ACELERANDO LA FUSIÓN DE IMÁGENES MEDIANTE COMPUTACIÓN HETEROGÉNEA

Rubén Javier Medina Daza
Andrés Ovidio Restrepo Rodríguez
Nelson Enrique Vera Parra

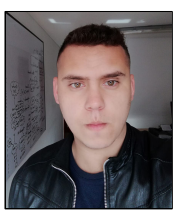
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Doctorado en Ingeniería
UNIVERSIDAD DISTRITAL "FRANCISCO JOSÉ DE CALDAS"

Rubén Javier Medina Daza

Doctor en Informática con énfasis en Sistemas de Información Geográfica, Magíster en Teleinformática, Licenciado en Matemáticas de la Universidad Distrital Francisco José de Caldas. Profesor Titular de Ingeniería Catastral y Geodesia, de la Maestría en Ciencias de la Información y las Comunicaciones y del Doctorado de Ingeniería.

Andrés Ovidio Restrepo Rodríguez



Ingeniero de Sistemas y actual estudiante de la Maestría en Ciencias de la Información y las Comunicaciones en la Universidad Distrital Francisco José de Caldas. Campos de investigación: Inteligencia Artificial, Learning Analytics, Entornos Inmersivos y Computación Heterogénea.

Nelson Enrique Vera Parra

Ingeniero Electrónico de la Universidad Surcolombiana, Magíster en Ciencias de la Información y las Comunicaciones y Doctor en Ingeniería de la Universidad Distrital Francisco José de Caldas. Profesor Titular de la misma Universidad. Investigador en HPC, Ciencia deDatos y Bioinformática.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Doctorado en Ingeniería
UNIVERSIDAD DISTRITAL "FRANCISCO JOSÉ DE CALDAS"

ACELERANDO LA FUSIÓN DE IMÁGENES MEDIANTE COMPUTACIÓN HETEROGÉNEA

Rubén Javier Medina Daza
Andrés Ovidio Restrepo Rodríguez
Nelson Enrique Vera Parra

Medina Daza, Rubén Javier
Acelerando la fusión de imágenes mediante computación heterogénea / Rubén Javier Medina Daza, Andrés Ovidio Restrepo Rodríguez, Nelson Enrique Vera Parra. -- 1a ed. -- Bogotá : Universidad Distrital Francisco José de Caldas, 2021.
119 p. ; 24 cm -- (Doctorado en Ingeniería)

Incluye reseña de los autores en la pasta -- Contiene referencias bibliográficas.

ISBN 978-958-49-4957-8 (Impreso) - 978-958-49-4958-5 (Digital)

1. Procesamiento de imágenes - Técnicas digitales
2. Computación heterogénea I. Restrepo Rodríguez, Andrés Ovidio II. Vera Parra, Nelson Enrique III. Título IV. Serie

CDD: 006.6 ed. 23 CO-BoBN– a1088110

ISBN Impreso: 978-958-49-4957-8
ISBN Digital: 978-958-49-4958-5

Primera edición: Bogotá, diciembre de 2021

Corrección de estilo, diseño gráfico y producción editorial:
IngeEdit Editores – Sandra Patricia Rodríguez Lamus - ingeeditorial@gmail.com

Impresión:
IngeEdit Editores – Sandra Patricia Rodríguez Lamus

Doctorado en Ingeniería
Carrera 7 No. 40B – 53 Bogotá
Correo electrónico: investigacion.doctoradoing@udistrital.edu.co

Hecho el depósito legal.

Impreso y hecho en Colombia
Printed and made in Colombia

Tabla de contenido

Índice de figuras

Índice de tablas

Prefacio

Durante los últimos años el procesamiento de imágenes ha tomado importancia en el campo científico, su principal objetivo es maximizar el uso de la información de una imagen para un contexto en particular. De acuerdo con lo anterior, uno de los principales temas en este campo es la fusión de imágenes, la cual hace referencia a la combinación de información relevante obtenida a partir de dos imágenes, esto con el fin de producir una imagen que contenga una calidad superior a las originales. Dentro de este campo, se pueden realizar fusiones de imágenes satelitales, donde se debe proporcionar una imagen pancromática para realizar una inyección de riqueza espacial en la información espectral asociada a la imagen multiespectral.

La fusión de imágenes al igual que la gran mayoría de operaciones con imágenes presentan una exigencia computacional dependiente del tamaño de la imagen, debido a la granularidad pixel a pixel presente en estas operaciones. Esta granularidad que aparentemente es un inconveniente termina convirtiéndose en una ventaja porque habilita la posibilidad de paralelización masiva sobre arquitecturas computacionales que ofrecen un alto número de núcleos de procesamiento, tales como las GPU (*Graphics Processing Unit*).

Este libro presenta una forma eficiente de acelerar la implementación de los principales métodos de fusión de imágenes mediante procesamiento heterogéneo, segmentando y distribuyendo tareas convenientemente entre cómputo secuencial sobre CPU y cómputo paralelo masivo sobre GPU.

Capítulo 1

Introducción, problema de investigación y objetivos

La fusión de imágenes de teledetección de muy alta resolución o *pan-sharpening,* consiste en añadir o inyectar la información espacial que contiene la imagen pancromática a las bandas espectrales de la imagen Multiespectral, preservando las características espectrales de esta. Sin embargo, en este proceso se introducen distorsiones, además de las inherentes al registro de los datos Multiespectral (MS) y Pancromática (PAN). En este contexto, para intentar evitar este inconveniente a lo largo de la última década se han desarrollado multitud de algoritmos de *pansharpening* (Vivone et al., 2015). Sin embargo, no existe en la actualidad ninguno que se postule como la solución óptima para la fusión de imágenes.

1.1 Introducción a la fusión de imágenes

El concepto de fusión de datos se remonta a los años 1950 y 1960 (Wang et al., 2005) cuando se inició la búsqueda de métodos prácticos que permitieran mezclar imágenes procedentes de diversos sensores, con el fin de proporcionar una imagen que facilitara una mejor identificación de objetos naturales y artificiales, de aquí, que actualmente se disponga de un gran número de metodologías y algoritmos para la fusión de imágenes

ópticas, siendo las técnicas basadas en análisis multirresolución (MRA) las más utilizadas.

Algunas técnicas son muy sencillas desde un punto de vista conceptual, como la transformada de Brovey, Multiplicación, el Análisis de Componentes Principales o la transformada HSI, sin embargo, como se demuestra en numerosos trabajos, estas metodologías proporcionan imágenes fusionadas con considerables distorsiones respecto al color de las imágenes multiespectrales originales. Para minimizar estas distorsiones se han presentado un gran número de métodos basados principalmente en técnicas de análisis multirresolución, que proporcionan una mínima distorsión espectral de las imágenes fusionadas con resultados superiores a los métodos citadas previamente.

> "La fusión de imágenes es una respuesta a la frecuente necesidad de tener en una sola imagen datos de alta resolución espectral y espacial a partir de imágenes multiespectrales y pancromáticas de diferente resolución espacial y diferentes sensores remotos. La fusión permite obtener información detallada sobre el medio ambiente urbano y rural, útil para una aplicación específica en estudio" (Wald, 1999; Alparone et al., 2007).

Corresponde a técnicas que permiten mezclar, a nivel de píxel, las virtudes de diversas imágenes mejorando la capacidad de discriminación digital de los fenómenos espaciales, permitiendo al usuario cambiar la escala del análisis espacial con la misma imagen. En pocas palabras, lo que se pretende es mejorar la calidad de los datos, lo que además sirve para mejorar la fiabilidad de las estimaciones de una determinada variable (Chuvieco, 2002).

La fusión de imágenes genera imágenes sintéticas, producto de la combinación de uno o más sensores, por ejemplo, imágenes de radar con ópticas, térmicas con ópticas, etc. Una de las aplicaciones más recurrentes es la de mejorar la resolución espacial de una imagen multiespectral, usando una imagen de resolución espectral pobre, pero de mayor resolución espacial. Hace unos años lo más natural era fusionar bandas de

Landsat 5 TM, de 30 m. de resolución espacial, con la banda pancromática de Spot, con píxel de 10 m (Chuvieco, 2008). El resultado poseía lo mejor de los dos mundos, la riqueza espectralidad Landsat junto a la riqueza espacial de la Spot. Este procedimiento también puede hacerse entre fotografías aéreas e imágenes de cualquier satélite.

Hoy lo más común es utilizar la banda pancromática, propia del mismo satélite y fusionarla con sus bandas espectrales. La ventaja de esto, es que ambas imágenes son de la misma fecha y tienen el mismo ángulo de inclinación de la toma, por lo tanto, tendrán las mismas características de sombras e igualdad de condiciones atmosféricas.

Generalmente la relación entre el tamaño del píxel de las bandas espectrales y la banda pancromática es de 1 a 2, es decir, si una banda espectral posee resolución espacial de 30 m. por píxel, la banda pancromática poseería una resolución de 15 m. (ver Figura 1).

Para realizar la fusión de imágenes se debe cumplir:

1. La georreferenciación o corregistración de las imágenes involucradas, debe ser la misma. Es decir, la ubicación de los objetos en el espacio debe coincidir.
2. La extensión de las imágenes debe ser la misma, en otras palabras, la cantidad de líneas y columnas debe ser igual.
3. El tamaño del píxel también debe ser igual en todas las bandas involucradas. Es decir, el tamaño del píxel de la imagen multiespectral debe coincidir con el tamaño de la imagen pancromática.

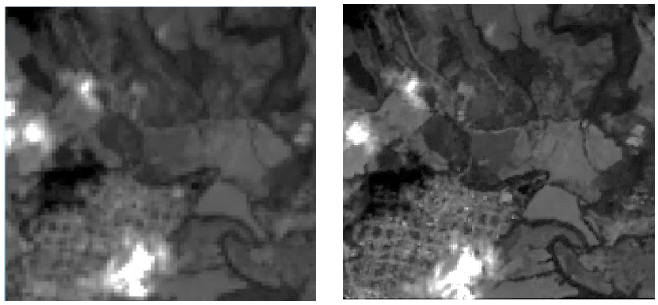


Figura 1. Comparación de la banda 3 y la imagen pancromática Landsat 8 OLI TIRS.

Para cumplir estos requisitos, las bandas espectrales deben ser procesadas. Lo primero es igualar las matrices en cuanto a tamaño del píxel y cantidad de las bandas espectrales a la imagen pancromática. El procedimiento se llama remuestreo e implica recalcular la matriz raster de las bandas para que esta sea igual a la matriz de la imagen pancromática.

La Figura 2, muestra lo que ocurre con los valores de los píxeles durante el proceso. En esta, la relación es 1 a 2, donde 1 píxel espectral se multiplica por 4, pero los valores asignados no cambian, se repiten, ya que no se está mejorando la imagen o no hay nueva información espectral que representar. Se debe mencionar que el peso de la nueva banda en el disco duro será de 4 veces mayor que la original. Para el caso de imágenes con relación 1 a 4 el peso aumenta 16 veces.

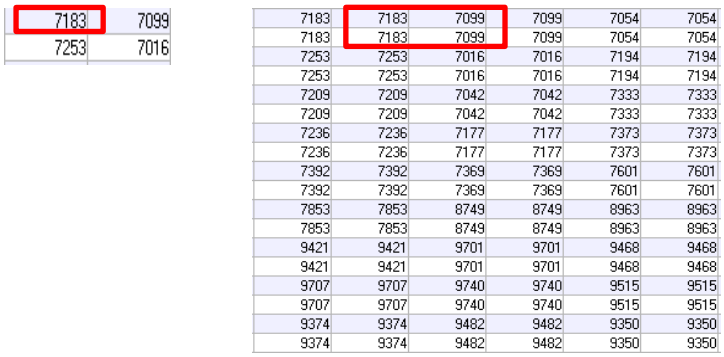


Figura 2. Valores de píxel durante el proceso de remuestreo.

La calidad de las imágenes a fusionar es muy relevante cuando éstas provienen de distintos sensores. Wald et al. (1997) recomiendan que se cumplan ciertas condiciones medibles matemáticamente, a través de índices estadísticos, pueden ser los ERGAS espectral (*Erreur Relative Globale Adimensionelle de Synthèse*), de Wald (2000) o ERGAS Espacial, para evaluar la calidad espacial de la fusión, de Lillo-Saavedra y Gonzalo (2005). También se puede utilizar la diferencia de los valores medios, la diferencia de varianzas, la desviación estándar o el error medio cuadrático (RMS),

correlaciones espaciales, entre otros, y que van más allá de mera inspección visual. Estas condiciones se pueden resumir en:

1. Cualquier imagen fusionada una vez degradada de su resolución original, debe ser lo más similar posible a la imagen original (antes de la fusión).
2. Cualquier imagen fusionada debe ser lo más similar posible a la imagen original del sensor que aporta la imagen de mayor resolución espacial.

Como ejemplo de fusión, con una imagen Ikonos de Bogotá, RGB verdadero color, 4 metros de resolución espacial, y una imagen pancromática 1 m de resolución, (ver Figura 3).

