la familia *Fermi*), la Tabla 3.1 muestra las especificaciones hardware de estos dos modelos de GPUs. El aspecto que tienen estas dos tarjetas puede verse en la Fig. 3.10. La Tesla C1060 es una GPU pensada exclusivamente para el cálculo científico de propósito general y, entre otras cosas, no tiene salida de vídeo. La GPU GTX 580 está diseñada para su utilización en el uso de videojuegos.

El hecho de que cada GPU pertenezca a una familia distinta implica que cada una presenta una arquitectura diferente. Esta característica resulta importante de cara a la programación, ya que diferentes arquitecturas presentan especificaciones técnicas distintas. Como ejemplo ilustrativo las Figs. 3.11 y 3.12 muestran un esquema de la arquitectura de las dos familias de GPUs utilizadas. Las principales diferencias residen en el número y el tipo de multiprocesadores. En caso de *Tesla* cada multiprocesador

esta formado por un grupo de 8 procesadores; un multiprocesador *Fermi* se compone de 2 grupos de 16 procesadores. Los últimos modelos de la familia Tesla (como es el caso de la C1060 utilizada en nuestros experimentos) constan de 30 multiprocesadores, lo que hace un total de 240 procesadores o cores. En el caso de la familia *Fermi*, los últimos modelos constan de 16 multiprocesadores, es decir 512 procesadores (como en el caso de la GPU GTX 580 utilizada en los experimentos). Como hemos visto en este apartado el código se ejecuta en grupos de 32 hilos, lo que *NVidia*^{TM} llama un *warp*. En una *Tesla* los 8 cores de un multiprocesador ejecutan una instrucción para un *warp* completo, 32 hilos, en 4 ciclos. Cada procesador de una *Tesla* tiene una unidad funcional de enteros y otra de coma flotante de precisión simple, además una unidad de operaciones especiales (funciones trigonométricas, coma flotante de doble precisión) se comparte para los 8 procesadores de un multiprocesador. Esto produce un ratio de productividad de 1/8 cuando se trabaja con doble precisión. Un multiprocesador *Fermi* ejecuta una instrucción para 2 *warps* completos (uno por cada grupo de 16 cores) en 2 ciclos para operaciones con enteros y coma flotante de simple precisión. Para operaciones de doble precisión un multiprocesador *Fermi* combina los dos grupos de cores para comportarse como un multiprocesador de 16 cores de doble precisión. Esto significa que el pico de productividad cuando se trabaja con doble precisión sólo desciende a 1/2. Como también hemos visto existe una memoria compartida en cada multiprocesador para que compartan datos los hilos que pertenecen a un mismo bloque. Se trata de una memoria manejada por software con baja latencia y un gran ancho de banda que funciona prácticamente a la velocidad de los registros. En el caso de la familia *Tesla* el tamaño de esta memoria es de 16KB, sin embargo en el caso de la familia *Fermi* es de 64KB, pudiendo configurarse de dos modos: 48KB de memoria compartida manejada por software y 16KB de caché de datos manejada por hardware (modo por defecto); o 16KB de memoria compartida manejada por software y 48KB de caché de datos manejada por hardware.

Al conjunto de especificaciones técnicas de una GPU se le conoce como *compute capability* o capacidad de cómputo. El conjunto de dispositivos de la familia *Tesla* tiene como versión de capacidad de cómputo la $1.x$ mientras que los de la familia *Fermi* tienen como versión la $2.x$. La Tabla 3.2 tiene como objetivo mostrar las diferencias que existen en las especificaciones técnicas de cada familia en función de la capacidad de cómputo.

Una vez descritos algunos conceptos básicos sobre arquitecturas GPU y las diferentes arquitecturas GPU utilizadas en el presente trabajo, procedemos a describir las nuevas implementaciones GPU de algoritmos de análisis hiperespectral desarrolladas, no sin antes destacar que la principal ventaja en cuanto a la utilización de GPUs para el tratamiento de imágenes hiperespectrales viene dada por el hecho de que las mayoría de las técnicas de análisis hiperespectral se basan en la realización de operaciones matriciales [72]. Si bien es cierto que dichas operaciones pueden resultar muy costosas desde el punto de vista computacional debido a la gran dimensionalidad de este tipo de imágenes, en contrapartida conviene destacar que el carácter repetitivo (y muchas veces regular) de estas operaciones las hace altamente susceptibles de ser implementadas en este tipo de procesadores, permitiendo así un incremento significativo de su rendimiento en términos computacionales y dotando a dichas técnicas de la capacidad de producir una respuesta

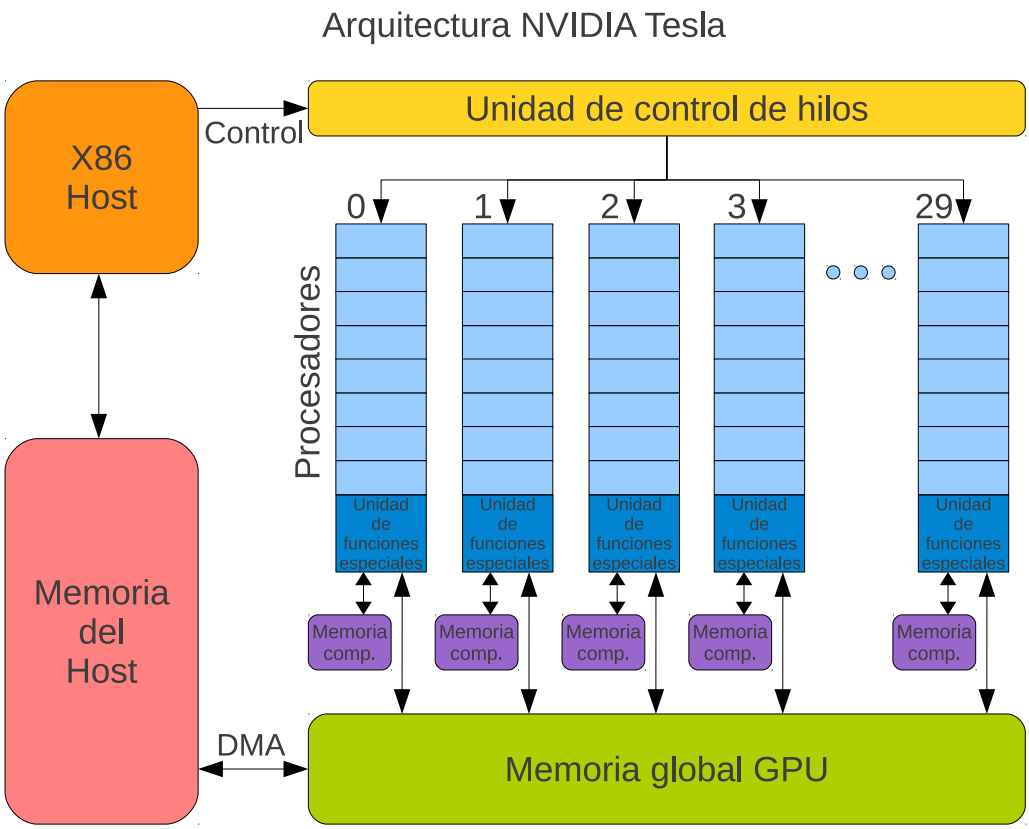

Figura 3.11: Arquitectura NVIDIA Tesla.

Especificaciones técnicas	Capacidad de cómputo (versión)				
	1.0	1.1	1.2	1.3	2.x
Dimensiones máximas x o y de un grid	65535				
N° máximo de hilos por bloque	512				1024
Dimensiones máximas x o y de un bloque	512				1024
Dimensión máxima z de un bloque	64				
Tamaño de warp	32				
N° máximo de bloques residentes por MP	8				
N° máximo de warps residentes por MP	24		32		48
N° máximo de hilos residentes por MP	768		1024		1536
N° de registros de 32 bits por MP	8K		16K		32K
Máxima memoria compartida por MP	16KB				48KB
N° de bancos de memoria compartida	16				32
Cantidad de memoria local por hilo	16KB				512KB
Número máximo de instrucciones por kernel	2 millones				

Tabla 3.2: Capacidad de cómputo de las arquitecturas Tesla y Fermi.

muy rápida cuando se implementan en arquitecturas GPU. Este aspecto es clave para la explotación de dichas técnicas en aplicaciones que requieren una respuesta en tiempo real. En el siguiente capítulo describimos las nuevas implementaciones GPU de algoritmos de análisis hiperespectral desarrolladas en el presente trabajo.

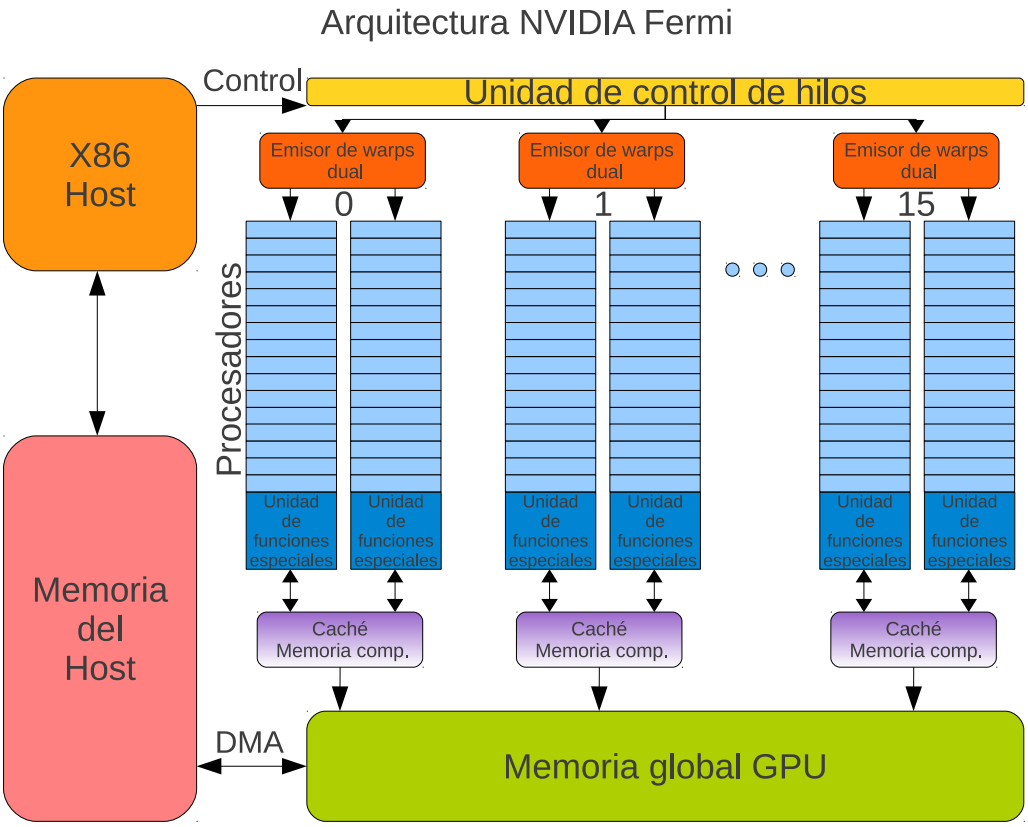

Figura 3.12: Arquitectura NVIDIA Fermi.

Capítulo 4

Implementaciones GPU

En este capítulo se realiza una descripción detallada de las nuevas implementaciones GPU de algoritmos de análisis hiperespectral desarrolladas en el presente trabajo utilizando CUDA. Los métodos considerados se basan en la resolución del problema de la mezcla descrito en el capítulo 2 de la presente memoria, y comprenden un conjunto de técnicas para estimación del número de *endmembers* (descritas en el apartado 2.3.1), reducción dimensional (ver apartado 2.3.2), identificación de *endmembers* (ver apartado 2.3.3) y estimación de abundancias (ver apartado 2.3.4). A continuación se describen los métodos implementados en cada uno de estos apartados, las cuales pueden utilizarse para formar cadenas completas de desmezclado espectral como ya se indicó en el capítulo 2 del presente documento.

4.1. Estimación del número de *endmembers*

En este apartado se describe la implementación GPU de dos métodos: VD (descrito en el subapartado 2.3.1.1) y HySime (descrito en el subapartado 2.3.1.2). Para cada método se indican en primer lugar las entradas y las salidas del mismo, y a continuación se proporciona una descripción de los pasos realizados para llevar a cabo su implementación eficiente en arquitecturas GPU.

4.1.1. *Virtual Dimensionality* (VD)

- **Entradas**: Imagen hiperespectral, $\mathbf{Y}$, y probabilidad de falsa alarma, P_F.
- **Salidas**: Estimación del número de *endmembers*, p.

Como se mostró en el subapartado 2.3.1.1, el primer paso en la implementación GPU del algoritmo VD es cargar la imagen $\mathbf{Y}$ de disco a la memoria principal de la CPU. A continuación se carga la imagen completa en la memoria global de la GPU. La primera operación necesaria es el cálculo de la matriz de covarianza $\mathbf{K}_{l\times l}$. Para este propósito necesitamos calcular primero el valor medio de cada banda de la imagen y restárselo a todos los píxeles de la misma banda, es decir, centrar los datos en la media restando a cada píxel de la imagen el píxel medio $\overline{\mathbf{Y}}$. Para llevar a cabo este cálculo en la GPU, utilizamos

un kernel llamado `centrarDatos` configurado con tantos bloques como bandas l tenga la imagen hiperespectral considerada. En cada bloque, todos los hilos cooperan para llevar a cabo un proceso de reducción en el que se van sumando los valores de todos los píxeles para cada banda. Para ello se hace uso de la memoria compartida y se realizan lecturas coalescentes. La coalescencia permite a un grupo de hilos (concretamente medio *warp*) realizar lecturas o escrituras con un único acceso, siempre que los datos se encuentren en un segmento de 32, 64 o 128 bytes. La Fig. 4.1 muestra como se lleva a cabo el proceso de reducción para el caso de una suma: en primer lugar se realiza una lectura coalescente de los datos en memoria global y se escriben en memoria compartida, a continuación cada hilo del bloque se encarga de sumar 2 elementos, seguidamente se reduce a la mitad el número de hilos y cada uno de estos vuelve a realizar la suma de otros 2 elementos (resultantes de la iteración anterior). El proceso se repite hasta que queda un solo hilo, el cual sumará 2 elementos para obtener el total. Una vez sumados se divide cada valor calculado por el número de píxeles de la imagen.

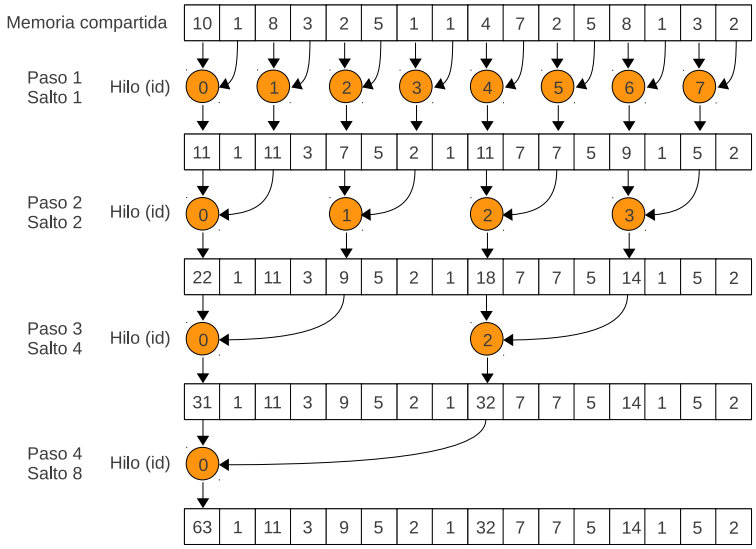

Figura 4.1: Proceso de reducción en la memoria compartida de la GPU.

Llegados a este punto, se ha obtenido el píxel medio de la imagen $\overline{\mathbf{Y}}$ el cual se almacena en una estructura para su posterior uso. La última tarea del kernel `centrarDatos` es restar ese píxel medio a cada píxel de la imagen. Como a todos los elementos de una banda se les va a restar el mismo valor, hacemos uso otra vez de la memoria compartida leyendo desde aquí el valor que se va a sustraer. Este proceso, en el que todos los hilos realizan una lectura de la misma posición de la memoria compartida, se denomina *broadcast* y no provoca ningún conflicto en dicha memoria. La Fig. 4.2 muestra como se lleva a cabo este proceso de centrado de datos. Podemos ver como los hilos del i-ésimo bloque se encargan de calcular el valor en la i-ésima banda espectral del píxel medio y a continuación restan ese valor al que se encuentra almacenado en la i-ésima banda espectral de cada píxel. El hecho de que en la Fig. 4.2 el último píxel se denote como $n-1$ y el último hilo se denote como $k-1$ se debe a que no podemos configurar un bloque con más de 512 hilos

en el caso de la arquitectura *Tesla*, o con más de 1024 en caso de la arquitectura *Fermi* (ver apartado 3.3), siendo estos números mucho menores que el número de píxeles en una imagen de este tipo. Por tanto, lo que realmente hace la implementación propuesta es procesar cada banda en tramos de k hilos, siendo k el número máximo de hilos soportados por la arquitectura considerada.

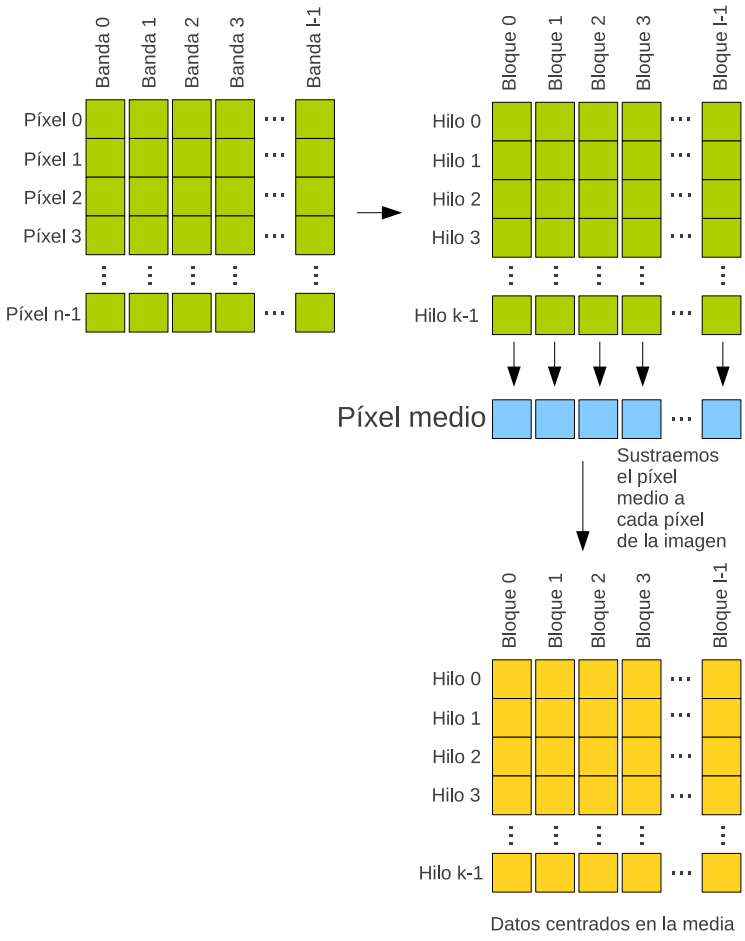

Figura 4.2: Esquema de procesamiento del kernel `centrarDatos` en la implementación CUDA del método VD.

El resultado de esta operación es una matriz $\mathbf{Y} - \overline{\mathbf{Y}}$ con los datos de la imagen centrados en la media. Esta matriz se utilizará para finalizar los cálculos de la matriz de covarianza en la GPU mediante una operación de multiplicación de matrices dada por: $(\mathbf{Y} - \overline{\mathbf{Y}})^T(\mathbf{Y} - \overline{\mathbf{Y}})$. Para realizar esta operación hacemos uso de la librería `cuBLAS`[1], una implementación de la librería *basic linear algebra subprograms* (BLAS)[2] optimizada por NVidia™ para su ejecución paralela en GPUs. Concretamente, utilizamos la función `cublasSgemm` de `cuBLAS`, que realiza multiplicación eficiente de matrices en GPUs. El

[1] https://developer.nvidia.com/cublas
[2] http://www.netlib.org/blas

siguiente paso es calcular la matriz de correlación $\mathbf{R}_{l\times l}$ en la GPU. Para ello, utilizamos un kernel llamado `correlación` que utiliza como entradas dos matrices: la matriz de covarianza $\mathbf{K}$, y la matriz resultante de la operación $\overline{\mathbf{Y}}\overline{\mathbf{Y}}$ (calculada mediante `cuBLAS`). El kernel utiliza tantos hilos como elementos tenga la matriz de correlación, es decir $l \times l$ hilos organizados en bloques con el mayor número de hilos permitido por la arquitectura, k. Cada hilo se encarga del cálculo de un elemento de la matriz de correlación conforme a esta expresión: $\mathbf{R}_{ij} = \mathbf{K}_{ij} + \overline{\mathbf{Y}}_i\overline{\mathbf{Y}}_j$.

Una vez calculadas las matrices de correlación y covarianza en la GPU, el siguiente paso es extraer de estas los autovalores y aplicar el detector de Neyman-Pearson utilizando el parámetro de entrada P_F para estimar el número de *endmembers*. Estos tres pasos se realizan de forma rápida en la CPU y, por tanto, en la presente implementación no realizamos este cálculo en la GPU ya que hemos comprobado experimentalmente que el envío de los datos necesarios para poder hacer el cálculo en la GPU supone más tiempo que el necesario para completar dichos cálculos en la CPU.

4.1.2. *Hyperspectral signal identification by minimum error* (HySime)

- **Entradas**: Imagen hiperespectral, $\mathbf{Y}$.
- **Salidas**: Número de *endmembers* estimados, p.

Como se mostró en el subapartado 2.3.1.2, el método HySime consta de dos partes, una dedicada a la estimación del ruido en la imagen hiperespectral $\mathbf{Y}$ (ver Algoritmo 1) y otra para la estimación del subespacio en el que residen los datos y, por tanto, del número de *endmembers* (ver Algoritmo 2). La implementación GPU comienza cuando se transfieren los datos de la imagen desde la memoria principal de la CPU a la memoria global de la GPU. Una vez hecho esto se comienza con la parte de estimación de ruido. El primer paso es calcular la matriz de correlación $\widehat{\mathbf{R}}$ y su inversa $\mathbf{R}'$. La primera se calcula en la GPU mediante el uso de la función para multiplicación de matrices `cublasSgemm` de `cuBLAS`, mientras que el cálculo de la matriz inversa se realiza en la CPU mediante una función serie (hemos comprobado experimentalmente que el envío de los datos necesarios para poder hacer el cálculo en la GPU supone más tiempo que el necesario para completar dichos cálculos en la CPU).

Seguidamente el bucle `for` en el paso 4 del Algoritmo 1 repite el mismo conjunto de operaciones para cada banda de la imagen hiperespectral. Como todas las iteraciones en el bucle son independientes y en cada una se calcula una parte de la matriz $\widehat{\xi}$ con la estimación de ruido, podemos ejecutar estas operaciones en la GPU. Para ello hacemos uso de dos kernels y una multiplicación de matrices de gran tamaño mediante `cublasSgemm`. El primer kernel `calcularBeta` realiza el cálculo de todos los vectores de regresión $\widehat{\beta}_i$ y los almacena en una matriz $\widehat{\beta}$ de tamaño $l \times l$, siendo l el número de bandas de la imagen hiperespectral $\mathbf{Y}$. La idea principal en este kernel es asignar la computación de la i-ésima iteración, es decir de $\widehat{\beta}_i$ en el bucle `for`, al i-ésimo bloque. De este modo, para este kernel tenemos tantos bloques como bandas en la imagen hiperespectral original, l, y el mismo número de hilos por bloque. El cálculo que se

lleva a cabo en cada bloque para obtener un $\widehat{\beta}_i$ se divide en varios pasos, partiendo de la expresión (4.1) que aparece en el paso 5 del Algoritmo 1:

$$\widehat{\beta}_i = ([\mathbf{R}']_{\alpha_i,\alpha_i} - [\mathbf{R}']_{\alpha_i,i}[\mathbf{R}']_{i,\alpha_i}/[\mathbf{R}']_{i,i})[\widehat{\mathbf{R}}]_{\alpha_i,i}, \tag{4.1}$$

donde $[\mathbf{R}']_{\alpha_i,\alpha_i}$ es la inversa de la matriz de correlación completa (la misma en todas las iteraciones del bucle), $[\mathbf{R}']_{\alpha_i,i}$ es la i-ésima columna de la misma matriz, $[\mathbf{R}']_{i,\alpha_i}$ es la i-ésima fila de la misma matriz, $[\mathbf{R}']_{i,i}$ es el elemento (i,i) de la misma matriz, y $[\widehat{\mathbf{R}}]_{\alpha_i,i}$ es la i-ésima columna de la matriz de correlación (no de su inversa). A partir de esta información, componemos los cálculos de la siguiente manera:

$$AA_i = [\mathbf{R}']_{\alpha_i,i}[\mathbf{R}']_{i,\alpha_i}/[\mathbf{R}']_{i,i}, \tag{4.2}$$

$$XX_i = [\mathbf{R}']_{\alpha_i,\alpha_i} - AA_i, \tag{4.3}$$

$$\widehat{\beta}_i = XX_i[\widehat{\mathbf{R}}]_{\alpha_i,i}. \tag{4.4}$$

La Fig. 4.3 muestra cómo se lleva a cabo el proceso descrito por la expresión (4.2). En concreto, dicha figura muestra cómo cada bloque se encarga de el cálculo de un AA_i, para ello se almacena la i-ésima columna de $\mathbf{R}'$ en la memoria compartida del multiprocesador en el que se aloja el bloque, posteriormente el j-ésimo hilo se encarga de calcular la j-ésima fila de la matriz AA_i, multiplicando los vectores $[\mathbf{R}']_{\alpha_i,i}$ y $[\mathbf{R}']_{i,\alpha_i}$ y dividiendo el valor resultante por el elemento $[\mathbf{R}']_{i,i}$. De esta manera, hilos consecutivos escriben en memoria global en posiciones consecutivas, lo que permite accesos coalescentes a esta memoria aumentando notablemente el rendimiento.

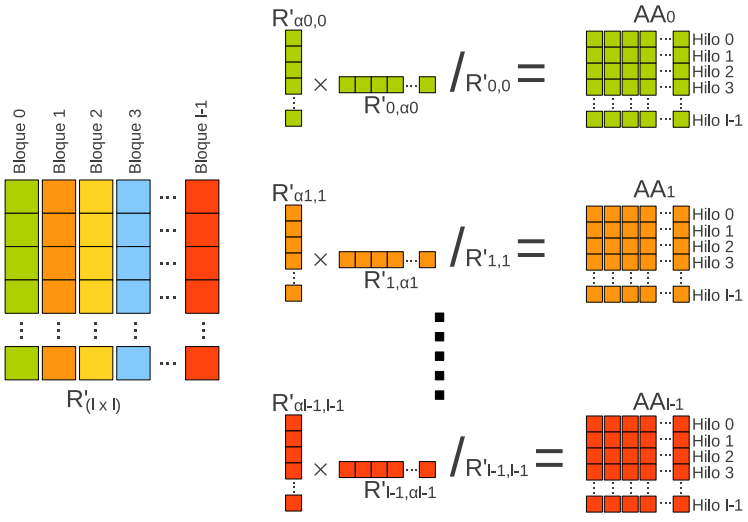

Figura 4.3: Esquema de procesamiento del kernel `calcularBeta` en el método HySime: cálculo de la expresión (4.2).

Por su parte, la Fig. 4.4 muestra el proceso seguido por el kernel `calcularBeta` para calcular expresión (4.3). En esta ocasión, el i-ésimo bloque se encarga del cálculo de matriz XX_i. El proceso se divide en l iteraciones (una por cada columna de la matriz $\mathbf{R}'$) y en cada iteración el j-ésimo hilo se encarga de calcular el j-ésimo elemento de la

columna que se está tratando en la iteración actual, realizando la resta de dos elementos de las matrices $[\mathbf{R}']_{\alpha_i,\alpha_i}$ y AA_i (calculada en el paso anterior). De esta manera, cada hilo calcula una fila completa de la matriz XX_i.

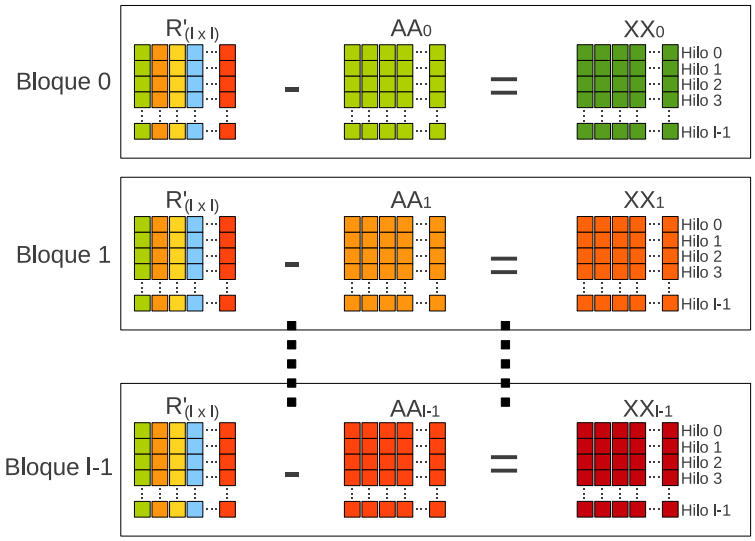

Figura 4.4: Esquema de procesamiento del kernel `calcularBeta` en el método HySime: cálculo de la expresión (4.3).

La etapa final del kernel `calcularBeta` consiste en el cálculo de la expresión (4.4), proceso que aparece ilustrado gráficamente en la Fig. 4.5. En este caso, el bloque i-ésimo se encarga de calcular el vector columna $\widehat{\beta}_i$. En un primer paso el i-ésimo bloque realiza una copia en memoria compartida de el vector $[\widehat{\mathbf{R}}]_{\alpha_i,i}$ cuyos elementos serán usados por todos los hilos del bloque. Al igual que en el paso anterior, el proceso se divide en l iteraciones, siendo l el número de columnas de XX_i. En una iteración concreta it, el j-ésimo hilo lee el j-ésimo elemento de la columna it de XX_i, lo multiplica por el j-ésimo elemento del vector almacenado en memoria compartida y almacena el resultado en la j-ésima posición del vector $\widehat{\beta}_i$. Los resultados parciales de cada iteración se suman para obtener el resultado final.

Una vez que hemos calculado la matriz $\widehat{\beta}$, el siguiente paso de la implementación consiste en llevar a cabo la operación $\widehat{\xi}_i = \mathbf{z}_i - \mathbf{Z}_{\alpha_i}\widehat{\beta}_i$ (ver Algoritmo 1). Podemos calcular todos los $\widehat{\xi}_i$ en paralelo, de manera que la expresión queda de la siguiente manera: $\widehat{\xi} = \mathbf{Z} - \mathbf{Z}\widehat{\beta}$. Para este cálculo hacemos nuevamente uso de la librería `cuBlas`, y en particular de la función `cublasSgemm`, para llevar a cabo la multiplicación $\mathbf{P} = \mathbf{Z}\widehat{\beta}$. Finalmente se utiliza otro kernel denominado `restarElementos` en el cual se maximiza el número de hilos lanzando (tantos como elementos haya en $\mathbf{Z}$) para realizar la resta $\widehat{\xi} = \mathbf{Z} - \mathbf{P}$ en paralelo. En este kernel, cada hilo se encarga de restar un único elemento. Llegados a este punto, se han realizado todos los pasos del Algoritmo 1 y tenemos como resultado la matriz $\widehat{\xi}$ de $n \times l$ elementos con la estimación de el ruido en la imagen hiperespectral $\mathbf{Y}$.

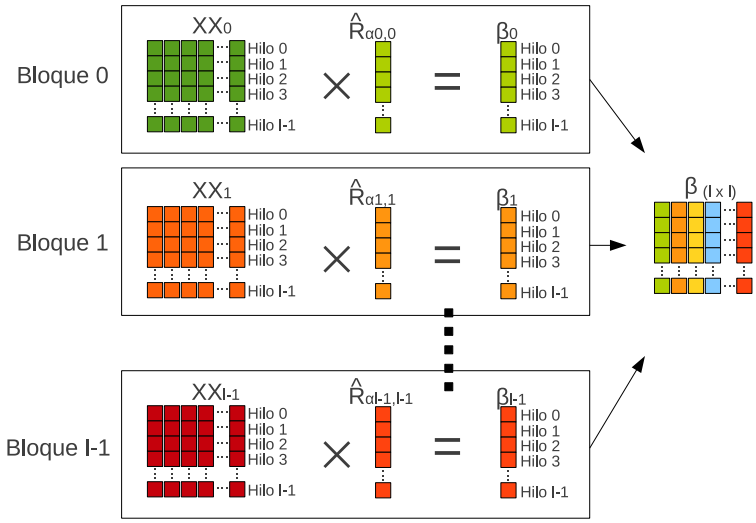

Figura 4.5: Esquema de procesamiento del kernel `calcularBeta` en el método HySime: cálculo de la expresión (4.4).

Pasamos a describir la segunda parte de la implementación, que consiste en la estimación del subespacio siguiendo los pasos indicados en el Algoritmo 2. Lo primero es calcular la matriz de correlación del ruido $\widehat{\mathbf{R}}_n = \left[\widehat{\xi}_1, ..., \widehat{\xi}_n\right]^T \left[\widehat{\xi}_1, ..., \widehat{\xi}_n\right]$. Esta operación podría realizarse mediante una multiplicación de matrices mediante `cuBLAS`. Sin embargo, como sólo se utilizan los elementos de la diagonal de esta matriz $\widehat{\mathbf{R}}_n$ para estimar el número de *endmembers*, no es necesario llevar a cabo una multiplicación de matrices completa. En su ligar, calculamos únicamente los elementos de la diagonal de la siguiente forma: $\widehat{\mathbf{R}}_{n_{i,i}} = \left[\widehat{\xi}_{1_i}, ..., \widehat{\xi}_{n_i}\right]^T \left[\widehat{\xi}_{1_i}, ..., \widehat{\xi}_{n_i}\right]$. Debido a la cantidad de estructuras intermedias que se utilizan en este paso (y al tamaño de las mismas) hemos concluido que puede haber problemas de espacio de memoria en la GPU (sobre todo en el caso de la arquitectura GTX 580 que cuenta con 1.5 GB de memoria). Por este motivo, se ha optado por realizar este paso en serie en la CPU.