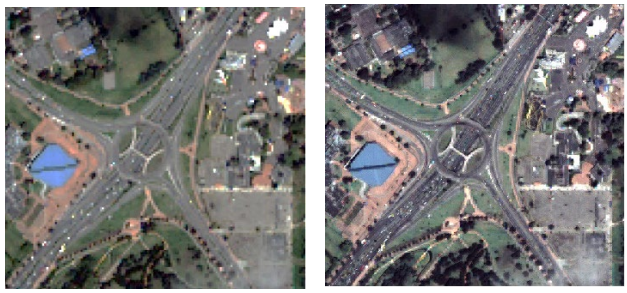


Figura 3. Imagen original y la imagen fusionada Ikonos.

1.2 Introducción a la computación heterogénea

A través de la historia de la computación, el paradigma de desarrollo y evolución de los procesadores se había enfocado en el aumento de su capacidad de cómputo mediante el incremento de la frecuencia de reloj, con el objeto de ejecutar una mayor cantidad de instrucciones en el menor tiempo posible. Sin embargo, desde 2003 debido al consumo de energía y los problemas de disipación de calor que limitan la construcción de procesadores que aumenten la frecuencia de reloj y el nivel de actividades productivas que puede ejecutarse en cada periodo de reloj en un único procesador, se cambió el enfoque integrando múltiples unidades de procesamiento en un mismo chip para aumentar el poder de procesamiento (De Antonio y Marina, 2005). Gracias al desarrollo de estos procesadores se abrió la posibilidad de resolver problemas

computacionales que antes hubieran sido imposibles (Alba, 2005). Estos problemas deben ser solucionados de una manera distinta a como se resuelven linealmente, tomando un problema cualquiera se divide en un conjunto de subproblemas para resolver éstos simultáneamente sobre diferentes unidades de procesamiento.

De acuerdo a lo expuesto en el párrafo anterior, en la actualidad el desarrollo de sistemas de procesamiento se ha enfocado en producir dispositivos con la capacidad de ejecución simultánea de dos manera diferentes: La primera opción es el diseño de CPUs multi-core, optimizadas para reducir el tiempo de ejecución de procesos secuenciales (*lactency cores*); la segunda opción, es el diseño de sistemas de procesamiento many-thread, como por ejemplo las GPUs (*Graphics Processing Unit* / Unidades de Procesamiento Gráfico) optimizadas para mejorar el desempeño (menos tiempo y menos consumo de energía eléctrica) en la ejecución de procesos paralelizables (*throughput cores*). Debido a que la mayoría de problemas computacionalmente intensivos poseen procesos tanto secuenciales como paralelizables, en los últimos años se ha iniciado el proceso de integración de los sistemas multi-core y los sistemas many-thread en plataformas computacionales denominadas heterogéneas (Kirk & Wen-mei, 2012).

Una plataforma de computación heterogénea se define como un sistema conformada por lo menos de dos tipos diferentes de procesadores, normalmente, con el objeto de incorporar capacidades de procesado especializadas para realizar tareas particulares (Shan, 2006). Un sistema heterogéneo se conforma habitualmente por una o más CPU que cumplen la función de unidad de procesamiento principal (llamado generalmente Host) y uno o más dispositivos de procesamiento diferentes, como por ejemplo GPUs (*Graphics Processing Units*), DSPs (*Digital Signal Processors*), FPGAs (*Field Programmable Gate Arrays*), que cumplen la función de aceleradores (ver Figura 4). También se puede encontrar la integración de dos o más tipos de procesadores en un solo chip, por ejemplo, un APU

(*accelerated processing unit*) es un microprocesador que integra una CPU multinúcleo y una GPU mediante un bus de alta velocidad.

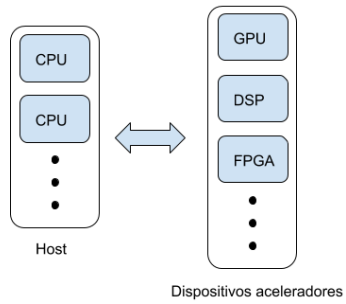


Figura 4. Plataforma heterogénea típica.

Así como la heterogeneidad entre dispositivos de procesamiento representa una ventaja al ofrecer capacidades de procesado especializadas para realizar tareas particulares, también representa una gran desventaja desde el punto de vista del desarrollo. La heterogeneidad entre dispositivos de procesamiento se centra principalmente en la diferencia entre arquitecturas de conjuntos de instrucciones ISA (*Instruction Set Architecture*), por tal motivo cada uno de los tipos de dispositivos podrá contar con modelos, paradigmas y herramientas de programación totalmente diferentes, lo que conlleva a procesos de desarrollo separados con tortuosas integraciones. Los limitantes en la integración de procesos de desarrollo para los diferentes tipos de dispositivos que pueden estar involucrados en un sistema heterogéneo, se han comenzado a mitigar con la creación de estándares de plataformas y modelos de programación tales como CUDA y OpenCL.

1.2.1 CUDA

CUDA es una plataforma de computación paralela de propósito general y un modelo de programación. Su principal objetivo es habilitar el uso de GPUs NVIDIA para soluciona problemas computacionales complejos de una forma más eficiente que como se hace sobre una CPU (CUDA C

Programming Guide, 2017). CUDA incluye un entorno de software que permite a los desarrolladores usar C como un lenguaje de alto nivel. También soporta otros lenguajes de programación y APIs.

1.2.1.1 Modelo de programación de CUDA

El modelo de programación de CUDA se soporta sobre 3 abstracciones claves: jerarquía de grupos de hilos, memorias compartidas y barreras de sincronización, que se presentan al programador como un conjunto mínimo de extensiones de lenguaje. Estas abstracciones guían al programador a dividir el problema en subproblemas gruesos que pueden resolverse de forma independiente en paralelo mediante bloques de hilos, y cada subproblema en piezas más finas que se pueden resolver cooperativamente en paralelo por todos los hilos dentro del bloque.

El modelo es escalable de forma automática, en el sentido que los bloques de hilos no van sujetos al número de multiprocesadores de la GPU. La ejecución de los bloques se adapta al número de multiprocesadores disponibles (ver Figura 5).

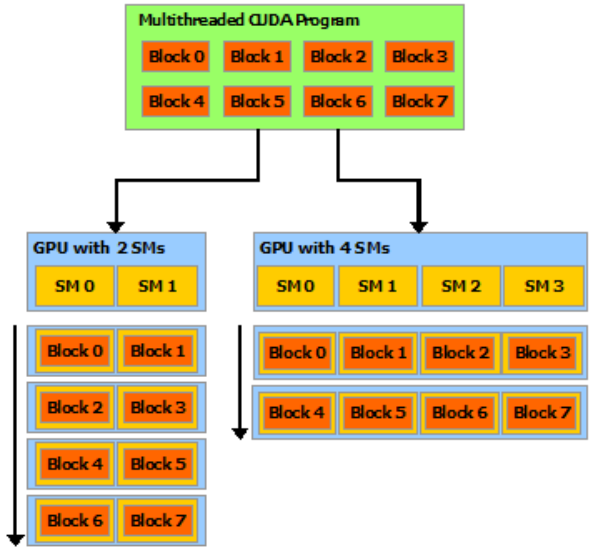


Figura 5. Escalabilidad automática de CUDA: los bloques de hilos se distribuyen de forma homogénea entre los SMs (Streaming Multiprocessors).
Fuente: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Figura 6. Ejemplo de definición y llamado de un kernel.
Fuente: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

Kernels: CUDA extiende C de tal forma que el programador pueda definir funciones denominadas kernels, que cuando sean llamadas, se ejecuten N veces en paralelo por N diferentes Hilos CUDA. El número de hilos a ejecutar la función se define en el momento de llamar el kernel. En la Figura 6 se puede observar un ejemplo de definición y llamado de un kernel.

Jerarquía de hilos: en CUDA los hilos se pueden agrupar en bloques de 1, 2 o 3 dimensiones y a su vez esos bloques se pueden agrupar en mallas de 1, 2 o 3 dimensiones. En la Figura 7 se puede observar una grilla de 2 dimensiones conformada por bloques de hilos también de 2 dimensiones.

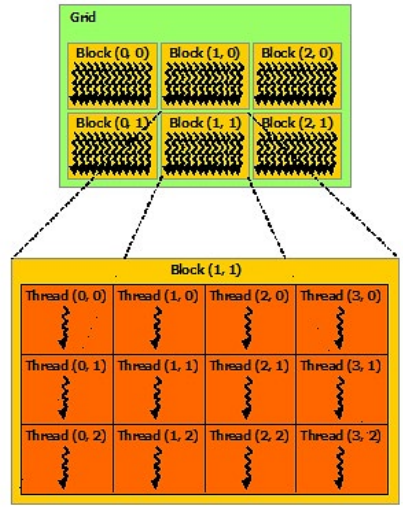


Figura 7. Malla de bloques de hilos.
Fuente: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Figura 8. Ejemplo de definición y llamado de un kernel con una malla bidimensional conformada por bloques bidimensionales de hilos.
Fuente: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

El número de hilos por bloque y el número de bloques por malla se determinan en el momento de llamar el kernel. Dentro del kernel tanto los bloques como los hilos tienen un identificador que se puede acceder a través de una variable (built-in). Para el caso de los bloques la variable es blockIdx y para el caso de los hilos es threadIdx. Adicionalmente se puede acceder a la dimensión de los bloques mediante la variable blockDim.

En el ejemplo de la figura 8 se definen bloques de tamaño 16x16 (256 hilos), que se agrupan en una malla bidimensional definida de tal forma que hallan suficientes bloques como para disponer de un hilo por cada elemento de la matriz a procesar.

Jerarquía de memoria: Las memorias con las cuales se cuenta en CUDA se organizan de forma jerárquica de acuerdo a su visibilidad. Cada hilo tiene una memoria privada de uso exclusivo, cada bloque de hilos tiene una memoria compartida a la cual pueden acceder todos los hilos de un bloque, pero no los de otros bloques, por último, todos los hilos sin importar de que bloque sean pueden acceder a una memoria denominada global. Adicional a esta memoria global existen otras dos memorias de acceso general para todos los hilos pero únicamente para su lectura, estas memorias son la de textura y la constante.

En la Figura 9 se pueden observar los diferentes tipos de memoria en CUDA con su visibilidad por parte de los hilos, los bloques y las mallas.

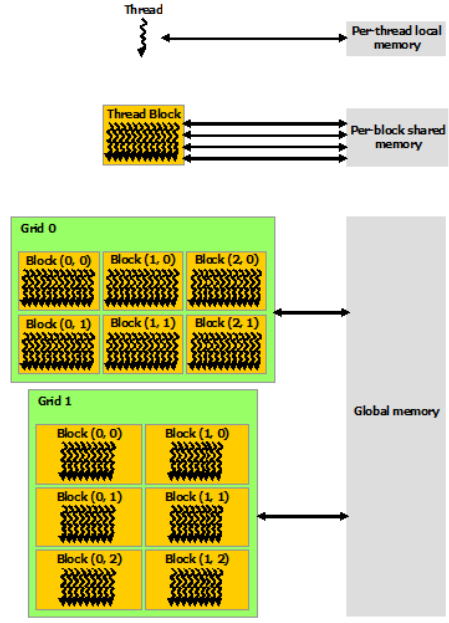


Figura 9. Jerarquía de memoria en CUDA.
Fuente: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

Programación heterogénea: el modelo de programación de CUDA asume que sus hilos se ejecutan en un dispositivo separado físicamente que actúa como coprocesador del host donde se ejecuta el programa C desde el cual se llaman los kernels. Para el caso de tener una CPU y una GPU, esta última actuará como coprocesador del host CPU.

El modelo también asume que tanto la GPU como la CPU poseen su propio espacio de memoria independiente en la DRAM y se refiere a estos espacios como memoria de dispositivo y memoria de host respectivamente. En la Figura 10 se puede observar el concepto de programación heterogénea: un programa en C que ejecuta de forma secuencial código serial que es ejecutado en el host y código paralelo que es ejecutado en el dispositivo (GPU).

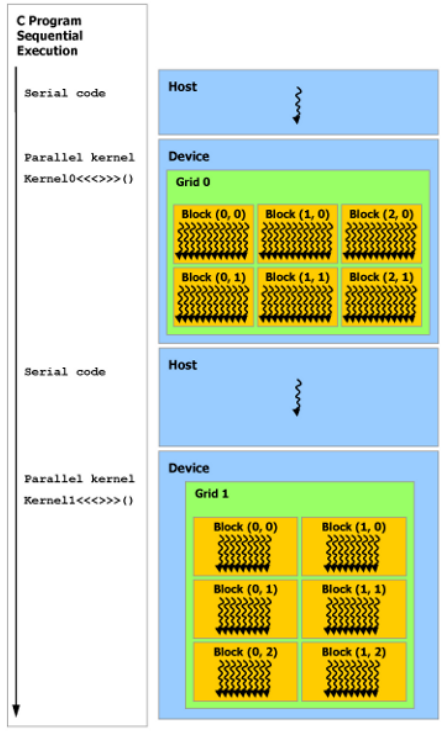


Figura 10. Programación heterogénea.
Fuente: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

1.3 Reto computacional para implementar la fusión de imágenes en arquitecturas computacionales heterogéneas

Como se mencionó en los párrafos anteriores, la fusión de imágenes utiliza una serie de algoritmos que tienen algo en común: involucran en gran medida operaciones pixel a pixel, lo que genera una dependencia directa entre el tamaño de la imagen y la exigencia computacional. Sin embargo, esas operaciones pixel a pixel presentan una baja o nula interdependencia, lo que habilita su parelización masiva para acelerar su cómputo mediante arquitecturas many-core, como por ejemplo las unidades de procesamiento gráfico o GPU como se puede evidenciar en las implementaciones de Yoo et al. (2009) y de Lu et al. (2011).