El siguiente paso del Algoritmo 2 es el cálculo de la matriz de correlación de la señal $\widehat{\mathbf{R}}_s = 1/n \sum_i ((\mathbf{y}_i - \widehat{\xi}_i)(\mathbf{y}_i - \widehat{\xi}_i^T))$. Para implementar esta etapa en la GPU, volvemos a hacer uso del kernel `restarElementos` que nos permite realizar la operación $(\mathbf{y}_i - \widehat{\xi}_i^T)$ de forma eficiente. Seguidamente se emplea la función `cublasSgemm` de `cuBLAS` para obtener $\widehat{\mathbf{R}}_s$. El paso 4 del Algoritmo 2 realiza la descomposición de valores singulares de $\widehat{\mathbf{R}}_s$ para obtener sus autovectores. Este paso se lleva a cabo en la CPU, al igual que los pasos 5 y 6 que ordenan estos valores de forma ascendente, ya que se ha comprobado experimentalmente que la transferencia y cálculo de los datos a la GPU consume más que la realización del cálculo en la CPU. La obtención de los términos $\widehat{\delta}_i$ se lleva a cabo conforme a la expresión (4.5) basada en formas cuadráticas dadas por $q_{i_j} = \mathbf{v}_{i_j}^T \widehat{\mathbf{R}}_s \mathbf{v}_{i_j}$ y $\sigma_{i_j}^2 = \mathbf{v}_{i_j}^T \widehat{\mathbf{R}}_n \mathbf{v}_{i_j}$, donde cada $\mathbf{v}_i$ es el i-ésimo autovector de la matriz de correlación de la

señal:

$$\widehat{\delta}_{i_j} = \sum_{j=1}^{l} -q_{i_j} + 2\sigma_{i_j}^2 \tag{4.5}$$

Estas operaciones se implementan haciendo uso de `cuBLAS`. Finalmente, aplicamos una sencilla función de minimización en la CPU para obtener el número de *endmembers*, p, calculando cuántos $\widehat{\delta}_i$ son menores que 0. El subespacio de la señal viene dado, por tanto, por los primeros p autovectores de $\widehat{\mathbf{R}}_s$.

4.2. Reducción dimensional

En este apartado se describe la implementación GPU de dos métodos: PCA (descrito en el subapartado 2.3.2.1) y SPCA (descrito en el subapartado 2.3.2.2). Para cada método se indican en primer lugar las entradas y las salidas del mismo, y a continuación se proporciona una descripción de los pasos realizados para llevar a cabo su implementación eficiente en arquitecturas GPU.

4.2.1. *Principal component analysis* (PCA)

- **Entradas**: Imagen hiperespectral $\mathbf{Y}$ con l bandas espectrales.
- **Salidas**: Imagen hiperespectral reducida con $p-1$ componentes.

Como se mostró en el subapartado 2.3.2.1, el primer paso para realizar la transformación PCA es calcular la matriz de covarianza $\mathbf{K}_{l\times l}$. Para ello, copiamos los datos de la imagen original $\mathbf{Y}$ a la memoria global de la GPU y utilizamos el kernel `centrarDatos` (explicado detalladamente en el apartado 4.1.1). El resultado de esta operación es una matriz $\mathbf{Y} - \overline{\mathbf{Y}}$ con los datos de la imagen centrados en la media. Esta matriz se utilizará para finalizar los cálculos de la matriz de covarianza en la GPU mediante una operación de multiplicación de matrices $(\mathbf{Y} - \overline{\mathbf{Y}})^T(\mathbf{Y} - \overline{\mathbf{Y}})$ realizada utilizando la función `cublasSgemm` de `cuBLAS`.

Una vez completada esta operación, a continuación se copia la matriz $\mathbf{K}_{l\times l}$ en la memoria de la CPU para realizar la descomposición en valores singulares de dicha matriz. Mediante esta descomposición extraemos los autovalores y los autovectores ($\mathbf{V}_{l\times l}$) de $\mathbf{K}_{l\times l}$. Este paso se realiza en la CPU. Posteriormente, se ordenan los autovalores y sus autovectores asociados decrecientemente, y se copian estos últimos de nuevo en la memoria global de la GPU para llevar a cabo el proceso de proyección de los datos de la imagen original $\mathbf{Y}$ sobre los autovectores, que son las direcciones de los nuevos ejes de coordenadas en las que la varianza es mayor. Este proceso se realiza mediante la siguiente multiplicación de matrices: $\mathbf{B} = \mathbf{YV}$. El resultado es una transformación (acorde a la varianza de los datos) de la imagen hiperespectral original $\mathbf{Y}$ en una nueva estructura $\mathbf{B}$ con las mismas dimensiones que la imagen original, donde la primera banda de esta nueva imagen será la banda con mayor varianza en los datos. Seleccionando las primeras $p-1$ bandas, donde p es el número de *endmembers* que se pretende extraer, tenemos una

imagen reducida dimensionalmente que mantiene una gran cantidad de la información presente en la imagen original. Es posible acelerar el proceso de proyección para obtener directamente la imagen dimensionalmente reducida si en lugar de utilizar la matriz $\mathbf{V}$ completa utilizamos solamente las $p-1$ primeras columnas de ésta, es decir los primeros $p-1$ autovectores.

4.2.2. *Simple principal component analysis* (SPCA)

- **Entradas**: Imagen hiperespectral $\mathbf{Y}$ y valor umbral de convergencia t_v.
- **Salidas**: Imagen hiperespectral reducida con $p-1$ componentes.

La SPCA comienza realizando una carga de los datos de la imagen de la memoria principal en CPU a la memoria global en GPU. Aquí los datos se centran y normalizan conforme a la desviación estándar, dividiendo cada píxel de la imagen hiperespectral $\mathbf{Y}$ por la varianza de la imagen calculada como $\sigma^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})^2\right]$. Como ya se comentó en el subapartado 2.3.2.2, esta operación mejora la estabilidad en la ejecución del método. Para realizar dicha normalización de datos en la GPU, utilizamos un kernel llamado `normalizarDatos` cuyo primer paso es idéntico al del kernel `centrarDatos` (explicado en la sección 4.1.1). En este paso se calcula el término $(\mathbf{y}_i - \bar{\mathbf{y}})$ de la varianza. El siguiente paso consiste en calcular la suma de los cuadrados de los datos centrados por columnas. Para ello, se utiliza un método de reducción con el que pasamos de n datos a 1. Seguidamente se divide este valor obtenido por $n-1$. Llegados a este punto, tenemos calculada la varianza de cada píxel (o dicho de otra forma, de cada columna de la matriz $\mathbf{Y}$ que contiene los píxeles de la imagen hiperespectral).

El kernel continúa dividiendo cada elemento de $\mathbf{Y}$ por la varianza de su columna. En este paso cada bloque trabaja sobre una columna. Como normalmente existen más elementos en una columna que el número de hilos que puede lanzar un bloque, es preciso dividir el proceso en iteraciones. En cada iteración cada hilo calcula un elemento de la columna. Como los elementos de una misma columna son consecutivos en la memoria y los hilos consecutivos acceden a posiciones consecutivas, estamos realizando accesos coalescentes a la memoria global, lo cual supone una mejora de rendimiento notable. Además cada hilo almacena el valor de la varianza en sus registros de manera que el acceso a este valor se produzca lo más rápido posible. En la configuración de este kernel utilizamos tantos bloques de hilos como bandas tenga la imagen hiperespectral, y maximizamos el número de hilos de acuerdo a la arquitectura ya que no se excede la utilización de recursos dentro de cada multiprocesador para este caso.

La siguiente fase de la transformación SPCA consiste en calcular la matriz de covarianza $\mathbf{K}$ de la imagen $\mathbf{Y}$ normalizada, multiplicando ésta misma por su traspuesta. Para realizar este proceso utilizamos la función de multiplicación de matrices `cublasSgemm` proporcionada por `cuBLAS`. A continuación repetimos una serie de pasos de forma iterativa hasta que se cumple una condición de convergencia:

1. Comenzamos construyendo la matriz de deflación Δ. Para ello primero calculamos $\mathbf{V}^T\mathbf{V}$ en la GPU, donde $\mathbf{V}_{l\times p-1}$ es una matriz aleatoria. El cálculo de Δ se puede

obtener de forma recursiva añadiendo elementos a cada fila o a cada columna tal y como se indica en la expresión (2.6).

2. A continuación realizamos las proyecciones de deflación mediante $\mathbf{V}_{dfl} = \mathbf{V}\Delta$ en la GPU, utilizando la función `cublasSgemm`.

3. En este paso se ejecuta el algoritmo *power iteration* (ver Algoritmo 3) y se calcula la corrección causada por el proceso de deflación. Ambas multiplicaciones matriciales $\Phi = \mathbf{K}\mathbf{V}_{dfl}$ y $\Psi = \mathbf{V}\text{triu}(\mathbf{V}_{dfl}^T\Phi, -1)$ se realizan en la GPU mediante el uso de `cuBLAS`.

4. El cálculo de $\mathbf{V} = \Phi - \Psi$ y de $\mathbf{V} = \mathbf{V}/norm(\mathbf{V})$ se realiza en la CPU ya que se ha observado experimentalmente que su computación en GPU no aporta una mejora en el tiempo total de ejecución. Hay que tener en cuenta que la matriz $\mathbf{V}$, con dimensiones $l \times (p-1)$, es más pequeña que $\mathbf{K}$, con dimensiones $l \times l$, que a su vez es más pequeña que la imagen original $\mathbf{Y}$, de tamaño $n \times l$.

5. Finalmente analizamos la condición de convergencia viendo si existe un cambio significativo en la dirección de los autovectores en relación con la iteración anterior. Esta operación se puede llevar a cabo de forma sencilla en la CPU. Si los cambios medidos están por encima de un umbral dado t_v, continuamos iterando. En caso contrario finalizamos y proporcionamos los autovectores finales en $\mathbf{V}$.

Una vez se han obtenido los autovectores finales, el último paso es multiplicar éstos por la imagen original $\mathbf{Y}$. De esta forma, se obtiene una imagen reducida $\mathbf{B}$ de tamaño $n \times (p-1)$. Esta operación se realiza en la GPU mediante el uso de la función `cublasSgemm` de `cuBLAS`.

4.3. Identificación de *endmembers*

En este apartado se describe la implementación GPU de dos métodos: PPI (descrito en el subapartado 2.3.3.1) y N-FINDR (descrito en el subapartado 2.3.3.2). Para cada método se indican en primer lugar las entradas y las salidas del mismo, y a continuación se proporciona una descripción de los pasos realizados para llevar a cabo su implementación eficiente en arquitecturas GPU.

4.3.1. *Pixel purity index* (PPI)

- **Entradas**: Imagen hiperespectral $\mathbf{Y}$ y número de iteraciones it.
- **Salidas**: Índice de pureza asociado a cada píxel de la imagen.

Como se mostró en el subapartado 2.3.3.1, el algoritmo PPI se puede dividir en varias fases bien diferenciadas. Por un lado tenemos la generación de los vectores aleatorios ó *skewers*, por otro lado la proyección de todos los píxeles de la imagen sobre cada *skewer* (lo cual define el número de iteraciones del algoritmo, it) y la detección de los

píxeles más extremos. Y por otro lado la agrupación de los resultados para formar la imagen de pureza que contiene el índice de pureza asociado a cada píxel.

Comencemos pues con la fase de generación de *skewers*. Como sabemos, un *skewer* es un vector unitario cuyas componentes se generan de forma aleatoria. El número de componentes de cada *skewer* debe ser igual al número de bandas de la imagen hiperespectral, l. En una iteración, el algoritmo proyecta todos los píxeles de la imagen sobre un *skewer*, por tanto necesitamos tantos *skewers* como iteraciones, it. En este sentido, necesitaremos generar un total de $it \times l$ números aleatorios. A la hora de generar tal cantidad de números hemos utilizado una función que realiza dicha acción haciendo uso de la GPU. Esta función se encuentra dentro de los ejemplos de uso de CUDA, concretamente se trata de una función llamada `RandomGPU` que aparece dentro del ejemplo `MersenneTwister`. Esta función permite generar y almacenar en la memoria de la GPU 24×10^6 elementos en tan solo 10 milisegundos. De esta manera con una sola llamada al kernel, podemos generar números aleatorios en muy poco tiempo. Cada grupo de l elementos formará un *skewer*. La configuración de la llamada al kernel se ha realizado mediante 32 bloques de 128 hilos cada bloque, tal y como viene especificada por defecto en el ejemplo. A continuación se describen los parámetros utilizados en la invocación a la función `RandomGPU`:

1. El primer parámetro es la dirección a la estructura de datos donde se van a almacenar los números generados. Este parámetro debe de ser un puntero a un tipo float y previamente se deberá reservar espacio en la GPU para albergar esta estructura. A esta estructura la denominamos en lo sucesivo `d_Skewers`.

2. El segundo parámetro es el número de elementos que generará cada hilo. Se trata de un entero cuyo valor indica el número de elementos que se va a generar durante la ejecución de un hilo.

3. El tercer parámetro es la semilla. Se trata de un valor que sirve de condición inicial para la función de generación de números aleatorios; su valor está en función de la hora y fecha del sistema. La asignación de un valor en función de la hora y fecha a la semilla nos permite obtener valores distintos en cada ejecución. De hecho, como los parámetros son comunes a todos los hilos del kernel, todos los hilos generarían el mismo conjunto de elementos aleatorios de no ser porque internamente este valor de semilla se modifica multiplicándose por el identificador interno del hilo. De esta forma, cada hilo cuenta con una semilla distinta y por lo tanto genera un conjunto de elementos distinto del resto de hilos. Por razones ilustrativas, la Fig. 4.6 muestra gráficamente la forma en que los *skewers* quedan almacenados en la estructura `d_Skewers`, siendo it el número de iteraciones (*skewers*) del algoritmo PPI.

Una vez finalizada la fase de generación de *skewers*, el siguiente paso es la proyección de todos los puntos de la imagen sobre cada *skewer* y la identificación de los píxeles extremos. Esta fase se realiza mediante una única llamada al kernel `PPI`. En dicha llamada, cada hilo realiza la proyección de todos los píxeles de la imagen sobre un único

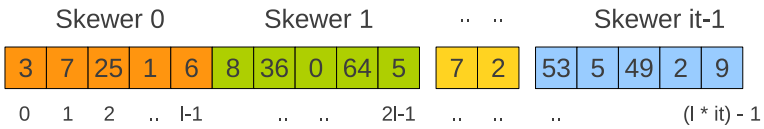

Figura 4.6: Almacenamiento de los *skewers* en la implementación GPU del método PPI.

skewer y detecta los dos píxeles más extremos para ese *skewer*, es decir los píxeles que resulten en el producto escalar con valor más alto y más bajo. La configuración para la ejecución del kernel es la siguiente: tendremos tantos hilos de ejecución como iteraciones queramos realizar en el algoritmo, se maximiza el número de hilos por bloque dependiendo de la arquitectura (512 para *Tesla* y 1024 para *Fermi*), por tanto el número de bloques es el número de iteraciones dividido por el número de hilos por bloque. El kernel necesita como parámetros de entrada la imagen hiperespectral $\mathbf{Y}$ y la estructura `d_Skewers`. La salida del mismo es una estructura de tipo vector que llamaremos `d_res_parcial`. El tamaño de esta estructura es $it \times 2$, esto es así porque en cada iteración se almacenan las dos proyecciones extremas sobre un *skewer*, esto es, los píxeles que obtienen el valor más alto y más bajo en el producto escalar con el *skewer*. La Fig. 4.7 representa de forma gráfica la forma en la que se almacenan estos valores en la estructura. En dicha estructura, cada color representa los resultados obtenidos en una iteración.

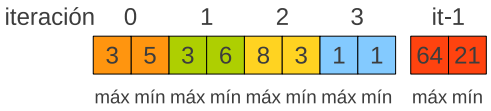

Figura 4.7: Almacenamiento de resultados parciales en la implementación GPU del método PPI.

Pasamos ahora a explicar el funcionamiento del kernel `PPI`. Como ya hemos comentado, pretendemos que cada hilo de ejecución realice una iteración del algoritmo, esto es, que calcule la proyección de todos los puntos de la imagen sobre un único *skewer* y obtenga los dos píxeles extremos. Para ello, además de las estructuras anteriormente comentadas, necesitamos dos estructuras más:

- La primera estructura es local a cada hilo y se almacena en memoria local, es decir, en los registros de la GPU. Se trata de una estructura de tipo vector con tantos elementos como bandas tenga la imagen hiperespectral, l. En esta estructura, el j-ésimo hilo cargará desde memoria global el j-ésimo *skewer* dentro de la estructura `d_Skewers` descrita anteriormente. Almacenamos esta estructura en memoria local por dos razones fundamentales. La primera es que los valores almacenados por cada hilo no van a ser necesitados por el resto de hilos, por tanto no ubicamos esta estructura en la memoria compartida. La segunda es que, a lo largo de la ejecución del algoritmo, se accede a esta estructura continuamente, concretamente se accede a cada elemento una vez por cada píxel de la imagen. Por este motivo, hacemos uso de la memoria local que es cientos de veces más rápida que la memoria global

de la GPU.

- La segunda estructura se almacena en memoria compartida. Se trata también de una estructura de tipo vector que tendrá capacidad para almacenar varios píxeles. Dependiendo de la arquitectura, el número de píxeles que podemos alojar cambia, ya que en el caso de *Tesla* el tamaño de la memoria compartida es de 16 KB y en el caso de *Fermi* el tamaño es de 48 KB. Como cada hilo necesita realizar el producto escalar por los mismos píxeles, es lógico hacer uso de una estructura compartida a la que puedan acceder todos los hilos de un bloque sin necesidad de que cada uno de ellos acceda a memoria global por separado. Además, esta memoria es también cientos de veces más rápida que la memoria global. Por tanto, se puede obtener una importante mejora en el rendimiento al hacer uso de ella. Por esta estructura irán pasando todos los píxeles de la imagen en grupos.

Teniendo en cuenta estas estructuras, el kernel `PPI` primero carga un *skewer* de `d_Skewers` y lo almacena en los registros y a continuación carga un grupo de píxeles y los almacena en la memoria compartida. Esta lectura de píxeles de la memoria global de la GPU se realiza de forma coalescente, lo cual permite mejorar el rendimiento en gran medida, sobre todo en el caso de la arquitectura *Tesla*. La coalescencia permite realizar lecturas o escrituras con un único acceso siempre que los datos se encuentren en un segmento de 32, 64 o 128 bytes. En nuestro caso cargamos de memoria píxeles cuyos valores en la misma banda se encuentran en posiciones consecutivas. Al tratarse la memoria global de una memoria lenta en comparación con la compartida o la local, el hecho de reducir el número de accesos supone una mejora en el rendimiento. A la hora de escribir en la memoria compartida lo hacemos accediendo cada hilo a un banco distinto, con el fin de que cada escritura pueda realizarse en paralelo.

Una vez que tenemos almacenados el *skewer* y un grupo de píxeles en un registro y en la memoria compartida, respectivamente, el siguiente paso que aplica el kernel `PPI` es calcular el producto escalar de cada píxel con el *skewer*. Cada hilo utiliza dos variables: `pemin` y `pemax`, que almacenan los valores máximo y mínimo obtenidos en el producto escalar, y otras dos variables: `imax` e `imin` que almacenan el índice (posición en la imagen) de los píxeles que han obtenido esos valores máximo y mínimo. Cuando finaliza la proyección de un grupo de píxeles se carga otro grupo y se repite el proceso hasta que se hayan procesado todos los píxeles de la imagen. Finalmente, el j-ésimo hilo almacena en la posición $j \times 2$ y $j \times 2 + 1$ de la estructura `d_res_parcial` los valores `imax` e `imin`, respectivamente, tal y como aparece ilustrado en la Fig. 4.7.

Por último, tiene lugar una fase de agrupación de resultados que se realiza en la CPU, ya que hemos comprobado experimentalmente que las operaciones involucradas no son costosas y se pueden efectuar de forma eficiente sin recurrir a la GPU. Esta operación genera una denominada imagen de pureza, es decir, una imagen que almacena, para cada píxel de la imagen hiperespectral $\mathbf{Y}$, el número de veces que ha sido seleccionado como extremo mediante el algoritmo PPI. Para generar esta imagen de pureza, se recorre la estructura `h_res_parcial` (que es simplemente una copia de `d_res_parcial`) y se copia en la memoria de la CPU, incrementando en una unidad el contador asociado a los

píxeles que aparecen en `h_res_parcial`. El proceso aparece ilustrado de forma gráfica en la Fig. 4.8. Como podemos ver en la figura, el píxel 1 (color azul en la Fig. 4.8) aparece 2 veces en `h_res_parcial` y por tanto su contador asociado se incrementa 2 veces en la estructura **P**, que contiene los índices de pureza asociados a cada píxel. Por su parte, el píxel 3 (color naranja en la Fig. 4.8) aparece 3 veces en `h_res_parcial` y por tanto su contador asociado se incrementa 3 veces en **P**. Del mismo modo, hay píxeles que no aparecen en `h_res_parcial`, como es el caso del píxel 0 (color blanco en la Fig. 4.8). En este caso, su contador asociado en **P** simplemente no se incrementa.

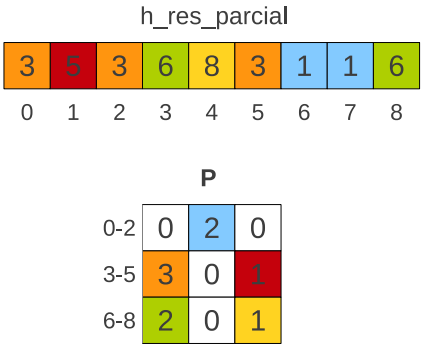

Figura 4.8: Agrupación de los resultados en el método PPI.

4.3.2. N-FINDR

- **Entradas**: Imagen hiperespectral reducida con $p-1$ componentes, siendo p el número de *endmembers* que se desea que el método extraiga.
- **Salidas**: Conjunto de p *endmembers*.

Antes de comenzar con la descripción de la implementación del método N-FINDR, descrito en el apartado 2.3.3.2 de la presente memoria, es importante destacar que se han realizado una serie de optimizaciones matemáticas al método original, dando como resultado una mejora de rendimiento incluso en la implementación serie. La parte que más tiempo consume del algoritmo N-FINDR es el cálculo de los determinantes necesarios para la estimación de volumen en la expresión (2.9). El determinante de una matriz no singular **V** se puede obtener mediante una factorización $\mathbf{PV} = \mathbf{LU}$ (donde **P** es una matriz de permutaciones, **L** es una matriz triangular inferior y **U** es una matriz triangular superior), multiplicando los elementos de la diagonal de **U**. Esta descomposición se conoce como *eliminación gaussiana* o factorización **LU** (con pivotamiento parcial de filas). Los cálculos de volumen repetidos en el algoritmo N-FINDR se pueden se pueden reducir haciendo uso de algunas propiedades de la factorización **LU** y de los determinantes de matrices. Consideremos por ejemplo las siguientes matrices de tamaño $p \times p$ y $p \times p-1$,

respectivamente:

$$\begin{aligned} V_{\mathbf{B}}^{(1)} &= \begin{bmatrix} 1 & \dots & 1 & 1 \\ \mathbf{m}_2^{(0)} & \dots & \mathbf{m}_p^{(0)} & \mathbf{y}_j \end{bmatrix} \text{ y} \\ \bar{V}_{\mathbf{B}}^{(1)} &= \begin{bmatrix} 1 & \dots & 1 \\ \mathbf{m}_2^{(0)} & \dots & \mathbf{m}_p^{(0)} \end{bmatrix}, \end{aligned} \tag{4.6}$$

donde $\mathbf{B}$ es una versión reducida de la imagen hiperespectral $\mathbf{Y}$ con $p-1$ componentes, obtenida por ejemplo utilizando la transformación PCA ó SPCA. Supongamos que hemos calculado la factorización $\mathbf{LU}$ (con pivotamiento parcial) $\mathbf{P_B}\bar{V}_{\mathbf{B}}^{(1)} = \mathbf{L_B U_B}$.

Teniendo esto en cuenta, la factorización $\mathbf{LU}$ (con pivotamiento parcial) de $V_{\mathbf{B}}^{(1)}$ viene dada por $\mathbf{P_B}V_{\mathbf{B}}^{(1)} = [\mathbf{U_B}(L_B^{-1}P_B^T\mathbf{y}_j)]$. Por lo tanto, las factorizaciones $\mathbf{LU}$ requeridas en el cálculo de los volúmenes del algoritmo N-FINDR pueden realizarse formando la matriz $p \times n$ $\hat{\mathbf{B}} = \begin{bmatrix} 1\ 1\ \dots\ 1 \\ \mathbf{B}^T \end{bmatrix}$, donde $\mathbf{B}$ es la imagen hiperespectral reducida dimensionalmente. Entonces necesitamos calcular $\mathbf{L_B^{-1}P_B^T}\hat{\mathbf{B}}$, lo cual podemos llevar a cabo en la GPU mediante el uso de el kernel `calcularVolumenes` que calcula el volumen obtenido por cada píxel en una iteración, esto es el volumen que resulta de reemplazar cada píxel por el mismo *endmember* de el conjunto de *endmembers* actual. Los n volúmenes requeridos en cada iteración se obtienen multiplicando el determinante de $\mathbf{U_B}$ por cada uno de los elementos de la última fila de $\mathbf{L_B^{-1}P_B^T}\hat{\mathbf{B}}$. Posteriormente, mediante el uso de el kernel `maxVolumen`, obtenemos el valor del volumen máximo y las coordenadas del píxel que produce dicho volumen. El proceso detallado se describe a continuación paso a paso:

- Primeramente formamos la matriz inicial $\bar{V}_{\mathbf{B}}^{(1)}$ de tamaño $p \times p$, inicializando a valores unitarios la primera fila y situando en cada columna (a partir de la segunda fila) un *endmember* inicial seleccionado aleatoriamente.

- Calculamos el determinante de esta matriz $\bar{V}_{\mathbf{B}}^{(1)}$ y almacenamos el valor en una variable denominada `volumenActual`. Al tratarse de una matriz de dimensiones pequeñas, el determinante puede calcularse de forma eficiente en la CPU.

- A continuación formamos un vector de tamaño n (número de píxeles de la imagen hiperespectral $\mathbf{Y}$) denominado `Vvolumenes` donde, en la k-ésima iteración, se almacena en la i-ésima posición el volumen conseguido al sustituir el i-ésimo píxel por un *endmember*. También modificamos la imagen reducida de entrada $\mathbf{B}$ insertando una primera banda formada exclusivamente por valores unitarios, obteniendo $\hat{\mathbf{B}}$.

- En cada iteración k cambiamos en la matriz $\bar{V}_{\mathbf{B}}^{(1)}$ el *endmember* en la k-ésima posición por el *endmembers* en la p-ésima posición. Además, sustituimos la p-ésima columna por una columna de tipo $[0, \cdots, 1]^T$.

- A continuación realizamos la factorización $\mathbf{LU}$ de esta matriz, obteniendo $\mathbf{L_B}$, $\mathbf{U_B}$ y $\mathbf{P_B}$. Seguidamente, calculamos el determinante de $\mathbf{U_B}$ e invertimos la matriz $\mathbf{L_B}$. Al tratarse de matrices pequeñas y triangulares, el cálculo del determinante como

el de la inversa puede realizarse en CPU sin apenas penalización en el cómputo del tiempo total.

- Llegados a este punto, tenemos los elementos que necesitamos para calcular los volúmenes conseguidos en una iteración del algoritmo N-FINDR. Recordemos que éstos se obtienen al multiplicar el determinante de $\mathbf{U_B}$ por cada una de las entradas de la ultima fila de $\mathbf{L_B^{-1}P_B^T\hat{B}}$. Dividiremos pues estos cálculos en dos etapas: en una primera etapa multiplicamos en CPU las matrices $\mathbf{S} = \mathbf{L_B^{-1}P_B^T}$. La segunda etapa es más costosa y se realiza en la GPU, haciendo uso del kernel `calcularVolumenes`.

- Como ya sabemos, los volúmenes se obtienen al multiplicar el determinante de $\mathbf{U_B}$ por los elementos en la última fila de $\mathbf{L_B^{-1}P_B^T\hat{B}}$ o lo que es lo mismo de $\mathbf{S\hat{B}}$. El resultado de esta operación es una matriz de tamaño $p \times n$ de la que sólo necesitamos la última fila. Es por esto que podemos ahorrarnos los cálculos para obtener las primeras $p-1$ filas multiplicando sólo la ultima fila de $\mathbf{S}$ por toda la matriz $\mathbf{\hat{B}}$.

El funcionamiento del kernel `calcularVolumenes` es el siguiente. Partimos de una configuración en la que se lanzan tantos hilos como píxeles tiene la imagen hiperespectral, es decir n hilos, maximizando el número de hilos por bloque en función de la arquitectura considerada (*Tesla* o *Fermi*). El i-ésimo hilo calcula el i-ésimo elemento del vector de volúmenes `Vvolumenes`, multiplicando cada elemento de la última fila de la matriz $\mathbf{S}$ por la i-ésima columna de $\mathbf{\hat{B}}$ y sumando los resultados. Por tanto, parece buena idea almacenar la última fila de $\mathbf{S}$ en la memoria compartida de cada multiprocesador y así evitar que los hilos accedan continuamente a las mismas posiciones de la memoria global. Tras esta operación se realiza el proceso de multiplicación propiamente dicho, obteniendo cada hilo un valor que almacena en su memoria local. El último paso es multiplicar éste valor por el determinante de $\mathbf{U_B}$ y almacenar el resultado en la i-ésima posición del vector `Vvolumenes` (en memoria global). Este proceso aparece ilustrado de forma gráfica mediante un ejemplo en la Fig. 4.9.

Una vez calculados los volúmenes para una iteración, el siguiente paso es determinar cuál ha sido el píxel que ha generado el mayor volumen y ver si éste es mayor que el volumen actual. Para ello hacemos uso de un nuevo kernel `maxVolumen`. Este kernel se configura maximizando el número de hilos por bloque, de manera que tengamos tantos hilos como elementos en el vector `Vvolumenes`. El kernel realiza un proceso de reducción en el que cada bloque reduce un fragmento del vector y obtiene el valor máximo local, así como la posición de ese valor dentro del vector. El proceso de reducción se explica detalladamente en la sección 4.1.1. Como cada bloque obtiene un valor propio, al final de la ejecución tendremos tantos valores como bloques (cada uno es el máximo local a su fragmento), y por tanto será necesario almacenar estos valores junto con su posición correspondiente en memoria global para posteriormente copiarlos en memoria principal de CPU y obtener el máximo global junto con su posición. Este proceso aparece ilustrado de forma gráfica en la Fig. 4.10, donde podemos ver cómo cada bloque se encarga de encontrar un máximo local junto con su posición, se copian estos elementos a la memoria principal y finalmente la CPU se encarga de encontrar el máximo global, procesando eso sí una estructura de tamaño j, donde j el número de bloques lanzados ($j << n$). Para

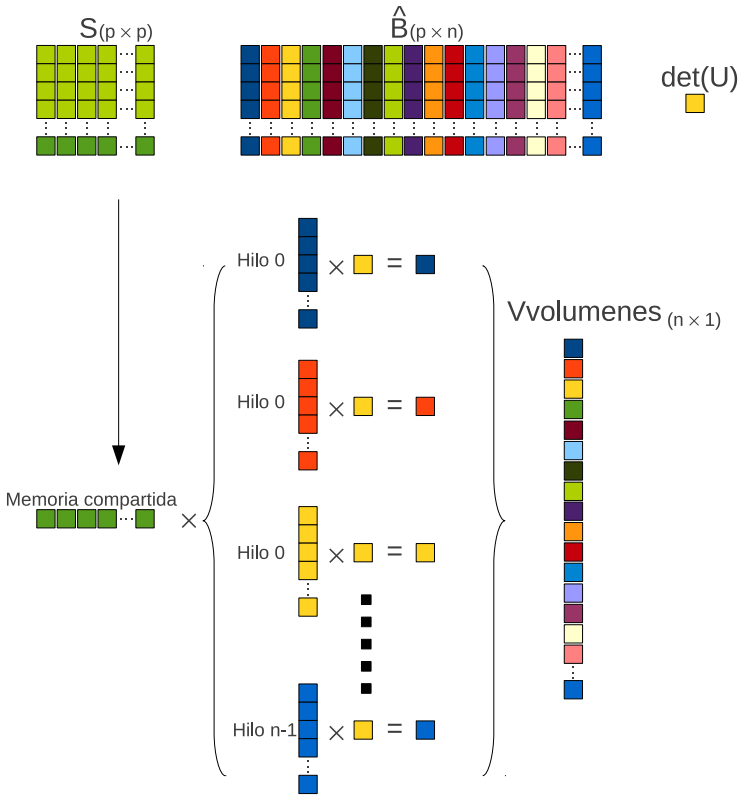

Figura 4.9: Cálculo de volúmenes en el kernel `calcularVolumenes` de la implementación GPU de N-FINDR.

finalizar cada iteración k, se compara el volumen actual con el obtenido en la iteración, si el segundo es mayor se sustituye el k-ésimo *endmember* por el píxel que generó dicho volumen. Además este volumen pasa a ser el volumen actual, repitiéndose este proceso hasta que se alcanza la condición de convergencia.

4.4. Estimación de abundancias

En este apartado se describe la implementación GPU de tres métodos: UCLS (descrito en el subapartado 2.3.4.1), NCLS (descrito en el subapartado 2.3.4.2) e IEA (descrito en el subapartado 2.3.4.3). Para cada método se indican en primer lugar las entradas y las salidas del mismo, y a continuación se proporciona una descripción de los pasos realizados para llevar a cabo su implementación eficiente en arquitecturas GPU.

4.4.1. *Unconstrained least squares* (UCLS)

- **Entradas**: Imagen hiperespectral $\mathbf{Y}$ y conjunto de p *endmembers*.
- **Salidas**: Estimación de la abundancia de cada *endmember* en cada píxel.

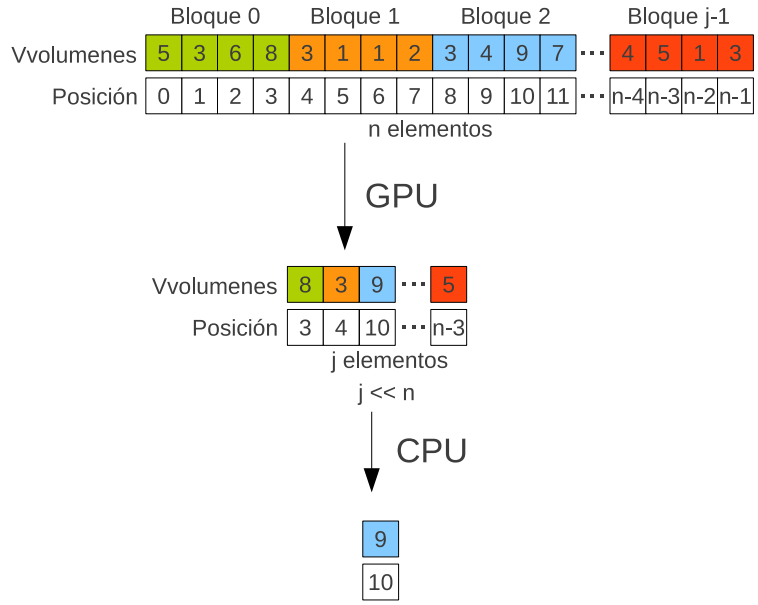

Figura 4.10: Búsqueda del máximo volumen en el kernel `maxVolumen` de la implementación GPU de N-FINDR.

Como se mostró en el subapartado 2.3.4.1, el estimador que resuelve el problema en la expresión (2.10) para un determinado píxel de la imagen hiperespectral $\mathbf{Y}$ sin aplicar ninguna restricción viene dado por la expresión (2.11). Para simplificar la presentación de la implementación descrita en el presente subapartado, podemos transformar la expresión (2.11) a un formato matricial, en el que $\mathbf{M} = \{\mathbf{m}_i\}_{i=1}^{p}$ denota una matriz que contiene los p *endmembers* identificados por un método como N-FINDR y $\mathbf{X}$ es una estructura p-dimensional que contiene las abundancias de cada uno de los p endmembers para cada píxel de la imagen, de la siguiente forma:

$$\mathbf{X} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{Y}^T, \tag{4.7}$$

donde $\mathbf{X}$ es la matriz de abundancias, $\mathbf{M}$ es la matriz de *endmembers* e $\mathbf{Y}$ es la imagen hiperespectral original. Dado el carácter matricial del estimador, una posible implementación consistiría en utilizar `cuBLAS` para todas las multiplicaciones de matrices y un proceso serie para resolver la inversa presente en el proceso. En nuestro caso se ha comprobado experimentalmente que resulta más eficiente realizar las primeras multiplicaciones en serie haciendo uso de CPU, así como la inversa, y utilizar un kernel llamado `ucls` para la multiplicación final de mayor tamaño que involucra a la imagen hiperespectral original. Así pues el proceso se divide en dos partes: en la primera se calcula en la CPU lo que denominamos como *matriz de cómputo*, una matriz de tamaño $p \times l$ que se obtiene como resultado de calcular $(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T$ y que es la misma para el calculo de las abundancias de todos los píxeles. La segunda parte es la llamada al kernel `ucls`, cuyo funcionamiento se explica a continuación.

La configuración de la llamada al kernel `ucls` se realiza con tantos hilos como píxeles haya en la imagen, es decir n hilos. Se maximiza el número de hilos por bloque en función

de lo que nos permita la arquitectura de la GPU (*Tesla* ó *Fermi*). La idea es multiplicar cada fila de la *matriz de cómputo* por cada píxel $\mathbf{y}_i$ de la imagen hiperespectral $\mathbf{Y}$, y así obtener un vector de abundancias $\mathbf{x}_i$ por cada píxel, de manera que al agruparlos todos en una matriz $\mathbf{X}$ podamos formar los mapas de abundancias. Por tanto, como cada hilo va a utilizar todas las filas de la *matriz de cómputo*, parece lógico almacenar ésta en la memoria compartida de los multiprocesadores. El primer paso del kernel `ucls` es realizar una lectura de tantas filas de la *matriz de cómputo* como podamos almacenar en la memoria compartida (el número de filas varía con la arquitectura GPU, ya que el tamaño de esta memoria es distinto). Una vez realizado este paso, el i-ésimo hilo almacenará en su memoria local (registros del multiprocesador) el i-ésimo píxel de la imagen hiperespectral, $\mathbf{y}_i$. Si suponemos que la imagen se almacena banda a banda (es decir, cada banda es una columna de la matriz $\mathbf{Y}$), al acceder todos los hilos a los valores en la misma banda se produce coalescencia en dichos accesos, lo cual permite incrementar notablemente el rendimiento de la implementación.

El siguiente paso aplicado por el kernel `ucls` es multiplicar una fila de la *matriz de cómputo* por el píxel y almacenar el resultado en la matriz $\mathbf{X}$. Como ésta matriz también se organiza por bandas, podemos obtener de nuevo accesos coalescentes. El proceso se repite con cada fila de la *matriz de cómputo* hasta que todas las filas han sido procesados. El funcionamiento del kernel `ucls` se ilustra gráficamente mediante un ejemplo en la Fig. 4.11. Conviene reiterar que la utilización de un kernel en nuestra implementación en lugar de la función de multiplicación de matrices `cublasSgemm` de `cuBLAS` no supone ninguna penalización (más bien al contrario, debido a las optimizaciones aplicadas).

4.4.2. *Non-negative constrained least squares* (NCLS)

- **Entradas**: Imagen hiperespectral $\mathbf{Y}$ y conjunto de p *endmembers*.
- **Salidas**: Estimación de la abundancia de cada *endmember* en cada píxel.

Como se mostró en el subapartado 2.3.4.2, el algoritmo NCLS sigue un proceso iterativo para calcular las abundancias de los p *endmembers* proporcionados por un método como N-FINDR en cada píxel $\mathbf{y}_i$ de la imagen hiperespectral $\mathbf{Y}$ utiliando la expresión (2.12), que aplicada iterativamente da como resultado abundancias que satisfacen la condición de no negatividad. Esta expresión se puede transformar a formato matricial de la siguiente forma:

$$\mathbf{X}^{k+1} = \mathbf{X}^k \otimes \frac{\mathbf{YM}}{\mathbf{X}^k\mathbf{M}^T\mathbf{M}}, \tag{4.8}$$

donde $M = \{\mathbf{m}_i\}_{i=1}^{p}$ es la matriz de *endmembers*, $\mathbf{X}$ denota una estructura p-dimensional que contiene las abundancias estimadas para cada píxel de la imagen $\mathbf{Y}$ (y que se va actualizando de forma iterativa) y el símbolo $\otimes$ representa una multiplicación elemento a elemento y no una multiplicación matricial (el cociente también se realiza elemento a elemento). A continuación detallamos cómo se ha llevado a cabo la implementación en GPU del método NCLS conforme a la expresión (4.8). Para ello se han utilizado dos

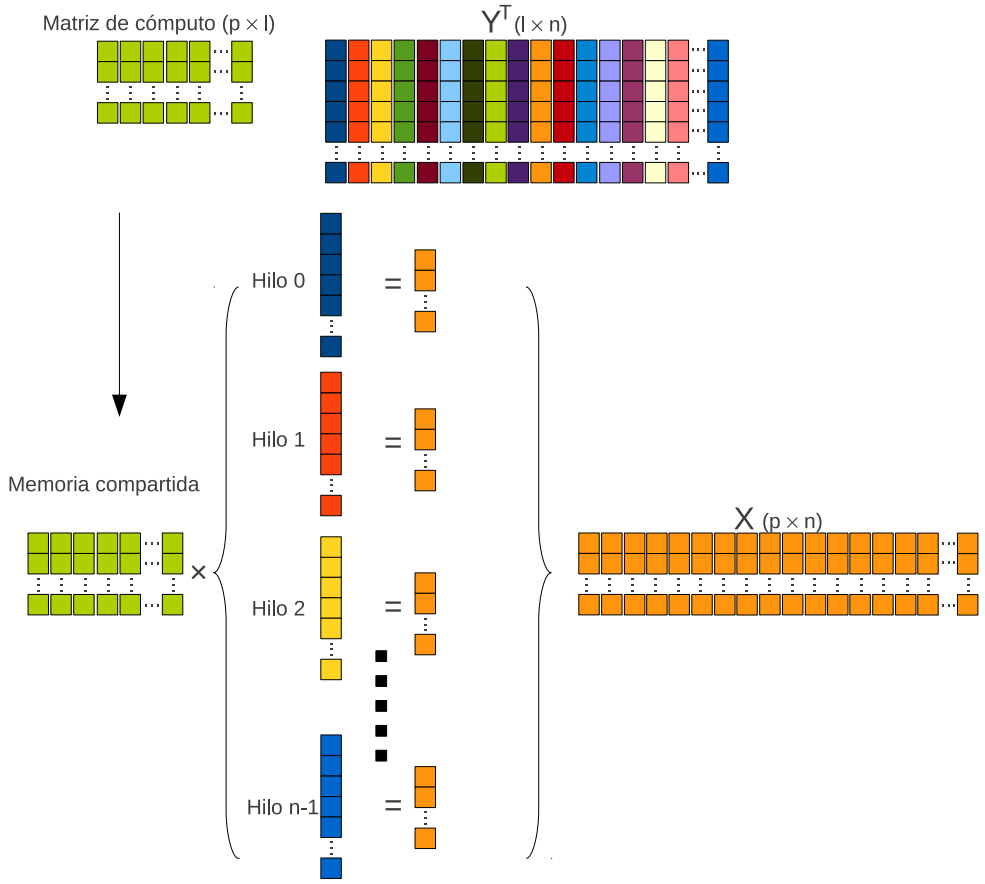

Figura 4.11: Cálculo de abundancias por parte del kernel `ucls` en la implementación GPU del método UCLS.

kernels y tres multiplicaciones matriciales haciendo uso de función `cublasSgemm` en la librería `cuBLAS`.

El primer paso en la implementación GPU de NCLS es inicializar la matriz de abundancias $\mathbf{X}$ para su utilización en la primera iteración. Teniendo en cuenta que esta matriz va a contribuir a los resultados del denominador de la expresión (4.8), hay que garantizar que sus valores sean mayores que cero ya que estos se arrastrarían con las multiplicaciones y al final tendríamos un cociente con denominador negativo o cero. Para ello, en esta primera iteración se inicializan todos los elementos de $\mathbf{X}$ a unos. De esto se encarga un kernel sencillo al que llamamos `inicializarX`. El i-ésimo hilo del kernel se encarga de asignar el valor 1 al i-ésimo elemento de $\mathbf{X}$. El kernel se configura con tantos hilos como elementos haya en $\mathbf{X}$ es decir, $n \times p$, maximizando el número de hilos por bloque dependiendo de la arquitectura GPU considerada.

Una vez inicializados los valores de las abundancias, comenzamos el proceso iterativo no sin antes comentar un detalle importante. En el proceso iterativo la parte del numerador $\mathbf{YM}$ de la expresión (4.8) permanece invariante en cada iteración, por lo que se puede calcular antes de comenzar a iterar y utilizar su valor (constante) en cada

iteración. El cálculo del numerador se realiza mediante una operación de multiplicación de matrices, para lo que hacemos uso de la función `cublasSgemm` en la librería `cuBLAS`. Tenemos entonces que $\mathbf{Num} = \mathbf{YM}$ es constante, donde **Num** es el numerador de la expresión (4.8). Durante el proceso iterativo, se calcula iterativamente el denominador de la expresión (4.8). Este cálculo se lleva a cabo de nuevo utilizando la librería `cuBLAS`, realizando primero la operación $\mathbf{Aux} = \mathbf{XM}^T$ y seguidamente la operación $\mathbf{Den} = \mathbf{AuxM}$, donde **Den** hace referencia al denominador de la expresión (4.8). Finalmente, se actualizan los valores de las abundancias en **X** mediante la operación $\mathbf{X}^{k+1} = \mathbf{X}^k \otimes \frac{\mathbf{Num}}{\mathbf{Den}}$. Para realizar esta tarea se utiliza un kernel `actualizarX` que se configura con tantos hilos como elementos haya en **X**, maximizando el número de hilos por bloque como siempre en función de lo que permita cada arquitectura. Así pues, el i-ésimo hilo actualiza el i-ésimo elemento de la matriz **X** multiplicando éste por el i-ésimo elemento de la matriz **Num** y dividiendo por el i-ésimo elemento de la matriz **Den**.

Una vez actualizadas las abundancias es preciso comprobar si se cumple la condición de parada. En nuestro caso, se han implementado dos condiciones de parada. La primera de ellas es muy simple, y consiste en utilizar un número de iteraciones fijo que se considera suficiente para que las abundancias sean lo más correctas posible. El segundo criterio de parada reconstruye la la imagen original (utilizando los *endmembers* en **M** y las abundancias estimadas en la iteración actual) y calcula el error en la reconstrucción. Si este error es menor que un umbral establecido, entonces se cumple la condición de parada. El cálculo del error de reconstrucción se efectúa de forma paralela en la GPU. Este proceso se describe en la implementación del algoritmo IEA, la cual se muestra a continuación.

4.4.3. *Iterative error analysis* (IEA)

- **Entradas**: Imagen hiperespectral **Y** y número de *endmembers* que se desea que el método extraiga, p.
- **Salidas**: Conjunto de p *endmembers* y estimación de la abundancia de cada *endmember* en cada píxel.

Como se mostró en el subapartado 2.3.4.3, el método IEA se divide en una serie de pasos claramente diferenciados, concretamente: inicialización, cálculo del *endmember* inicial y proceso iterativo con la condición de parada. En nuestra implementación GPU, la fase de inicialización se efectúa mediante la expresión (2.14), que calcula el vector promedio de la imagen hiperespectral, $\bar{Y}$, mediante un kernel llamado `pixelMedio` en el cual se utilizan tantos bloques como bandas haya en la imagen hiperespectral, maximizando el número de hilos por bloque en función de la arquitectura. El i-ésimo bloque se encarga de calcular el valor medio de la i-ésima banda, para lo cual se lleva a cabo un proceso de reducción de los valores de cada banda exactamente igual al que se realiza en el kernel `centrarDatos` utilizado en la implementación GPU del algoritmo VD. La sección 4.1.1 . contiene una explicación detallada de este proceso. Una vez que se ha extraído el píxel medio, el siguiente paso es encontrar el primer *endmember*, para lo cual se efectúa en primer lugar el desmezclado de la imagen hiperespectral **Y** mediante el

método UCLS, utilizando únicamente el vector promedio $\bar{Y}$ como *endmember*. Llegados a este punto, es importante comentar que se podría utilizar alternativamente el método NCLS para dicho proceso, pero hemos comprobado experimentalmente que la utilización de uno u otro método no tiene un impacto significativo en la selección de *endmembers* por parte del método IEA, y el método UCLS es más rápido que NCLS.

Utilizando las abundancias estimadas, realizamos la reconstrucción por mínimos cuadrados de la imagen original y calculamos el píxel que resulta en mayor error de error de reconstrucción, etiquetándolo como el perimer *endmember*. Esta operación se lleva a cabo mediante la utilización del kernel `reduccionError`, que se configura con tantos hilos como píxeles haya en la imagen hiperespectral (maximizando el número de hilos por bloque en función de lo que nos permita cada arquitectura). En un primer paso, el kernel proporciona los errores de reconstrucción en una estructura en la que el i-ésimo elemento representa el error que se ha producido al reconstruir el $\mathbf{y}_i$. Para ello, cada hilo se encarga de calcular un elemento tomando los valores de la imagen original y de la reconstruida banda a banda, restándolos y elevando el resultado al cuadrado. El resultado se va acumulando hasta procesar todas las bandas. Finalmente, se almacena el resultado en la posición correspondiente de la estructura. Los valores de la misma banda para píxeles consecutivos están en posiciones consecutivas de la memoria global, por lo que los accesos a dichos elementos son coalescentes. Este proceso se ilustra gráficamente mediante un ejemplo en la Fig. 4.12.

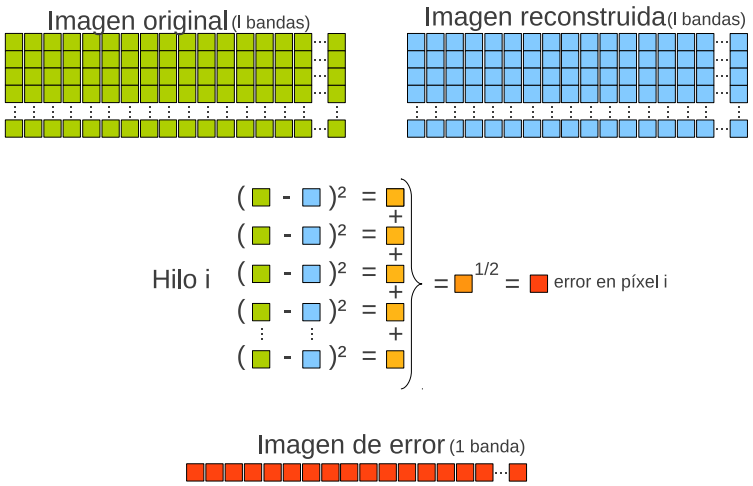

Figura 4.12: Cálculo eficiente de los errores de reconstrucción en el kernel `reduccionError` de la implementación GPU de IEA.

Tras obtener los errores de reconstrucción para todos los píxeles de la imagen $\mathbf{Y}$ se procede a extraer la posición del píxel con mayor error de reconstrucción. Para ello, cada bloque de hilos procesa un fragmento de la imagen (exactamente de la misma forma que el kernel `maxVolumen` de la implementación GPU del algoritmo N-FINDR en la sección 4.3.2). Es decir, cada bloque obtiene el valor máximo y la posición que ocupa dicho valor máximo en su fragmento, se copia el resultado de cada bloque a la memoria principal de

CPU, y finalmente la CPU se encarga de encontrar el máximo global de entre todos los máximos locales. Éste valor determina cuál es el primer *endmember*. Una vez localizado dicho *endmember* se realiza un proceso iterativo en el que se añade el *endmember* al conjunto **M**, que va creciendo en cada iteración, y se repiten los pasos de estimación de las abundancias, reconstrucción de la imagen, cálculo de los errores de reconstrucción y extracción del píxel con mayor error hasta p *endmembers*. Conviene reiterar que, a lo largo del proceso (en cada iteración), se obtienen las abundancias asociadas al conjunto actual de *endmembers*. Por este motivo, el método IEA es capaz de proporcionar no solamente las firmas espectrales de los *endmembers* sino también sus abundancias asociadas, integrando dos etapas fundamentales de la cadena de desmezclado espectral lineal.

Capítulo 5

Resultados Experimentales

En este capítulo describimos los resultados obtenidos en la validación experimental de los algoritmos e implementaciones paralelas propuestas. En primer lugar, describimos las imágenes hiperespectrales reales que se han utilizado en el estudio. Dichas imágenes ofrecen la oportunidad de analizar escenarios reales en los que la aplicación de técnicas de identificación de *endmembers* y estimación de abundancias constituye un objetivo relevante. Seguidamente se realiza un estudio de precisión en la aplicación de técnicas de desmezclado, basado en una serie de métricas para determinar la calidad de los resultados obtenidos por los métodos propuestos. Para finalizar el capítulo se realiza un estudio del rendimiento computacional de las versiones serie y GPU de los métodos considerados, haciendo especial énfasis en la posibilidad de obtener resultados de análisis en tiempo real.

5.1. Imágenes hiperespectrales consideradas

En el presente trabajo se han utilizado dos imágenes hiperespectrales, ambas tomadas por el sensor AVIRIS y ampliamente utilizadas en aplicaciones de desmezclado espectral. A continuación se describen las imágenes hiperespectrales consideradas.

5.1.1. AVIRIS Cuprite

Esta imagen fue adquirida en 1995 por el sensor AVIRIS sobre la región minera denominada Cuprite, en el estado de Nevada, Estados Unidos. La Fig. 5.1 muestra la ubicación de la imagen sobre una fotografía aérea tomada en la zona. Visualmente, puede apreciarse la existencia de zonas compuestas por minerales, así como suelos desnudos y una carretera interestatal (I-95) que cruza la zona en dirección sur-norte, con dirección a la ciudad de Las Vegas, Nevada.

Las características de la imagen AVIRIS Cuprite están resumidas en la Tabla 5.1, la cual se encuentra disponible online[1] en datos de reflectancia, es decir, la imagen se encuentra corregida atmosféricamente mediante el método denominado *atmospheric removal* (ATREM). El rango espectral cubre la región de 0,4 a 2,5μm (es decir, parte

[1] http://aviris.jpl.nasa.gov/free-data

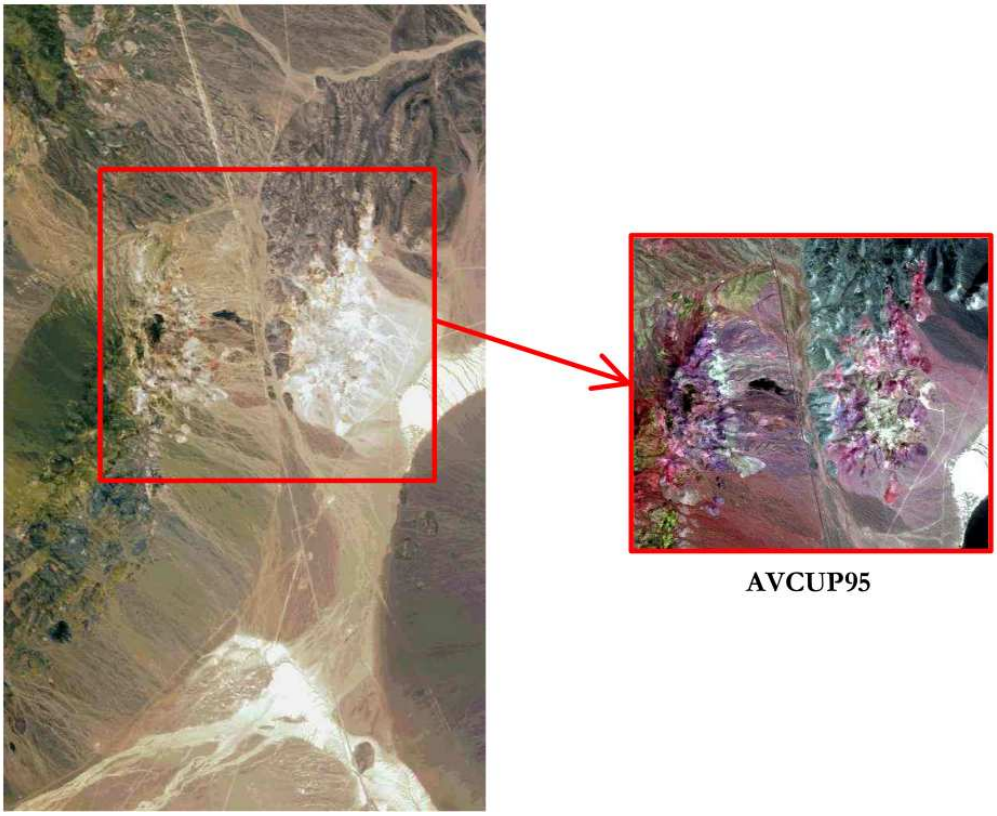

Figura 5.1: Ubicación de la imagen AVIRIS Cuprite, obtenida en 1995, sobre una fotografía aérea de alta resolución.

visible e infrarrojo cercano) del espectro. En particular, la región de 2 a 2,5 μm se denomina *showrt wave infra-red* (SWIR) y en ella se manifiestan singularidades que permiten discriminar entre una amplia gama de minerales de tipo calizo [11].

Líneas	350
Muestras	350
Bandas espectrales	188
Rango espectral	0.4 - 2.5 μm
Resolución espacial	20 metros/píxel
tamaño	44 MBytes aproximadamente

Tabla 5.1: Características de la imagen hiperespectral AVIRIS obtenida sobre la región minera de Cuprite, en el estado de Nevada.

La imagen AVIRIS Cuprite se ha utilizado extensamente en aplicaciones de desmezclado espectral. El motivo es la disponibilidad de información de referencia para la misma, obtenida por el Instituto Norteamericano de Estudios Geológicos ó U.S. Geological Survey (USGS en lo sucesivo). Esta información de referencia se encuentra disponible online[2] y consta de los siguientes elementos:

[2]http://speclab.cr.usgs.gov

1. Un conjunto de firmas espectrales de referencia, obtenidas a partir de estudios de campo y laboratorio realizados por expertos del USGS, utilizando un espectro-radiómetro Beckman con resolución radiométrica de 0,5 μm en el infrarrojo cercano y 0,2 μm en el visible [13]. La elevada resolución radiométrica de este instrumento ha motivado la creación de una librería espectral de firmas espectrales (convolucionadas con las longitudes de onda de AVIRIS) de forma que se pueda establecer una comparación realista entre los datos adquiridos por ambos instrumentos de medida en longitudes de onda comunes. La librería espectral (original y convolucionada) ha sido hecha pública por USGS [14]. La Fig. 5.2 muestra cinco firmas espectrales correspondientes a cinco de los minerales más representativos en la zona de estudio.

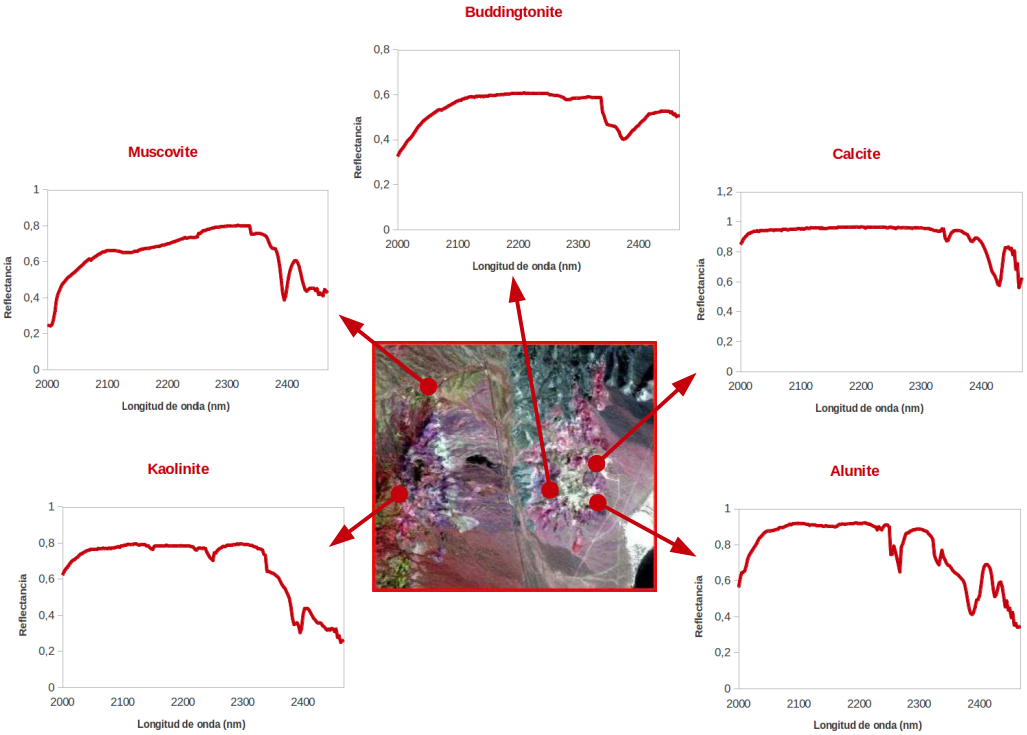

Figura 5.2: Firmas espectrales de referencia para cinco minerales representativos de la región Cuprite en Nevada.

2. Además, existe un mapa de clasificación para los minerales más característicos que pueden encontrarse en la zona, realizado por USGS [98]. En dicho mapa, se asigna un color diferente a cada píxel correspondiente a los diferentes minerales presente en la escena. Conviene destacar que estos mapas han sido obtenidos mediante el algoritmo `Tetracorder` [12] de USGS, utilizando las firmas espectrales contenidas en la librería USGS. Conviene destacar que este algoritmo no es un algoritmo de desmezclado sino de clasificación, por lo que no se dispone de mapas de abundancia de referencia que contengan las proporciones reales (a nivel sub-píxel) de los diferentes materiales en el terreno. En la validación, se utilizará la similaridad entre

los mapas de abundancia obtenidos por los algoritmos propuestos y los mapas de clasificación de referencia como indicador cualitativo de calidad de dichos mapas de abundancia. Por motivos ilustrativos, la Fig. 5.3 muestra el mapa de clasificación para todos los minerales de la escena, donde cada color representa un mineral. La zona marcada por una línea blanca se corresponde con la zona considerada en nuestros experimentos. Por otra parte, la Fig. 5.4 muestra cinco mapas que indican la presencia o ausencia de cinco minerales más representativos de la zona de estudio, donde los píxeles blancos indican la presencia del mineral y los píxeles en negro indican su ausencia. De nuevo, insistimos que estos mapas no contienen información acerca de la abundancia a nivel sub-píxel de minerales en la imagen ya que dicha información resulta muy difícil de obtener en la práctica.

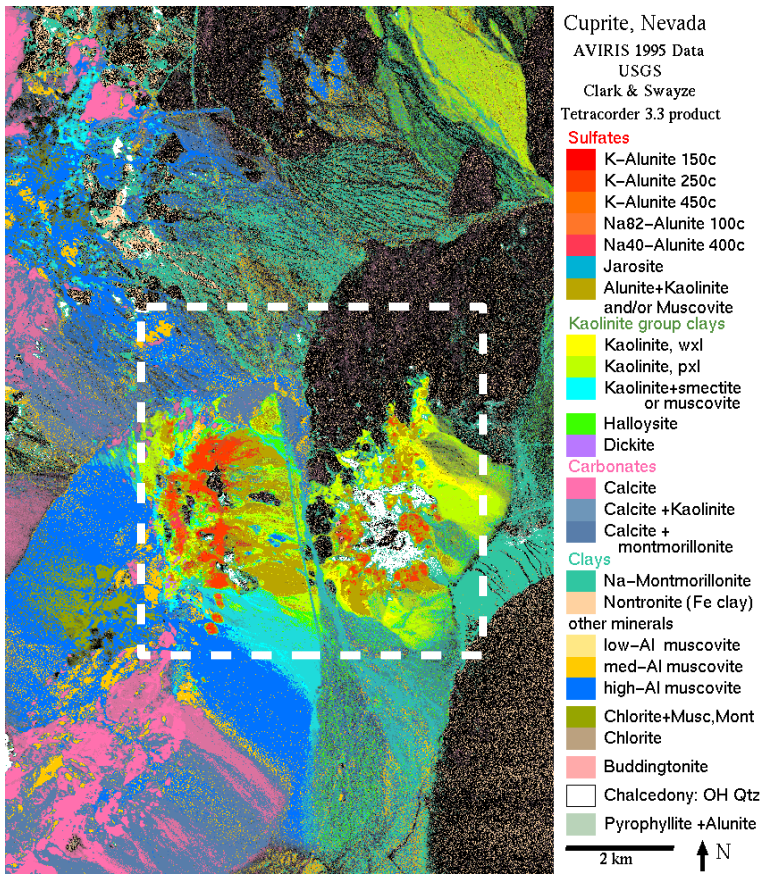

Figura 5.3: Mapa de clasificación de la imagen AVIRIS Cuprite (obtenido mediante el algoritmo `Tetracorder` de USGS).

5.1.2. AVIRIS World Trade Center

La segunda imagen utilizada en el presente trabajo fue adquirida por el sensor AVIRIS el día 16 de septiembre de 2001, justo cinco días después de los ataques terroristas que

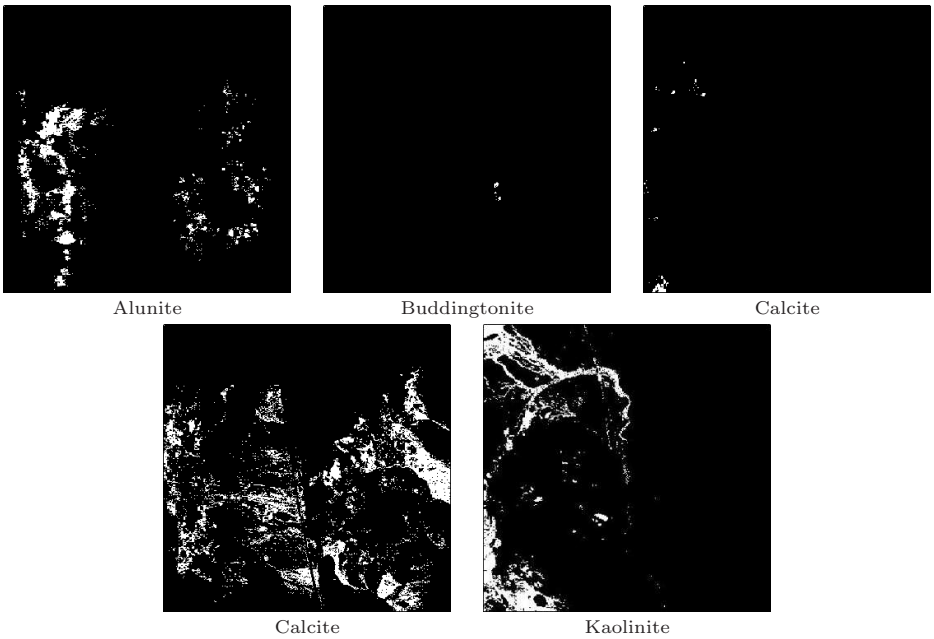

Figura 5.4: Mapas indicando la presencia o ausencia de cinco minerales representativos en la imagen AVIRIS Cuprite.

derrumbaron las dos torres principales y otros edificios del complejo *World Trade Center* (WTC) en la ciudad de Nueva York. Las características principales de la imagen aparecen resumidas en la Tabla 5.2. Conviene destacar que la resolución espacial de la imagen es muy elevada (1.7 metros por píxel) para lo que suele ser habitual en el caso de AVIRIS, en el que la resolución espacial habitual suele rondar los 20 metros por píxel. Esto se debe a que la imagen corresponde a un vuelo de baja altura, mediante el cual se pretendía obtener la mayor resolución espacial posible sobre la zona de estudio al contrario de otros estudios con el mismo sensor, en los que se pretende cubrir un área más extensa.

Líneas	614
Muestras	512
Bandas	224
Rango espectral	0,4 - 2,5 μm
Resolución espacial	1.7 metros/píxel
tamaño	150 MBytes aproximadamente

Tabla 5.2: Características de la imagen hiperespectral AVIRIS obtenida sobre la zona del World Trade Center en la ciudad de Nueva York, cinco días después del atentado terrorista del 11 de Septiembre de 2001.

La Fig. 5.5 muestra una composición en falso color de la imagen AVIRIS WTC. En dicha composición, utilizamos las bandas espectrales localizadas en 1682, 1107 y 655 nanómetros como rojo, verde y azul, respectivamente. En función de la composición en falso color utilizada, las zonas de la imagen con predominancia de vegetación reciben tonalidades verdes, mientras que las zonas con fuegos aparecen con tonos rojos. El humo

que proviene del área del WTC (enmarcada en el rectángulo de color rojo) y que se dirige al sur de la isla de Manhattan recibe un tono azul claro en la composición de falso color, debido a la elevada reflectancia del humo en la longitud de onda correspondiente a 655 nanómetros.

Para finalizar este subapartado, conviene destacar que la imagen AVIRIS WTC se encuentra disponible en unidades de radiancia, lo cual quiere decir que la imagen no ha sido corregida atmosféricamente. Esta situación nos permite analizar un escenario de procesamiento a bordo, en el cual los datos son analizados inmediatamente a continuación de ser adquiridos por el sensor sin aplicar ningún proceso de corrección atmosférica a bordo. En este caso concreto, la tarea de discriminación de *endmembers* relevantes como el de fuego, junto con sus mapas de abundancia asociados, constituyen un objetivo que debe ser realizado en tiempo real pero que resulta bastante difícil en este caso concreto debido, principalmente, a la complejidad del fondo, que comprende numerosas sustancias espectrales distintas como cabría esperar en un paisaje de tipo urbano.

Figura 5.5: Composición en falso color de la imagen hiperespectral AVIRIS obtenida sobre la zona del WTC en la ciudad de Nueva York, cinco días después del atentado terrorista del 11 de Septiembre de 2001. El recuadro en rojo marca la zona donde se sitúa el WTC en la imagen.

5.2. Métricas comparativas utilizadas

En el presente apartado describimos las métricas comparativas utilizadas para evaluar la precisión en el desmezclado de las cadenas consideradas. Las métricas consideradas son dos: el ángulo espectral ó *spectral angle distance* (SAD) y el error cuadrático medio ó *root mean square error* (RMSE) [40]. A continuación se describen en detalle dichas medidas.