Evidentemente la paralelización masiva mediante arquitecturas many-core es el camino a seguir para enfrentar la alta exigencia computacional de la fusión de imágenes, y mucho más si se tiene en cuenta que actualmente se dispone de un gran número de librerías eficientes para el cómputo matricial sobre GPU, como por ejemplo CUBLAS (Toolkit, 2011) o clBLAS (Nugteren, 2018).

Sin embargo los algoritmos de fusión de imágenes no están conformados exclusivamente de procesos paralelizables eficientemente, sino que involucran: a) procesos secuenciales con cierto grado de interdependencia que conlleva a que su ejecución en una plataforma many-core no represente ninguna aceleración sino que por el contrario implique tiempos y recursos de memoria adicionales; b) procesos paralelizables que por su alta transferencia de datos entre las plataformas multi-core y many-core es más eficiente su ejecución secuencial. Esto exige un modelo de procesamiento heterogéneo que segmente y distribuya convenientemente tareas entre los dos tipos de arquitecturas, teniendo en cuenta no solo la capacidad de paralelización de los procesos sino también el costo de la

transferencia de los datos y el uso eficiente de las estructuras de memoria disponibles.

1.4 Objetivo de este libro

Este libro busca diseñar e implementar modelos de procesamiento heterogéneos que permitan la aceleración eficiente de los principales métodos de fusión de imágenes sobre plataformas computacionales que integren arquitecturas multi-core (CPU) y arquitecturas many-core (GPU).

En los capítulos 2, 3 y 4, el lector encontrará los conceptos de los métodos de fusión por transformada de Brovey, por multiplicación, por análisis de componentes principales y por el algoritmo de À trous; así como también encontrará el modelo de procesamiento y el código que permite su implementación eficiente en arquitecturas heterogéneas (CPU/GPU). Adicionalmente el libro cuenta con un repositorio (https://github.com/Parall-UD/libro_fusion_imagenes_satelitales_GPU) donde se encuentran los scripts y las imágenes de prueba. En el capítulo 5 se exponen las diferentes métricas que permiten evaluar la calidad de la imagen fusionada tanto espacial como espectralmente. En el capítulo 6 se presentan y analizan los resultados de la evaluación de los modelos y su implementación tanto a nivel de aceleración como a nivel de calidad de la imagen fusionada. Finalmente se presentan las conclusiones.

Capítulo 2

Fusión de imágenes basado en operaciones algebraicas

En función del algoritmo de fusión aplicado se obtendrán imágenes con mayor o menor calidad espacial, pero estableciéndose siempre un compromiso entre esta y la calidad espectral de la imagen fusionada, ya que cuanto mayor será la cantidad de información proveniente de la imagen pancromática que se le inyecta a la multiespectral mejor será su calidad espacial, pero también mayor será la distorsión de las características espectrales de la multiespectral original y viceversa. En la mayoría de los casos, el objetivo es obtener una imagen con una resolución espacial próxima a la de la imagen pancromática, introduciendo la mínima distorsión espectral posible.

2.1 Algoritmos de fusión de imágenes

La fusión puede definirse como la combinación simultanea de información procedente de fuentes distintas que se complementan y cuyo resultado permite mejorar la calidad y la interpretabilidad de los datos originales. En el contexto de la teledetección, la fusión consiste en la combinación de dos o más imágenes con el fin de obtener una nueva

imagen que contenga la información deseada de cada una de ellas. Este proceso de fusión puede llevarse a cabo a distintos niveles: de píxel, de objeto y de decisión (Stathaki 2008, Zhang 2010).

La fusión a nivel de píxel es el nivel de procesado más bajo y consiste en generar una imagen fusionada donde la información asociada a cada píxel se obtiene a partir de los pixeles de las imágenes de origen. La fusión a nivel de objeto se basa en la extracción previa de los objetos en las imágenes origen en base a criterios como tamaño, forma o vecindad, empleando técnicas de segmentación. Finalmente, la fusión a nivel de decisión consiste en fusionar la información al nivel más alto de abstracción. Así, las imágenes fuente son procesadas independientemente para extraer la información que a continuación se combina aplicando reglas de decisión para reforzar la interpretación común.

En este escenario, una de las herramientas de procesado novedosas, y que presentan gran interés por parte de la comunidad científica, son las técnicas de fusión a nivel de píxel o pansharpening, que permiten obtener imágenes de varias bandas del espectro con el máximo nivel de detalle espacial. Así, el principal objetivo de la fusión a nivel de píxel consiste en la aplicación de algoritmos de procesado para mejorar la resolución espacial de las diferentes bandas multiespectrales sin alterar sus características espectrales (Li, Lixin y Mingyi, 2012).

2.2 Revisión de métodos de fusión de imágenes satelitales

De forma genérica, el proceso básico para la fusión de imágenes multiespectrales y pancromáticas de un mismo sensor es el que se muestra en la Figura 11. Lógicamente, y como paso previo a la fusión, es importante garantizar el perfecto registro de las diferentes imágenes. Se aprecia que la primera transformación consiste en la interpolación para ajustar el tamaño de la imagen multiespectral (MS) al de la pancromática (PAN), para seguidamente aplicar el algoritmo de fusión correspondiente.

La fusión de imágenes a nivel de píxel es un campo de investigación muy activo. Si bien, es verdad que desde hace bastantes años se habían comenzado a estudiar, es a partir del año 2000 cuando ha despertado un interés creciente asociado a la disponibilidad de datos procedentes de sensores ópticos de diferentes resoluciones espaciales.

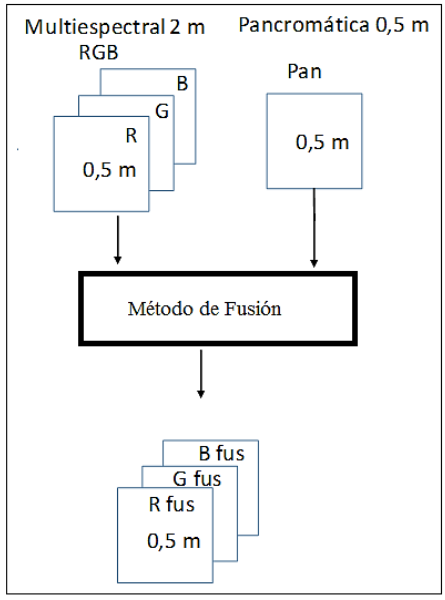


Figura 11. Diagrama genérico del proceso de fusión a nivel de píxel entre las bandas MS y PAN. Fuente: elaboración propia.

Existen diferentes formas de clasificar los distintos algoritmos de fusión (Kpalma et al., 2013; Amro et al., 2011; Zhang, 2010; González-Audícana, 2007).

A continuación, se muestra una de ellas atendiendo a los detalles de su implementación:

- Métodos basados en operaciones algebraicas: las imágenes fusionadas se obtienen como resultado de operaciones aritméticas entre bandas de la imagen MS y la PAN.

- Métodos basados en sustitución de componentes: el principio teórico de estos métodos es la realización de una transformación de la imagen MS original en una serie de componentes transformadas, de tal forma que al sustituir una de dichas componentes por la imagen PAN y realizar la operación de transformación inversa se consiga una imagen fusionada de alta resolución espectral y espacial.
- Métodos basados en la inyección de altas frecuencias: estos métodos se basan en extraer las componentes de alta frecuencia de la imagen PAN, por ejemplo, usando un filtrado paso alto, que posteriormente se inyectan a la MS.
- Métodos basados en el análisis multirresolución: estas técnicas descomponen las bandas MS y PAN a diferentes escalas para extraer los detalles espaciales que se importan a las bandas MS a la escala más fina. Los métodos basados en la transformada wavelet discreta son los algoritmos más empleados en este ámbito de la fusión de imágenes.

Con independencia de la clasificación utilizada por diversos autores, en la actualidad se dispone de un gran número de algoritmos de fusión (Vivone et al., 2015; Amro et al., 2011; Marcello-Ruiz et al., 2011; Ehlers et al. 2010; Stathaki, 2008) principalmente aplicados para la fusión de imágenes ópticas e IR cercanas.

En sus inicios, las técnicas más populares fueron las basadas en operaciones aritméticas, destacando los algoritmos de *Brovey*, *Synthetic Variable Ratio* o *Ratio Enhancement*, y las basadas en la sustitución de bandas tras la aplicación de una transformada, destacando el Análisis de Componentes Principales (PCA, *Principal Component Analysis*), la transformada Intensidad-Brillo-Saturación (IHS, *Intensity-Hue-Saturation*) o el algoritmo Gram-Schmidt (GS).

La utilización de estos algoritmos está muy extendida dada la baja complejidad computacional que presentan. Sin embargo, proporcionan imágenes fusionadas de menor calidad espectral, es decir cuyo color presenta distorsiones respecto al color de las imágenes multiespectrales originales. Esto impide su uso en diferentes aplicaciones en el área de la teledetección, como son la clasificación de imágenes o la detección de

cambios. Más recientemente, para el tratamiento de datos de satélite de nuevos sensores con mayor número de bandas, como es el caso de *Worldview-2* o de los sensores hiperespectrales, se ha desarrollado nuevos algoritmos como, por ejemplo, el *Hyperspecral Colour Sharpening* (HCS) (Li, He et al., 2013, Padwick, Deskevich et al., 2010).

Para solventar las limitaciones espectrales de los algoritmos mencionados, surgieron técnicas que inyectan la información de alta frecuencia, destacando los métodos HPF (*High- Pass-Filtering*), HPM (*High-Pass-Modulation*) o el basado en la aplicación de filtros paso alto en el dominio de Fourier (Lillo-Saavedra, Gonzalo et al., 2005). Sin embargo, los métodos que utilizan el análisis multirresolución, y fundamentalmente la Transformada Wavelet Discreta (TWD) son los más populares para disminuir la distorsión espectral. En particular, para lograr resultados óptimos de fusión, diversos esquemas basados en wavelets han sido propuestos por varios investigadores (Hong, Zhang 2008; Amolins, Zhang y Dare, 2007; Lillo - Saavedra, Gonzalo, 2006; Otazu et al., 2005), destacando los algoritmos de Mallat y À trous, cuya principal diferencia se refiere al sentido en el que se realiza la estrategia multirresolución, pues en el primer caso se diezma la imagen mientras que para À trous no se aplica ningún diezmado.

En el ámbito de esta investigación se han seleccionados los algoritmos de *pan-sharpening* que a continuación se detallan para llevar a cabo el proceso de evaluación de la calidad espacial y espectral empleando las métricas existentes. Se han seleccionado algoritmos pertenecientes a diferentes categorías y, en especial, aquellos cuyo comportamiento espectral y espacial es conocido al estar ampliamente documentado en la literatura científica. Hay que destacar que el objetivo es fusionar imágenes para analizar las prestaciones de los índices de calidad.

2.3 Transformada de Brovey

Es un algoritmo de bajo coste computacional basado en operaciones aritméticas y que da como resultado imágenes de buena calidad espacial, pero baja calidad espectral. Utiliza un método que multiplica cada píxel de la imagen multiespectral por la relación entre la intensidad de cada píxel de la pancromática y la suma de las intensidades de todas las bandas de la multiespectral.

Fue originariamente diseñado para imágenes de satélites de tres bandas (composiciones RGB). Así, la transformada de Brovey (Hallada and Cox, 1983) inicial puede ser implementada según la expresión matemática siguiente:

$$NB1 = (3B1/(B1 + B2 + B3)) * PAN \quad (1)$$

$$NB2 = (3B2/(B1 + B2 + B3)) * PAN \quad (2)$$

$$NB3 = (3B3/(B1 + B2 + B3)) * PAN \quad (3)$$

donde *NB*1, *NB*2 y *NB*3 son las bandas fusionadas y *PAN* es la pancromática. Al realizar la implementación del algoritmo ha de tenerse en cuenta que los valores a utilizar deben estar normalizados para evitar desbordamientos de rango. A continuación, se muestra la ecuación extendida del algoritmo para imágenes con *N* bandas:

$$ND_{FUS,bi} = \left(\frac{NB \times ND_{bi}}{ND_{bi} + ND_{bi} + \cdots + BND_{bNB}}\right) * ND_{PAN} \quad (4)$$

Donde:

NB es el número de bandas espectrales.