5.2.1. Ángulo espectral ó *spectral angle distance* (SAD)

El ángulo espectral mide la similaridad espectral de dos firmas espectrales en términos del ángulo definido por sus vectores representativos en el espacio l-dimensional. Para describir esta medida en términos matemáticos, consideramos dos firmas espectrales $\mathbf{y}_i$ e $\mathbf{y}_j$, que pueden venir asociadas a píxeles de una imagen hiperespectral $\mathbf{Y}$. Si expresamos estos píxeles de forma completa como $\mathbf{y}_i = [y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(l)}]$ e $\mathbf{y}_j = [y_j^{(1)}, y_j^{(2)}, \ldots, y_j^{(l)}]$, donde el término $y_i^{(k)}$ hace referencia al k-ésimo valor espectral del píxel $\mathbf{y}_i$ y el término $y_j^{(k)}$ hace referencia al k-ésimo valor espectral del píxel $\mathbf{y}_j$, siendo $k \in \{1, 2, \ldots, l\}$, el SAD entre $\mathbf{y}_i$ e $\mathbf{y}_j$ viene dado por el arco coseno del ángulo espectral formado por dichos píxeles en el espacio l-dimensional, es decir:

$$\mathrm{SAD}(\mathbf{y}_i, \mathbf{y}_j) = \arccos \frac{\mathbf{y}_i \cdot \mathbf{y}_j}{\|\mathbf{y}_i\| \cdot \|\mathbf{y}_j\|} = \arccos \frac{\sum_{k=1}^{l} y_i^{(k)} \cdot y_j^{(k)}}{\sqrt{\sum_{k=1}^{l} y_i^{(k)^2}} \cdot \sqrt{\sum_{k=1}^{l} y_j^{(k)^2}}} \tag{5.1}$$

A continuación, destacamos algunas características interesantes relacionadas con esta medida:

1. El valor de la medida SAD suele venir dada por un valor entre 0 y $\frac{\pi}{2}$, ya que las firmas espectrales de una imagen suelen venir dadas por valores positivos, independientemente de si se encuentran expresadas en radiancia (radiación obtenida por el sensor) o reflectancia (resultado de un proceso de corrección atmosférica sobre los datos).

2. Por otra parte, la medida SAD es invariante frente a la multiplicación de los vectores $\mathbf{y}_i$ e $\mathbf{y}_j$ por valores constantes. De este modo, se trata de una medida robusta frente a cambios en la escala de las firmas espectrales debidos a diferentes condiciones de iluminación de la escena, así como a divergencias en la orientación angular, aspectos que la convierten en una medida muy apropiada para analizar el grado de similitud de dos firmas espectrales independientemente de la magnitud de las mismas (la cual está directamente relacionada con las condiciones de iluminación presentes en la escena).

5.2.2. Error cuadrático medio ó *root mean square error* (RMSE)

El error RMSE se ha utilizado habitualmente para medir el error que se produce al reconstruir la imagen en base a los endmembers extraídos y a las abundancias estimadas. En nuestro caso, el error RMSE se mide multiplicando la matriz de *endmembers* identificados, $\mathbf{M}$, por la matriz de abundancias estimadas, $\mathbf{X}$, de forma que obtenemos

una versión aproximada $\hat{\mathbf{Y}}$ de la imagen original. Seguidamente, se calcula el error de reconstrucción utilizando la siguiente expresión, obteniendo un valor para cada píxel de la imagen original:

$$\mathrm{RMSE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=1}^{l} [\mathbf{y}_i^{(k)} - \hat{\mathbf{y}}_i^{(k)}]^2 \right)^{1/2}, \tag{5.2}$$

donde n es el número de píxeles de la imagen hiperespectral, l es el número de bandas, $\mathbf{y}_i^{(k)}$ representa la k-ésima banda del píxel $\mathbf{y}_i$ de la imagen hiperespectral original, e $\hat{\mathbf{y}}_i^{(k)}$ representa la k-ésima banda del píxel $\hat{\mathbf{y}}_i$ de la imagen reconstruida, siendo $k \in \{1, 2, \ldots, l\}$. Teniendo en cuenta esta definición, podemos obtener el error de reconstrucción para cada píxel de la imagen y el RMSE total para la imagen completa. Obviamente, los píxeles con error de reconstrucción más próximos a cero estarán mejor representados por la combinación lineal del conjunto de *endmembers* extraídos, $\mathbf{M}$, y sus correspondientes abundancias, $\mathbf{X}$.

5.3. Precisión de las técnicas de desmezclado

Una vez descritas las imágenes utilizadas en los experimentos y las métricas para el análisis de resultados de precisión, procedemos a mostrar los resultados obtenidos por las cadenas de procesamiento que pueden formarse con los métodos mostrados en la sección 2 con las imágenes hiperespectrales consideradas. La Tabla 5.3 muestra la composición de las cadenas consideradas en términos de los métodos empleados en cada etapa. Debido al carácter aleatorio en alguno de los métodos utilizados en los experimentos (por ejemplo, N-FINDR), los resultados pueden variar tanto en el tiempo de ejecución como en la precisión alcanzada de una ejecución a otra. Es por esto que a la hora de realizar los experimentos, se han realizado 50 ejecuciones de cada cadena para posteriormente realizar un promediado de los resultados obtenidos que refleje de manera más correcta los resultados. Conviene destacar que en este proceso se ha observado que la desviación estándar de los resultados con respecto a la media es muy pequeña, lo cual indica que las diferentes ejecuciones proporcionan prácticamente los mismos resultados. También conviene destacar que, en los experimentos con la imagen AVIRIS Cuprite, disponemos de firmas espectrales de referencia con las que podemos comparar los *endmembers* obtenidos. Esto se debe a que la imagen se encuentra disponible en unidades de reflectancia (es decir, la imagen está corregida atmosféricamente) y a la existencia de firmas espectrales de referencia obtenidas por USGS. En la imagen AVIRIS WTC no disponemos de tal información, por tanto en los experimentos con AVIRIS Cuprite se realizará una comparativa de las cadenas en base a las dos métricas consideradas: SAD y RMSE, mientras que en los experimentos con AVIRIS WTC únicamente se utiliza el RMSE que no necesita información de referencia ya que se basa en estimar cómo de bien el modelo lineal de mezcla aproxima la imagen original, que en este caso se utiliza como referencia en el proceso de validación.

Cadena	Número de *endmembers*	Reducción dimensional	Idenficación de *endmembers*	Estimación de abundancias
1	VD	PCA	N-FINDR	UCLS
2	VD	PCA	N-FINDR	NCLS
3	VD	SPCA	N-FINDR	UCLS
4	VD	SPCA	N-FINDR	NCLS
5	HySime	PCA	N-FINDR	UCLS
6	HySime	PCA	N-FINDR	NCLS
7	HySime	SPCA	N-FINDR	UCLS
8	HySime	SPCA	N-FINDR	NCLS

Tabla 5.3: Composición de las ocho cadenas de desmezclado completas consideradas en los experimentos.

Cadena	Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite	Media
1	6.00	4.22	5.94	10.11	5.96	6.45
2	5.90	4.23	5.91	10.01	5.89	6.39
3	5.95	4.23	6.66	10.03	5.29	6.43
4	5.79	4.23	6.74	9.77	5.35	6.38
5	6.48	4.25	5.97	10.73	6.11	6.71
6	5.98	4.23	6.21	10.41	6.09	6.58
7	5.68	4.22	7.11	10.73	5.30	6.61
8	5.64	4.22	6.92	10.74	5.28	6.56

Tabla 5.4: Similaridad espectral entre los *endmembers* obtenidos por diferentes cadenas de desmezclado y las firmas espectrales de referencia USGS para cinco minerales representativos en la imagen AVIRIS Cuprite.

5.3.1. Experimentos con la imagen AVIRIS Cuprite

La Tabla 5.4 muestra los resultados de SAD para la imagen hiperespectral AVIRIS Cuprite con cada una de las 8 cadenas de desmezclado consideradas (los resultados mostrados en la Tabla 5.4 corresponden a la media tras 50 ejecuciones). Para obtener estos resultados, se ha efectuado un proceso de emparejamiento en el que se han identificado las firmas USGS que resultan más parecidas (en términos de SAD) con respecto a los *endmembers* obtenidos por las diferentes cadenas de desmezclado, de forma que la Tabla 5.4 muestra el valor SAD resultante del mejor emparejamiento para cada mineral. En el estudio, hemos considerado los cinco minerales más representativos en la zona, y los valores de SAD en la Tabla 5.4 se expresan en grados (recordamos que valores de SAD cercanos a cero indican mayor similaridad espectral de los *endmembers* con respecto a las correspondientes firmas de referencia USGS).

Como puede apreciarse en la Tabla 5.4, los resultados obtenidos en cuanto a SAD por cada una de las diferentes cadenas son bastante similares entre sí, situándose el ángulo medio de los cinco materiales en torno a los 6°. Recordemos que el mejor resultado para esta métrica sería 0° y el peor 90°, lo que nos lleva a concluir que todas las cadenas consiguen identificar *endmembers* bastante similares a las firmas espectrales USGS de referencia. Esta similitud en los resultados se debe a que el método de extracción

de *endmembers* considerado en todos los casos es N-FINDR (ver Tabla 5.4). Hemos comprobado experimentalmente que la utilización de otros métodos de identificación de *endmembers* en la literatura, incluido el método IEA descrito en el capítulo 2 de la presente memoria, da como resultado *endmembers* muy similares, espectralmente, a los proporcionados por N-FINDR. Por tanto, concluimos que N-FINDR es un método adecuado para la identificación de *endmembers* en imágenes hiperespectrales.

Por otra parte, la Fig. 5.6 muestra los mapas de error que se han obtenido en cada píxel de la imagen AVIRIS Cuprite con las diferentes cadenas de procesamiento al aplicar la métrica descrita en la expresión (5.2). Cada uno de estos mapas de error viene acompañado del RMSE medio obtenido de estas imágenes tras realizar 50 ejecuciones de cada cadena, así como de la desviación estándar. El mapa mostrado para cada cadena en la Fig. 5.6 es el más cercano a la media de las 50 ejecuciones. A la hora de representar cada una de estos mapas, se ha utilizado una escala de colores cuyo valor mínimo es 0 y cuyo valor máximo es el máximo global de todos los mapas obtenidos. En la representación de la Fig. 5.6, los colores frios denotan menor error y los colores cálidos denotan mayor error.

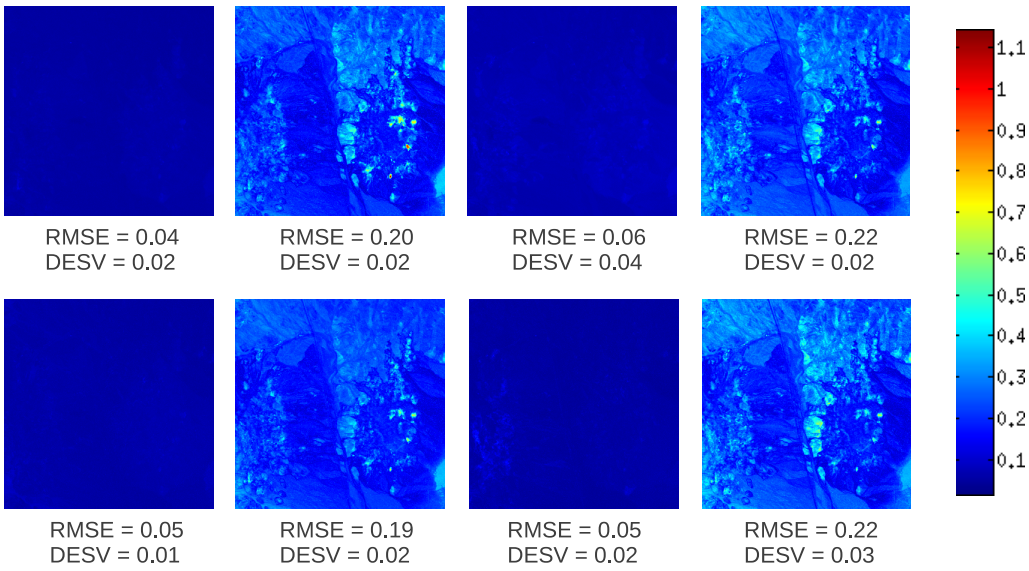

Figura 5.6: Errores de reconstrucción para cada píxel de la imagen AVIRIS Cuprite y error RMSE global obtenido para cada cadena de desmezclado considerada. Los mapas se encuentran ordenados por número de cadena, de izquierda a derecha y de arriba a abajo, con la primera cadena en la esquina superior izquierda y la octava cadena en la esquina inferior derecha.

Como puede apreciarse en la Fig. 5.6, los errores de reconstrucción obtenidos por las diferentes cadenas son muy similares y relativamente bajos, lo que indica que el modelo lineal de mezcla permite aproximar de manera precisa la imagen original en todos los casos. Además se observa que la desviación estándar es prácticamente cero en todos los casos, lo que nos lleva a afirmar que (a pesar del componente aleatorio en algunos de los métodos utilizados como N-FINDR) los resultados obtenidos son muy estables.

Conviene destacar que, en las cadenas que se emplea el método NCLS para la estimación de abundancias (cadenas pares en la Fig. 5.6) el RMSE es más alto. Esto se debe principalmente al proceso de corrección que sufre el proceso de estimación de abundancias asegurar que dichas abundancias resulten positivas. También conviene destacar el hecho de que las cadenas que usan el mismo método de estimación de abundancias obtienen resultados prácticamente idénticos. En concreto, la cadena con menor error RMSE en los experimentos realizados es la primera (VD+PCA+N-FINDR+UCLS) mientras que la cadena con mayor error RMSE es la última (HySime+SPCA+N-FINDR+NCLS). Por motivos ilustrativos, la Fig. 5.7 muestra los mapas de abundancias (asociados a los cinco minerales más representativos de la zona) obtenidos por la primera cadena de desmezclado. La Fig. 5.7 pretende realizar una comparación cualitativa de los resultados de dicha cadena con los mapas de presencia/ausencia de minerales obtenidos a partir del algoritmo `Tetracorder` de USGS. Como puede apreciarse en la Fig. 5.7, existe gran correlación entre los mapas correspondientes a los cinco minerales considerados, ofreciendo los mapas de abundancia más información ya que su contenido describe el porcentaje de cobertura de cada *endmember* en cada píxel de la imagen, mientras que los mapas de presencia/ausencia simplemente indican la presencia o no de cada mineral sin tener en cuenta su abundancia.

5.3.2. Experimentos con la imagen AVIRIS World Trade Center

Dado que no existe información de referencia en forma de firmas espectrales para la imagen AVIRIS WTC, debido al hecho de que la imagen se encuentra disponible en unidades de radiancia (y, por tanto, sin corrección atmosférica) a continuación presentamos los errores de reconstrucción obtenidos por las diferentes cadenas tras utilizar el modelo lineal de mezcla para aproximar la imagen original utilizando los *endmembers* identificados y las abundancias estimadas. Dichos resultados se muestran en la Fig. 5.8. De nuevo, la métrica aplicada es la descrita en la expresión (5.2) y cada uno de los mapas de error en la Fig. 5.8 viene acompañado del RMSE medio obtenido de estas imágenes tras realizar 50 ejecuciones de cada cadena, así como de la desviación estándar. El mapa mostrado para cada cadena en la Fig. 5.8 es el más cercano a la media de las 50 ejecuciones. A la hora de representar cada una de estos mapas, se ha utilizado una escala de colores cuyo valor mínimo es 0 y cuyo valor máximo es el máximo global de todos los mapas obtenidos, utilizando colores frios para denotar menor error y colores cálidos para denotar mayor error.

Como se aprecia en la Fig. 5.8, la cadena 5 (HySime+PCA+N-FINDR+UCLS) obtiene el error de reconstrucción más bajo y la cadena 2 (VD+PCA+N-FINDR+NCLS) obtiene el error más alto. De nuevo, se observa que las cadenas que utilizan el método NCLS para estimación de abundancias obtienen valores de RMSE más elevado. De cara a investigar este fenómeno en más detalle, la Fig. 5.9 muestra los mapas de abundancias obtenidos para los *endmembers* fuego, vegetación y humo por la mejor cadena en términos de RMSE (número 5, en la fila superior de la figura) y por la peor cadena en términos de RMSE (número 2, en la fila inferior de la figura). Como puede apreciarse en la Fig. 5.9, la cadena 2 (basada en el método NCLS) obtiene mapas de abundancia que parecen

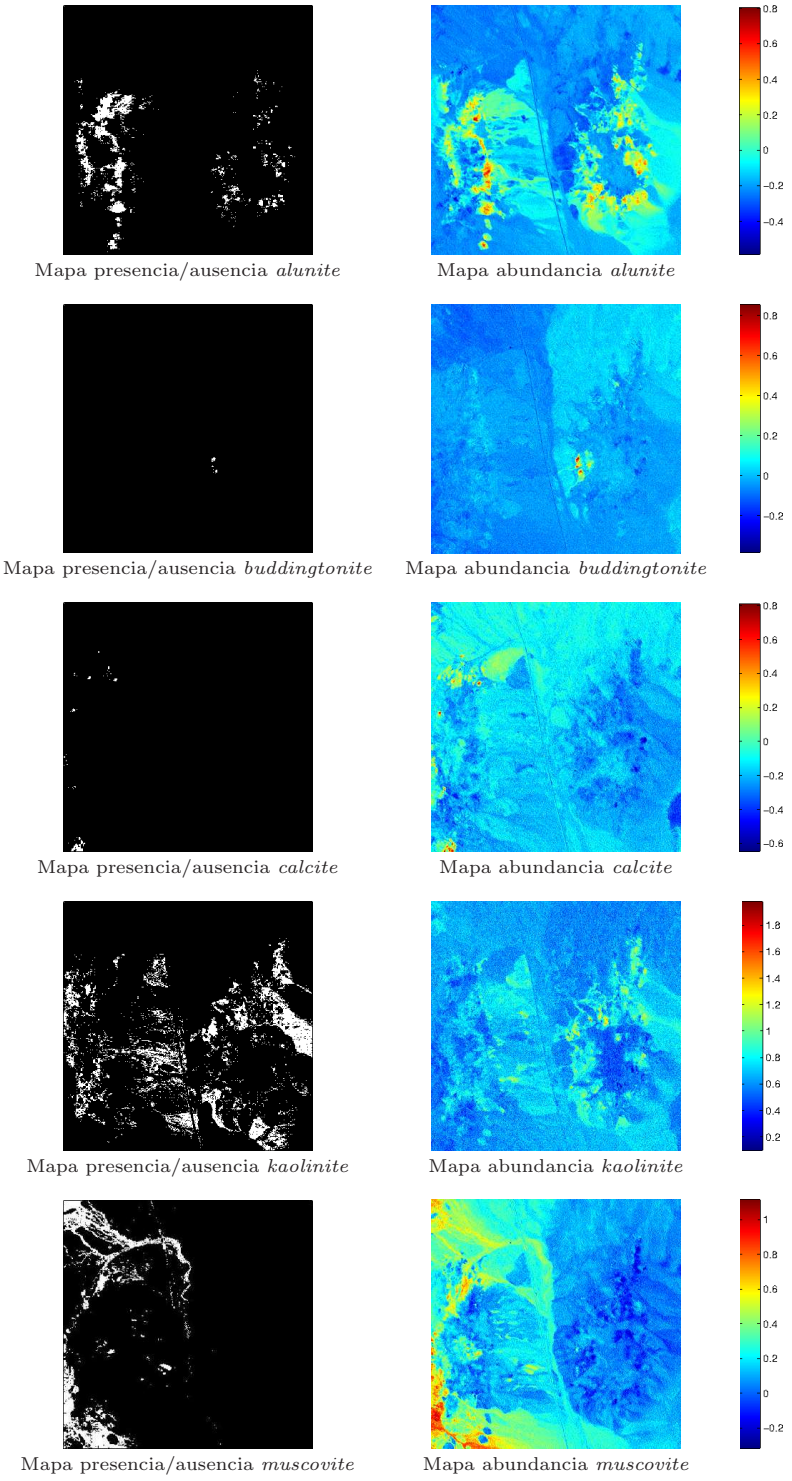

Figura 5.7: Comparativa entre los mapas de presencia/ausencia (obtenidos por el algoritmo `Tetracorder` de USGS) y los mapas de abundancia (obtenidos por la cadena de desmezclado que aplica los métodos VD+PCA+N-FINDR+UCLS) para cinco minerales representativos en la imagen AVIRIS Cuprite.

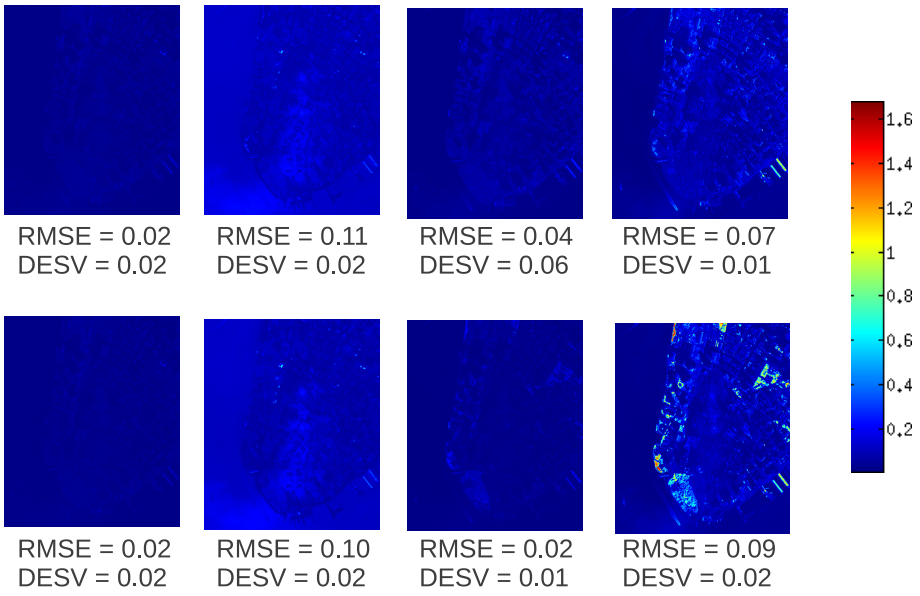

Figura 5.8: Errores de reconstrucción para cada píxel de la imagen AVIRIS World Trade Center y error RMSE global obtenido para cada cadena de desmezclado considerada. Los mapas se encuentran ordenados por número de cadena, de izquierda a derecha y de arriba a abajo, con la primera cadena en la esquina superior izquierda y la octava cadena en la esquina inferior derecha.

mejores desde un punto de vista cualitativo que los proporcionados por la cadena 5 (basada en el método UCLS), al menos para los *endmembers* de fuego y vegetación. El *endmember* asociado al humo aparece representado de forma similar en ambos métodos, pero al tratarse de un *endmember* muy abundante en la imagen este *endmember* puede determinar las diferencias existentes en cuanto a RMSE, ya que los mapas de abundancia asociados a los *endmembers* de fuego y vegetación parecen más precisos en la cadena 2, a pesar de que esta cadena resulta en un valor RMSE global más elevado que la cadena 5. Este ejemplo pone de manifiesto que las métricas globales, como SAD y RMSE, no son capaces de reflejar casos particulares dependientes de la aplicación, por ejemplo un usuario interesado en el *endmember* fuego (quizá el elemento más importante en la imagen AVIRIS World Trade Center) seleccionaría los resultados de la cadena 2 a pesar de que esta cadena resulta en un mayor valor de RMSE global que la cadena 5. Teniendo en cuenta estos aspectos, que resultan muy importantes en cuanto a la valoración de las diferentes cadenas consideradas, procedemos en el siguiente apartado a analizar las cadenas desde el punto de vista del rendimiento computacional de las diferentes implementaciones desarrolladas en arquitecturas GPU.

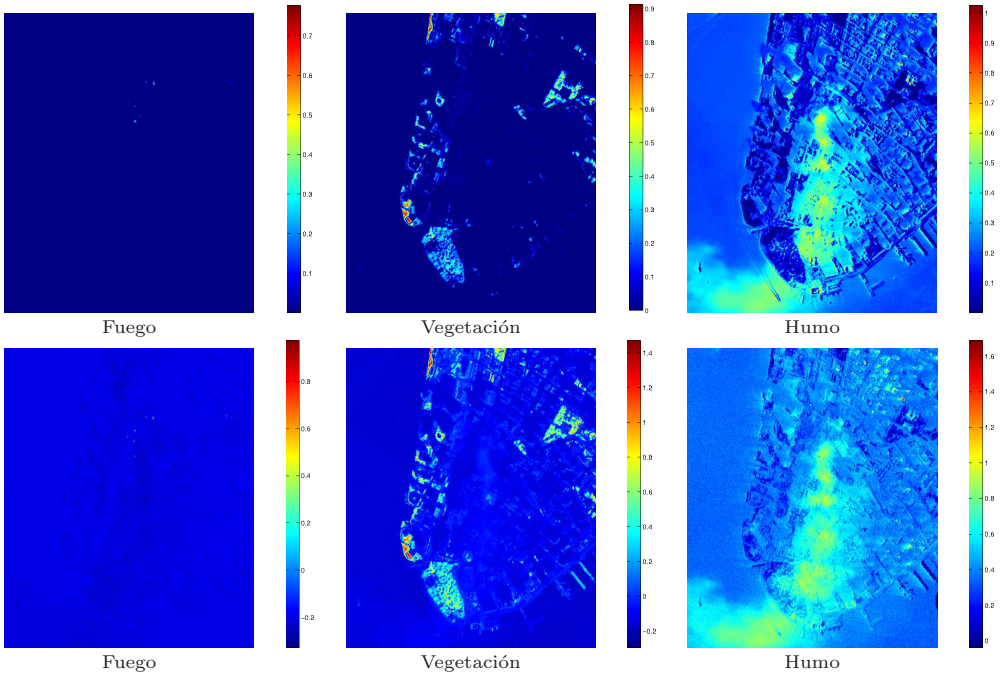

Figura 5.9: Mapas de abundancia estimados por la cadena 2 (VD+PCA+N-FINDR+NCLS) [fila superior] y por la cadena 5 (HySime+PCA+N-FINDR+UCLS) [fila inferior] para los *endmembers* fuego, vegetación y humo en la imagen AVIRIS World Trade Center.

5.4. Rendimiento computacional de las implementaciones GPU

En este apartado mostramos el rendimiento computacional de las implementaciones GPU desarrolladas para las cadenas completas de procesamiento utilizando distintas arquitecturas GPU. Estos experimentos se han realizado sobre el mismo procesador multi-núcleo, utilizando para las mediciones en serie uno de los núcleos del procesador y para las mediciones en paralelo una de las dos tarjetas gráficas consideradas en nuestros experimentos (NVidia Tesla C1060 y NVidia GTX 580, descritas en la sección 3.3). Las especificaciones técnicas del procesador multi-núcleo se muestran en la Tabla 5.5.

Conviene destacar que las mediciones de tiempo de ejecución se han realizado utilizando la función `gettimeofday()` del lenguaje C. Debido al carácter aleatorio en alguna de las etapas de las cadenas de procesamiento consideradas, se ha optado por realizar 50 ejecuciones para cada cadena de forma que los resultados que se muestran en el presente apartado corresponden a la media de las ejecuciones realizadas (en todos los casos, se ha observado que la desviación estándar es despreciable dado que los valores de las diferentes ejecuciones son prácticamente idénticos). En todos los casos, la toma de tiempos comienza justo antes de la carga de la imagen desde disco y finaliza tras el almacenamiento de los resultados en disco. Cada una de las 50 ejecuciones se ha realizado sobre las tres plataformas, es decir, en la CPU (serie, usando solo un núcleo),

Tipo de computador	ACPI x64-based PC
Sistema operativo	Ubuntu 11.04 (natty) 64bits
Tipo de CPU	QuadCore Intel Core i7 920, 2666 MHz
Tamaño de caché L1	128KB
Tamaño de caché L2	1024KB
Tamaño de caché L3	8192KB
Nombre de la placa madre	Asus P6T7 WS Supercomputer
Chipset de la placa madre	Intel Tylersburg X58, Intel Nehalem
Memoria del sistema	6135 MB (DDR3-1333 DDR3 SDRAM)
Tipo de BIOS	AMI (07/01/09)
Placa de video	NVIDIA Quadro NVS 295 (256 MB)
Disco duro 1 (sistema)	ATA Device (150 GB, 10000 RPM, SATA)
Disco duro 2	ATA Device (500 GB, 7200 RPM, SATA-II)

Tabla 5.5: Especificaciones técnicas del procesador multi-núcleo utilizado en los experimentos.

en la GPU NVidia Tesla C1060 y en la GPU NVidia GTX 580. Además del compilador `NVCC`, utilizado para compilar el código que se ejecuta en la GPU, se han utilizado dos compiladores diferentes para la versión serie y las partes serie de las versiones paralelas, estos son: `GCC`, un compilador libre para sistemas GNU, e `ICC`, un compilador propietario diseñado para compilar código C/C++ para su ejecución en arquitecturas Intel. En todas las pruebas, se ha realizado la compilación utilizando el *flag* `-O3` para optimizar al máximo el rendimiento de las versiones serie.

Antes de mostrar los resultados obtenidos con las dos imágenes consideradas en los experimentos, conviene destacar que una de las principales motivaciones de este trabajo es conseguir procesar dichas imágenes (mediante cadenas completas de desmezclado) en tiempo real. Por tanto, llegados a este punto es preciso establecer qué significa procesar dichas imágenes en tiempo real. En este sentido, el sensor AVIRIS es un sensor de empuje que obtiene una línea completa de 512 muestras o píxeles (cada uno con 224 bandas espectrales) en 8.3 milisegundos. Además el número de líneas consecutivas que el sensor recoge antes de volcar una imagen completa a disco es de 614. Este tamaño (aproximadamente 140 MB) coincide exactamente con el de una de las dos imágenes consideradas, la imagen AVIRIS World Trade Center. Por tanto el tiempo que necesita el sensor para obtener esta imagen es de 5.096 segundos. Consideramos por tanto que las cadenas de procesamiento serán capaces de funcionar en tiempo real si pueden procesar una imagen hiperespectral de 140 MB (como la imagen AVIRIS World Trade Center) y devolver los resultados de desmezclado en tiempo inferior a 5.096 segundos. Por otra parte, hablar de procesamiento en tiempo real en la imagen AVIRIS Cuprite es algo más complicado ya que ninguna de sus dimensiones (350 líneas, 350 muestras y 188 bandas espectrales tras eliminar algunas bandas ruidosas en el proceso de corrección atmosférica) coincide con las dimensiones estándares del sensor. Aún así, se puede estimar de forma sencilla que el procesamiento en tiempo real para esta imagen dependería del hecho de que las líneas de la imagen AVIRIS Cuprite constan de 350 píxeles en lugar de 512, y suponiendo que el sensor emplea el mismo tiempo en capturar un píxel de 224 bandas

espectrales que uno de 188, podemos establecer el tiempo que se necesitaría para obtener una línea de esta imagen en 5,67 milisegundos. Si tenemos en cuenta que el número de líneas de esta imagen es de 350, entonces el tiempo total empleado para capturar esta imagen por parte del sensor AVIRIS sería de 1.985 segundos que constituiría el límite de tiempo para procesar esta imagen en tiempo real.

Una vez establecidos los umbrales para considerar tiempo real en el procesamiento de las dos imágenes consideradas, procedemos a mostrar los resultados obtenidos por cada una de las cadenas para cada imagen. En todos los casos, se muestra una tabla con los tiempos obtenidos en cada etapa de la cadena y el tiempo total para cada plataforma utilizada. En total hablamos de 6 plataformas: 2 plataformas serie (dependiendo del compilador utilizado, ya sea `gcc` ó `icc`, y 4 plataformas paralelas, también dependientes del compilador y de la arquitectura GPU (Tesla `gcc`, Tesla `icc`, GTX `gcc` y GTX `icc`. En todos los casos, se muestran los resultados de aceleración ó *speedup* conseguidos con el mismo compilador, es decir los que resultan de relacionar los tiempos serie `gcc` frente a Tesla `gcc` ó GTX `gcc` y serie `icc` frente a Tesla `icc` ó GTX `icc`. En el caso de que se consiga alcanzar el tiempo real en el procesamiento de la cadena, éste aparecerá con fondo azul en la tabla. Además, en cada caso se presenta una gráfica en la que se puede apreciar la contribución de cada etapa al tiempo total de la cadena. Esta gráfica aparece escalada, de tal manera que a la izquierda aparecen las ejecuciones en serie y a la derecha las ejecuciones en paralelo. Se realiza este escalado debido a la gran diferencia de tiempo entre las versiones serie y paralela de algunas cadenas.

5.4.1. Resultados para la imagen AVIRIS Cuprite

A continuación analizamos los resultados obtenidos por las diferentes implementaciones de cada una de las cadenas de desmezclado completas consideradas en el presente trabajo al procesar la imagen hiperespectral AVIRIS Cuprite.

5.4.1.1. Cadena de procesamiento 1

La Tabla 5.6 muestra los resultados de tiempo obtenidos en cada una de las plataformas para esta cadena. Puede verse que los tiempos obtenidos en la versión serie varían bastante en función del compilador que se utilice, resultando ser la versión compilada con `icc` (7.470 segundos) unos 5 segundos más rápida que la versión compilada con gcc (12.346 segundos). En cuanto a las versiones paralelas no se aprecia una diferencia significativa en el tiempo total tanto si varía el compilador utilizado como si lo que varía es la tarjeta gráfica. En este caso el tiempo se sitúa en torno a 0.9 segundos consiguiendo una ejecución en tiempo real de la cadena de procesamiento. En este sentido el *speedup* conseguido en la versión compilada con `gcc` es de 13x en ambas tarjetas y el de la versión compilada con `icc` es de 8x también en ambas tarjetas.

La Fig 5.10 nos permite apreciar la contribución de cada etapa de la cadena al tiempo total de ejecución desde un punto de vista gráfico. Como ya se ha comentado el gráfico

	LOAD	VD	PCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.186	5.555	5.401	0.883	0.313	0.008	12.346	
Tesla gcc	0.202	0.411	0.147	0.112	0.054	0.008	0.934	13.219
GTX gcc	0.204	0.423	0.137	0.107	0.045	0.010	0.926	13.331
Serie icc	0.170	3.940	2.253	0.847	0.251	0.008	7.470	
Tesla icc	0.183	0.420	0.149	0.115	0.054	0.012	0.933	8.007
GTX icc	0.194	0.441	0.132	0.094	0.041	0.012	0.914	8.174

Tabla 5.6: Tiempos de procesamiento de las diferentes etapas de la cadena 1 [AVIRIS Cuprite].

aparece escalado para poder apreciar bien la parte correspondiente a las ejecuciones paralelas. De esta manera en la parte izquierda, con los nombres y los tiempos en azul, se usan dos barras para las ejecuciones serie. Y en la parte derecha se usan cuatro barras, con los nombres y los tiempos en naranja, para las ejecuciones paralelas. Puede verse como en las ejecuciones serie las etapas con más peso son la de estimación de el número de endmembers y reducción dimensional, sumando entre ambas aproximadamente el 80 % del tiempo total. En el caso de las ejecuciones paralelas la etapa que abarca la mayor parte del tiempo es la de reducción dimensional que toma aproximadamente el 50 %. Un factor importante a tener en cuenta son los tiempos de carga de la imagen y de almacenamiento de los resultados, éstos son bastante significativos en las ejecuciones paralelas. En este caso el porcentaje de tiempo dedicado a entrada/salida es de aproximadamente el 20 %

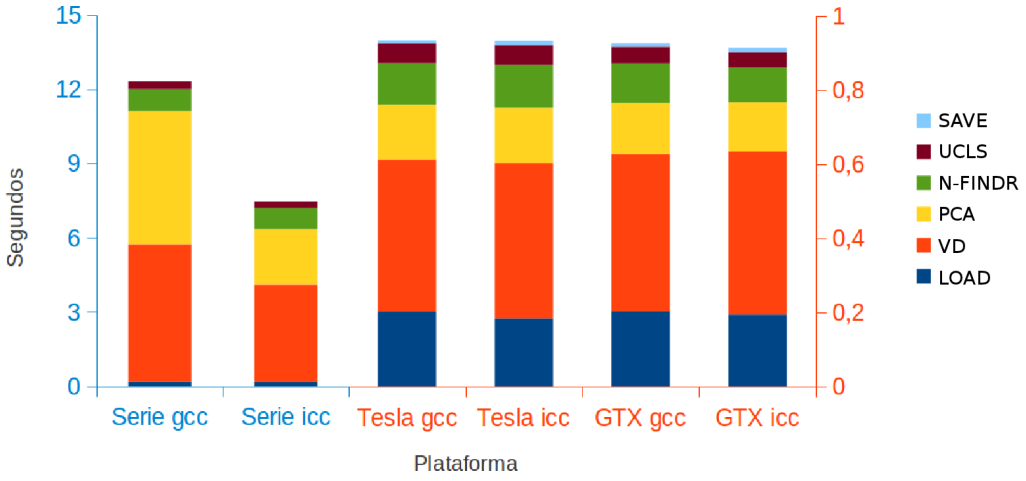

Figura 5.10: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 1 [AVIRIS Cuprite]

5.4.1.2. Cadena de procesamiento 2

La Tabla 5.7 muestra los resultados de tiempo obtenidos en cada una de las plataformas para esta cadena. En el caso de las ejecuciones serie existe una gran diferencia

en utilizar un compilador u otro, concretamente de 23 segundos, siendo el tiempo empleado por la versión compilada con `gcc` de 104.595 segundos y el tiempo empleado por la versión compilada con `icc` de 81.215 segundos. En el caso de las ejecuciones paralelas la diferencia no la marca el compilador empleado sino la tarjeta gráfica utilizada. El tiempo de ejecución para el caso de la GPU Tesla C1060 es de 2.3 segundos mientras que el tiempo de ejecución en el caso de la GPU GTX 580 es de 1.8 segundos lo que hace que esta ejecución se realice en tiempo real. En cuanto a los *speedups* conseguidos éstos son diferentes para cada tarjeta y compilador, siendo el más alto de 57x que se obtiene con la GTX 580 usando el compilador `gcc`.

	LOAD	VD	PCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.179	5.516	5.393	0.891	92.607	0.008	104.595	
Tesla gcc	0.194	0.391	0.147	0.111	1.539	0.013	2.395	43.674
GTX gcc	0.199	0.404	0.133	0.096	0.987	0.008	1.828	57.215
Serie icc	0.163	3.921	2.250	0.832	74.041	0.008	81.215	
Tesla icc	0.184	0.401	0.148	0.113	1.539	0.008	2.394	33.924
GTX icc	0.180	0.406	0.132	0.092	0.987	0.008	1.805	45.002

Tabla 5.7: Tiempos de procesamiento de las diferentes etapas de la cadena 2 [AVIRIS Cuprite]

Podemos ver la cantidad de tiempo con la que contribuye cada etapa al tiempo total para esta cadena en la Fig. 5.11. Lo más llamativo es el gran porcentaje de tiempo que se dedica a la estimación de abundancias sobre todo en las ejecuciones serie, donde este paso alcanza un 90 % aproximadamente. En esta cadena se utiliza NCLS como método de estimación de abundancias. Como ya vimos, su algoritmo utiliza varias multiplicaciones matriciales de forma iterativa. Ésta es la principal causa de este incremento de tiempo en las ejecuciones serie, así como de los *speedups* conseguidos. En el caso de las ejecuciones paralelas el porcentaje de tiempo empleado en esta etapa de estimación de abundancias es aproximadamente del 65 % al utilizar la GPU Tesla C1060 y del 50 % al utilizar la GPU GTX 580. El porcentaje de tiempo empleado en entrada salida es del 8 % en el caso de la Tesla C1060 y del 11 % en el caso de la GTX 580.

5.4.1.3. Cadena de procesamiento 3

La Tabla 5.8 muestra los tiempos de ejecución de la cadena de desmezclado número 3. Podemos ver que existe una diferencia de 5 segundos entre las dos ejecuciones serie, siendo el tiempo de la versión compilada con `gcc` de 12.369 segundos y el de la versión compilada con `icc` de 7.498 segundos. En el caso de las ejecuciones paralelas no existe una diferencia de tiempos significativa situándose todas ellas en torno a los 0.9 segundos, por tanto se alcanza el tiempo real para esta imagen con las dos tarjetas gráficas consideradas. Los *speedups* en esta cadena son de 13x para el caso de las versiones compiladas con `gcc` y de 8x para las compiladas con `icc`.

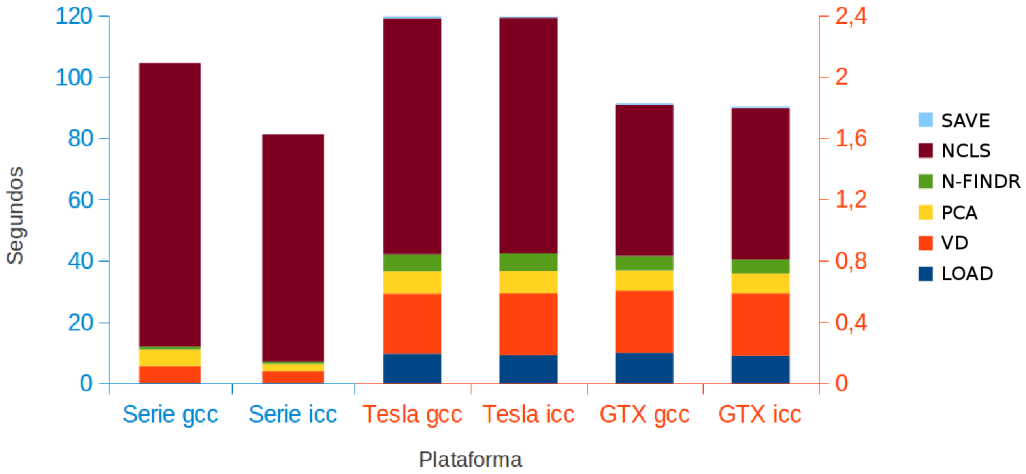

Figura 5.11: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 2 [AVIRIS Cuprite]

	LOAD	VD	SPCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.180	5.508	5.493	0.869	0.311	0.008	12.369	
Tesla gcc	0.193	0.392	0.172	0.114	0.054	0.008	0.933	13.260
GTX gcc	0.200	0.402	0.147	0.103	0.042	0.009	0.903	13.704
Serie icc	0.164	3.941	2.268	0.864	0.252	0.008	7.498	
Tesla icc	0.183	0.396	0.174	0.111	0.054	0.009	0.926	8.101
GTX icc	0.182	0.406	0.146	0.097	0.041	0.009	0.881	8.515

Tabla 5.8: Tiempos de procesamiento de las diferentes etapas de la cadena 3 [AVIRIS Cuprite]

La Fig. 5.12 muestra gráficamente la contribución de cada etapa de la cadena al tiempo total de ejecución. Para la ejecución serie compilada con `gcc` las etapas de estimación del número de *endmembers* y de reducción dimensional emplean un 45 % del tiempo total cada una. En la version serie compilada con `icc` la etapa de estimación del número de endmembers es la que más tiempo emplea con un 52 % del tiempo total. En el caso de las ejecuciones paralelas también es esta etapa la que más tiempo consume, empleando en torno a un 42 % en todos los casos. Finalmente el tiempo empleado en entrada/salida para estas versiones está en torno al 18 %.

5.4.1.4. Cadena de procesamiento 4

Los resultados de tiempo de la cadena número 4 se muestran en la Tabla 5.9. En dicha tabla podemos ver que existe una diferencia de 23 segundos entre las dos ejecuciones serie, siendo la versión compilada con `icc` la más rápida empleando 81.174 segundos y la versión compilada con `gcc` la más lenta empleando un total de 104.610 segundos. En el caso de las ejecuciones paralelas la diferencia de tiempos entre las distintas versiones no la marcan los compiladores utilizados sino las tarjetas gráficas. Para el caso de la GPU

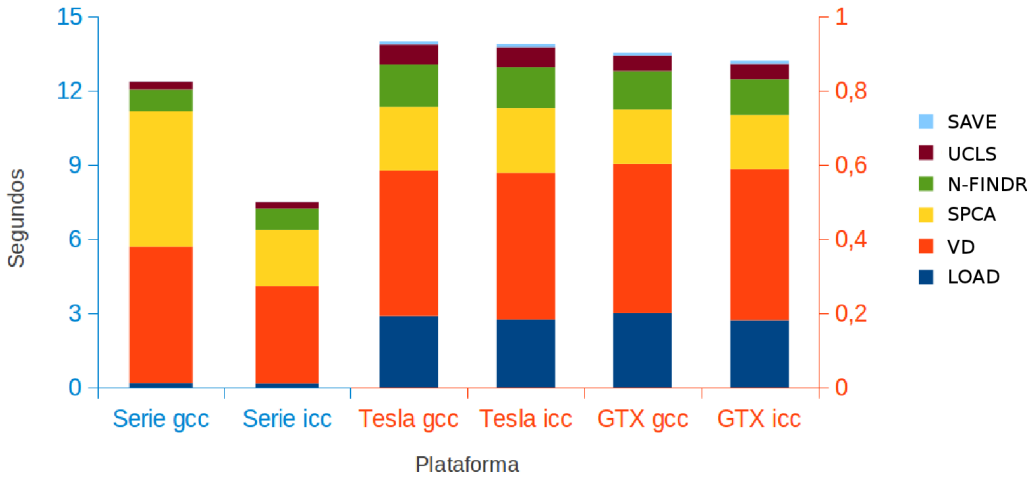

Figura 5.12: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 3 [AVIRIS Cuprite]

Tesla C1060 el tiempo empleado es de 2.4 segundos en ambos compiladores, y para el caso de la GPU GTX 580 el tiempo total está en torno a 1.8 segundos lo que supone alcanzar tiempo real. El *speedup* más alto es de 56x y se obtiene con la GPU GTX 580 utilizando `gcc`.

	LOAD	VD	SPCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.180	5.505	5.491	0.847	92.580	0.008	104.610	
Tesla gcc	0.192	0.390	0.172	0.112	1.540	0.008	2.414	43.340
GTX gcc	0.200	0.405	0.148	0.100	0.988	0.008	1.848	56.594
Serie icc	0.164	3.920	2.266	0.815	74.000	0.008	81.174	
Tesla icc	0.183	0.392	0.173	0.117	1.539	0.009	2.414	33.633
GTX icc	0.183	0.402	0.146	0.098	0.987	0.008	1.825	44.471

Tabla 5.9: Tiempos de procesamiento de las diferentes etapas de la cadena 4 [AVIRIS Cuprite]

Podemos hacernos una idea de la contribución al tiempo total de cada una de las etapas en esta cadena gracias a la Fig. 5.13. Al igual que la cadena número 2 y las cadenas pares, se utiliza NCLS como método de estimación de abundancias. Éste se trata de un método costoso sobre todo en las versiones serie, lo que hace que se produzcan *speedups* altos. Puede verse como la parte de la barra correspondiente a NCLS ocupa prácticamente el 90 % de toda la barra en las versiones serie. En el caso de las versiones paralelas depende de la tarjeta utilizada empleando un 50 % del tiempo en la GPU GTX 580 y un 63 % en la GPU Tesla C1060. Las etapas de entrada/salida emplean un 8 % del tiempo total en caso de la Tesla C1060 y un 10 % en el caso de la GTX 580.

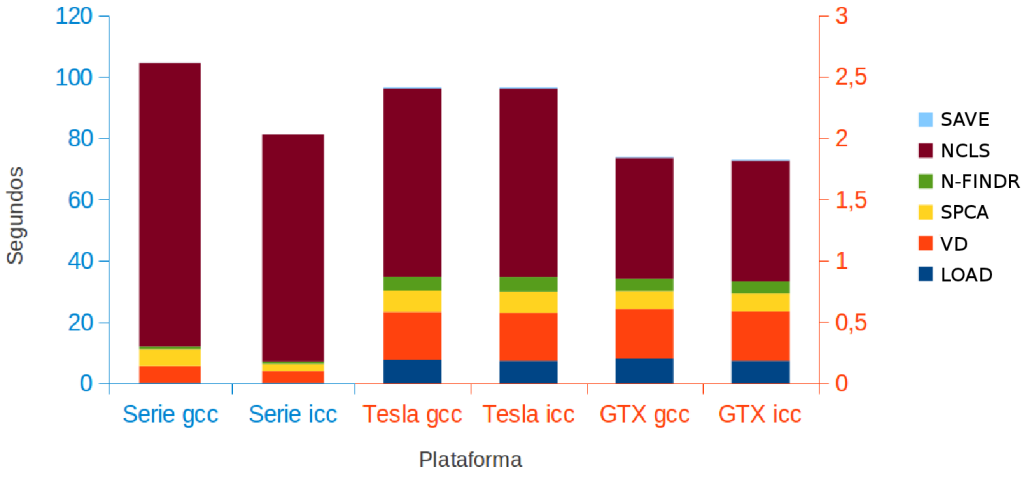

Figura 5.13: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 4 [AVIRIS Cuprite]

5.4.1.5. Cadena de procesamiento 5

Desde esta cadena hasta la número 8 todas usan HySime como método de estimación del número de *endmembers*. Este método es bastante más costoso lo que produce un incremento del tiempo en esta etapa que lleva consigo no alcanzar tiempo real en ninguna de las siguientes cadenas. Los tiempos de ejecución para la cadena número 5 en cada una de las plataformas quedan reflejados en la Tabla 5.10. Existe una diferencia de 11 segundos entre la versión que se compiló con `gcc` (34.662 segundos) y la que se compiló con `icc` (23.241 segundos). Viendo los tiempos de las ejecuciones paralelas podemos deducir que la diferencia entre las distintas versiones viene dada por la tarjeta utilizada y no por el compilador. Así, el tiempo empleado por las versiones que utilizan la GPU Tesla C1060 es de 2.3 segundos y el empleado por las versiones que utilizan la GPU GTX 580 es de 2.1 segundos. El *speedup* más alto conseguido en esta cadena es de 16x y se consigue con la GPU GTX 580 con respecto a la versión serie compilada con `gcc`.

	LOAD	HYSIME	PCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.180	28.278	5.344	0.591	0.262	0.007	34.662	
Tesla gcc	0.195	1.852	0.144	0.078	0.049	0.007	2.324	14.915
GTX gcc	0.201	1.661	0.131	0.071	0.038	0.007	2.108	16.443
Serie icc	0.165	20.106	2.208	0.543	0.212	0.007	23.241	
Tesla icc	0.183	1.839	0.145	0.079	0.049	0.007	2.300	10.104
GTX icc	0.183	1.662	0.130	0.068	0.037	0.007	2.087	11.134

Tabla 5.10: Tiempos de procesamiento de las diferentes etapas de la cadena 5 [AVIRIS Cuprite]

Como podemos ver en la Fig 5.14, la etapa que más tiempo emplea en todas las versiones es la de estimación del número de *endmembers*, variando desde un 81 % en la version serie `gcc`, un 86 % en la versión serie `icc`, un 79 % para ambas GPUs. Para esta cadena el tiempo dedicado a entrada/salida en las versiones paralelas está en torno al 9 %.

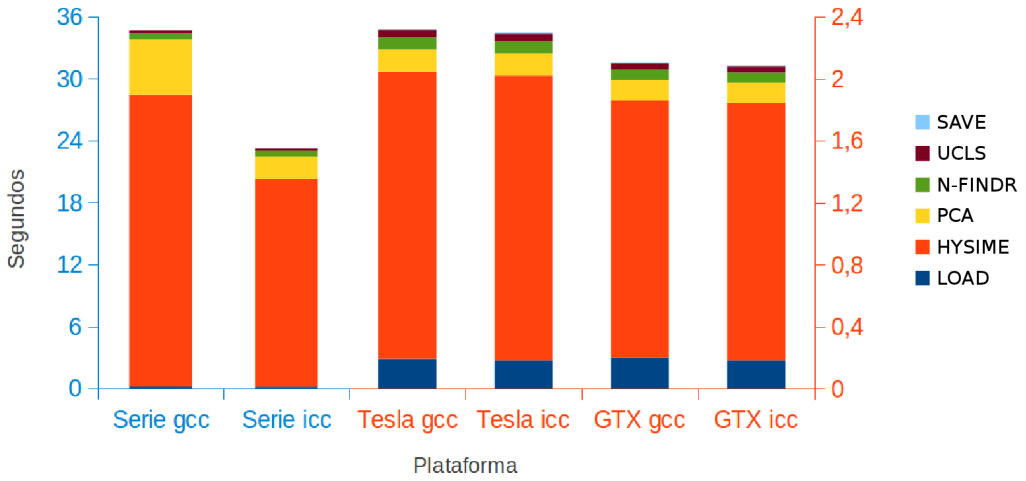

Figura 5.14: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 5 [AVIRIS Cuprite]

5.4.1.6. Cadena de procesamiento 6

La Tabla 5.11 muestra los resultados de tiempo que se han obtenido en la cadena número 6. Como se viene repitiendo en el caso de las ejecuciones en serie, la diferencia en los tiempos viene marcada por el uso de un compilador u otro, no siendo así en el caso de las ejecuciones paralelas donde la diferencia de tiempos viene marcada por el uso de una GPU u otra. En este sentido existe una diferencia de 18 segundos entre la versión serie `gcc` (112.032 segundos) y la versión serie `icc` (84.486 segundos). Para las versiones paralelas tenemos un tiempo de 3.3 segundos usando la GPU Tesla C1060 y un tiempo de 2.9 segundos usando la GPU GTX 580. Como comentamos en la cadena anterior el hecho de usar HySime en la etapa de estimación del número de endmembers ralentiza mucho las ejecuciones. Si a eso le sumamos el tiempo empleado por NCLS en la etapa de estimación de abundancias tenemos como resultado una cadena que no consigue alcanzar el tiempo real. El *speedup* más alto conseguido en esta cadena es de 38x y se produce cuando usamos la GPU GTX 580 en comparación con la versión serie `gcc`.

La Fig. 5.15 nos da un punto de vista gráfico de el tiempo empleado en cada etapa con respecto al total. En esta cadena se produce un fenómeno curioso, si comparamos

	LOAD	HYSIME	PCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.180	28.274	5.342	0.600	77.630	0.007	112.032	
Tesla gcc	0.196	1.861	0.144	0.081	1.074	0.007	3.363	33.317
GTX gcc	0.201	1.657	0.131	0.075	0.858	0.007	2.928	38.256
Serie icc	0.164	20.101	2.207	0.528	61.479	0.007	84.486	
Tesla icc	0.183	1.835	0.145	0.084	1.074	0.007	3.328	25.387
GTX icc	0.182	1.664	0.130	0.067	0.858	0.007	2.907	29.059

Tabla 5.11: Tiempos de procesamiento de las diferentes etapas de la cadena 6 [AVIRIS Cuprite]

las versiones series con las paralelas podemos ver que en las versiones serie un 25 % del tiempo total se emplea en HySime y un 70 % es empleado en NCLS. Sin embargo en las versiones paralelas se produce el fenómeno contrario, aportando el método HySime un 55 % del tiempo total y el método NCLS un 32 %. Por otro lado el tiempo dedicado a entrada/salida en el caso de las versiones paralelas supone un 5 % del total.

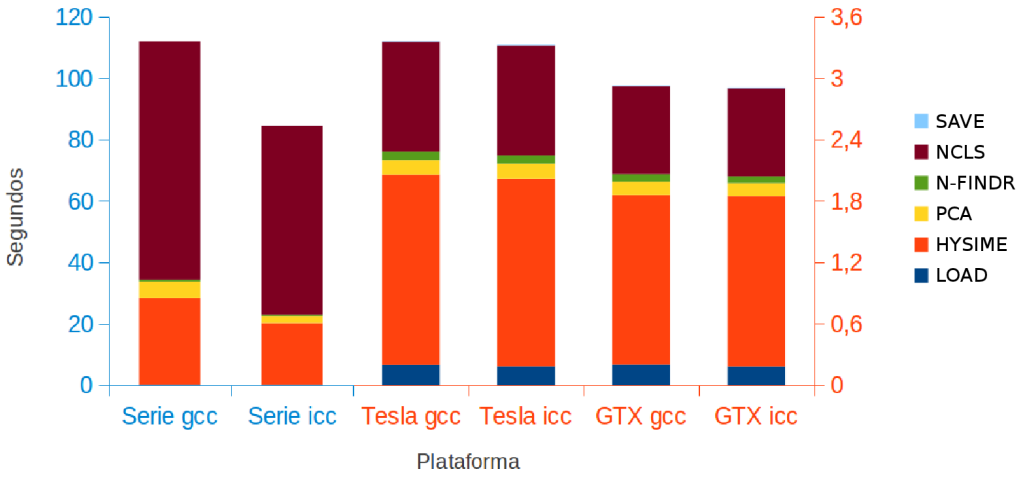

Figura 5.15: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 6 [AVIRIS Cuprite]

5.4.1.7. Cadena de procesamiento 7

Los resultados de tiempo que se han conseguido en las distintas ejecuciones de la cadena número 7 se muestran en la Tabla 5.12. En dicha tabla podemos apreciar una diferencia significativa de tiempos producida en las versiones serie y que se deben al hecho de utilizar un compilador u otro. Concretamente esta diferencia es de 10 segundos, teniendo un tiempo total de 34.732 segundos la versión Serie gcc y un tiempo total de 23.234 segundos la versión serie `icc`. En el caso de las versiones paralelas la diferencia de tiempos no es tan significativa y se debe sobre todo al hecho de utilizar una GPU u otra. En este caso el tiempo total para la versión que utiliza la Tesla C1060 es de 2.3 segundos y el tiempo para la versión que utiliza la GTX 580 es de 2.1 segundos. A pesar de ser

tiempos muy bajos no se alcanza la barrera de tiempo real en esta cadena. Se consigue un *speedup* de 16x al comparar la ejecución que utiliza la GTX con la serie `gcc`.

	LOAD	HYSIME	SPCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.180	28.276	5.431	0.578	0.261	0.007	34.732	
Tesla gcc	0.197	1.852	0.168	0.087	0.049	0.007	2.359	14.725
GTX gcc	0.201	1.665	0.144	0.070	0.038	0.007	2.124	16.351
Serie icc	0.165	20.105	2.225	0.521	0.212	0.007	23.234	
Tesla icc	0.184	1.848	0.169	0.082	0.049	0.007	2.339	9.933
GTX icc	0.183	1.662	0.143	0.069	0.037	0.007	2.102	11.054

Tabla 5.12: Tiempos de procesamiento de las diferentes etapas de la cadena 7 [AVIRIS Cuprite]

La Fig. 5.16 muestra un gráfico de barras donde se puede apreciar la contribución al tiempo total de ejecución de cada una de las etapas de la cadena número 7. Puede verse como la etapa dominante en tiempo es, en todos los casos, la de estimación del número de *endmembers* que utiliza el método HySime y que supera el 75 % del tiempo total. Concretamente en el caso de la ejecución serie `gcc` supone un 81 %, para la ejecución serie `icc` un 86 %, y para el caso de las ejecuciones paralelas un 78 %. Para estas últimas el tiempo dedicado a entrada/salida es de un 9 % aproximadamente.

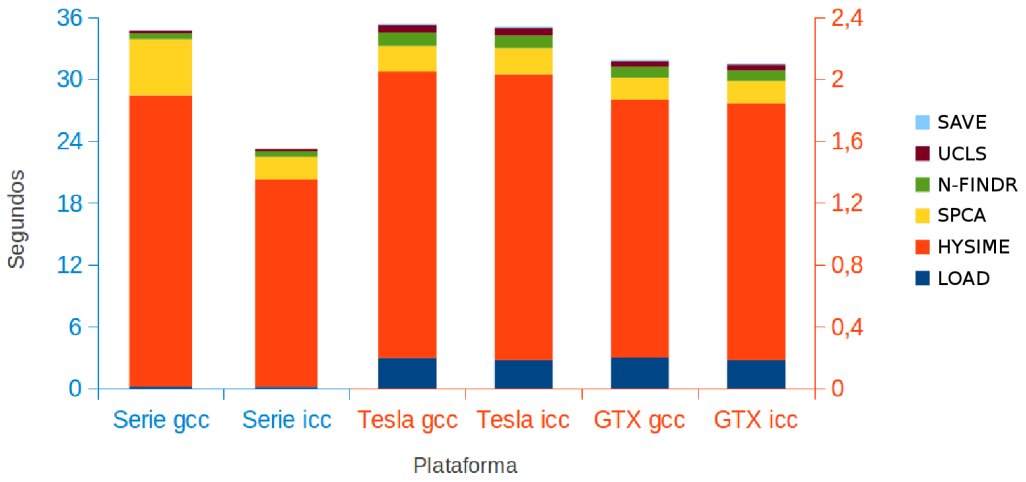

Figura 5.16: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 7 [AVIRIS Cuprite]

5.4.1.8. Cadena de procesamiento 8

La Tabla 5.13 nos muestra los resultados de tiempo que se han obtenido al ejecutar las diferentes pruebas con distintas plataformas. Un factor que se mantiene constante en

todas las cadenas es el hecho de que existe una diferencia de tiempos producida por el uso de distintos compiladores en las versiones serie, y producida por el uso de distintas GPUs en las versiones paralelas. Así tenemos que esta diferencia es de 28 segundos en las versiones serie, teniendo la versión serie `gcc` un tiempo de 112.181 segundos y la versión serie `icc` un tiempo de 84.453 segundos. En hecho de usar la GPU Tesla C1060 en esta cadena supone un tiempo de 3.3 segundos que es independiente del compilador utilizado. Para el caso de la GTX 580 el tiempo total es de 2.9 segundos. Como puede verse en ningún caso se consigue una ejecución en tiempo real para esta cadena. El *speedup* más alto se consigue, como es habitual, al relacionar la ejecución que utiliza la GTX 580 con la serie `gcc`, siendo de 38x en esta cadena.

	LOAD	HYSIME	SPCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.179	28.282	5.428	0.600	77.685	0.007	112.181	
Tesla gcc	0.197	1.853	0.169	0.083	1.075	0.007	3.383	33.161
GTX gcc	0.201	1.655	0.145	0.069	0.859	0.007	2.936	38.207
Serie icc	0.164	20.100	2.222	0.526	61.435	0.007	84.453	
Tesla icc	0.183	1.842	0.171	0.085	1.075	0.007	3.363	25.116
GTX icc	0.183	1.664	0.143	0.069	0.859	0.007	2.926	28.867

Tabla 5.13: Tiempos de procesamiento de las diferentes etapas de la cadena 8 [AVIRIS Cuprite]

Para hacernos una idea de el tiempo que emplea cada una de las etapas de esta cadena en relación al tiempo total de ejecución podemos fijarnos en la Fig. 5.17. Puede apreciarse el mismo fenómeno que ocurría en la cadena número 6, y es que si comparamos las versiones serie con las versiones paralelas podemos ver que en las serie un 25 % del tiempo total se emplea en HySime y un 70 % es empleado en NCLS. Sin embargo en las versiones paralelas se produce el fenómeno contrario, aportando el método HySime un 55 % del tiempo total y el método NCLS un 32 %. Por otro lado el tiempo dedicado a entrada/salida en el caso de las versiones paralelas supone un 5 % del total.

5.4.2. Resultados para la imagen AVIRIS World Trade Center

A continuación analizamos los resultados obtenidos por las diferentes implementaciones de cada una de las cadenas de desmezclado completas consideradas en el presente trabajo al procesar la imagen hiperespectral AVIRIS World Trade Center.

5.4.2.1. Cadena de procesamiento 1

La Tabla 5.14 muestra los resultados de tiempo obtenidos en cada una de las plataformas para esta cadena. Puede verse que los tiempos obtenidos en la versión serie varían bastante en función del compilador que se utilice, resultando ser la versión compilada con `icc` (31.034 segundos) unos 17 segundos más rápida que la versión compilada con `gcc` (48.694 segundos). En cuanto a las versiones paralelas no se aprecia una diferencia significativa en el tiempo total si varía el compilador utilizado, no siendo

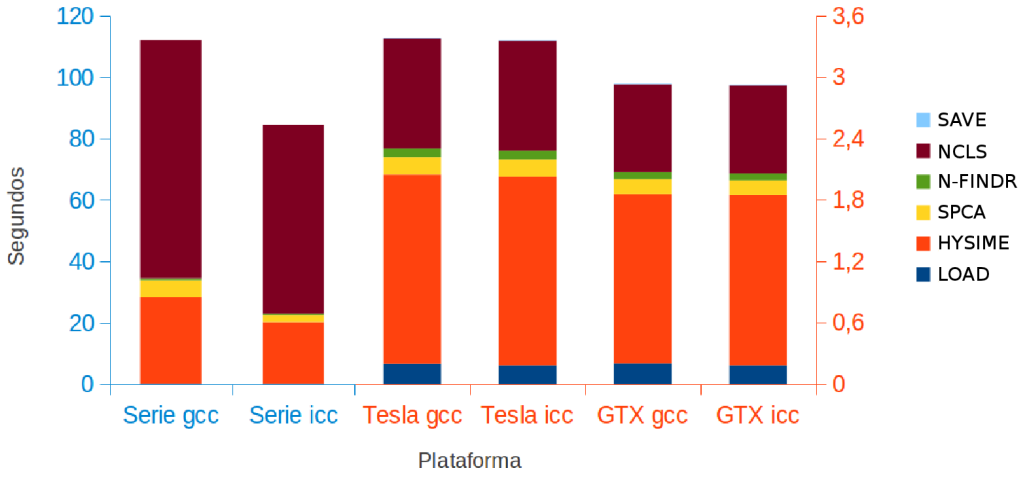

Figura 5.17: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 8 [AVIRIS Cuprite]

así cuando lo que varía es la tarjeta gráfica. En este caso el tiempo de la GPU Tesla C1060 se sitúa en torno a 2.6 segundos y el tiempo de la GPU GTX 580 en torno a 2.3 segundos consiguiendo una ejecución en tiempo real de la cadena de procesamiento. Se obtiene un *speedup* de 20x comparando la versión GTX 580 con la serie `gcc`.

	LOAD	VD	PCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.514	19.986	19.977	6.643	1.547	0.027	48.694	
Tesla gcc	0.568	0.869	0.325	0.659	0.215	0.028	2.664	18.281
GTX gcc	0.568	0.680	0.331	0.564	0.160	0.038	2.343	20.787
Serie icc	0.500	13.881	8.283	7.069	1.273	0.027	31.034	
Tesla icc	0.550	0.872	0.327	0.656	0.216	0.032	2.652	11.700
GTX icc	0.567	0.690	0.330	0.505	0.160	0.029	2.281	13.604

Tabla 5.14: Tiempos de procesamiento de las diferentes etapas de la cadena 1 [AVIRIS World Trade Center]

La Fig. 5.18 nos permite apreciar la contribución de cada etapa de la cadena al tiempo total de ejecución desde un punto de vista gráfico. Puede verse como en las ejecuciones serie las etapas con más peso son la de estimación de el número de *endmembers* y reducción dimensional en el caso de la versión compilada con `gcc`, abarcando un 40 % del tiempo cada una dejando un 15 % para la etapa de extracción de *endmembers*. Para la versión serie compilada con `icc` la etapa de estimación del número de endmembers necesita un 44 % del tiempo total, mientras que las etapas de reducción dimensional y extracción de *endmembers* necesitan en torno a un 25 % cada una. En el caso de las ejecuciones paralelas y para ambas tarjetas, la etapa que abarca la mayor parte del tiempo es la de estimación del número de *endmembers* que toma aproximadamente el 33 % y la

etapa de extracción de los *endmembers* que toma un 25 % del tiempo total. Un factor importante a tener en cuenta son los tiempos de carga de la imagen y de almacenamiento de los resultados, éstos son bastante significativos en las ejecuciones paralelas. En este caso el porcentaje de tiempo dedicado a entrada/salida es de aproximadamente el 20 %.

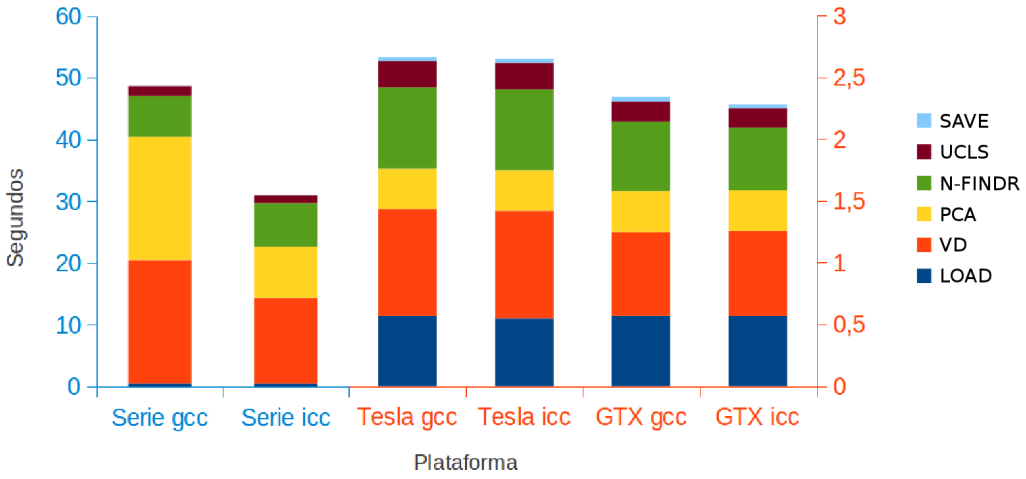

Figura 5.18: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 1 [AVIRIS World Trade Center]

5.4.2.2. Cadena de procesamiento 2

La Tabla 5.15 muestra los resultados de tiempo obtenidos en cada una de las plataformas para esta cadena. En el caso de las ejecuciones serie existe una gran diferencia en utilizar un compilador u otro, concretamente de 100 segundos, siendo el tiempo empleado por la versión compilada con `gcc` de 506.151 segundos y el tiempo empleado por la versión compilada con `icc` de 406.467 segundos. En el caso de las ejecuciones paralelas la diferencia no la marca el compilador empleado sino la tarjeta gráfica utilizada. El tiempo de ejecución para el caso de la GPU Tesla C1060 es de 7.6 segundos mientras que el tiempo de ejecución en el caso de la GPU GTX 580 es de 4.7 segundos lo que hace que esta ejecución se realice en tiempo real. En cuanto a los *speedups* conseguidos éstos son diferentes para cada tarjeta y compilador, siendo el más alto de 107x que se obtiene con la GTX580 usando el compilador `gcc`.

Podemos ver la cantidad de tiempo con la que contribuye cada etapa al tiempo total para esta cadena en la Fig. 5.19. Lo que más llama la atención es el gran porcentaje de tiempo que se dedica a la estimación de abundancias sobre todo en las ejecuciones serie, donde alcanza un 90 % aproximadamente. En esta cadena se utiliza NCLS como método de estimación de abundancias. Como ya vimos, su algoritmo utiliza

	LOAD	VD	PCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.495	20.024	19.976	7.827	457.802	0.027	506.151	
Tesla gcc	0.546	0.863	0.325	0.626	5.258	0.027	7.646	66.198
GTX gcc	0.548	0.678	0.331	0.536	2.579	0.030	4.702	107.656
Serie icc	0.481	13.918	8.278	7.597	376.165	0.027	406.467	
Tesla icc	0.549	0.870	0.328	0.622	5.257	0.030	7.656	53.091
GTX icc	0.538	0.681	0.330	0.483	2.578	0.028	4.638	87.632

Tabla 5.15: Tiempos de procesamiento de las diferentes etapas de la cadena 2 [AVIRIS World Trade Center]

varias multiplicaciones matriciales de forma iterativa. Ésta es la principal causa de este incremento de tiempo en las ejecuciones series, así como de los *speedups* conseguidos. En el caso de las ejecuciones paralelas el porcentaje de tiempo empleado en esta etapa de estimación de abundancias es aproximadamente del 68 % al utilizar la GPU Tesla C1060 y del 50 % al utilizar la GPU GTX 580. El porcentaje de tiempo empleado en entrada/salida es del 7 % en el caso de la Tesla C1060 y del 12 % en el caso de la GTX580.