$ND_{FUS,bi}$ es el valor digital de la banda fusionada *i*.

ND_{bi} es el valor digital de la banda multiespectral *i*.

ND_{PAN} es el valor digital de la banda *PAN*.

2.3.1 *Modelo de procesamiento heterogéneo para la transformada de Brovey*

El modelo de procesamiento heterogéneo para implementar la transformada de Brovey sobre una arquitectura CPU/GPU se presenta en la Figura 12. El primer paso es la separación de bandas para la imagen multiespectral. Posteriormente, se realiza la transferencia de los niveles digitales a memoria global de la GPU, con el fin de realizar una normalización de las bandas. Este proceso consiste en tomar cada una de las bandas, multiplicarlas por un factor, el cual corresponde al número total de bandas y finalmente dividir este valor entre la suma de cada una de las bandas. Acto seguido, se multiplica elemento a elemento cada una de las bandas normalizadas con la imagen pancromática, esto con el propósito de inyectar la riqueza espacial en cada una de las bandas. Después, se calcula el valor máximo y mínimo de las bandas con inyección espacial, para posteriormente en el último paso, realizar un ajuste de riqueza espectral, el cual consiste en restar el valor mínimo a cada elemento de una banda, multiplicarlos por un factor de 255 y este resultado, debe ser dividido por la resta entre el valor máximo y mínimo. Este ajuste se realiza por cada una de las bandas.

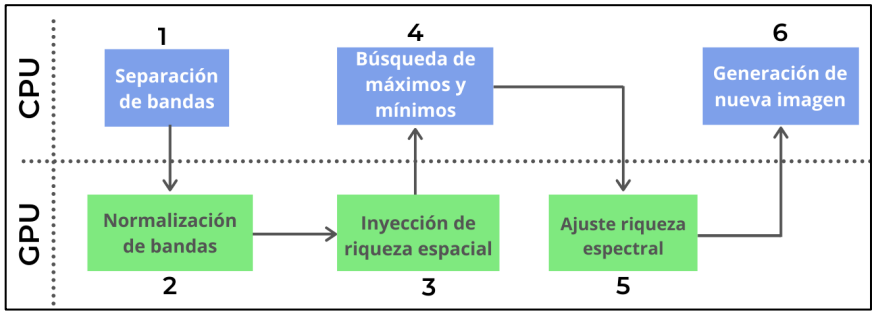


Figura 12. Modelo de procesamiento heterogéneo para la transformada de Brovey

2.3.2 Implementación de la transformada de Brovey en Python

A continuación, se presentan fragmentos secuenciales de código en Python, utilizados para poder llevar a cabo la fusión de imágenes satelitales

mediante la transformada de Brovey. En el repositorio del libro se encuentra el script completo con las imágenes de prueba: https://github.com/Parall-UD/libro_fusion_imagenes_satelitales_GPU.

Definición de dependencias - Líneas 1 – 7:

```
1. import skimage.io
2. import pycuda.autoinit
3. import pycuda.driver as drv
4. import pycuda.gpuarray as gpuarray
5. import numpy as np
6. import skcuda.linalg as linalg
7. from pycuda.elementwise import ElementwiseKernel
```

En estas líneas de código se importan las librerías necesarias para llevar a cabo la fusión de imágenes satelitales mediante la transformada de Brovey haciendo uso de una arquitectura CPU/GPU. Por un lado, la librería ***skimage*** mediante el módulo ***io***, nos permite leer imágenes con extensión TIFF. Asimismo, la librería ***pycuda***, permite acceder a la interfaz de programación de aplicaciones (API) de computación paralela CUDA del Nvidia desde Python. En este orden de ideas, ***pycuda*** admite el manejo de arreglos en memoria de GPU, mediante su módulo ***gpuarray***, y el módulo ***elementwise*** contiene herramientas para la generación de núcleos para la evaluación de expresiones de etapas múltiples en uno o varios operandos en un solo recorrido. También, se importa la librería ***Numpy***, la cual es un paquete fundamental para la computación científica en Python, proporcionando herramientas para el manejo de objetos matriciales multidimensionales y poder realizar rutinas de operaciones rápidas entre matrices. Por último, se encuentra la librería ***Scikit-Cuda,*** proporcionando interfaces de Python para muchas de las funciones de dispositivo/tiempo de CUDA, CUBLAS, CUFFT y CUSOLVER, propias del Kit de programación de CUDA de NVIDIA. En este caso, mediante su módulo ***linalg*** se proporciona la posibilidad de realizar operaciones de álgebra lineal en GPU.

Función para normalización de bandas - Líneas 8 – 10:

```
8.  def step_1(matrix_color, matrix_suma):
9.  matrix_1=gpuarray.if_positive(matrix_suma, (3*matrix_color)
     /matrix_suma,matrix_suma)
10. return matrix_1
```

En estas líneas de código se realiza la declaración de una función nombrada ***step_1***, la cual tiene como propósito realizar la división de una banda entre la suma de todas las bandas. Mediante la función ***if_positive*** del módulo ***gpuarray*** se realiza la evaluación de cada posición de la matriz, tomando como criterio si el valor es positivo. De acuerdo a este valor, se realiza la primera operación o la segunda constatando una sentencia de condicional.

Función para la inyección de riqueza espacial - Líneas 11 – 13:

```
11.  def step_2(matrix_1, matrix_image_pan):
12.  matrix_2 = linalg.multiply(matrix_1, matrix_image_pan)
13.  return matrix_2
```

Asimismo, se define la función ***step_2,*** la cual permite realizar una multiplicación posición a posición entre dos matrices. Lo anterior, mediante la función ***multiply*** propia del módulo ***linalg***. Esta función recibe como parámetros las dos matrices que se desean multiplicar posición a posición.

Función para obtener máximos y mínimos - Líneas 14– 17:

```
14.  def step_3(matrix_1):
15.  mat_max = np.amax(matrix_1.get())
16.  mat_min = np.amin(matrix_1.get())
17.  return mat_max, mat_min
```

Adicionalmente, se debe establecer una función que permita calcular el valor máximo y mínimo a partir de una matriz de entrada. Debido a esto se define la función ***step_3.***

Núcleo para ajuste espectral - Líneas 18 – 21:

```
18  lin_comb = ElementwiseKernel(
19.   "float a, float *x, float b, float *z",
20.   "z[i] = ((x[i]-a)*255)/(b-a)",
21.    "linear_combination")
```

En estas líneas de código se establece la variable ***lin_comb*** la cual almacena una función ElementwiseKernel, recibiendo como parámetros dos valores tipo **float** y tres matrices flotantes. Donde la matriz Z se convertirá en la matriz de salida de esta función. Cada vez que se haga un llamado a esta función se generará un núcleo y se realizará una operación de ajuste de riqueza espectral.

Función para ajuste espectral - Líneas 22 – 24:

```
22.  def step_4(matrix_1, matrix_color, mat_max, mat_min):
23.  lin_comb(mat_min, matrix_1, mat_max, matrix_color)
24.  return matrix_color
```

Sin embargo, para mantener uniformidad en el código, se define una función propia en el lenguaje de Python, llamada ***step_4.*** Esta función, estará encargada de realizar el llamado a la función de ***Elementwise***, establecida previamente.

Lectura y carga de imágenes - Líneas 25 – 26:

```
25.  multispectral = skimage.io.imread('multispectral.tiff', plugin = 'tifffile')
26.  panchromatic = skimage.io.imread('panchromatic.tiff', plugin = 'tifffile')
```

A partir de estas líneas de código, se realiza la lectura de la imagen multiespectral y pancromática. Lo anterior, mediante la función ***imread***, perteneciente al módulo ***io*** de la librería ***scikit-image.*** Esta función convierte las imágenes que se desean leer a un arreglo multidimensional de

numpy, con el propósito de poder ser utilizadas y manejadas mediante su representación matricial.

Conversión de tipo de dato de las bandas - Líneas 27 – 32:

```
27. multispectral = multispectral.astype(np.float32)
28.  r = multispectral[:,:,0].astype(np.float32)
29.  g = multispectral[:,:,1].astype(np.float32)
30.  b = multispectral[:,:,2].astype(np.float32)
31.  panchromatic = panchromatic.astype(np.float32)
32.  msuma = r+g+b
```

Una vez se han leído y cargado las imágenes, se procede a especificar el tipo de dato de cada uno de los píxeles de la imagen, en este caso para manejar uniformidad se especifica un tipo flotante de 32 bits haciendo uso del tipo de ***float32*** de **numpy.** Adicionalmente, en las líneas 28 a 30, se realiza una indexación sobre la matriz que contiene la información de la imagen multiespectral, con el fin de obtener cada una de las bandas de su espacio de color, que en este caso es rojo, verde, azul (RGB). Para finalizar, la transformada de Brovey, requiere conformar un arreglo bidimensional que reúna la suma pixel a pixel de cada una de las bandas extraídas anteriormente. Esto se almacena en la variable ***msuma***.

Transferencia de variables a memoria global de GPU - Líneas 33 – 38:

```
33. r_gpu = gpuarray.to_gpu(r)
34. g_gpu = gpuarray.to_gpu(g)
35. b_gpu = gpuarray.to_gpu(b)
36. panchromatic_gpu = gpuarray.to_gpu(panchromatic)
37. msuma_gpu = gpuarray.to_gpu(msuma)
38. linalg.init()
```

Durante todo este momento, se ha venido trabajando sobre la CPU del equipo. Sin embargo, en estas líneas de código se realiza la transferencia de los arreglos de ***numpy*** que contienen las diferentes bandas a arreglos de ***pycuda***, es decir, esta transferencia se realiza de memoria CPU a memoria

global de la GPU. Esto se logra mediante la función ***to_gpu*** propia del módulo ***gpuarray.*** Dicha función recibe por parámetro el arreglo que deseamos transferir. En esta oportunidad se realiza la transferencia a GPU de la banda roja, verde, azul, la imagen pancromática y la matriz que tiene la suma de las bandas. Por último, se inicializa el módulo de operaciones de álgebra lineal de ***scikit-cuda***.

Normalización e inyección espacial de bandas - Líneas 39 – 44:

```
39. m11_gpu = step_1(r_gpu, msuma_gpu)
40. m22_gpu = step_2(m11_gpu, panchromatic_gpu)
41. m33_gpu = step_1(b_gpu, msuma_gpu)
42. m44_gpu = step_2(m33_gpu, panchromatic_gpu)
43. m55_gpu = step_1(g_gpu, msuma_gpu)
44. m66_gpu = step_2(m55_gpu, panchromatic_gpu)
```

En esta línea de código se realiza la división de las bandas (R, G, B) entre la matriz que contiene la suma de estas bandas, esto mediante la función ***step_1()*** declarada al inicio de este proceso. Acto seguido, a través de la función ***step_2***, se toma el resultado de la división de matrices para cada una de las bandas y se realiza una multiplicación posición a posición con la imagen pancromática. Es importante resaltar que todo este proceso se realizó en la GPU.

Ajuste espectral de bandas - Líneas 45 – 53:

```
45. Amax_host, Amin_host = step_3(m22_gpu)
46. rr_gpu = gpuarray.empty_like(r_gpu)
47. step_4(m22_gpu, rr_gpu, Amax_host, Amin_host)
48. Amax_host, Amin_host = step_3(m66_gpu)
49. gg_gpu = gpuarray.empty_like(g_gpu)
50. step_4(m66_gpu, gg_gpu, Amax_host, Amin_host)
51. Amax_host, Amin_host = step_3(m44_gpu)
52. bb_gpu = gpuarray.empty_like(b_gpu)
53. step_4(m44_gpu, bb_gpu, Amax_host, Amin_host)
```

Acto seguido, se obtienen los valores máximos y mínimos de la matriz resultado de la multiplicación de la banda y la imagen pancromática. Asimismo, se separa espacio en memoria para las matrices que se obtendrán en esta línea de código, eso mediante la función ***empty_like()*** la cual recibe como parámetro el tamaño de la matriz que se desea separar, en este caso se toma una matriz como guía. Esto quiere decir que se separará en memoria y se creará un arreglo en GPU exactamente del tamaño de la matriz que se pasa por parámetro. Para finalizar, se aplica el proceso de ajuste espectral mediante la función ***step_4()*** la cual hace el llamado a la función ***Elementwise.*** Ese proceso de ajuste espectral consiste en que el valor mínimo se resta de cada banda generada mediante la función ***step_2()*** y los datos resultantes se multiplican por un coeficiente de 255 para la posterior normalización (división) por la diferencia entre los valores máximo y mínimo. Ese proceso se realiza para cada una de las bandas (R, G, B) procesadas anteriormente.

Transferencia de bandas resultantes a memoria CPU - Líneas 54 – 56:

```
54. ggg_host = gg_gpu.get().astype(np.uint8)
55. rrr_host = rr_gpu.get().astype(np.uint8)
56. bbb_host = bb_gpu.get().astype(np.uint8)
```

El proceso anterior se realizó sobre la GPU, sin embargo, es necesario realizar una conversión de tipo de datos, donde se pasa de un flotante de 32 bits a un entero de 8 bits, para que la imagen resultado puede ser visualizada fácilmente. Este proceso de conversión se realiza para cada una de las bandas resultantes y adicionalmente, se realiza sobre CPU. Mediante la función ***get()***, se realiza la transferencia de datos desde memoria GPU a memoria CPU.

Generación de nueva imagen - Líneas 57 – 58:

```
57. fusioned_image = np.stack((rrr_host, ggg_host, bbb_host),axis=2)
58. skimage.io.imsave('broveygpu_image.tif',fusioned_image,plugin = 'tifffile')
```

Para finalizar, se realiza el proceso de concatenación de las bandas procesadas mediante la función ***stack*** de ***numpy***. Eso produce una nueva imagen que mantiene la riqueza espectral ajustada de la imagen multiespectral junto con la resolución especial de la imagen pancromática. Por último, mediante la función ***imsave*** de ***skimage*** se guarda localmente la imagen generada a partir de la fusión de estas imágenes. La Figura 13C, presenta la imagen resultado, al realizar la fusión de la imagen multiespectral (Figura 13A) y pancromática (Figura 13B), ambas con dimensión de 1024 píxeles por 1024 píxeles. Lo anterior mediante la transformada de Brovey.

A) B) C)

Figura 13. Imagen fusionada de 1024x1024 píxeles, mediante la transformada de Brovey

2.4 Método de multiplicación

Este método aplica un algoritmo simple de multiplicación, para incorporar el contenido de la imagen pancromática en la imagen multiespectral (Pohl and Van Genderen, 1998).

$$R_{ijk} = M_{ijk} \times PAN \quad (5)$$

En donde

R_{ijk} es la imagen fusionada

M_{ijk} imagen multiespectral banda k

2.4.1 Modelo de procesamiento heterogéneo para el método de multiplicación

La Figura 14 presenta la interacción entre la CPU y la GPU implementada para el método de multiplicación. Como primer paso, se realiza la separación de las bandas de la imagen multiespectral en CPU. Acto seguido, se realiza la transferencia de las bandas a la memoria global de la GPU, para inyectar la riqueza espacial en cada banda. Este proceso se realiza mediante la multiplicación elemento a elemento de la imagen pancromática con cada una de las bandas. Posteriormente, se calcula el valor máximo y mínimo de las bandas con inyección espacial, para finalmente realizar un ajuste de riqueza espectral, el cual consiste en restar el valor mínimo a cada elemento de una banda multiplicarlos por un factor de 255, este resultado debe ser dividido por la resta entre el valor máximo y mínimo. Este ajuste se realiza por cada una de las bandas.

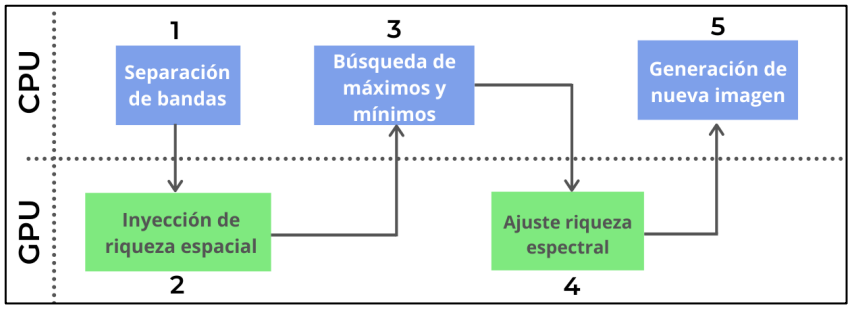


Figura 14. Modelo de procesamiento heterogéneo para el método de multiplicación.

2.4.2 Implementación del método de multiplicación en Python

A continuación, se presentan fragmentos secuenciales de código en Python, utilizados para poder llevar a cabo la fusión de imágenes satelitales mediante el método de multiplicación. En el repositorio del libro (https://github.com/Parall-UD/libro_fusion_imagenes_satelitales_GPU) se encuentra el script completo con las imágenes de prueba.

Definición de dependencias - Líneas 1 – 7:

```
1. import skimage.io
2. import numpy as np
3. import pycuda.autoinit
4. import pycuda.driver as drv
5. import pycuda.gpuarray as gpuarray
6. import skcuda.linalg as linalg
7. from pycuda.elementwise import ElementwiseKernel
```

De igual manera en el que se presentó al inicio de la implementación de la transformada de Brovey, para el método de multiplicación también es necesario importar un conjunto de librerías como ***scikit-image***, ***numpy***, ***pycuda*** y ***scikit-cuda.*** Estas librerías se explican de mejor manera en las primeras líneas de la transformada de Brovey, sin embargo, estas librerías nos permiten la lectura y almacenamiento de imágenes con extensión .TIFF, la interacción con herramientas para realizar operaciones matriciales, una interfaz para la interacción con computación paralela mediante el framework CUDA, entre otras funcionalidades.

Función para la inyección de riqueza espacial - Líneas 8 – 10:

```
8.  def step_1(color_matrix, image_matrix):
9.    matrix_sal = linalg.multiply(color_matrix, image_matrix)
10.  return matrix_sal
```

Como primera instancia, es necesario realizar la inyección de riqueza espacial de la imagen pancromática a la imagen multiespectral. Debido a esto, se requiere establecer una función que permita realizar esta tarea, mediante la multiplicación posición a posición entre dos matrices. Es por esto que, se crea la función ***step_1()***, la cual recibe por parámetros las dos matrices que se desean multiplicar. Por último, esta función retorna la matriz resultante de la multiplicación posición a posición.

Función para obtener máximos y mínimos - Líneas 11 – 14:

```
11. def step_2(matrix_1):
12.  mat_max = np.amax(matrix_1)
13.  mat_min = np.amin(matrix_1)
14.  return mat_max, mat_min
```

En estas líneas de código se define una función nombrada como ***step_2()***, cuyo propósito es obtener el máximo y mínimo valor entre un arreglo bidimensional. A su vez, esta función retorna estos dos valores. Lo anterior se realiza mediante las funciones ***amax()*** y ***amin()*** de ***numpy***.

Núcleo para ajuste espectral - Líneas 15 – 18:

```
15.  lin_comb = ElementwiseKernel(
16.  "float a, float *x, float b, float *z",
17.  "z[i] = ((x[i]-a)*255)/(b-a)",
18.   "linear_combination")
```

Posteriormente, mediante este fragmento de código, donde se hace uso de la función ***ElementwiseKernel()***, se establece el procedimiento de ajuste espectral. Este proceso consiste en tomar el valor mínimo y restarlo de su respectiva matriz para poder ser multiplicada por un valor constante de 255. Después, se toman los valores obtenidos y se realiza una normalización respecto a la diferencia entre los valores máximos y mínimos de dicha matriz inicial. Es importante resaltar que, lo que se encuentra dentro de la función es un pequeño fragmento de código de C-CUDA, embebido dentro de Python.

Función para ajuste espectral - Líneas 19 – 21:

```
19.  def step_3(matrix_1, matrix_color, mat_max, mat_min):
20.  lin_comb(mat_min, matrix_1, mat_max, matrix_color)
21.  return matrix_color.get()
```

Estas líneas tienen como objetivo definir una función propia de Python, donde se realice el llamado al núcleo ***Elementwise*** que se encarga de realizar el ajuste espectral. Esta función recibe los siguientes parámetros:

- ***matrix_1:*** es la matriz que se desea ajustar espectralmente, es decir en este caso serán las distintas bandas que han sido procesadas durante la aplicación de este método.
- ***matrix_color:*** es una matriz vacía donde se almacenará la matriz resultado de aplicar esta función de Elementwise.
- ***mat_max:*** es el valor máximo de la ***matrix_1.***
- ***mat_min:*** es el valor mínimo de la ***matrix_1.***

Finalmente, esta función retorna la ***matrix_color.*** Sin embargo, se debe tener en cuenta que este proceso se realiza en GPU, en este caso se desea realizar la transferencia de esta variable a memoria de CPU, por lo tanto, utilizamos la función ***get().***

Lectura y carga de imágenes - Líneas 22 – 23:

```
22. multispectral = skimage.io.imread('multispectral.tiff', plugin='tifffile')
23. panchromatic = skimage.io.imread('panchromatic.tiff', plugin='tifffile')
```

En estas líneas de código, se realiza la lectura y carga de las imágenes de punto de partida para fusión de imágenes satelitales. Estas son, la imagen multiespectral y pancromática. Además de esto, la librería **scikit-image**, lee estas imágenes y las presenta al público mediante su representación matricial de tipo ***numpy.***

Conversión de tipo de dato de las bandas - Líneas 24 – 28:

```
24. multispectral = multispectral.astype(np.float32)
25. r = multispectral[:,:,0].astype(np.float32)
26. g = multispectral[:,:,1].astype(np.float32)
27. b = multispectral[:,:,2].astype(np.float32)
28. panchromatic = panchromatic.astype(np.float32)
```

Posteriormente, al leer la imagen multiespectral y pancromática, y tenerlas en su representación matricial, se realiza la conversión de tipos de datos a un flotante de 32 bits, lo anterior con el propósito de mantener homogeneidad en las operaciones matriciales. Esta conversión, se realiza mediante la función propia de todo elemento de numpy como lo es ***astype()***, donde por parámetro que recibe es el tipo de dato. Así mismo, la imagen multiespectral está compuesta por un conjunto de bandas dependiendo de su espacio de color. Para esta ocasión el espacio de color es RGB, lo cual indica que tiene tres bandas, una roja, una verde y una azul (*red, green, blue*). De acuerdo a esto, mediante la indexación de matrices en ***numpy*** se extraen cada una de estas bandas.

Transferencia de variables a memoria global de GPU - Líneas 29 – 33:

```
29. r_gpu = gpuarray.to_gpu(r)
30. g_gpu = gpuarray.to_gpu(g)
31. b_gpu = gpuarray.to_gpu(b)
32. panchromatic_gpu = gpuarray.to_gpu(panchromatic)
33. linalg.init()
```

Como se ha mencionado anteriormente, se desea realizar la implementación de esta técnica de fusión sobre una arquitectura GPU, por lo tanto, es necesario realizar la transferencia de las variables necesarias para este proceso de memoria de CPU y a memoria global de GPU. En este orden de ideas, mediante la función ***to_gpu()*** se envía a GPU cada una de las bandas extraídas anteriormente y la imagen pancromática. Para finalizar, se inicializa el módulo de álgebra lineal de la librería ***scikit-cuda***, lo anterior al ejecutar ***linalg.init()***. Después de realizar esto, ya se pueden ejecutar funcionalidades de este módulo.

Inyección de riqueza espacial a bandas - Líneas 34 – 36:

```
34. m33_gpu = step_1(r_gpu, panchromatic_gpu)
35. m44_gpu = step_1(g_gpu, panchromatic_gpu)
36. m55_gpu = step_1(b_gpu, panchromatic_gpu)
```

Mediante estas líneas de código, se busca tomar la riqueza espacial de la imagen pancromática e inyectarla en cada una de las bandas, todo esto en GPU. Este proceso, se realiza mediante la función ***step_1()*** definida previamente. Dicha función, realiza la multiplicación píxel a píxel entre las bandas (R, G, B) y la representación matricial de la imagen pancromática.

Ajuste espectral de bandas - Líneas 37 – 45:

```
37. Amax, Amin = step_2(m33_gpu.get())
38. br_gpu = gpuarray.empty_like(r_gpu)
39. br_host = step_3(m33_gpu, br_gpu, Amax, Amin)
40. Amax, Amin = step_2(m44_gpu.get())
41. bg_gpu = gpuarray.empty_like(g_gpu)
42. bg_host = step_3(m44_gpu, bg_gpu, Amax, Amin)
43. Amax, Amin = step_2(m55_gpu.get())
44. bb_gpu = gpuarray.empty_like(b_gpu)
45. bb_host = step_3(m55_gpu, bb_gpu, Amax, Amin)
```

Posteriormente, mediante estas líneas de código, se obtienen los valores máximos y mínimos de la matriz resultado de la multiplicación de cada banda y la imagen pancromática. Asimismo, mediante la función ***empty_like(),*** se separa espacio en memoria de la GPU para las matrices que se obtendrán en estas líneas de código. La función ***empty_like*** recibe como parámetro el tamaño de la matriz que se desea separar, en este caso se toma una matriz como guía. Esto quiere decir que se separará en memoria y se creará un arreglo en GPU exactamente del tamaño de la matriz que se pasa por parámetro. Para finalizar, utilizando la función ***step_3()*** se aplica el proceso de ajuste espectral, dicha función hace el llamado al núcleo simple de ***Elementwise*** creado con anterioridad, para realizar el ajuste espectral de cada una de las bandas enriquecidas espacialmente**.** Ese proceso de ajuste espectral consiste en que el valor mínimo se resta de cada banda generada mediante la función ***step_2()*** y los datos resultantes se multiplican por 255 para la posterior normalización (división) por la

diferencia entre los valores máximo y mínimo. Ese proceso se realiza para cada una de las bandas (R,G,B) procesadas anteriormente.