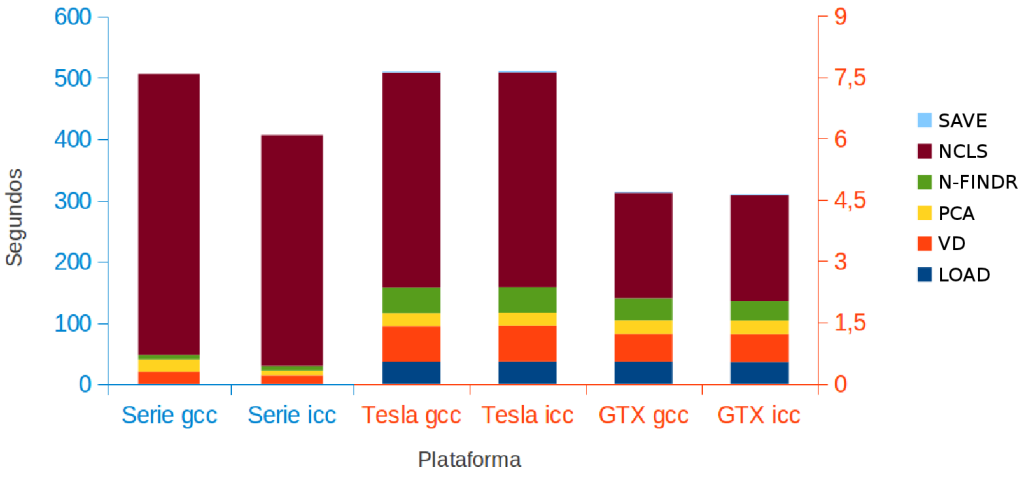

Figura 5.19: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 2 [AVIRIS World Trade Center]

5.4.2.3. Cadena de procesamiento 3

La Tabla 5.16 muestra los tiempos de ejecución de la cadena número 3. Podemos ver que existe una diferencia de 17 segundos entre las dos ejecuciones serie, siendo el tiempo de la versión compilada con `gcc` de 50.188 segundos y el de la versión compilada con `icc` de 33.364 segundos. En el caso de las ejecuciones paralelas existe una diferencia de tiempos de tan solo 0.4 segundos, situándose las ejecuciones que utilizan la GPU Tesla C1060 en 2.8 segundos y las que utilizan la GPU GTX 580 en 2.4 segundos, por tanto se alcanza el tiempo real para esta imagen con las dos tarjetas gráficas. El *speedup* más alto para esta cadena es de 20x y se consigue al comparar la ejecución que utiliza la GPU

GTX 580 con la ejecución serie `gcc`.

	LOAD	VD	SPCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.499	20.275	20.578	7.261	1.549	0.027	50.188	
Tesla gcc	0.549	0.866	0.467	0.769	0.215	0.028	2.892	17.353
GTX gcc	0.552	0.682	0.390	0.618	0.160	0.032	2.434	20.623
Serie icc	0.486	14.528	8.616	8.434	1.273	0.027	33.364	
Tesla icc	0.552	0.866	0.465	0.691	0.217	0.030	2.821	11.827
GTX icc	0.540	0.677	0.384	0.553	0.159	0.028	2.342	14.248

Tabla 5.16: Tiempos de procesamiento de las diferentes etapas de la cadena 3 [AVIRIS World Trade Center]

La Fig. 5.20 muestra gráficamente la contribución de cada etapa al tiempo total de ejecución. De forma parecida a lo que ocurre en la cadena número 1 puede verse como en las ejecuciones serie las etapas con más peso son la de estimación de el número de *endmembers* y reducción dimensional en el caso de la versión compilada con `gcc`, abarcando un 40 % del tiempo cada una dejando un 14 % para la etapa de extracción de *endmembers*. Para la versión compilada con `icc` la etapa de estimación del número de *endmembers* necesita un 43 % del tiempo total mientras que las etapas de reducción dimensional y extracción de *endmembers* necesitan en torno a un 25 % cada una. En el caso de las ejecuciones paralelas y para ambas tarjetas, la etapa que abarca la mayor parte del tiempo es la de estimación del número de *endmembers* que toma aproximadamente el 29 % y la etapa de extracción de los *endmembers* que toma un 24 % del tiempo total. Un factor importante a tener en cuenta son los tiempos de carga de la imagen y de almacenamiento de los resultados, éstos son bastante significativos en las ejecuciones paralelas. En este caso el porcentaje de tiempo dedicado a entrada/salida es de aproximadamente el 24 %.

5.4.2.4. Cadena de procesamiento 4

Los resultados de tiempo de la cadena número 4 se muestran en la Tabla 5.17. En ésta podemos ver que existe una diferencia de 98 segundos entre las dos ejecuciones serie, siendo la versión compilada con `icc` la más rápida empleando 405.126 segundos y la versión compilada con `gcc` la más lenta empleando un total de 503.135 segundos. En el caso de las ejecuciones paralelas la diferencia de tiempos entre las distintas versiones no la marcan los compiladores utilizados sino las tarjetas gráficas. Para el caso de la GPU Tesla C1060 el tiempo empleado es de 7.8 segundos en ambos compiladores, y para el caso de la GPU GTX 580 el tiempo total está en torno a 4.8 segundos lo que supone el alcance de tiempo real. El *speedup* más alto es de 103x y se obtiene con la GPU GTX 580 utilizando `gcc`.

Podemos hacernos una idea de la contribución al tiempo total de cada una de las etapas en esta cadena gracias a la Fig. 5.21. Al igual que la cadena número 2 y las

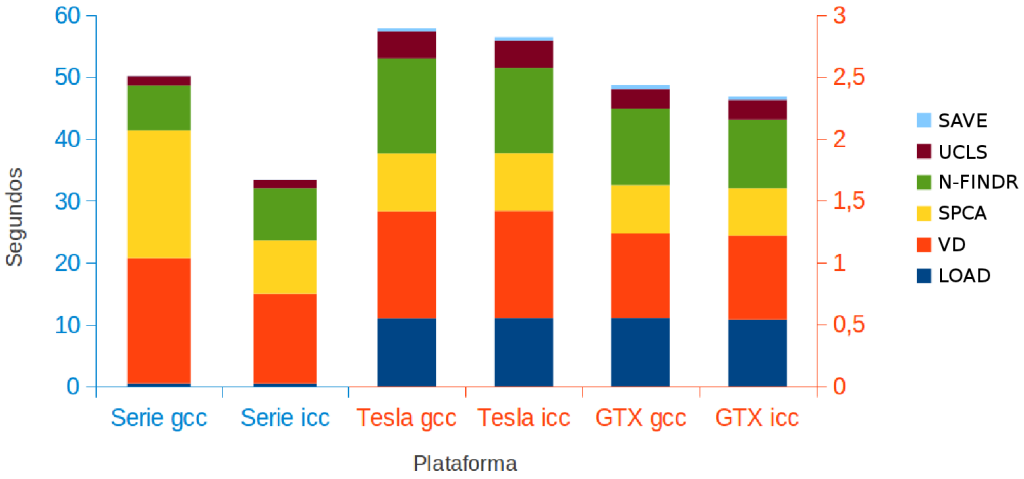

Figura 5.20: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 3 [AVIRIS World Trade Center]

	LOAD	VD	SPCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.500	20.422	20.566	8.347	453.272	0.028	503.135	
Tesla gcc	0.550	0.868	0.466	0.725	5.255	0.027	7.891	63.759
GTX gcc	0.556	0.675	0.388	0.609	2.579	0.031	4.838	103.991
Serie icc	0.488	14.914	8.758	8.638	372.301	0.028	405.126	
Tesla icc	0.552	0.868	0.467	0.690	5.258	0.028	7.863	51.523
GTX icc	0.539	0.672	0.384	0.559	2.578	0.027	4.761	85.099

Tabla 5.17: Tiempos de procesamiento de las diferentes etapas de la cadena 4 [AVIRIS World Trade Center]

cadenas pares, se utiliza NCLS como método de estimación de abundancias. Éste se trata de un método costoso sobre todo en las versiones serie, lo que hace que se produzcan *speedups* altos. Puede verse como la parte de la barra correspondiente a NCLS ocupa prácticamente el 90 % de toda la barra en las versiones serie. En el caso de las versiones paralelas depende de la tarjeta utilizada empleando un 53 % del tiempo en la GPU GTX 580 y un 66 % en la GPU Tesla C1060. Las etapas de entrada/salida emplean un 7 % del tiempo total en caso de la GPU Tesla C1060 y un 12 % en el caso de la GPU GTX 580.

5.4.2.5. Cadena de procesamiento 5

Desde esta cadena hasta la número 8 todas usan HySime como método de estimación del número de *endmembers* que es bastante más costoso que el método VD empleado en otras cadenas, lo que produce un incremento del tiempo en esta etapa que lleva consigo no alcanzar tiempo real en ninguna de las siguientes cadenas. Los tiempos de ejecución para la cadena número 5 en cada una de las plataformas quedan reflejados en la Tabla 5.18. Existe una diferencia de 41 segundos entre la version que se compiló con `gcc` (121.566 segundos) y la que se compiló con `icc` (80.677 segundos). Viendo los tiempos

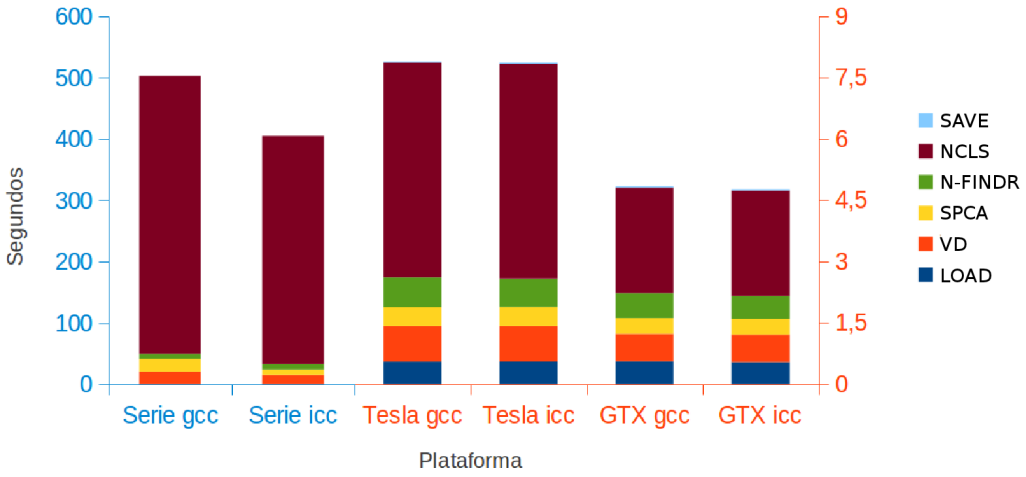

Figura 5.21: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 4 [AVIRIS World Trade Center]

de las ejecuciones paralelas podemos deducir que la diferencia entre las distintas versiones viene dada por la tarjeta utilizada y no por el compilador. Así el tiempo empleado por las versiones que utilizan la GPU Tesla C1060 es de 6.4 segundos y el empleado por las versiones que utilizan la GPU GTX 580 es de 5.3 segundos. El *speedup* más alto conseguido en esta cadena es de 22x y se consigue con la GPU GTX 580 con respecto a la versión serie compilada con `gcc`.

	LOAD	HYSIME	PCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.502	97.581	19.631	2.681	1.151	0.020	121.566	
Tesla gcc	0.547	5.051	0.317	0.317	0.175	0.020	6.427	18.915
GTX gcc	0.547	4.077	0.324	0.270	0.132	0.025	5.375	22.618
Serie icc	0.490	68.011	8.280	2.931	0.945	0.020	80.677	
Tesla icc	0.548	5.084	0.320	0.327	0.176	0.022	6.477	12.456
GTX icc	0.545	4.056	0.323	0.257	0.131	0.021	5.332	15.130

Tabla 5.18: Tiempos de procesamiento de las diferentes etapas de la cadena 5 [AVIRIS World Trade Center]

Como podemos ver en la Fig. 5.22 la etapa de la cadena 5 que más tiempo emplea en todas las versiones es la de estimación del número de *endmembers*, variando desde un 80 % en la version serie `gcc`, un 84 % en la versión serie `icc`, un 75 % para ambas GPUs. Para esta cadena el tiempo dedicado a entrada/salida en las versiones paralelas está en torno al 10 %.

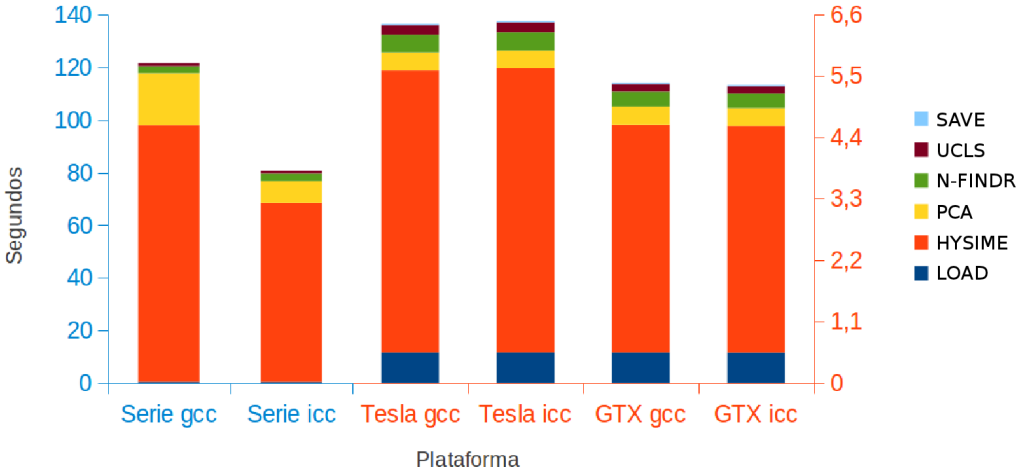

Figura 5.22: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 5 [AVIRIS World Trade Center]

5.4.2.6. Cadena de procesamiento 6

La Tabla 5.19 muestra los resultados de tiempo que se han obtenido en la cadena número 6. Como se viene repitiendo en el caso de las ejecuciones en serie la diferencia en los tiempos viene marcada por el uso de un compilador u otro, no siendo así en el caso de las ejecuciones paralelas donde la diferencia de tiempos viene marcada por el uso de una GPU u otra. En este sentido existe una diferencia de 102 segundos entre la versión serie `gcc` (462.246 segundos) y la versión serie `icc` (360.599 segundos). Para las versiones paralelas tenemos un tiempo de 10.9 segundos usando la GPU Tesla C1060 y un tiempo de 7.6 segundos usando la GPU GTX 580. Como comentamos en la cadena anterior el hecho de usar HySime en la etapa de estimación del número de *endmembers* ralentiza mucho las ejecuciones. Si a eso le sumamos el tiempo empleado por NCLS en la etapa de estimación de las abundancias tenemos como resultado una cadena que no consigue alcanzar procesamiento en tiempo real. El *speedup* más alto conseguido en esta cadena es de 60x y se produce cuando usamos la GPU GTX 580 en comparación con la versión serie `gcc`.

	LOAD	HYSIME	PCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.507	97.679	19.653	2.583	341.804	0.020	462.246	
Tesla gcc	0.551	5.049	0.317	0.326	4.595	0.020	10.858	42.571
GTX gcc	0.549	4.068	0.326	0.289	2.348	0.021	7.602	60.806
Serie icc	0.491	68.049	8.275	2.793	280.970	0.020	360.599	
Tesla icc	0.557	5.092	0.322	0.319	4.597	0.021	10.907	33.061
GTX icc	0.552	4.049	0.323	0.269	2.347	0.020	7.560	47.695

Tabla 5.19: Tiempos de procesamiento de las diferentes etapas de la cadena 6 [AVIRIS World Trade Center]

La Fig. 5.23 nos da un punto de vista gráfico del tiempo empleado en cada etapa con respecto al total. En esta cadena se produce un fenómeno interesante: si comparamos las versiones serie con las paralelas podemos ver que en las versiones serie un 20 % del tiempo total se emplea en ejecutar HySime y un 73 % se emplea en ejecutar NCLS. Sin embargo, en las versiones paralelas se produce el fenómeno contrario, aportando el método HySime un 46 % del tiempo total y el método NCLS un 30 %. Por otro lado el tiempo dedicado a entrada/salida en el caso de las versiones paralelas supone un 7 % del total.

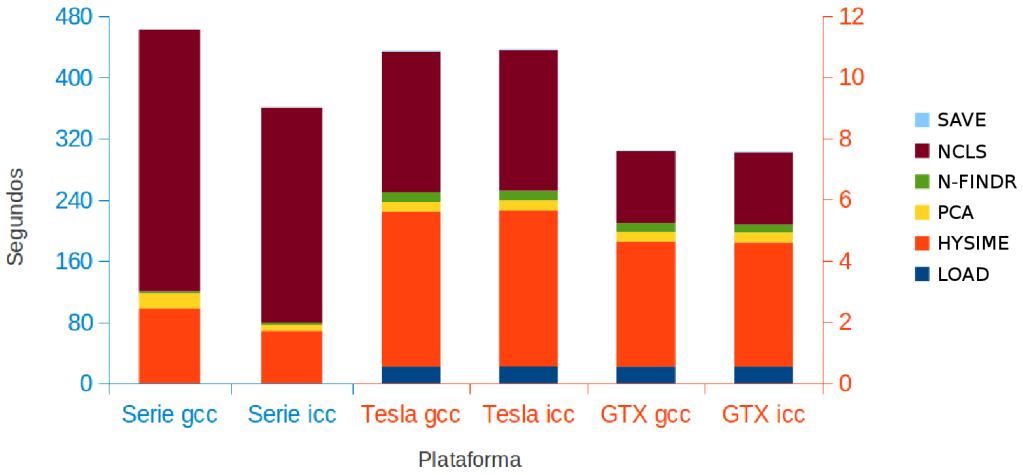

Figura 5.23: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 6 [AVIRIS World Trade Center]

5.4.2.7. Cadena de procesamiento 7

Los resultados de tiempo que se han conseguido en las distintas ejecuciones de la cadena número 7 se muestran en la Tabla 5.20. En dicha tabla podemos apreciar una diferencia significativa de tiempos producida en las versiones serie y que se deben al hecho de utilizar un compilador u otro. Concretamente esta diferencia es de 41 segundos, teniendo un tiempo total de 122.738 segundos la versión serie `gcc` y un tiempo total de 81.554 segundos la versión serie `icc`. En el caso de las versiones paralelas, la diferencia de tiempos no es tan significativa y se debe sobre todo al hecho de utilizar una GPU u otra. En este caso, el tiempo total para la versión que utiliza la GPU Tesla C1060 es de 606 segundos y el tiempo para la versión que utiliza la GPU GTX 580 es de 5.4 segundos. A pesar de ser tiempos muy bajos, no se alcanza la barrera de tiempo real en esta cadena. Se consigue un *speedup* de 22x al comparar la ejecución que utiliza la GPU GTX 580 con la serie `gcc`.

	LOAD	HYSIME	SPCA	N-FINDR	UCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.507	97.563	20.053	3.444	1.151	0.021	122.738	
Tesla gcc	0.551	5.040	0.451	0.372	0.175	0.020	6.608	18.575
GTX gcc	0.551	4.062	0.378	0.299	0.132	0.024	5.446	22.539
Serie icc	0.490	68.154	8.359	3.585	0.946	0.021	81.554	
Tesla icc	0.553	5.085	0.454	0.361	0.176	0.022	6.651	12.262
GTX icc	0.547	4.052	0.374	0.310	0.131	0.020	5.435	15.006

Tabla 5.20: Tiempos de procesamiento de las diferentes etapas de la cadena 7 [AVIRIS World Trade Center]

La Fig. 5.24 muestra un gráfico de barras donde se puede apreciar la contribución al tiempo total de ejecución de cada una de las etapas de la cadena número 7. Puede verse como la etapa dominante en tiempo es, en todos los casos, la de estimación del número de *endmembers* que utiliza el método HySime y que supera el 70 % del tiempo total. Concretamente, en caso de la ejecución serie `gcc` supone un 79 %, para serie `icc` un 83 %, y para el caso de las ejecuciones paralelas un 74 %. Para estas últimas el tiempo dedicado a entrada/salida es de un 10 % aproximadamente.

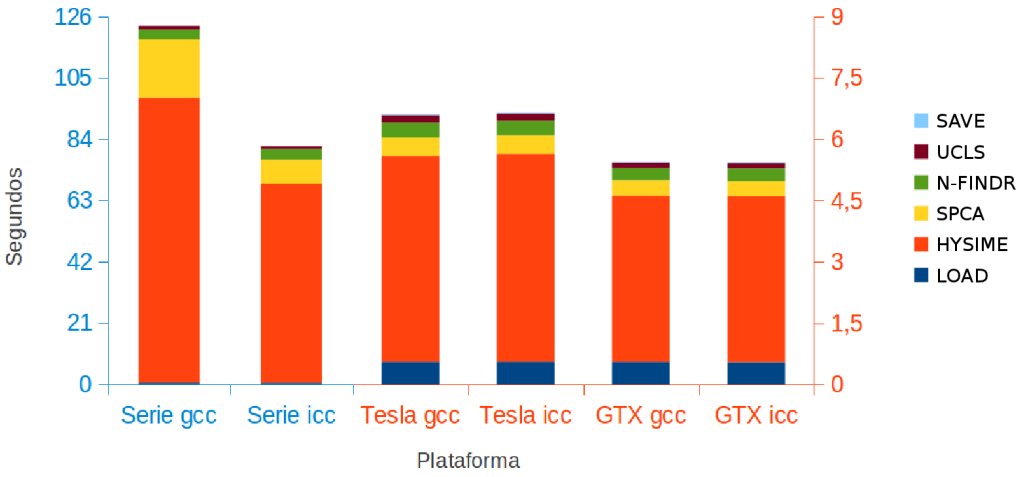

Figura 5.24: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 7 [AVIRIS World Trade Center]

5.4.2.8. Cadena de procesamiento 8

La Tabla 5.21 muestra los resultados de tiempo que se han obtenido al ejecutar las diferentes pruebas con distintas plataformas. Un aspecto que se mantiene constante en todas las cadenas es que existe una diferencia de tiempos producida por el uso de distintos compiladores en las versiones serie, producida por el uso de distintas GPUs en las versiones paralelas. Así, tenemos que esta diferencia es de 101 segundos en las versiones serie, teniendo la versión serie `gcc` un tiempo de 461.301 segundos y la versión serie `icc` un tiempo de 360.673 segundos. En hecho de usar la GPU Tesla C1060 en esta

cadena supone un tiempo de 11 segundos que es independiente del compilador utilizado. Para el caso de la GTX 580 el tiempo total es de 7.6 segundos. Como puede verse, en ningún caso se consigue una ejecución en tiempo real para esta cadena. El *speedup* más alto se consigue, de nuevo, al relacionar la ejecución que utiliza la GPU GTX 580 con la versión serie `gcc`, en este caso es de 60x.

	LOAD	HYSIME	SPCA	N-FINDR	NCLS	SAVE	TOTAL	SPEEDUP
Serie gcc	0.506	97.551	20.063	3.242	339.918	0.020	461.301	
Tesla gcc	0.553	5.055	0.448	0.357	4.603	0.020	11.036	41.801
GTX gcc	0.551	4.069	0.377	0.300	2.348	0.021	7.667	60.170
Serie icc	0.491	68.175	8.341	3.795	279.852	0.020	360.673	
Tesla icc	0.553	5.068	0.454	0.360	4.605	0.021	11.061	32.609
GTX icc	0.549	4.045	0.373	0.283	2.348	0.020	7.618	47.344

Tabla 5.21: Tiempos de procesamiento de las diferentes etapas de la cadena 8 [AVIRIS World Trade Center]

Para hacernos una idea del tiempo que emplea cada una de las etapas de esta cadena en relación al tiempo total de ejecución podemos fijarnos en la Fig. 5.25. En dicha figura puede verse como se produce el mismo fenómeno que ocurría en la cadena número 6, y es que si comparamos las versiones serie con las versiones paralelas podemos ver que en las versiones serie un 20 % del tiempo total se emplea en HySime y un 73 % se emplea en NCLS. Sin embargo, en las versiones paralelas se produce el fenómeno contrario, aportando el método HySime un 46 % del tiempo total y el método NCLS un 30 %. Por otro lado, el tiempo dedicado a entrada/salida en el caso de las versiones paralelas supone un 7 % del total.

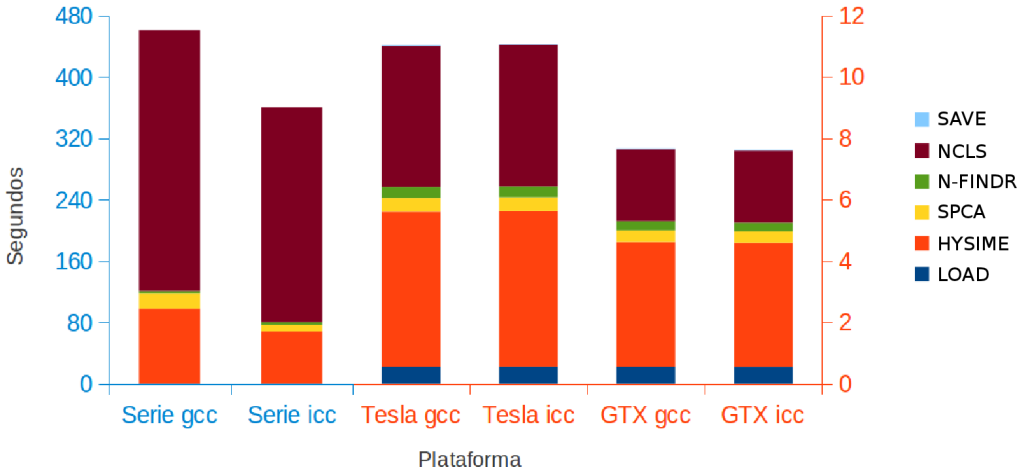

Figura 5.25: Representación gráfica de los tiempos de procesamiento de las diferentes etapas de la cadena 8 [AVIRIS World Trade Center]

5.4.3. Discusión de resultados

En el presente subapartado realizamos una discusión global de los resultados obtenidos por las diferentes implementaciones consideradas con las dos imágenes hiperespectrales utilizadas en el estudio. En concreto, a partir de los resultados presentados en el presente apartado se pueden realizar las siguientes observaciones de carácter general:

- En todos los casos existe una diferencia significativa de tiempos en las versiones serie debida al uso de una compilador u otro. Utilizar el compilador `icc` de Intel en las versiones serie siempre ofrece mejores resultados que utilizar el compilador `gcc`.

- Por otra parte, no se aprecian diferencias significativas de tiempo por el hecho de utilizar compiladores distintos en las versiones paralelas, ni siquiera en la escala reducida de tiempos en la que se encuentran estas ejecuciones. Esto se debe a que el compilador se encarga de la parte serie de las versiones paralelas. En este trabajo se ha intentado maximizar el uso de la GPU en el tiempo de ejecución, por lo que la parte que se ejecuta en serie en estas versiones es muy reducida (básicamente, carga y almacenamiento) y el hecho de utilizar un compilador u otro no supone una diferencia significativa en los tiempos. La diferencia de tiempos en las versiones paralelas se debe básicamente a la utilización de una arquitectura GPU u otra, resultando ser la arquitectura GTX 580 la que ofrece mejores resultados frente a la arquitectura Tesla C1060.

- Si bien no se trata del objetivo principal de este trabajo, alcanzar un *speedup* alto cuando se comparan arquitecturas paralelas frente a arquitecturas serie siempre es importante. Hay que decir que siempre se ha intentado comparar las versiones paralelas con las mejores versiones serie. De ahí que se hayan utilizado distintos compiladores y flags de compilación que reducen en gran medida los tiempos de ejecución serie.

- La Fig. 5.26 muestra un gráfico con los *speedups* que se han conseguido en la imagen AVIRIS Cuprite con cada GPU en cada cadena, tras comparar los tiempos con las versiones serie en cada compilador. Éstos varían desde los 8x hasta los 57x, siendo siempre mayores en las cadenas pares que utilizan el método NCLS como método de estimación de abundancias. Los *speedups* calculados frente a la versión serie compilada con `gcc` son siempre mayores que los calculados frente a la versión serie compilada con `icc`. Esto se debe a que, como ya se ha indicado previamente, este último compilador ofrece mejores resultados de tiempo en la versión serie, lo que reduce notablemente el el factor de aceleración.

- Por su parte, los *speedups* conseguidos para el caso de la imagen AVIRIS World Trade Center pueden verse de forma gráfica en la Fig 5.27. La figura muestra que las proporciones similares a las que aparecen en la Fig. 5.26, cambiando principalmente la escala. Esto indica que las cadenas presentan cierta estabilidad, a pesar de cambiar el tamaño de la imagen procesada. En este caso tenemos *speedups* que

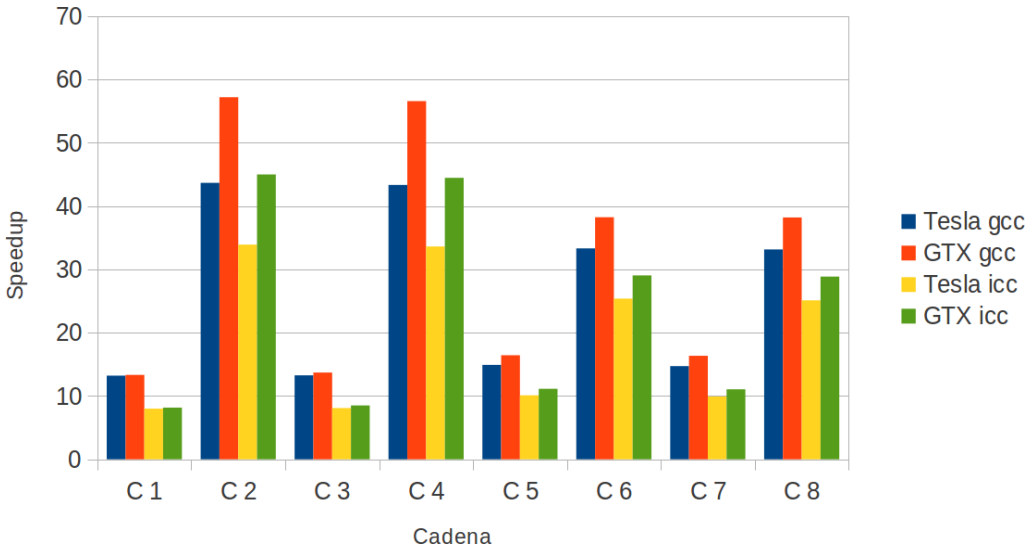

Figura 5.26: Resumen de *speedups* conseguidos en las pruebas con la imagen AVIRIS Cuprite.

van desde 14x a 108x. Como ocurre en los experimentos realizados con la imagen AVIRIS Cuprite, las cadenas que ofrecen mayor *speedup* son aquellas en las que se usa NCLS como método de estimación de abundancias.

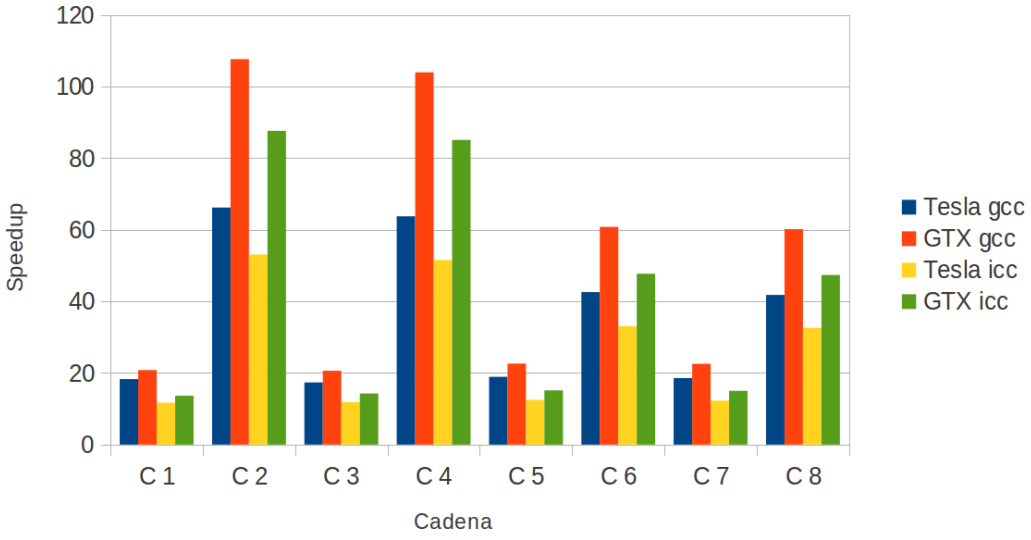

Figura 5.27: Resumen de *speedups* conseguidos en las pruebas con la imagen AVIRIS World Trade Center.

- Uno de los principales objetivos del presente trabajo es alcanzar tiempo real en el procesamiento de imágenes hiperespectrales utilizando cadenas completas de procesamiento. En este sentido, conviene destacar que este objetivo se ha cumplido en un gran número de pruebas realizadas sobre las dos imágenes y utilizando las

dos GPUs consideradas. La Fig. 5.28 muestra un gráfico de barras con una línea en color naranja que representa el umbral de tiempo real. En concreto, en esta figura se aprecia que existen dos cadenas donde ambas GPUs consiguen procesamiento en tiempo real, éstas son la número 1 y la número 3. En ambas se usa VD como método de estimación del número de *endmembers*, N-FINDR como método de extracción de *endmembers* y UCLS como método de estimación de abundancias. La única variante es el método de reducción dimensional, donde se usa PCA en la cadena número 1 y SPCA en la cadena número 3.

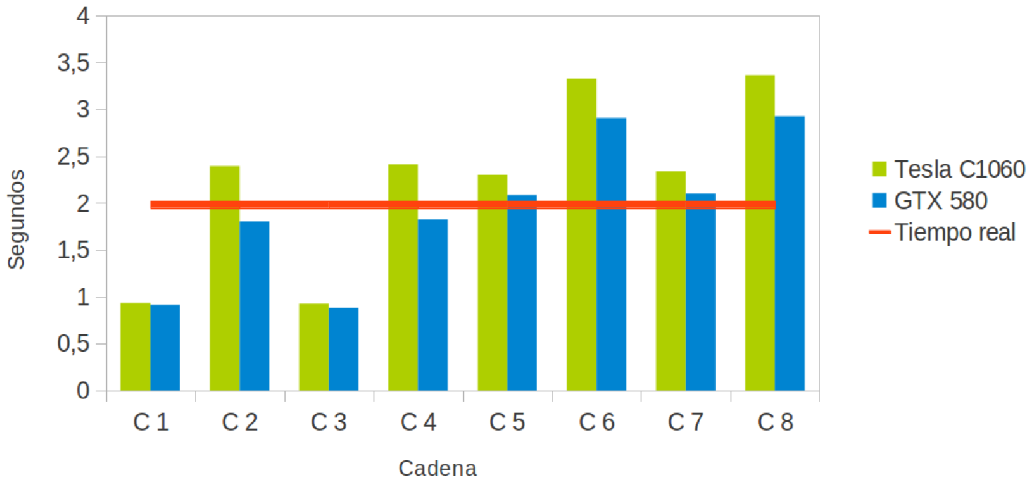

Figura 5.28: Procesamiento en tiempo real de la imagen AVIRIS Cuprite.

- Por otra parte, la Fig. 5.28 indica que existen dos cadenas en las que sólo una de las GPUs, la GTX 580, consigue un procesamiento en tiempo real. Éstas son las cadenas 2 y 4, donde otra vez la única variante esta en utilizar PCA o SPCA como método de reducción dimensional. Ambas cadenas utilizan el método NCLS para estimar las abundancias y, como se mostró en las Figs. 5.11 y 5.13, éste método es el que más porcentaje de tiempo aporta al total en ambas cadenas.

- La Fig. 5.28 indica que existen cuatro cadenas donde ninguna de las GPUs alcanza procesamiento en tiempo real, éstas son las cadenas 5, 6, 7 y 8. Todas ellas tienen como etapa común la de estimación del número de *endmembers*, usando como método HySime que es bastante costoso computacionalmente. Por tanto, parece claro que es preciso optimizar el rendimiento de dicho método para conseguir tiempo real en las cadenas anteriormente mencionadas.

- Finalmente, la Fig. 5.29 muestra una gráfica similar a la de la Fig. 5.28 pero con los tiempos referidos al procesamiento de la imagen AVIRIS World Trade Center. Como ya hemos visto al discutir los *speedups*, los tiempos en cada cadena están muy relacionados y son en cierto modo proporcionales a los que se obtienen con la

imagen AVIRIS Cuprite. Resulta significativo que los resultados de procesamiento en tiempo real se producen exactamente para las mismas cadenas que en los experimentos con la imagen AVIRIS Cuprite. Esto indica claramente la flexibilidad y adaptabilidad de las implementaciones desarrolladas a imágenes hiperespectrales con distintas características y tamaños.

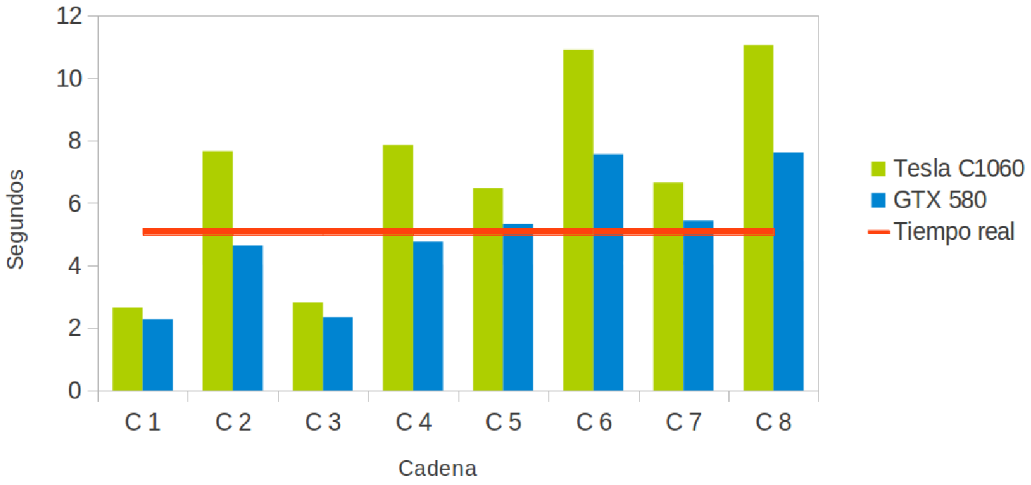

Figura 5.29: Procesamiento en tiempo real de la imagen AVIRIS World Trade Center

- A la vista de los resultados presentados en este apartado, podemos concluir que las cadenas de procesamiento más rápidas son la número 1 y la número 3. A pesar de no conseguir *speedups* muy altos, son las que menos tiempo total emplean en procesar toda la imagen, obteniendo además buenos resultados en lo que a precisión se refiere para las dos imágenes consideradas. Estas dos cadenas son las dos únicas para las que las dos arquitecturas GPU consideradas alcanzan resultados de procesamiento en tiempo real.

5.4.4. Evaluación de la implementación GPU del algoritmo PPI

Como ya se ha comentado, además del método N-FINDR en el presente trabajo se ha implementado también otro algoritmo de extracción de *endmembers*, el método PPI. Dicho método no se ha integrado en las cadenas de procesamiento cuyos resultados aparecen descritos en anteriores subapartados, debido principalmente al carácter semi-supervisado de PPI que hace que el método se utilice principalmente como preprocesado a la etapa de identificación de *endmembers*. En definitiva, el método PPI proporciona una serie de píxels candidatos a ser identificados como *endmembers* que, si bien puede resultar útil para reducir el rendimiento computacional de las implementaciones serie, en las implementaciones paraleas de las cadenas de procesamiento completas (con las que se pretende alcanzar tiempo real) supone más bien una penalización ya que el método N-FINDR puede identificar de forma rápida los *endmembers* a partir de la imagen

original sin preprocesado. Sin embargo, en el presente apartado hemos decidido ilustrar el rendimiento de las implementaciones desarrolladas del algoritmo PPI que pueden ser empleadas en combinación con métodos serie o bien para acelerar el procesamiento semi-supervisado de una imagen hiperespectral.

La Fig. 5.30 muestran dos imágenes de pureza resultantes de aplicar el algoritmo PPI. En concreto, . La Fig. 5.30(a) se ha generado mediante la ejecución del método en su versión serie. La Fig. 5.30(b) se ha generado mediante la ejecución en GPU. Puede apreciarse que los resultados no coinciden exactamente en ambas versiones, esto se debe a que la generación de los *skewers* es totalmente aleatoria, por tanto variará el resultado entre dos ejecuciones ya sean en serie o en paralelo. Un modo de evaluar la precisión es ver que se han seleccionado puntos como extremos en aquellas zonas donde se sabe que existen determinados minerales. En la Fig. 5.30 se han rodeado con círculos de colores aquellas zonas donde están presentes los cinco minerales mas significativos de la escena. Puede verse que en ambas versiones se detectan puntos en estas zonas lo que nos permite suponer que en ambos casos se identifican los píxeles más puros correspondientes a dichos minerales.

Con respecto a los resultados obtenidos en cuanto a aumento de prestaciones en la GPU, conviene destacar que la aceleración por la implementación GPU con respecto al código serie optimizado es muy significativa. En concreto, el código serie fue compilado con el *flag* `-O3` y y ejecutado en el procesador Intel core i7 920 con $k = 15360$ *skewers* sobre la imagen AVIRIS Cuprite, obteniendo un tiempo de procesamiento de 1892.22 segundos. Por su parte, el código GPU implementado en CUDA y ejecutado en la GPU GTX 580 permitió obtener los resultados en un total de 16.51 segundos, lo cual supone un factor de aceleración de 114.67x. Sin embargo, a pesar del *speedup* alcanzado, este tiempo no nos permitiría alcanzar tiempo real en ninguna de las cadenas en caso de ser utilizado como preprocesado antes de N-FINDR. Como se ha comentado anteriormente, el método N-FINDR puede obtener los mismos resultados sin necesidad de utilizar PPI como paso previo, por tanto hemos decidido no incluir PPI en las cadenas completas de desmezclado comentadas en el presente apartado.

5.4.5. Evaluación de la implementación GPU del algoritmo IEA

Por su parte, el método IEA tiene como característica principal el hecho de que abarca dos etapas de la cadena de procesamiento, es decir: extracción de *endmembers* y estimación de abundancias. Como ya se ha comentado en la presente memoria, IEA tiene otra característica fundamental y es el hecho de tratarse de un método iterativo que realiza tantas iteraciones como *endmembers* se pretenden estimar, y que calcula las abundancias en cada iteración. Por estos motivos, se ha decidido evaluar este método aparte de las cadenas de procesamiento completas anteriormente comentadas.

En concreto, al evaluar el método IEA con la imagen AVIRIS Cuprite se observó que obtiene resultados muy similares en cuanto a precisión en la identificación de *endmembers* con respecto a los resultados obtenidos por otras cadenas, obteniendo un valor de similaridad espectral promedio de 6.94° con respecto a las cinco firmas espectrales de referencia USGS consideradas en la Tabla 5.4. Por su parte, la Fig. 5.31 muestra

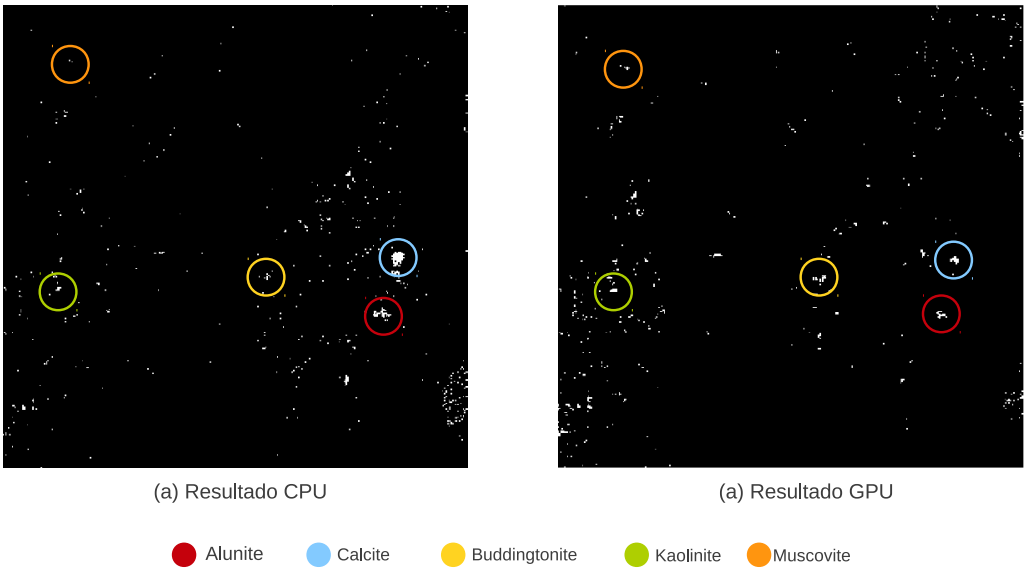

Figura 5.30: Resultados obtenidos por la implementación serie (a) y GPU (b) del método PPI al procesar la imagen AVIRIS Cuprite.

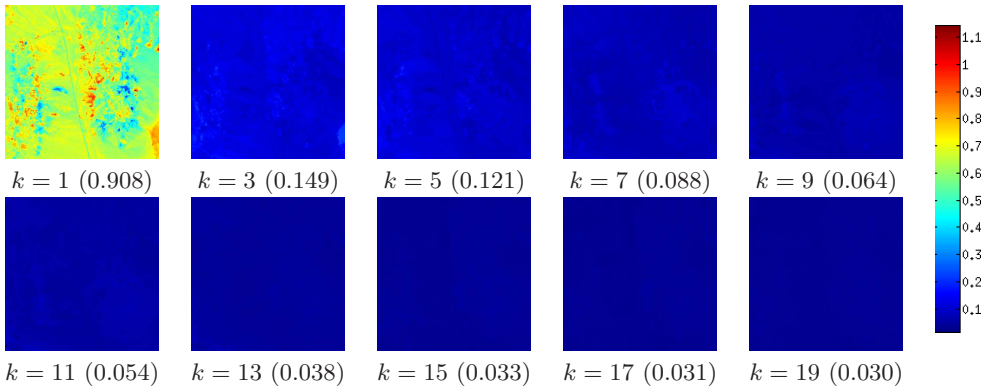

Figura 5.31: RMSE de reconstrucción (entre paréntesis) entre la imagen original y la reconstruida para diferentes iteraciones (k) del método IEA al procesar la imagen AVIRIS Cuprite.

los resultados de error de reconstrucción RMSE obtenidos por IEA para la imagen AVIRIS Cuprite dependiendo del número de iteraciones k realizado. Podemos ver cómo a medida que aumenta el número de iteraciones realizadas el error de reconstrucción RMSE disminuye debido a que cada vez se cuenta con más información para reconstruir la imagen. Los resultados de RMSE son comparables e incluso superan a los obtenidos por cualquiera de las cadenas anteriormente comentadas, lo cual indica que IEA es un método altamente interesante para desmezclar datos hiperespectrales. Dicho método necesita ser combinado con un método de estimación del número de *endmembers* como VD ó HySime para formar una cadena completa de desmezclado.

Para finalizar este apartado, es importante destacar que la aceleración obtenida por la

implementación GPU de IEA con respecto al código serie optimizado es muy significativa. En concreto, el código serie fue compilado con el *flag* `-O3` y y ejecutado en el procesador Intel core i7 920 sobre la imagen AVIRIS Cuprite, obteniendo un tiempo de procesamiento de 69.33 segundos. Por su parte, el código GPU implementado en CUDA y ejecutado en la GPU GTX 580 permitió obtener los resultados en un total de 0.68 segundos, lo cual supone un factor de aceleración de 101.02x. Este tiempo nos permite alcanzar tiempo real si empleamos VD para la estimación del número de *endmembers*, pero todavía no permitiría procesamiento en tiempo real si se utiliza HySime para dicha tarea.

Capítulo 6

Conclusiones y Líneas Futuras

El análisis de imágenes hiperespectrales constituye una parte muy activa de la observación remota de la Tierra, con numerosas nuevas misiones activas y en desarrollo que permitirán una cobertura total y multi-temporal de nuestro planeta, con potenciales aplicaciones en áreas de gran relevancia como el estudio del cambio climático o la evolución de los ecosistemas terrestres. Uno de los principales problemas a la hora de analizar de forma precisa la información proporcionada por imágenes hiperespectrales de la superficie terrestre es el problema de la mezcla, que resulta de la gran complejidad existente en la superficie del planeta a la hora de realizar un modelado de la misma a gran escala (como es el objetivo de las misiones hiperespectrales actuales). En este proceso, cobra especial importancia la eficiencia en el procesamiento de los datos, debido a que la resolución espacial, espectral y temporal de los mismos es muy elevada y es preciso aplicar técnicas de procesamiento hiperespectral eficientes capaces de solventar el problema de la mezcla.

En el presente trabajo se ha abordado el problema de la mezcla desde la perspectiva de diversas técnicas de análisis hiperespectral que permiten expresar los píxeles de la imagen como una combinación (lineal o no) de elementos espectralmente puros (*endmembers*) ponderados por sus correspondientes fracciones de abundancia. En concreto, se han estudiado en detalle numerosas aproximaciones existentes en la literatura y se han implementado, por primera vez, en arquitecturas de tipo GPU, alcanzando resultados de procesamiento en tiempo real para cadenas completa de desmezclado espectral también por primera vez en la literatura. En la validación realizada con imágenes hiperespectrales reales, se ha conseguido alcanzar gran precisión en el análisis (a nivel sub-píxel) en tiempo real y de forma portable entre diferentes arquitecturas GPU, realizando una contribución altamente novedosa a la literatura existente en análisis hiperespectral y que se espera que tenga un gran impacto en las nuevas misiones hiperespectrales de observación remota de la Tierra.

En concreto, para llevar a cabo el exhaustivo estudio comparativo realizado con vistas a validar la precisión y rendimiento computacional de las técnicas desarrolladas, dichas técnicas (en sus versiones secuencial y GPU) se han aplicado casos de estudio en los que la necesidad de obtener resultados en tiempo real es manifiesta. Por un lado, el estudio se

ha centrado en el análisis de una imagen hiperespectral obtenida sobre la zona del World Trade Center de Nueva York, días después del atentado terrorista del 11 de Septiembre de 2001. Por otro lado, también se ha utilizado una imagen obtenida sobre la región minera de Cuprite en Nevada en la que se intenta detectar minerales para su extracción. En dicha imagen, la necesidad de procesamiento en tiempo real no es tan crítica pero dicha imagen dispone de abundante información de referencia que nos ha permitido evaluar la precisión de los métodos utilizados. En ambos casos las imágenes fueron obtenidas por el sensor AVIRIS de NASA, uno de los sensores de referencia en la comunidad científica dedicada al análisis de datos hiperespectrales. En este sentido, es importante destacar que las prestaciones de los algoritmos desarrollados han sido demostradas desde la perspectiva de aplicaciones reales de gran actualidad, quedando patentes las posibilidades de explotación de las técnicas propuestas con motivo del presente proyecto en el contexto específico de dichas aplicaciones.

Las versiones paralelas de los algoritmos presentados en este trabajo se han implementado sobre dos GPUs de NVidia de dos familias diferentes, como son la Tesla C1060 (de la familia Tesla) y la GTX 580 (de la familia Fermi). En la actualidad, y en parte a raíz de desarrollos como los presentados este trabajo, la computación GPU aplicada a análisis de imágenes hiperespectrales representa un área en el que compañías y agencias internacionales están comenzando a realizar desarrollos orientados a funciones específicas, desde la implementación de algoritmos de desmezclado tratados en la presente memoria, pasando por otras aplicaciones en los dominios de clasificación, detección de objetivos y compresión de imágenes hiperespectrales. Por tanto, se espera que los desarrollos realizados sirvan como referencia a nuevos desarrollos en otros campos relacionados con análisis de imágenes hiperespectrales. Los resultados obtenidos en cuanto a aceleración y precisión en el análisis reflejan las muy elevadas prestaciones que las tarjetas gráficas programables ofrecen en cuanto al procesamiento de datos masivamente dimensionales, tales como las imágenes hiperespectrales que hoy en día proporcionan de forma rutinaria numerosos sensores de observación remota de la Tierra operados por agencias internacionales como NASA o la Agencia Europea del Espacio (ESA).

Entre las líneas futuras de trabajo que tenemos previsto abordar para ampliar los estudios desarrollados en la presente memoria destacan en primer lugar, la posibilidad de desarrollar versiones paralelas capaces de funcionar en otros tipos de arquitecturas tales como sistemas multi-núcleo ó *clusters* de GPUs, un tipo de arquitectura que cada vez es más popular (prueba de ello, destacamos que las arquitecturas que ocupan las primeras posiciones en la lista de supercomputadores más rápidos[1] permiten una explotación híbrida basada en paralelismo funcional entre nodos complementado con procesamiento local en GPUs). Esto nos permitiría procesar grandes cantidades de datos, tales como las que se encuentran disponibles en repositorios de imágenes hiperespectrales en NASA y ESA. También pretendemos desarrollar nuevos algoritmos de análisis hiperespectral susceptibles de ser utilizados en aplicaciones de clasificación, detección de objetivos y compresión a bordo de datos hiperespectrales. En este sentido, anticipamos que el

[1] http://www.top500.org

principal obstáculo a la incorporación plena de la tecnología GPU en misiones reales de observación remota de la tierra se centra en el elevado consumo de las GPUs en comparación con otro tipo de arquitecturas como FPGAs o bien procesadores digitales de señal (DSP). En este sentido, conviene destacar los esfuerzos que están siendo realizados por agencias internacionales de relevancia en aplicaciones de observación remota de la tierra, tales como ESA ó NASA, orientados a incorporar la tecnología GPU en las nuevas misiones de observación remota de la tierra desde el espacio mediante la apuesta por el desarrollo de GPUs de bajo consumo y tolerantes a las radiaciones que se encuentran en el espacio, área en la que anticipamos importantes desarrollos en los próximos años. En este sentido, los desarrollos realizados en el presente trabajo permiten al candidato y al grupo de investigación encontrarse en una posición privilegiada a la hora de contribuir con algoritmos funcionales y prácticos, capaces de solventar problemas reales de gran relevancia social como el utilizado como caso de estudio en las comparativas realizadas en el presente trabajo.

Bibliografía

[1] J. B. Adams, M. O. Smith, and P. E. Johnson. Spectral mixture modeling: a new analysis of rock and soil types at the viking lander 1 site. *Journal of Geophysical Research*, 91:8098–8112, 1986. [Citado en págs. 10 y 16]

[2] J. M. Bioucas-Dias and J. M. P. Nascimento. Hyperspectral subspace identification. *IEEE Trans. Geosci. and Remote Sens.*, 46(8):2435–2445, 2008. [Citado en pág. 20]

[3] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot. Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5:354–379, 2012. [Citado en págs. 12, 13, 15, 17, 19 y 26]

[4] J. Boardman, F. A. Kruse, and R. O. Green. Mapping target signatures via partial unmixing of AVIRIS data. In *Proc. JPL Airborne Earth Science Workshop*, pages 23–26. JPL Publication, 1995. [Citado en pág. 26]

[5] C. C. Borel and S. A. W. Gerslt. Nonlinear spectral mixing models for vegetative and soil surfaces. *Remote Sens. Environment*, 47:403–416, 1994. [Citado en págs. 11, 12 y 14]

[6] J. Brazile, R. A. Neville, K. Staenz, D. Schlaepfer, L. Sun, and K. I. Itten. Cluster versus grid for operation generation of atcor's modtran-based look up table. *Parallel Computing*, 34:32–46, 2008. [Citado en págs. 37 y 38]

[7] R. Brightwell, L. Fisk, D. Greenberg, T. Hudson, M. Levenhagen, A. Maccabe, and R. Riesen. Massively parallel computing using commodity components. *Parallel Computing*, 26:243–266, 2000. [Citado en pág. 35]

[8] C.-I Chang. *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. Kluwer Academic/Plenum Publishers: New York, 2003. [Citado en págs. 9 y 20]

[9] C.-I Chang and Q. Du. Estimation of number of spectrally distinct signal sources in hyperspectral imagery. *IEEE Trans. Geosci. and Remote Sens.*, 42(3):608–619, 2004. [Citado en pág. 20]

[10] R. N. Clark and T. L. Roush. Reflectance spectroscopy: Quantitative analysis techniques for remote sensing applications. *J. Geophys. Res.*, 89(7):6329–6340, 1984. [Citado en pág. 13]

[11] R.N Clark. *Spectroscopy of Rocks and Minerals, and Principles of Spectroscopy.* John Wiley and Sons, 1999. [Citado en pág. 78]

[12] R.N. Clark, G. Swayze, T.V.V. King, E. Livo, J.B. Dalton, and R.F. Kokaly. Tetracorder and expert system feature identification rules for reflectance (and emittance) spectroscopy analysis 1: Visible to near-infrared detection of minerals, organics, vegetation, water, amorphous and other materials. *Proc. XI NASA/JPL Airborne Earth Science Workshop*, 1999. [Citado en pág. 79]

[13] R.N. Clark and G.A. Swayze. Evolution in imaging spectroscopy analysis and sensor signal-to-noise: An examination of how far we have come. *Proc. XI NASA/JPL Airborne Earth Science Workshop*, 1996. [Citado en pág. 79]

[14] R.N. Clark, G.A. Swayze, A.J. Gallagher, T.V.V. King, and W.M. Calvin. The u. s. geological survey, digital spectral library: Version 1: 0.2 to 3.0 microns. Open file report 93-592, USGS, 1993. [Citado en pág. 79]

[15] M. Clint and A. Jenning. The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration. *The Computer Journal*, 13:76–80, 1970. [Citado en pág. 25]

[16] M. D. Craig. Minimum-volume transforms for remotely sensed data. *IEEE Trans. Geosci. and Remote Sens.*, 32:542–552, 1994. [Citado en pág. 18]

[17] J. Dorband, J. Palencia, and U. Ranawake. Commodity computing clusters at goddard space flight center. *Journal of Space Communication*, 3:1, 2003. [Citado en pág. 35]

[18] Q. Du and J. E. Fowler. Low-complexity principal component analysis for hyperspectral image compression. *International Journal High Performance Computing Applications*, 22:273–286, 2009. [Citado en pág. 38]

[19] Q. Du and R. Nekovei. Fast real-time onboard processing of hyperspectral imagery for detection and classification. *Journal of Real-Time Image Processing*, 22:438–448, 2009. [Citado en pág. 38]

[20] E. El-Araby, T. El-Ghazawi, J. L. Moigne, and R. Irish. Reconfigurable processing for satellite on-board automatic cloud cover assessment. *Journal of Real-Time Image Processing*, 5:245–259, 2009. [Citado en pág. 39]

[21] T. El-Ghazawi, S. Kaewpijit, and J. L. Moigne. Parallel and adaptive reduction of hyperspectral data to intrinsic dimensionality. *Cluster Computing*, 1:102–110, 2001. [Citado en pág. 35]

[22] M. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, September 1972. [Citado en pág. 41]

[23] A. R. Gillespie, M. O. Smith, J. B. Adams, S. C. Willis, A. F. Fisher, and D. E. Sabol. Interpretation of residual images: Spectral mixture analysis of AVIRIS images, Owens Valley, California. In R. O. Green, editor, *Proc. 2nd AVIRIS Workshop*, volume 90–54, pages 243–270, 1990. [Citado en pág. 10]

[24] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock. Imaging spectrometry for Earth remote sensing. *Science*, 228:1147–1153, 1985. [Citado en pág. 9]

[25] G.H. Golub and C.F. Van Loan. Matrix computations. *Johns Hopkins University Press*, 3:406–408, 1996. [Citado en pág. 24]

[26] C. Gonzalez, J. Resano, D. Mozos, A. Plaza, and D. Valencia. FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis. *EURASIP Journal of Advances in Signal Processing*, 969806:1–13, 2010. [Citado en págs. 38, 39 y 40]

[27] C. Gonzalez, S. Sanchez, A. Paz, J. Resano, D. Mozos, and A. Plaza. Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing. *Integration, the VLSI Journal*, 2012. [Citado en págs. 38 y 39]

[28] 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. *IEEE Trans. Geosci. and Remote Sens.*, 26:65–74, 1988. [Citado en pág. 24]

[29] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis, et al. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS). *Remote Sens. Environment*, 65(3):227–248, 1998. [Citado en pág. 10]

[30] B. Hapke. Bidirection reflectance spectroscopy. I. theory. *J. Geophys. Res.*, 86:3039–3054, 1981. [Citado en pág. 13]

[31] B. Hapke. *Theory of Reflectance and Emittance Spectroscopy*. Cambridge Univ. Press, 1993. [Citado en págs. 13 y 14]

[32] S. Hauck. The roles of FPGAs in reprogrammable systems. *Proceedings of the IEEE*, 86:615–638, 1998. [Citado en pág. 39]

[33] D. Heinz and C.-I Chang. Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery. *IEEE Trans. Geosci. and Remote Sens.*, 39:529–545, 2001. [Citado en pág. 30]

[34] E. M. T. Hendrix, I. García, J. Plaza, and A. Plaza. Minimum volume simplicial enclosure for spectral unmixing. Technical Report MWP-50, Mansholt Graduate School, Wageningen University, The Netherlands, 2010. [Citado en pág. 18]

[35] M. D. Iordache, J. Bioucas-Dias, and A. Plaza. Sparse unmixing of hyperspectral data. *IEEE Trans. Geosci. and Remote Sens.*, 49(6):2014–2039, 2011. [Citado en págs. 18 y 29]

[36] P. E. Johnson, M. O. Smith, S. Taylor-George, and J. B. Adams. A semiempirical method for analysis of the reflectance spectra for binary mineral mixtures. *J. Geophys. Res.*, 88:3557–3561, 1983. [Citado en pág. 10]

[37] I. T. Jolliffe. *Principal Component Analysis*. New York: Spriger Verlag, 1986. [Citado en pág. 22]

[38] N. Keshava. A survey of spectral unmixing algorithms. *Lincoln Lab. J.*, 14(1):55–78, 2003. [Citado en pág. 10]

[39] N. Keshava, J. Kerekes, D. Manolakis, and G. Shaw. An algorithm taxonomy for hyperspectral unmixing. In *Proc. SPIE AeroSense Conference on Algorithms for Multispectral and Hyperspectral Imagery VI*, volume 4049, pages 42–63, 2000. [Citado en pág. 16]

[40] N. Keshava and J. F. Mustard. Spectral unmixing. *IEEE Signal Process. Mag.*, 19(1):44–57, 2002. [Citado en págs. 12, 13, 14, 16 y 83]

[41] F. A. Kruse, J. W. Boardman, and J. F. Huntington. Comparison of airborne hyperspectral data and EO-1 Hyperion for mineral mapping. *IEEE Trans. Geosci. and Remote Sens.*, 41(6):1388–1400, 2003. [Citado en pág. 10]

[42] D. A. Landgrebe. Hyperspectral image data analysis. *IEEE Signal Processing Magazine*, 19(1):17–28, 2002. [Citado en pág. 35]

[43] D. A. Landgrebe. *Signal Theory Methods in Multispectral Remote Sensing*. John Wiley & Sons: New York, 2003. [Citado en pág. 9]

[44] A. Lastovetsky and J. Dongarra. *High-Performance Heterogeneous Computing*. New York: Wiley, 2009. [Citado en pág. 37]

[45] J. Li and J. Bioucas-Dias. Minimum volume simplex analysis: a fast algorithm to unmix hyperspectral data. In *Proc. IEEE Int. Conf. Geosci. Remote Sens. (IGARSS)*, volume 3, pages 250–253, 2008. [Citado en pág. 18]

[46] S. Liangrocapart and M. Petrou. Mixed pixels classification. In *Proc. SPIE Image and Signal Process. Remote Sensing IV*, volume 3500, pages 72–83, 1998. [Citado en págs. 13 y 16]

[47] F. Liu, F. Seinstra, and A. Plaza. Parallel hyperspectral image processing on multi-cluster systems. *SPIE Journal of Applied Remote Sensing*, 2011. Aceptado para publicación. [Citado en pág. 37]

[48] G. Martin and A. Plaza. Region-based spatial preprocessing for endmember extraction and spectral unmixing. *IEEE Geosci. Remote Sens. Lett.*, 8:745–749, 2011. [Citado en pág. 18]

[49] L. Miao and H. Qi. Endmember extraction from highly mixed data using minimum volume constrained nonnegative matrix factorization. *IEEE Trans. Geosci. and Remote Sens.*, 45(3):765–777, 2007. [Citado en pág. 18]

[50] J. M. Molero, A. Paz, E. M. Garzón, J.A. Martínez, A. Plaza, and I. García. Fast anomaly detection in hyperspectral images with RX method on heterogenous clusters. *Journal of Supercomputing*, 2011. [Citado en pág. 37]

[51] J. Nascimento and J. Bioucas-Dias. Does independent component analysis play a role in unmixing hyperspectral data? *IEEE Trans. Geosci. and Remote Sens.*, 43(1):175–187, 2005. [Citado en pág. 24]

[52] R. A. Neville, K. Staenz, T. Szeredi, J. Lefebvre, and P. Hauff. Automatic endmember extraction from hyperspectral data for mineral exploration. In *Proc. Canadian Symp. Remote Sens.*, pages 21–24, 1999. [Citado en pág. 32]

[53] M. Parente, J. L. Bishop, and J. F. Bell. Spectral unmixing for mineral identification in pancam images of soils in Gusev crater, Mars. *Icarus*, 203:421–436, 2009. [Citado en pág. 38]

[54] B.N. Parlett. The Symmetric Eigenvalue Problem. In *Society for Industrial and Applied Mathematics*, 1998. [Citado en págs. 24 y 25]

[55] B.N. Parlett and I.S. Dhillon. Relatively robust representations of symmetric tridiagonals. *Linear Algebra and its Applications*, 309(1–3):121–151, 2000. [Citado en págs. 24 y 25]

[56] A. Paz, S. Blázquez, S. Muñoz, A. Plaza, and P. Martínez. Implementación paralela y validación preliminar de un nuevo algoritmo para detectar targets en imágenes hiperespectrales. In *XX Jornadas de Paralelismo*, Zaragoza, 2007. [Citado en pág. 40]

[57] A. Paz and A. Plaza. Cluster versus GPU implementation of an orthogonal target detection algorithm for remotely sensed hyperspectral images. In *IEEE International Conference on Cluster Computing (Cluster'10)*, Heraklion, Greece, 2010. [Citado en pág. 40]

[58] M. Petrou and P. G. Foschi. Confidence in linear spectral unmixing of single pixels. *IEEE Trans. Geosci. and Remote Sens.*, 37:624–626, 1999. [Citado en pág. 12]

[59] A. Plaza. Special issue on architectures and techniques for real-time processing of remotely sensed images. *Journal of Real-Time Image Processing*, 4:191–193, 2009. [Citado en págs. 38 y 39]

[60] A. Plaza, J. A. Benediktsson, J. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, J. Gualtieri, M. Marconcini, J. C. Tilton, and G. Trianni. Recent advances in techniques for hyperspectral image processing. *Remote Sensing of Environment*, 113:110–122, 2009. [Citado en págs. 9 y 35]

[61] A. Plaza and C.-I. Chang. *High Performance Computing in Remote Sensing*. Chapman & Hall, 2007. [Citado en pág. 34]

[62] A. Plaza and C.-I. Chang. Clusters versus FPGA for parallel processing of hyperspectral imagery. *International Journal High Performance Computing Applications*, 22(4):366–385, 2008. [Citado en págs. 35 y 39]

[63] A. Plaza, Q. Du, J. Bioucas-Dias, X. Jia, and F. Kruse. Foreword to the special issue on spectral unmixing of remotely sensed data. *IEEE Trans. Geosci. and Remote Sens.*, 49(11):4103–4110, 2011. [Citado en págs. 11, 12 y 13]

[64] A. Plaza, P. Martinez, R. Perez, and J. Plaza. Spatial/spectral endmember extraction by multidimensional morphological operations. *IEEE Trans. Geosci. and Remote Sens.*, 40:2025–2041, 2002. [Citado en pág. 18]

[65] A. Plaza, J. Plaza, and A. Paz. Improving the scalability of parallel algorithms for hyperspectral image analysis using adaptive message compression. *Computers & Geosciences*, 36(10):1283–1291, 2010. [Citado en pág. 35]

[66] A. Plaza, J. Plaza, A. Paz, and S. Sanchez. Parallel hyperspectral image and signal processing. *IEEE Signal Processing Magazine*, 28:119–126, 2011. [Citado en págs. 34, 39 y 40]

[67] A. Plaza, J. Plaza, S. Sanchez, and A. Paz. Lossy hyperspectral image compression tuned for spectral mixture analysis applications on NVidia graphics processing units. In *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, California, 2009. [Citado en pág. 40]

[68] A. Plaza, J. Plaza, S. Sanchez, and A. Paz. Optimizing a hyperspectral image processing chain using heterogeneous and GPU-based parallel computing architectures. In *Proceedings of the 9th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE'09)*, Gijón, Spain, 2009. [Citado en pág. 40]

[69] A. Plaza, J. Plaza, and D. Valencia. Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data. *The Journal of Supercomputing*, 40(1):81–107, 2007. [Citado en pág. 37]

[70] A. Plaza, S. Sanchez, A. Paz, and J. Plaza. GPUs versus FPGAs for onboard compression of hyperspectral data. In *2nd International Workshop on On-Board Payload Data Compression (OBPDC'10)*, Toulouse, France, 2010. [Citado en pág. 38]

[71] A. Plaza, D. Valencia, and J. Plaza. An experimental comparison of parallel algorithms for hyperspectral analysis using homogeneous and heterogeneous networks of workstations. *Parallel Computing*, 34(2):92–114, 2008. [Citado en pág. 37]