Transferencia de bandas resultantes a memoria CPU - Líneas 46 – 48:

```
46. brr = br_host.astype(np.uint8)
47. bgg = bg_host.astype(np.uint8)
48. bbb = bb_host.astype(np.uint8)
```

Estas líneas de código realizan una conversión de tipo de datos, donde se pasa de un ***float32*** a un ***uint8***, es decir se convierte de un flotante de 32 bits a entero de 8 bits. Esta conversión se realiza para cada una de las bandas que ha sido ajustada espectralmente.

Generación de nueva imagen - Líneas 49 – 50:

```
49. fusioned_image = np.stack((brr, bgg, bbb),axis=2)
50. skimage.io.imsave('multiplicativegpu_image.tif',fusioned_image, plugin='tifffile')
```

Finalmente, se realiza el proceso de concatenación de las bandas procesadas, mediante la función ***stack*** de ***numpy***. Eso produce una nueva imagen que mantiene la riqueza espectral ajustada de la imagen multiespectral junto con la resolución espacial de la imagen pancromática. Adicionalmente, la función ***imsave*** de ***skimage*** permite guardar localmente la imagen generada a partir de la fusión de estas imágenes. La Figura 15C, presenta la imagen resultado, al realizar la fusión de la imagen multiespectral (Figura 15A) y pancromática (Figura 15B), ambas con dimensión de 1024 píxeles por 1024 píxeles. Lo anterior, mediante el método de multiplicación.

A) B) C)

Figura 15. Imagen fusionada de 1024x1024 píxeles, mediante el método Multiplicación

Capítulo 3

Métodos basados en transformadas: métodos de sustitución de componentes

La mayoría de los sensores MS recogen información en bandas adyacentes del espectro electromagnético, lo que habitualmente implica detectar información redundante, ya que muchas de las cubiertas existentes sobre la superficie terrestre tienden a presentar comportamientos similares en regiones próximas del espectro.

3.1 Fusión de imágenes usando análisis de componente principales

El análisis en componente principales, también denominado transformación PCA (de sus siglas en inglés *Principal Component Analysis*) o transformada de Karhunen-Loève o Hotelling (Shettigara, 1992), crea nuevas imágenes a partir de las originales llamadas componentes principales (CP), no correlacionadas entre sí, que reorganizan la información original. Con las componentes principales se elimina esa información redundante entre componentes, de forma que la primera CP se define como la dirección a lo largo de la cual la varianza de los datos tiene

su máximo. Es decir, la esencia del análisis en componentes principales es la transformación de un conjunto de variables correlacionadas en un nuevo conjunto de variables no correlacionadas.

El método de fusión PCA (Shettigara, 1992) es similar al IHS en cuanto a que se basa en la transformación de las bandas de la imagen multiespectral en una serie de componentes, para luego sustituir una de ellas por la imagen pancromática adaptada, buscando de esta manera añadir la información espacial a la espectral. Tal y como se puede observar en la Figura 16, el proceso a seguir para desarrollar este método de fusión es el que se presenta a continuación. En primer lugar, se obtienen tantas componentes principales como bandas tenga la imagen multiespectral. De este modo, la CP1 contiene información espacial y las CP restantes la información espectral. A continuación, se iguala el histograma de la imagen pancromática al de la primera componente principal CP1, es decir, a aquella que contiene información relativa al conjunto de las bandas.

La imagen pancromática modificada (una vez ajustado su histograma) sustituye a la primera componente principal CP1. Aplicando a estas bandas la transformación inversa se obtienen las bandas de la imagen fusionada.

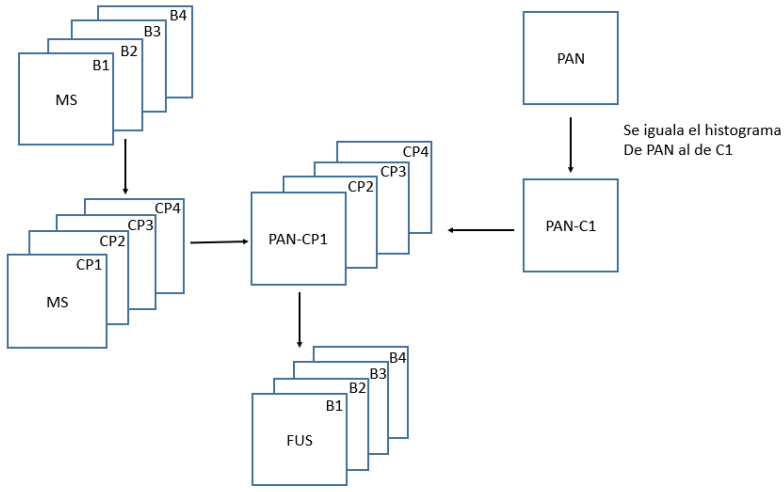


Figura 16. Algoritmo de fusión PCA. Fuente Autor

Una de las ventajas de este método es que no está limitado a imágenes de 3 bandas, sino que puede utilizarse para un número ilimitado de bandas. Sin embargo, introduce distorsión espectral en la imagen fusionada, esto es así pues se parte de la base de que tras la transformación PCA, la disociación entre información espacial y espectral de la imagen multiespectral es total, pero esto no es así. Computacionalmente este algoritmo de fusión es pesado ya que implica la realización del cálculo de la matriz de covarianza, el cálculo de los autovalores y autovectores, generar la matriz ortogonal y diversas operaciones algebraicas (producto de matrices, matrices inversas, transposiciones, etc.) para generar las componentes principales CP1, CP2 hasta CPN, donde N corresponderá al número de bandas de la imagen multiespectral.

3.1.1 Modelo de procesamiento heterogéneo para PCA

La Figura 17 presenta la implementación de PCA sobre una arquitectura de computación heterogénea CPU/GPU. Como primera instancia en el paso número uno, se le realiza a la imagen multiespectral la descomposición en sus bandas, en este caso (R, G, B). Después en el paso número dos se lleva a cabo la transferencia de los niveles digitales a memoria global de la tarjeta gráfica, con el propósito calcular el promedio de cada uno y restar dicho valor de cada uno de los pixeles de las bandas. Lo anterior, con el propósito de calcular la matriz de covarianza para cada una de las bandas en GPU. Posterior a esto en el paso número tres, se carga la matriz de covarianza a la memoria de la CPU, con el fin de calcular el coeficiente de diagonalización ortogonal, determinar el polinomio característico y calcular vectores y valores propios, para obtener la matriz ortogonalizada. En el paso siguiente, se transfiere la matriz ortogonalizada a memoria global de la GPU, para calcular los componentes principales mediante las bandas R, G, B originales. A partir de los componentes calculados, la imagen pancromática y la inversa de la matriz ortogonalizada se calculan los

componentes finales. Finalmente, se transfieren a CPU para realizar el stack de los componentes y generar la imagen resultante.

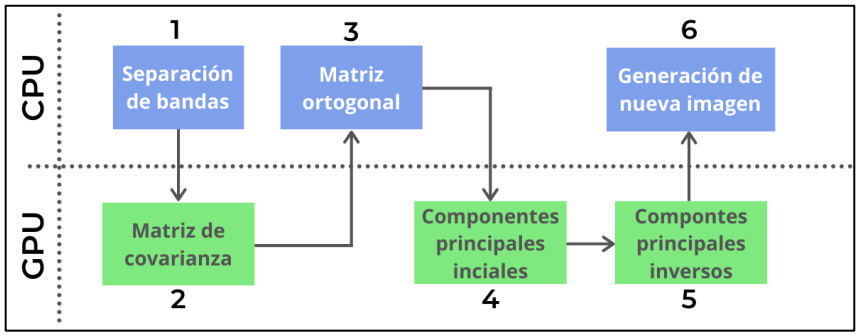


Figura 17. Modelo de procesamiento heterogéneo para PCA

3.1.2 Implementación de PCA en Python

A continuación, se presentan fragmentos secuenciales de código en Python, utilizados para poder llevar a cabo la fusión de imágenes satelitales mediante la técnica de análisis de componentes principales. En el repositorio del libro (https://github.com/Parall-UD/libro_fusion_imagenes_satelitales_ GPU) se encuentra el script completo con las imágenes de prueba.

Definición de dependencias - Líneas 1 – 10:

```
1. import skimage.io
2. import numpy as np
3. from numpy import linalg as la
4. import pycuda.autoinit
5. import pycuda.driver as drv
6. import pycuda.gpuarray as gpuarray
7. from pycuda import compiler
8. import skcuda.misc as misc
9. from pycuda.elementwise import ElementwiseKernel
```

Como se realizó en las dos técnicas de fusión anteriores, estas primeras líneas de código tienen como objetivo importar las librerías necesarias para la correcta ejecución de los siguientes fragmentos de código. De igual

manera, se hace uso de librerías mencionadas anteriormente como lo son ***Pycuda, Numpy, Scikit-image*** y ***Scikit-cuda***. Sin embargo, para esta técnica se hace necesario importar un nuevo módulo de esta última librería. El módulo es ***misc,*** el cual nos proporciona rutinas misceláneas, es decir, utilidades que no se han contemplado en otros módulos. Finalmente, para la librería ***Pycuda*** se importa un nuevo módulo llamado ***compiler***, el cual nos permite compilar bloques de código escritos en lenguaje C-CUDA y así poder ser ejecutados en Python.

Núcleo para calcular la matriz de varianza-covarianza - Líneas 11 – 44:

```
10.  kernel_var_cov = """
11.  #include <stdio.h>
12.  __global__ void CovarianceKernel(float *R, float *G, float *B, float *D)
13.  {
14.          const uint tx = threadIdx.x;
15.          const uint ty = threadIdx.y;
16.          __shared__ float prueba_salida;
17.          if (threadIdx.x == 0) prueba_salida = 0;
18.          float valor_temp = 0;
19.          float salida_temp[9];
20.          __syncthreads();
21.          const int size = 3;
22.          float arreglo[size];
23.          arreglo[0] = R[ty * %(BLOCK_SIZE)s + tx];
24.          arreglo[1] = G[ty * %(BLOCK_SIZE)s + tx];
25.          arreglo[2] = B[ty * %(BLOCK_SIZE)s + tx];
26.          __syncthreads();
27.          for(int k = 0; k < 3; k++){
28.                  for(int h = 0; h < 3; h++){
29.                          valor_temp = arreglo[k]*arreglo[h];
30.                          salida_temp[k*3+h] = valor_temp;
31.                          valor_temp = 0;
32.                  }
33.            }
4.           __syncthreads();
5.           for (int i = 0; i < 9; ++i){
36.                  atomicAdd(&prueba_salida,salida_temp[i]);
37.                  __syncthreads();
38.                  D[i] += prueba_salida;
39.                  __syncthreads();
```

```
40.                     prueba_salida = 0.0;
41.                     __syncthreads();
42.                 }
43.     }
44.     """
```

En este fragmento de código se establece un núcleo escrito en lenguaje de C-CUDA. El objetivo de este núcleo es calcular la matriz de varianza-covarianza a partir de las bandas RGB de la imagen multiespectral. Para realizar este proceso se utiliza la palabra reservarda de C-CUDA ***threadIdx*** mediante sus atributos ***x***, ***y***, para obtener la posición del hilo que se está ejecutando en los bloques de hilos de la GPU. Adicionalmente, mediante la función ***syncthreads()*** se realiza la sincronización de todos los hilos que se estén ejecutando paralelamente. Lo anterior, con el objetivo de coordinar los accesos a memoria, es decir, que ninguno de los hilos puede seguir realizando su tarea hasta que el resto de hilos hayan terminado. Asimismo, realiza una operación atómica de lectura-modificación-escritura con datos que residan en memoria global o compartida. Por ejemplo, la función ***atomicAdd()*** nos permite tomar un valor en memoria global o compartida y añadirle un número y escribir el resultado exactamente en la misma dirección, lo que se conoce como sobreescritura. La operación es atómica, dado que garantiza que se realizará sin interferencia de otros hilos. En otras palabras, ningún otro hilo puede acceder a esta dirección hasta que se complete la operación. La función ***CovarianceKernel*** escrita en C-CUDA se almacenada en la variable global ***kernel_var_cov*** de Python y recibe como parámetros la matriz que almacena la banda Roja, la matriz de la banda verde y la matriz de la banda azul (R, G, B) y por último, recibe una matriz ***D***, la cual será la matriz de salida de esta operación.

Núcleo para calcular componentes principales iniciales - Líneas 45 – 74:

```
45. kernel_componentes_principales_original = """
46. #include <stdio.h>
47.     __global__ void componentesPrincipalesOriginal(float *R, float *G, float *B,
        float *Q, float *S1, float *S2, float *S3)
48. {
49.             const uint tx = threadIdx.x;
50.             const uint ty = threadIdx.y;
51.             const int size = 3;
52.             float salida_temp [size];
53.             float valor_temp = 0.0;
54.             float arreglo[size];
55.             arreglo[0] = R[ty * %(BLOCK_SIZE)s + tx];
56.             arreglo[1] = G[ty * %(BLOCK_SIZE)s + tx];
57.             arreglo[2] = B[ty * %(BLOCK_SIZE)s + tx];
58.             __syncthreads();
59.             for(int i = 0; i < 3; ++i){
60.                 for(int j = 0; j < 3; ++j){
61.                     valor_temp += (Q[i*3+j] * arreglo[j]);
62.                 }
63.                 salida_temp[i] = valor_temp;
64.                 valor_temp = 0.0;
65.             }
66.             __syncthreads();
67.             S1[ty * %(BLOCK_SIZE)s + tx] = salida_temp[0];
68.             __syncthreads();
69.             S2[ty * %(BLOCK_SIZE)s + tx] = (-1.0)*salida_temp[1];
70.             __syncthreads();
71.             S3[ty * %(BLOCK_SIZE)s + tx] = salida_temp[2];
72.             __syncthreads();
73.     }
74.     """
```

En estas líneas de código, se define un núcleo para el cálculo de los componentes principales a partir de las bandas originales de la imagen multiespectral. De igual manera, este núcleo que se ejecutará en la GPU mediante el lenguaje C-CUDA. En este núcleo, se reitera el uso de los identificadores para cada uno de los hilos del bloque mediante ***threadIdx.x*** y ***threadIdx.y***. Asimismo, se interpretan los arreglos bidimensionales como

arreglos unidimensionales, tal como sucede en el contexto natural del lenguaje de programación **C**. De nuevo, se hace uso de la función reservada ***syncthreads(),*** para sincronizar todos los hilos que se estén ejecutando en cierto momento de la rutina. Por último, se va llenando posición a posición por cada hilo, las matrices resultantes, que en este caso hacen referencia a los tres componentes principales obtenidos. La función ***componentesPrincipalesOriginal()*** escrita en C-CUDA y almacenada en la variable global ***kernel_componentes_principales_original*** de Python, recibe como parámetros la matriz que almacena las bandas originales (R, G, B) de la imagen multiespectral, un arreglo ***Q*** que contiene valores propios y por último, recibe las matrices ***S1, S2*** y ***S3*** las cuales serán la matrices de salida de esta operación.

Núcleo para calcular los componentes principales finales - Líneas 75 – 104:

```
75. kernel_componentes_principales_pancromatica = """
76. #include <stdio.h>
77.   __global__ void componentesPrincipalesPancromatica(float *R, float *G, float
      *B, float *E, float *S1, float *S2, float *S3)
78.   {
79.   const uint tx = threadIdx.x;
80.   const uint ty = threadIdx.y;
81.   const int size = 3;
82.   float salida_temp [size];
83.   float valor_temp = 0.0;
84.   float arreglo[size];
85.   arreglo[0] = R[ty * %(BLOCK_SIZE)s + tx];
86.   arreglo[1] = G[ty * %(BLOCK_SIZE)s + tx];
87.   arreglo[2] = B[ty * %(BLOCK_SIZE)s + tx];
88.   __syncthreads();
89.   for(int i = 0; i < 3; ++i){
90.   for(int j = 0; j < 3; ++j){
91.   valor_temp += (E[i*3+j] * arreglo[j]);
92.   }
93.   salida_temp[i] = valor_temp;
94.   valor_temp = 0.0;
95.   }
96.   __syncthreads();
97.   S1[ty * %(BLOCK_SIZE)s + tx] = salida_temp[0];
```

```
98.  __syncthreads();
99.  S2[ty * %(BLOCK_SIZE)s + tx] = salida_temp[1];
100. __syncthreads();
101. S3[ty * %(BLOCK_SIZE)s + tx] = salida_temp[2];
102. __syncthreads();
103. }
104. """
```

En este mismo orden de ideas, este fragmento de código tiene como propósito establecer un núcleo para el cálculo de nuevos componentes principales a partir de la matriz inversa de los vectores propios, el segundo y tercer componente principal calculados inicialmente y de la representación matricial de la imagen pancromática. Este núcleo también hace uso de los identificadores para cada uno de los hilos del bloque mediante ***threadIdx.x*** y ***threadIdx.y***. Adicionalmente, se hace uso de la función reservada ***syncthreads()*** para sincronizar todos los hilos que se estén ejecutando en cierto momento de la rutina, tal como se ha presentado en núcleos anteriores. Por último, se van llenando posición a posición por cada hilo las matrices resultantes, que en este caso hacen referencia a los tres nuevos componentes principales obtenidos. La función ***componentesPrincipalesPancromatica()*** es escrita en lenguaje C-CUDA y es almacenada en la variable global ***kernel_componentes_principales_pancromatica*** de Python. Dicha función, recibe como parámetros la matriz de la inversa de los vectores propios, el componente principal 2 y 3 y la imagen pancromática. Finalmente, recibe las matrices ***S1, S2*** y ***S3*** las cuáles serán las matrices de salida de esta operación.

Función para la división de una matriz en submatrices - Líneas 105 – 107:

```
105. def split(array, nrows, ncols):
106.         r, h = array.shape
107.         return (array.reshape(h//nrows,nrows,-1,ncols).swapaxes(1,2) .reshape(-
             1, nrows, ncols))
```

En estas líneas de código, se define la función ***split()***, la cual permite dividir una matriz cuadrada en submatrices de igual tamaño que cumplan su estructura N x N. Esta función recibe los siguientes parámetros:

- ***array:*** hace referencia a la matriz o arreglo bidimensional que se desea segmentar en submatrices.
- ***nrows:*** número de filas que deben tener las submatrices.
- ***ncols:*** número de columnas que deben tener las submatrices.

Función para calcular la matriz de varianza-covarianza - Líneas 108 – 118:

```
108.  def varianza_cov( R_s, G_s, B_s):
109.          kernel_code = kernel_var_cov % {'BLOCK_SIZE': BLOCK_SIZE}
110.          mod = compiler.SourceModule(kernel_code)
111.          covariance_kernel = mod.get_function("CovarianceKernel")
112.          salida_gpu = gpuarray.zeros((3, 3), np.float32)
113.          Rs_gpu = gpuarray.to_gpu(R_s)
114.          Gs_gpu = gpuarray.to_gpu(G_s)
115.          Bs_gpu = gpuarray.to_gpu(B_s)
116.          for i in range(len(R_s)):
117.                  covariance_kernel(Rs_gpu[i], Gs_gpu[i],
                      Bs_gpu[i],salida_gpu,block = (32, 32, 1)
118.          return salida_gpu.get()
```

Anteriormente, se ha creado un núcleo para calcular en GPU la matriz de varianza-covarianza de las bandas originales de la imagen multiespectral. Sin embargo, ese núcleo debe ser llamado mediante funciones de Python. Debido a esto, se define la función ***varianza_cov()***, la cual establece un tamaño del bloque de hilo que se va a ejecutar paralelamente en la GPU y compila el núcleo mediante ***compiler.SourceModule***. Asimismo, se obtiene el núcleo a través de la función ***get_function()*** propia del módulo compilado. Además de esto, se separa espacio en memoria para la matriz de salida, se hace transferencia de las submatrices de las bandas R,G,B a memoria global de la GPU y se calcula iterativamente la matriz de varianza-covarianza para cada submatriz de las bandas.

Función para el apilamiento de submatrices - Líneas 119 – 131:

```
119. def stack_values(list_cp, array_split, size, block_size):
120.         block_size = block_size
121.         valor_inicial = 0
122.         valor_final = 0
123.         list_cp_nueva = []
124.         factor_div = (size//block_size)
125.         factor_ite = len(array_split)//factor_div
126.         for i in range(factor_ite):
127.             valor_final = valor_final + factor_div
128.             list_cp_nueva.append(np.hstack(list_cp[valor_inicial:
                 valor_final]))
129.             valor_inicial = valor_inicial + factor_div
130.         cp_final = np.vstack(list_cp_nueva)
131.         return cp_final
```

En este caso, este fragmento de código establece la función ***stack_values()*** para poder apilar las submatrices resultantes de los procesos asociados al cálculo de la matriz de varianza-covarianza, los componentes principales haciendo uso de los vectores propios y con la imagen pancromática. Para llevar a cabo esto, se hace uso de funciones como ***hstack()*** y ***vstack()*** propias de la librería de ***numpy.***

Función para calcular componentes principales iniciales - Líneas 132 – 154:

```
132. def componentes_principales_original(r_s,g_s,b_s,q,size, block_size):
133. cp1_temp, cp2_temp,cp3_temp  = []
134. size = size
135. block_size = block_size
136. kernel_code = kernel_componentes_principales_original % { 'BLOCK_SIZE':
     BLOCK_SIZE }
137. mod = compiler.SourceModule(kernel_code)
138. kernel = mod.get_function("componentesPrincipalesOriginal")
139. s1_gpu = gpuarray.zeros((block_size,block_size),np.float32)
140. s2_gpu = gpuarray.zeros((block_size,block_size),np.float32)
141. s3_gpu = gpuarray.zeros((block_size,block_size),np.float32)
142. q_gpu = gpuarray.to_gpu(q)
143. Rs_gpu_t = gpuarray.to_gpu(r_s)
144. Gs_gpu_t = gpuarray.to_gpu(g_s)
145. Bs_gpu_t = gpuarray.to_gpu(b_s)
```

```
146.  for i in range(len(r_s)):
147.  kernel(Rs_gpu_t[i],Gs_gpu_t[i],Bs_gpu_t[i],q_gpu,
      s1_gpu,s2_gpu,s3_gpu,block=(block_size, block_size,1))
148.  cp1_temp.append(s1_gpu.get())
149.  cp2_temp.append(s2_gpu.get())
150.  cp3_temp.append(s3_gpu.get())
151.  cp1 = stack_values(cp1_temp, r_s, size, block_size)
152.  cp2 = stack_values(cp2_temp, r_s, size, block_size)
153.  cp3 = stack_values(cp3_temp, r_s, size, block_size)
154.  return cp1, cp2, cp3
```

Anteriormente, se ha creado un núcleo para realizar el cálculo de los componentes principales en GPU. Sin embargo, ese núcleo debe ser llamado mediante funciones de Python. Debido a esto, se define la función ***componentes_principales_original ()*** la cual establece un tamaño del bloque de 32x32 de hilos que se van a ejecutar paralelamente en la GPU y se compila el núcleo mediante ***compiler.SourceModule.*** Este número de hilos será definido posteriormente mediante un variable global. Asimismo, se obtiene el núcleo a través de la función ***get_function()*** propia del módulo compilado, pasando como parámetro el nombre de la función de C-CUDA *("componentesPrincipalesOriginal")*. Además de esto, se separa espacio en memoria para las submatrices de salida de cada componente principal. Además, se hace transferencia de las submatrices de las bandas R, G, B a memoria global de la GPU y se calcula iterativamente la submatrices que contienen los componentes principales. Estas submatrices se almacenan en diferentes listas de Python. Finalmente, mediante la función ***stack_values()*** definida anteriormente, se realiza el apilamiento de cada submatriz y así poder tener los tres componentes principales en su totalidad. Dichos componentes se consolidan en las variables ***cp1***, ***cp2*** y ***cp3.***

Función para calcular componentes principales finales - Líneas 155 – 177:

```
155.  def componentes_principales_panchromartic(r_s , g_s, b_s, q, size,
      block_size):
156.      block_size = block_size
157.      nb1_temp, nb2_temp, nb3_temp = []
158.      size = size
159.      kernel_code = kernel_componentes_principales_pancromatica % {
          'BLOCK_SIZE': BLOCK_SIZE }
160.      mod = compiler.SourceModule(kernel_code)
161.      kernel =
          mod.get_function("componentesPrincipalesPancromatica")
162.      s1_gpu = gpuarray.zeros((block_size,block_size),np.float32)
163.      s2_gpu = gpuarray.zeros((block_size,block_size),np.float32)
164.      s3_gpu = gpuarray.zeros((block_size,block_size),np.float32)
165.      Rs_gpu_t = gpuarray.to_gpu(r_s)
166.      Gs_gpu_t = gpuarray.to_gpu(g_s)
167.      Bs_gpu_t = gpuarray.to_gpu(b_s)
168.      q_gpu = gpuarray.to_gpu(q)
169.      for i in range(len(r_s)):
170.          kernel(Rs_gpu_t[i], Gs_gpu_t[i], Bs_gpu_t[i], q_gpu,
              s1_gpu, s2_gpu, s3_gpu, block = (block_size, block_size, 1))
171.          nb1_temp.append(s1_gpu.get())
172.          nb2_temp.append(s2_gpu.get())
173.          nb3_temp.append(s3_gpu.get())
174.      nb1 = stack_values(nb1_temp, g_s, size, block_size)
175.      nb2 = stack_values(nb2_temp, g_s, size, block_size)
176.      nb3 = stack_values(nb3_temp, g_s, size, block_size)
177.      return nb1, nb2, nb3
```

De igual manera, en fragmentos anteriores se ha definido un núcleo para realizar el cálculo de los componentes principales a partir de la imagen pancromática en GPU. Sin embargo, ese núcleo también debe ser llamado mediante funciones de Python. Debido a esto, se define la función ***componentes_principales_panchromatic()*** la cual establece un tamaño del bloque de 32x32 de hilos que se van a ejecutar paralelamente en la GPU y se compila el núcleo mediante ***compiler.SourceModule.*** Adicionalmente, se obtiene el núcleo a través de la función ***get_function()*** propia del módulo compilado, pasando como parámetro el nombre de la función de C-CUDA

("componentesPrincipalesPancromatica"). Además de esto, se separa espacio en memoria para las submatrices de salida de cada componente principal, se hace transferencia de la matriz inversa de los vectores propios y de las submatrices de los componentes principales iniciales 2 y 3 a memoria global de la GPU. Una vez se realiza esto, se calcula iterativamente la submatrices que contienen los nuevos componentes principales. Estas submatrices se almacenan en diferentes listas de Python. Finalmente, mediante la función ***stack_values()*** definida anteriormente, se realiza el apilamiento de cada submatriz, para consolidar los tres componentes principales en su totalidad. Dichos componentes se consolidan en las variables ***nb1***, ***nb2*** y ***nb3.***

Núcleo para restar de una matriz un valor constante - Líneas 178 – 181:

```
178.  substract = ElementwiseKernel(
179.              "float *x, float y, float *z",
180.              "z[i] = x[i]-y",
181.              "substract_value")
```

En estas líneas de código se utiliza la función ***ElementwiseKernel***, para poder establecer un núcleo simple, el cual va a tomar una matriz de entrada ***x*** junto con un valor flotante ***y***. Esto con el propósito de realizar en GPU, la resta posición a posición de la matriz ***x*** y el valor de ***y.*** Adicionalmente, el parámetro ***z*** tan solo es la matriz de salida de esta operación.