[72] A. Plaza, D. Valencia, J. Plaza, and C.-I Chang. Parallel implementation of endmember extraction algorithms from hyperspectral data. *IEEE Geoscience and Remote Sensing Letters*, 3(3):334–338, 2006. [Citado en pág. 50]

[73] A. Plaza, D. Valencia, J. Plaza, and P. Martínez. Commodity cluster-based parallel processing of hyperspectral imagery. *Journal of Parallel and Distributed Computing*, 66(3):345–358, 2006. [Citado en págs. 34, 35 y 39]

[74] J. Plaza, A. Plaza, D. Valencia, and A. Paz. Massively parallel processing of hyperspectral images. In *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, California, 2009. [Citado en págs. 35 y 36]

[75] A. Remon, S. Sanchez, A. Paz, E. S. Quintana-Orti, and A. Plaza. Real-time endmember extraction on multi-core processors. *IEEE Geoscience and Remote Sensing Letters*, 8:924–928, 2011. [Citado en pág. 40]

[76] J. A. Richards and X. Jia. *Remote Sensing Digital Image Analysis: An Introduction*. Springer, 2006. [Citado en pág. 22]

[77] W. Rivera, C. Carvajal, and W. Lugo. Service oriented architecture Grid based environment for hyperspectral imaging analysis. *International Journal of Information Technology*, 11(4):104–111, 2005. [Citado en pág. 37]

[78] D. M. Rogge, B. Rivardand, J. Zhangand A. Sanchez, J. Harrisand, and J. Feng. Integration of spatial–spectral information for the improved extraction of endmembers. *Remote Sensing of Environment*, 110:287–303, 2007. [Citado en pág. 18]

[79] Y. Saad. Numerical Methods for Large Eigenvalue Problems, Revised Edition. In *Society for Industrial and Applied Mathematics*, 2011. [Citado en págs. 24 y 25]

[80] S. Sanchez and A.Plaza. Fast determination of the number of endmembers for real-time hyperspectral unmixing on GPUs. *Journal of Real-Time Image Processing*, 2012. [Citado en págs. 38 y 40]

[81] S. Sanchez, G. Martin, and A. Plaza. Parallel implementation of the n-findr endmember extraction algorithm on commodity graphics processing units. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'10)*, Hawaii, USA, 2010. [Citado en pág. 39]

[82] S. Sanchez, G. Martin, A. Plaza, and C.-I Chang. GPU implementation of fully constrained linear spectral unmixing for remotely sensed hyperspectral data exploitation. In *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2010. [Citado en pág. 39]

[83] S. Sanchez, G. Martin, A. Plaza, and J. Plaza. Near real-time endmember extraction from remotely sensed hyperspectral data using nvidia GPUs. In *SPIE Photonics Europe, Real-Time Image and Video Processing Conference*, Brussels, Belgium, 2010. [Citado en pág. 38]

[84] S. Sanchez, A. Paz, G. Martin, and A. Plaza. Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units. *Concurrency*

and Computation: Practice & Experience, 2011. Aceptada para publicación. [Citado en pág. 40]

[85] S. Sanchez, A. Paz, and A. Plaza. Real-time spectral unmixing using iterative error analysis on commodity graphics processing units. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'11)*, Vancouver, Canada, 2011. [Citado en pág. 38]

[86] S. Sanchez and A. Plaza. GPU implementation of the pixel purity index algorithm for hyperspectral image analysis. In *IEEE International Conference on Cluster Computing (Cluster'10)*, Heraklion, Greece, 2010. [Citado en pág. 39]

[87] S. Sanchez and A. Plaza. Implementación paralela del algoritmo pixel purity index para análisis hiperespectral en GPUs. In *Jornadas de Paralelismo, Congreso Español de Informática (CEDI'2010)*, Valencia, Spain, 2010. [Citado en pág. 38]

[88] S. Sanchez and A. Plaza. A comparative analysis of GPU implementations of spectral unmixing algorithms. In *SPIE Remote Sensing Europe, High Performance Computing in Remote Sensing Conference*, Prague, Czech Republic, 2011. [Citado en pág. 38]

[89] S. Sanchez and A. Plaza. Real-time implementation of a full hyperspectral unmixing chain on graphics processing units. In *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2011. [Citado en pág. 38]

[90] S. Sanchez and A. Plaza. GPU implementation of the image space reconstruction algorithm for remotely sensed hyperspectral unmixing. In *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2012. [Citado en pág. 38]

[91] S. Sanchez and A. Plaza. Parallel hyperspectral image compression using iterative error analysis on graphics processing units. In *IEEE Geoscience and Remote Sensing Symposium (IGARSS'12)*, Munich, Germany, 2012. [Citado en pág. 38]

[92] S. Sanchez and A. Plaza. Real-time lossy compression of hyperspectral images using iterative error analysis on GPUs. In *SPIE Photonics Europe, Real-Time Image and Video Processing Conference*, Brussels, Belgium, 2012. [Citado en pág. 38]

[93] S. Sanchez, R. Ramalho, L. Sousa, and A. Plaza. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. *Journal of Real-Time Image Processing*, 2012. [Citado en págs. 38 y 40]

[94] M. E. Schaepman, S. L. Ustin, A. Plaza, T. H. Painter, J. Verrelst, and S. Liang. Earth system science related imaging spectroscopy – an assessment. *Remote Sens. Environment*, 113:123–137, 2009. [Citado en pág. 9]

[95] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado. GPU for parallel on-board hyperspectral image processing. *International Journal High Performance Computing Applications*, 22(4):424–437, 2008. [Citado en pág. 39]

[96] G.L. Sleijpen and H.A. Van der Vorst. A jacobi-davidson iteration method for linear eigenvalue problems. *SIAM Review*, 42(2):267–293, 2000. [Citado en págs. 24 y 25]

[97] M. O. Smith, P. E. Johnson, and J. B. Adams. Quantitative determination of mineral types and abundances from reflectance spectra using principal component analysis. In *Proc. Lunar and Planetary Sci. Conf.*, volume 90, pages 797–904, 1985. [Citado en pág. 10]

[98] G.A. Swayze. *The Hydrothermal and Structural History of the Cuprite Mining District, Southwestern Nevada: An Integrated Geological and Geophysical Approach*. PhD thesis, University of Colorado, Boulder, 1997. [Citado en pág. 79]

[99] Y. Tarabalka, T. V. Haavardsholm, I. Kasen, and T. Skauli. Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and GPU processing. *Journal of Real-Time Image Processing*, 4:1–14, 2009. [Citado en pág. 39]

[100] S. Tehranian, Y. Zhao, T. Harvey, A. Swaroop, and K. Mckenzie. A robust framework for real-time distributed processing of satellite data. *Journal of Parallel and Distributed Computing*, 66:403–418, 2006. [Citado en pág. 37]

[101] U. Thomas, D. Rosenbaum, F. Kurz, S. Suri, and P. Reinartz. A new software/hardware architecture for real time image processing of wide area airborne camera images. *Journal of Real-Time Image Processing*, 5:229–244, 2009. [Citado en pág. 39]

[102] J. C. Tilton, W. T. Lawrence, and A. Plaza. Utilizing hierarchical segmentation to generate water and snow masks to facilitate monitoring change with remotely sensed image data. *GIScience and Remote Sensing*, 43(1):39–66, 2006. [Citado en pág. 35]

[103] M. Vélez. Iterative Algorithms for Unmixing of Hyperspectral Imagery. In *Proceeding of SPIE*, 2003. [Citado en págs. 31 y 32]

[104] M. E. Winter. A proof of the N-FINDR algorithm for the automated detection of endmembers in a hyperspectral image. 5425:31–41, 2004. [Citado en pág. 28]

[105] M.E. Winter. N-FINDR: an algorithm for fast autonomous spectral end-member determination in hyperspectral data. In *Proc. SPIE*, volume 3753, pages 266–270, 1999. [Citado en pág. 28]

[106] M. Zortea and A. Plaza. Spatial preprocessing for endmember extraction. *IEEE Trans. Geosci. and Remote Sens.*, 47:2679–2693, 2009. [Citado en pág. 18]

Apéndice A

Publicaciones

Los años en los que el doctorando ha desarrollado su trabajo de Tesis Doctoral han dado como fruto un conjunto de publicaciones en revistas de impacto, así como en capítulos de libro, congresos internacionales y nacionales y publicaciones online. En particular, se han conseguido 6 publicaciones en revistas indexadas en *journal citation reports* (JCR), 3 publicaciones JCR enviadas (en proceso de revisión), 2 capítulos de libro, 18 publicaciones en congresos internacionales revisados por pares, 1 publicación en congreso nacional y 2 publicaciones online directamente relacionadas con el trabajo de Tesis Doctoral. Durante este periodo el candidato ha trabajado como doctorando en el grupo de investigación "Computación Hiperespectral" (HyperComp) de la Universidad de Extremadura, en el Departamento de Tecnología de los Computadores y de las Comunicaciones de la Universidad de Extremadura. A continuación se detallan las publicaciones aportadas por el candidato y una breve descripción tanto del foro en el que fueron presentadas como de la contribución específica del candidato a cada publicación.

A.1. Revistas internacionales de impacto

1. S. Sanchez, R. Ramalho, L. Sousa and A. Plaza. "Real-Time Implementation of Remotely Sensed Hyperspectral Image Unmixing on GPUs". *Journal of Real-Time Image Processing*, aceptado para publicación, 2012 [JCR(2010)=1.020] (doi: 10.1007/s11554-012-0269-2). Este trabajo publicado en la revista *Journal of Real-Time Image Processing* que cuenta con un índice de impacto de 1.020 y que está situada en el segundo cuartil de la categoría *Imaging Science and Photographic Technology*, presenta el procesamiento en tiempo real de parte de la cadena completa en GPU. La única etapa que no se contempla en este trabajo es la de estimación del número de *endmembers*. La realización de este trabajo se llevo durante una estancia realizada por el candidato con investigadores de INESC-ID, en el Instituto Superior Tecnico, de la Universidad Técnica de Lisboa, realizada en el marco del proyecto *Open European Network on High Performance Computing in Complex Environments* (ComplexHPC). El papel del candidato en éste trabajo fue el de desarrollar versiones más eficientes de los algoritmos N-FINDR y UCLS

para su implementación en GPU y agruparlos junto con métodos de reducción dimensional para formar una cadena casi completa con la cual recoger tiempos de ejecución y avanzar un poco más hacia la consecución de tiempo real. Parte de estos resultados han sido empleados en el desarrollo de este trabajo de Tesis Doctoral.

2. S. Sanchez and A.Plaza. Fast Determination of the Number of Endmembers for Real-Time Hyperspectral Unmixing on GPUs. *Journal of Real-Time Image Processing*, aceptado para publicación, 2012 [JCR(2010)=1.020] (doi: 10.1007/s11554-012-0276-3). Este trabajo, publicado en la revista *Journal of Real-Time Image Processing* que cuenta con un índice de impacto de 1.020 y que está situada en el segundo cuartil de la categoría *Imaging Science and Photographic Technology*, presenta la implementación GPU de dos algoritmos pertenecientes a la etapa de estimación del número de *endmembers* como son VD y HySime. La principal labor del candidato en este trabajo fue la de implementar las versiones serie y paralelas de ambos métodos así como una comparativa entre ambas versiones. Tras el desarrollo de este artículo se abrió la posibilidad de incluir métodos de estimación del número de endmembers en una cadena completa para su ejecución en tiempo real. Todos los resultados obtenidos en este artículo se usan en el presente trabajo de Tesis Doctoral.

3. C. Gonzalez, S. Sanchez, A. Paz, J. Resano, D. Mozos and A. Plaza. "Use of FPGA or GPU-Based Architectures for Remotely Sensed Hyperspectral Image Processing". *Integration, the VLSI Journal*, aceptado para publicación, 2012 [JCR(2011)=0.646] (doi: 10.1016/j.vlsi.2012.04.002). Este artículo, publicado en la revista *Integration, the VLSI Journal* que cuenta con un índice de impacto de 0.646 y que se sitúa en el tercer cuartil de la categoría *Computer Science, Hardware and Architecture*, presenta una comparativa de métodos de la etapa de extracción de endmembers y de la etapa de estimación de las abundancias de dos plataformas, GPUs y FPGAs. En concreto se utilizan los algoritmos PPI e NCLS para realizar esta comparativa. La labor del candidato en el desarrollo de este artículo fue la de aportar las versiones GPUs de estos algoritmos, así como de realizar las pruebas pertinentes para la comparación de resultados. Los resultados obtenidos en la implementación de estos algoritmos se presentan en este trabajo de Tesis Doctoral.

4. A. Remon, S. Sanchez, A. Paz, E. S. Quintana-Orti y A. Plaza, "Real-time endmember extraction on multi-core processors", *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 5, pp. 924-928, Septiembre 2011 [JCR(2011)=1.485]. Este artículo, publicado en la revista *IEEE Geoscience and Remote Sensing Letters* cuyo índice de impacto es de 1.560 y que se encuentra ubicada en el primer cuartil en la categoría *Remote Sensing*, describe diferentes implementaciones multi-núcleo del algoritmo NFINDR para estracción de endmembers. El trabajo fue realizado en colaboración con investigadores de la Universidad Jaume I de Castellón durante una visita corta del candidato a dicho grupo. La contribución del candidato consistió en colaborar en el desarrollo de las implementaciones realizadas

en sistemas multi-núcleo con memoria compartida, y en la realización del estudio de validación de los algoritmos desarrollados utilizando un conjunto de imágenes AVIRIS que incluyen la imagen tomada sobre el World Trade Center la imagen de Cuprite que se emplean en el presente trabajo de Tesis Doctoral, comparando dichos resultados con los ya obtenidos en investigaciones previas desarrolladas por el candidato en colaboración con expertos del grupo HyperComp. Además el candidato contribuyó en la realización del estudio orientado a analizar la posibilidad de obtener una respuesta en tiempo real en la arquitectura paralela considerada, que reveló que la versión multi-núcleo de N-FINDR puede ejecutarse en tiempo real para procesar imágenes hiperespectrales en sistemas de este tipo.

5. A. Plaza, J. Plaza, A. Paz y S. Sanchez, "Parallel hyperspectral image and signal processing", *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 119-126, Mayo 2011 [JCR(2010)=6.000]. Este artículo, publicado en la revista *IEEE Signal Processing Magazine*, que actualmente es una de las revistas con mayor índice de impacto de IEEE con un factor de 6.000 y que ocupa la primera posición entre todas las revistas ubicadas en la categoría *Electrical and Electronic Engineering*, describe las implementaciones realizadas por el grupo HyperComp de algoritmos de análisis de imágenes hiperespectrales en diferentes tipos de arquitecturas paralelas, incluyendo un clusters de computadores, redes heterogéneas, FPGAs y GPUs. La contribución del candidato consistió en el desarrollo e implementación de técnicas de análisis hiperespectral en GPUs de NVidia, las cuales fueron muy relevantes en la comparativa multiplataforma presentada en este artículo, de muy reciente aparición y que demuestra la relevancia y aplicabilidad de la computación de altas prestaciones en aplicaciones relacionadas con la observación remota de la Tierra. La presentación de esta temática en una revista de gran relevancia a nivel internacional revela el reciente auge y potencial crecimiento futuro de esta línea de investigación, que ofrece importantes perspectivas de futuro en cuanto a la explotación eficiente de la cantidad ingente de datos que hoy en día proporcionan los satélites hiperespectrales de observación remota de la tierra. Estos resultados han sido presentados de forma parcial en el presente trabajo de Tesis Doctoral.

6. S. Sanchez, A. Paz, G. Martín and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units", *Concurrency and Computation: Practice and Experience*, 2011 [JCR(2011)=0.952]. Este trabajo, publicado en la revista *Concurrency and Computation: Practice & Experience* cuyo índice de impacto es 0.636 y que se encuentra ubicada en el tercer cuartil en la categoría *Computer Science: Theory and Methods*, describe una comparativa de algoritmos de análisis hiperespectral en diferentes tipos de tarjetas gráficas programables de NVidia. La aplicación descrita se centra en el desmezclado (interpretación a nivel sub-píxel) de datos hiperespectrales de la superficie terrestre, obtenidos de forma remota. El caso de estudio utilizado en la comparativa se centra en el análisis de la imagen AVIRIS sobre el World Trade Center que se ha empleado como caso de estudio en el presente trabajo de Tesis Doctoral. La contribución

del candidato a este trabajo consistió en la aportación de los algoritmos UCLS e NCLS (en su versión secuencial y GPU) a la comparativa. Dicho algoritmos, implementado en diferentes GPUs de NVidia, se utilizaron el la etapa de estimación de las abundancias, posterior a la de extracción de endmembers. Estos resultados han sido presentados de forma parcial en el presente trabajo de Tesis Doctoral.

A.2. Revistas enviadas (en proceso de revisión)

1. Pat Cappelaere, Sergio Sánchez, Sergio Bernabe, Antonio Scuri, Dan Mandl and Antonio Plaza. "Cloud Implementation of a Full Hyperspectral Unmixing Chain within the NASA Web Coverage Processing Service for EO-1" *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, en proceso de revisión, 2012 [JCR(2011)=1.489]. Este trabajo enviado a la revista *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, que cuenta con un índice de impacto de 1.489 y que está situada en el primer cuartil en la categoría de *Imaging Science & Photographic Technology*, muestra un estudio de implementación de la cadena de procesamiento hiperespectral completa aplicada a imágenes de gran tamaño. Dicha implementación está concebida para ser ejecutada en entorno cloud como parte de los servicios *SensorWeb* de NASA. El trabajo del candidato consistió en la aportación de varios algoritmos de la cadena en sus versiones serie para su posterior adaptación.

2. Sergio Bernabé, Sergio Sánchez, Antonio Plaza, Sebastián López, Jón Atli Benediktsson y Roberto Sarmiento. "Hyperspectral Unmixing on GPUs and Multi-Core Processors: A Comparison" *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, en proceso de revisión, 2012 [JCR(2011)=1.489]. Este trabajo enviado a la revista *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, que cuenta con un índice de impacto de 1.489 y que está situada en el primer cuartil en la categoría de *Imaging Science & Photographic Technology*, muestra una comparativa de ejecución de una cadena de desmezclado completa en GPU y en multi-core. Dicha cadena esta formada por los métodos VD, OSP y UCLS. El trabajo del candidato consistió en la aportación de los métodos VD y UCLS en sus versiones GPU para su inclusión en la cadena.

3. Alfredo Remón, Sergio Sánchez, Sergio Bernabé, Enrique S. Quintana-Ortí, and Antonio Plaza. "Performance versus Energy Consumption of Hyperspectral Unmixing Algorithms on Multi-Core Platforms" *EURASIP Journal on Advances in Signal Processing*, en proceso de revisión, 2012 [JCR(2011)=0.811]. Este trabajo enviado a la revista *EURASIP Journal on Advances in Signal Processing*, que cuenta con un índice de impacto de 0.811 y que está situada en el tercer cuartil en la categoría de *Engineering, Electrical & Electronic*, muestra un estudio que relaciona el rendimiento obtenido al ejecutar diferentes cadenas de procesamiento

hiperespectral con el consumo eléctrico en multi-cores. El trabajo del candidato consistió en la aportación de varios algoritmos de la cadena en sus versiones serie y GPUs para la realización del estudio.

A.3. Capítulos de libro

1. A. Plaza, J. Plaza, G. Martin and S. Sanchez. "Hyperspectral Data Processing Algorithms, in: Hyperspectral Remote Sensing of Vegetation". Edited by P. S. Thenkabail, J. G. Lyon and A. Huete, Taylor and Francis, 2011, ISBN: 978-1-4398-4537-0, pp. 121-137. Este capítulo realiza un repaso de las principales técnicas de análisis hiperespectral teniendo como objetivo principal el estudio de la vegetación. La labor del candidato en este capítulo fue de aportar diversos resultados conseguidos con algunos métodos implementados en GPU.

2. A. Plaza, G. Martin, J. Plaza, M. Zortea and S. Sanchez. "Recent Developments in Spectral Unmixing and Endmember Extraction, in: Optical Remote Sensing - Advances in Signal Processing and Exploitation Techniques". Edited by S. Prasad, L. Bruce and J. Chanussot, Springer, 2011, ISBN: 978-3-642-1241-6, pp. 235-268. Este capítulo realiza un repaso de las principales técnicas de análisis hiperespectral. La labor del candidato en este capítulo fue de aportar diversos resultados conseguidos con algunos métodos implementados en GPU.

A.4. Congresos internacionales revisados por pares

1. S. Sanchez, A. Paz y A. Plaza, "Real-Time Spectral Unmixing Using Iterative Error Analysis on Commodity Graphics Processing Units", *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'11)*, Vancouver, Canadá, 2011. Este trabajo, presentado en forma de comunicación oral en el congreso *IEEE IGARSS* en Vancouver, Canadá (el cual congrega a miles de investigadores a nivel mundial en el área de observación remota de la tierra, constituyendo el principal foro a nivel internacional en dicha disciplina) muestra un estudio que analiza las posibilidades de utilizar algoritmos de extracción de endmembers y estimación de abundancias (IEA) en comparación con otras técnicas para desmezclado en tiempo real de imágenes hiperespectrales de la superficie terrestre. La contribución del candidato a este trabajo consistió en aportar resultados que fueron utilizados en la validación experimental de las técnicas propuestas en el trabajo. Estos resultados han sido presentados de forma parcial en el presente trabajo de Tesis Doctoral.

2. A. Plaza, S. Sanchez, A. Paz y J. Plaza, "GPUs versus FPGAs for Onboard Compression of Hyperspectral Data", *2nd International ESA Workshop on On-Board Payload Data Compression (OBPDC'10)*, CNES, Toulouse, Francia, 2010. Este trabajo, presentado en forma de comunicación oral en el congreso *OBPDC de ESA* celebrado en Toulouse, Francia (el cual congrega a numerosos expertos en el área de compresión de imágenes obtenidas vía satélite) analiza las posibilidades de

las arquitecturas GPU (frente a FPGAs) en compresión de datos hiperespectrales obtenidos por satélites de observación remota de la tierra, evaluando la posibilidad de realizar dicha compresión a bordo del sensor. La contribución del candidato consistió en aportar las implementaciones de algoritmos de compresión en GPUs basadas en algoritmos de extracción de endmembers. Estos resultados han sido presentados de forma parcial en el presente trabajo de Tesis Doctoral.

3. S. Sanchez, G. Martín, A. Paz, A. Plaza y J. Plaza, "Near Real-Time Endmember Extraction from Remotely Sensed Hyperspectral Data Using NVidia GPUs", *SPIE Real-Time Image and Video Processing 2010*, Bruselas, Bélgica, 2010. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Real-Time Image and Video Processing* en Bruselas (el cual congregó a cerca de 500 investigadores a nivel mundial en el área de procesamiento de imágenes) muestra resultados referentes a la posibilidad de procesar en tiempo real imágenes hiperespectrales de la superficie terrestre, incluyendo el algoritmo NFINDR descrito en el presente trabajo. Por tanto, el trabajo se encuentra directamente relacionado con los resultados descritos en el presente trabajo de Tesis Doctoral.

4. A. Plaza, J. Plaza, S. Sanchez y A. Paz, "Lossy Hyperspectral Image Compression Tuned for Spectral Mixture Analysis Applications on NVidia Graphics Processing Units", *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2009. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Optics and Photonics* en San Diego (el cual congrega a más de 2000 investigadores a nivel mundial en el área de procesamiento de imágenes de satélite, constituyendo uno de los principales foros a nivel internacional en esta disciplina) muestra algunos de los primeros estudios realizados en el área de GPUs, en el que se utilizaron los algoritmos PPI (*Pixel Purity Index*), AMEE (*Automatic Morphological Endmember Extraction*) y FCLSU (*Fully Constrained Linear Spectral Unmixing*) logrando obtener *speedups* del orden de 26 utilizando técnicas de compresión. La contribución del candidato consistió en colaborar en las implementaciones paralelas de los algoritmos considerados en diferentes arquitecturas GPUs. Estos resultados no han sido presentados en la presente memoria.

5. A. Plaza, J. Plaza, S. Sanchez y A. Paz, "Optimizing a Hyperspectral Image Processing Chain Using Heterogeneous and GPU-Based Parallel Computing Architectures", *9th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE'09)*, Gijón, 2009. Este trabajo, presentado en forma de comunicación oral en el CMMSE'09 (congreso orientado a la presentación de resultados de implementación de técnicas computacionales en diversas aplicaciones y que acogió una reunión de la red nacional CAPAP-H de supercomputación en entornos heterogéneos, permitiendo la presentación de los resultados de investigación obtenidos a dicha comunidad) muestra una comparativa de resultados obtenidos en redes heterogéneas de computadores y GPUs en el procesamiento de imágenes hiperespectrales. La contribución

del candidato consistió en colaborar en las implementaciones paralelas de los algoritmos considerados en diferentes arquitecturas GPUs. Estos resultados han sido presentados de forma parcial en el presente trabajo de Tesis Doctoral.

6. J. M. Rodriguez-Alves, J. Nascimento, A. Plaza, S. Sanchez, J. Bioucas-Dias and V. Silva. "Vertex Component Analysis GPU-Based Implementation for Hyperspectral Unmixing". *IEEE GRSS Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS'12)*, Shanghai, China, 2012. Este trabajo, presentado en forma de comunicación oral en el congreso *WHISPERS* en Shanghai, China (el cual congrega a cerca de 200 investigadores a nivel mundial) presenta un estudio de implementación del algoritmo Vertex Component Analysis en GPU. Este algoritmo, descrito en los antecedentes del presente trabajo de Tesis Doctoral, se divide en dos etapas : una primera etapa de estimación del número de endmembers y reducción de ruido y una segunda de extracción. La contribución del candidato fue la de aportar y adaptar el método HYSIME como primera etapa del método VCA.

7. S. Sanchez and A. Plaza. "Parallel Hyperspectral Image Compression Using Iterative Error Analysis on Graphics Processing Units". *IEEE Geoscience and Remote Sensing Symposium (IGARSS'12)*, Munich, Alemania, 2012. Este trabajo, presentado en forma de comunicación oral en el congreso *IEEE IGARSS* en Munich, Alemania (el cual congrega a miles de investigadores a nivel mundial en el área de observación remota de la tierra, constituyendo el principal foro a nivel internacional en dicha disciplina) presenta un trabajo de compresión de imágenes hiperespectrales utilizando el método IEA en GPU. El trabajo del candidato fue el de implementar esta versión del método en la GPU y realizar un estudio de los resultados obtenidos con imágenes reales. Los resultados de compresión no se muestran en el Trabajo de Tesis pero si otros resultados obtenidos con este método.

8. P. Cappelaere, S. Sanchez, S. Bernabe and A. Plaza. "Multi-Core Implementation of a Full Hyperspectral Unmixing Chain within the NASA Web Coverage Processing Service for HyspIRI". *IEEE Geoscience and Remote Sensing Symposium (IGARSS'12)*, Munich, Alemania, 2012. Este trabajo, presentado en forma de comunicación oral en el congreso *IEEE IGARSS* en Munich, Alemania (el cual congrega a miles de investigadores a nivel mundial en el área de observación remota de la tierra, constituyendo el principal foro a nivel internacional en dicha disciplina) muestra un estudio de implementación multi-core de la cadena de procesamiento hiperespectral casi completa aplicada a imágenes de gran tamaño tomadas desde satélite para su posible utilización en el satélite HyspIRI. El trabajo del candidato consistió en la aportación de varios algoritmos de la cadena en sus versiones serie para su posterior adaptación.

9. S. Sanchez and A. Plaza. "Real-Time Lossy Compression of Hyperspectral Images Using Iterative Error Analysis on GPUs". *SPIE Photonics Europe, Real-Time Image and Video Processing Conference*, Bruselas, Bélgica, 2012. Este trabajo,

presentado en forma de comunicación oral en el congreso *SPIE Photonics Europe, Real-Time Image and Video Processing Conference* (el cual congregó un gran número investigadores a nivel mundial en el área de procesamiento de imágenes) muestra una implementación mejorada en GPU del algoritmo IEA para compresión de imágenes hiperespectrales en tiempo real. El trabajo del candidato fue el de desarrollar la versión optimizada de este método y realizar las pruebas pertinentes para obtener resultados. Resultados de este método se muestran en el presente Trabajo de Tesis.

10. S. Sanchez and A. Plaza. "GPU Implementation of the Image Space Reconstruction Algorithm for Remotely Sensed Hyperspectral Unmixing". *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2012. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference* (el cual congregó a un gran número investigadores a nivel mundial en el área de procesamiento de imágenes) presenta la implementación del método NCLS para estimación de abundancias en GPU. El trabajo del candidato consistió en el desarrollo de la versión paralela del método así como de la realización de las pruebas para obtener resultados. Los resultados obtenidos con este método se presentan en este Trabajo de Tesis.

11. S. Sanchez and A. Plaza. "A Comparative Analysis of GPU Implementations of Spectral Unmixing Algorithms".*SPIE Remote Sensing Europe, High Performance Computing in Remote Sensing Conference*, Praga, República Checa, 2011. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Remote Sensing Europe, High Performance Computing in Remote Sensing Conference* (el cual congregó a un gran número investigadores a nivel mundial en el área de teledetección) presenta una comparativa de varios algoritmos de la etapa de extracción de endmembers y de la etapa de estimación de abundancias implementados en GPU. El trabajo del candidato consistió en el desarrollo de las versiones paralelas de varios de estos métodos como son NFINDR, IEA, UCLS e NCLS. Resultados parciales de este trabajo se exponen en el presente Trabajo de Tesis Doctoral.

12. S. Sanchez and A. Plaza. "Real-Time Implementation of a Full Hyperspectral Unmixing Chain on Graphics Processing Units". *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2011. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference* (el cual congregó a un gran número investigadores a nivel mundial en el área de teledetección) presenta la implementación de la cadena completa de procesamiento hiperespectral en tiempo real utilizando la GPU. El trabajo del candidato consistió en el desarrollo de las versiones paralelas de los distintos métodos así como la integración de estos en la cadena de procesamiento.

Los resultados obtenidos en este trabajo se muestran de forma parcial en el presente Trabajo de Tesis.

13. S. Sanchez and A. Plaza. "GPU Implementation of the Pixel Purity Index Algorithm for Hyperspectral Image Analysis". *IEEE International Conference on Cluster Computing (Cluster'10)*, Heraklion, Greece, 2010. Este trabajo, presentado en forma de comunicación oral en el congreso *IEEE International Conference on Cluster Computing* presenta una implementación GPU del algoritmo PPI consiguiendo speedups superiores a 100 unidades. La contribución del candidato en el desarrollo de este trabajo consistió en la implementación optimizada en GPU del método PPI para extracción de endmembers. Los resultados obtenidos en este trabajo se muestran de forma parcial en el presente Trabajo de Tesis.

14. S. Sanchez, G. Martin and A. Plaza. "Parallel Implementation of the N-FINDR Endmember Extraction Algorithm on Commodity Graphics Processing Units". *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'10)*, Hawaii, USA, 2010. Este trabajo, presentado en forma de póster en el congreso *IEEE IGARSS* en Hawaii, USA (el cual congrega a miles de investigadores a nivel mundial en el área de observación remota de la tierra, constituyendo el principal foro a nivel internacional en dicha disciplina) presenta la primera implementación en GPU del método NFINDR para extracción de endmembers. El trabajo del candidato consistió en el desarrollo de la versión paralela de este método y la toma de resultados con una imagen real. Los resultados obtenidos en este trabajo se muestran de forma parcial en el presente Trabajo de Tesis.

15. S. Sanchez, G. Martin, A. Plaza and C.-I Chang. "GPU Implementation of Fully Constrained Linear Spectral Unmixing for Remotely Sensed Hyperspectral Data Exploitation". *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, San Diego, CA, 2010. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference* (el cual congregó un gran número investigadores a nivel mundial en el área de teledetección) muestra la implementación en GPU del método FCLSU para estimación de abundancias con restricciones de no negatividad y suma unitaria. El papel del candidato en el desarrollo de este trabajo consistió en el desarrollo de la versión paralela para su implementación en GPU.

16. A. Plaza, J. Plaza, I. Dopido, G. Martin, M. D. Iordache and S. Sanchez. "New Hyperspectral Unmixing Techniques in the Framework of the Earth Observation Optical Data Calibration and Information Extraction (EODIX) Project". *3rd International Symposium on Recent Advances in Quantitative Remote Sensing (RAQRS'10)*, Valencia, España, 2010. Este trabajo muestra un estudio de las principales técnicas que se utilizan en el análisis de imágenes hiperespectrales. La principal contribución del candidato fue la de aportar ciertos resultados obtenidos por versiones paralelas en GPU de algunos métodos.

17. S. Sanchez, G. Martin, A. Plaza and J. Plaza. "Near Real-Time Endmember Extraction from Remotely Sensed Hyperspectral Data Using NVidia GPUs". *SPIE Photonics Europe, Real-Time Image and Video Processing Conference*, Bruselas, Bélgica, 2010. Este trabajo, presentado en forma de comunicación oral en el congreso *SPIE Photonics Europe, Real-Time Image and Video Processing Conference* (el cual congregó un gran número investigadores a nivel mundial en el área de teledetección) muestra un estudio detallado sobre la implementación del algoritmo NFINDR en GPU para extracción de endmembers. La labor principal del candidato consistió en desarrollar y optimizar una versión paralela del método así como de realizar experimentos para recoger resultados. Algunos de los resultados de este trabajo se muestran en el presente Documento de Tesis.

18. A. Plaza, J. Plaza, H. Vegas and S. Sanchez. "Parallel Implementation of Endmember Extraction Algorithms Using NVidia Graphical Processing Units". *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'09)*, Ciudad del Cabo, Sudáfrica, 2009. Este trabajo presenta implenemtaciones paralelas en GPU de tres métodos de extracción de endmembers: PPI, KPPI (una versión de PPI que trabaja sobre ventanas) y AMME. La principal función del candidato en el desarrollo de este trabajo fue la de desarrollar e implementar las versiones paralelas de estos métodos. Algunos de los resultados de este trabajo se muestran en el presente Documento de Tesis.

A.5. Congresos nacionales

1. S. Sanchez y A. Plaza. "Implementación paralela del algoritmo Pixel Purity Index para análisis hiperespectral en GPUs". *Jornadas de Paralelismo, Congreso Español de Informática (CEDI'2010)*, Valencia, España, 2010. Este trabajo, presentado en forma de comunicación oral, muestra la implementación de una versión optimizada del algoritmo PPI en GPU que alcanza speedups muy superiores a 100 unidades. La labor principal del candidato fue la del desarrollo de la versión optimizada de la versión paralela del método para su implementación en GPU. Resultados parciales de este trabajo se muestran en el presente Trabajo de Tesis.

A.6. Publicaciones online

1. A. Plaza, J. Plaza and S. Sanchez. "Hyperspectral Image Compression on NVidia GPUs". *NVidia CUDA Zone*[1], 2009. Este artículo presenta la implementación de los métodos PPI y AMEE en GPU y su utilización para comprimir imágenes hiperespectrales.

2. A. Plaza, S. Sanchez and J. Plaza. "Hyperspectral Unmixing on NVidia GPUs". *NVidia CUDA Zone*, 2009. Este artículo muestra la implementación de algoritmos de extracción de endmembers en GPU. Concretamente se muestran resultados

[1] http://www.nvidia.com/object/cuda_showcase_html.html

obtenidos con PPI y Kernel PPI (una versión de PPI basada en ventanas espaciales).