Núcleo para ajuste espectral - Líneas 182 – 185:

```
182.  negative_adjustment = ElementwiseKernel(
183.              "float *x, float *z",
184.              "if(x[i] < 0){z[i] = 0.0;}else{z[i] = x[i];}",
185.              "adjust_value")
```

De igual manera, en estas líneas de código, se establece un nuevo núcleo de tipo **ElementwiseKernel.** Esta función tiene como propósito realizar un ajuste de valores negativos. Por lo tanto, tomará una matriz y evaluará cada una de sus posiciones, si el valor de una posición específica resulta ser

negativa se convertirá a un valor de cero. Este núcleo escrito en C-CUDA, se almacena en la variable ***negative_adjustment*** para poder ser invocada posteriormente.

Función para obtener traza de potencias sucesivas - Líneas 186 – 193:

```
186.  def successive_powers(ortogonal_matrix):
187.          size_mat_ort = len(ortogonal_matrix)
188.          s = np.zeros((size_mat_ort,1))
189.          B = np.zeros((size_mat_ort,size_mat_ort))
190.          for i in range(1, (size_mat_ort+1)):
191.                  B=la.matrix_power(ortogonal_matrix,i)
192.                  s[i-1]=np.trace(B)
193.          return s
```

De acuerdo con este fragmento de código, lo que se busca es establecer una función de Python nombrada ***successive_powers()***, la cual encontrará la traza de potencias sucesivas a partir de una matriz proporcionada por parámetro.

Función para calcular coeficientes de un polinomio - Líneas 194 – 202:

```
194.    def polynomial_coefficients(polynomial_trace, ortogonal_ matrix):
195.          n_interations = len(ortogonal_matrix)
196.          polynomial = np.zeros((n_interations))
197.          polynomial[0] = -polynomial_trace[0]
198.          for i in range(1,n_interations):
199.                  polynomial[i]=-polynomial_trace[i]/(i+1)
200.                  for j in range(i):
201.                          polynomial[i]=polynomial[i]-(polynomial[j]*
                            (polynomial_trace[(i-j)-1])/(i+1))
202.          return polynomial
```

En este conjunto de líneas se pretende establecer una función que se ejecute en CPU, cuyo objetivo sea calcular los coeficientes del polinomio característico a partir de una matriz ortogonal y su respectiva traza polinómica. Para esto, se define la función ***polynomial_coefficients()***.

Función para normalizar vectores propios - Líneas 203 – 212:

```
203. def eigenvectors_norm(mat_eigenvalues, ortogonal_matrix,
     mat_eigenvectors):
204.     n = len(mat_eigenvalues)
205.     V = np.zeros((n,n))
206.     S = np.zeros((n,1))
207.     for i in range(n):
208.         B= ortogonal_matrix[1:n,1:n]-mat_eigenvalues[i,i]* np.eye(n-1)
209.         temp_s=la.lstsq(B,mat_eigenvectors,rcond=-1)[0].transpose()
210.         S=np.insert(temp_s,0,1);
211.         V[0:n,i]=S/la.norm(S)
212.     return V, V.transpose()
```

En estas líneas de código se define la función ***eigenvectors_norm()***. Esta función busca calcular los vectores propios normalizados. Lo anterior, recibiendo como parámetros la matriz ortogonal, la matriz de vectores propios y la matriz de valores propios. Donde, cada vector propio es una columna de la matriz ortogonal base. El retorno de esta función es un arreglo con los vectores propios normalizados y su respectiva transpuesta.

Lectura y carga de imágenes - Líneas 213 – 217:

```
213. multispectral = skimage.io.imread('multispectral.tiff', plugin='tifffile')
214. panchromatic = skimage.io.imread('panchromatic.tiff', plugin='tifffile')
215. size_rgb = multispectral.shape
216. BLOCK_SIZE = 32
217. n_bands = size_rgb[2]
```

Una vez se han definido las funciones presentadas a lo largo de esta implementación, se procede a invocarlas secuencialmente haciendo saltos entre memoria de CPU y GPU. Sin embargo, en estas líneas de código, se realiza la lectura de la imagen multiespectral y pancromática. Esto, mediante la función ***imread*** perteneciente al módulo ***io*** de la librería ***scikit-image.*** Esta función consolida las imágenes a un arreglo multidimensional de **numpy,** por lo tanto, quedan listas para ser utilizadas y manipuladas. Adicionalmente, se crea la variable ***size_rgb*** la cual almacena la dimensión

con sus respectivas bandas de la imagen multiespectral. Además de esto, se define la variable ***BLOCK_SIZE*** con un valor por defecto de 32. Este valor, nos ayudará a lo largo de la implementación a establecer el tamaño del bloque de hilos que se ejecutará en GPU. Por último, se extrae el número de bandas de las que se compone la imagen multiespectral. En este caso, al manejar un espacio de color RGB se debe obtener un total de 3 bandas.

Conversión de tipo de dato de las bandas - Líneas 218 – 222:

```
218.  m_host = multispectral.astype(np.float32)
219.  r_host = m_host[:,:,0].astype(np.float32)
220.  g_host = m_host[:,:,1].astype(np.float32)
221.  b_host = m_host[:,:,2].astype(np.float32)
222.  panchromatic_host = panchromatic.astype(np.float32)
```

Posteriormente, en este fragmento de código, mediante la función ***astype()***, se define que el tipo de datos de las matrices multiespectral y pancromática será flotante de 32 bits. Adicionalmente, se extraen las bandas R, G, B (Red, Blue, Green) de la imagen multiespectral a partir de la indexación de arreglo de ***numpy.***

Transferencia de variables a memoria global de GPU - Líneas 223 – 229:

```
223.  r_gpu = gpuarray.to_gpu(r_host)
224.  g_gpu = gpuarray.to_gpu(g_host)
225.  b_gpu = gpuarray.to_gpu(b_host)
226.  p_gpu = gpuarray.to_gpu(panchromatic_host)
227.  mean_r_gpu = misc.mean(r_gpu)
228.  mean_g_gpu = misc.mean(g_gpu)
229.  mean_b_gpu = misc.mean(b_gpu)
```

En las cuatro primeras líneas de este fragmento de código, se realiza la transferencia de cada una de las bandas extraídas anteriormente y de la imagen pancromática, a memoria global de GPU. Posteriormente, mediante la función ***mean()*** del módulo ***misc*** propio de la librería ***scikit-cuda***, se calcula el promedio de cada una de los arreglos que almacenan las

bandas en GPU. Estos promedios son esenciales para poder obtener la matriz de varianza-covarianza de la imagen multiespectral.

Resta de bandas y promedio en GPU - Líneas 230 – 235:

```
230.  r_gpu_subs = gpuarray.zeros_like(r_gpu,np.float32)
231.  g_gpu_subs = gpuarray.zeros_like(g_gpu,np.float32)
232.  b_gpu_subs = gpuarray.zeros_like(b_gpu,np.float32)
233.  substract( r_gpu, mean_r_gpu.get(), r_gpu_subs)
234.  substract( g_gpu, mean_g_gpu.get(), g_gpu_subs)
235.  substract( b_gpu, mean_b_gpu.get(), b_gpu_subs)
```

En estas líneas se realizan arreglos llenos de ceros mediante la función ***zeros_like*** del módulo ***gpuarray*** de Pycuda. Estos arreglos son de la misma dimensión que los que consolidan las bandas R, G, B en GPU. Después, se invoca la función ***substract()*** la cual realiza la resta entre cada una de las bandas y su respectivo promedio. Todo lo anterior se lleva a cabo en GPU.

División de bandas en submatrices - Líneas 236 – 238:

```
236.  r_subs_split = split(r_gpu_subs.get(),BLOCK_SIZE,BLOCK_SIZE)
237.  g_subs_split = split(g_gpu_subs.get(),BLOCK_SIZE,BLOCK_SIZE)
238.  b_subs_split = split(b_gpu_subs.get(),BLOCK_SIZE,BLOCK_SIZE)
```

Posteriormente ya en CPU, se realiza la división de las matrices resultado del fragmento de código anterior. El resultado de esta operación es un arreglo de arreglos con las submatrices de un tamaño de 32 x 32. Esto para cada una de las bandas (R, G, B).

Cálculo de la matriz de covarianza y derivados - Líneas 239 – 244:

```
239.  mat_var_cov = varianza_cov(r_subs_split,g_subs_split, b_subs_split)
240.   coefficient = 1.0/((size_rgb[0]*size_rgb[1])-1)
241.   ortogonal_matrix = mat_var_cov*coefficient
242.   polynomial_trace = successive_powers(ortogonal_matrix)
243.  characteristic_polynomial = polynomial_coefficients(polynomial_
      trace,ortogonal_matrix)
244.  characteristic_polynomial_roots = np.roots(np.insert(
      characteristic_polynomial,0,1))
```

Acto seguido, se hace uso de estas bandas divididas para poder calcular en GPU la matriz de varianza-covarianza mediante la función ***varianza_cov()***. En este orden de ideas, se toma cada submatriz de cada banda y se calcula su matriz de varianza-covarianza, así hasta recorrerlas completamente y al final poder realizar una concatenación de estas matrices. Después, se calcula el coeficiente requerido para poder diagonalizar ortogonalmente la matriz de varianza-covarianza. Además de esto, al multiplicar la matriz de varianza-covarianza con este coeficiente se obtiene lo que se llamará matriz ortogonal. Posterior a ello, pasando como parámetro esta matriz a la función ***successive_powers()*** se genera la traza de las potencias sucesivas de la matriz ortogonal. Es necesario calcular los coeficientes del polinomio característico a partir de la matriz ortogonal y de la traza polinómica. Lo anterior, invocando la función ***polynomial_coefficients()*** descrita con anterioridad. Por último, mediante la función ***roots*** de ***numpy*** se hallan las raíces reales del polinomio característico.

Procesamiento de valores y vectores propio - Líneas 245 – 253:

```
245.  eigenvalues_mat = np.diag(characteristic_polynomial_roots)
246.  eigenvectors_mat = -1*ortogonal_matrix[1:n_bands,0]
247.  mat_ortogonal_base, q_matrix = eigenvectors_norm
      (eigenvalues_mat,ortogonal_matrix, eigenvectors_mat)
248.  q_matrix_list = q_matrix.tolist()
249.  q_matrix_cpu = np.array(q_matrix_list).astype(np.float32)
250.  w1 = q_matrix_cpu[0,:]
251.  w2 = (-1)*q_matrix_cpu[1,:]
252.  w3 = q_matrix_cpu[2,:]
253.  eigenvectors = np.array((w1,w2,w3))
```

En estas líneas, se obtiene la matriz diagonal de las raíces del polinomio característico esto mediante la función ***diag()*** de ***numpy***. Lo anterior se realiza dado que en la diagonal de esta matriz se encuentran los valores propios. Una vez se han obtenido estos valores, se procede a calcular los vectores propios a partir de la matriz ortogonal. En este orden de ideas, se cuenta un vector propio para cada valor propio. Posteriormente, en el resto

de líneas se generan los vectores propios normalizados, donde cada columna de la matriz ***mat_ortogonal_base*** es un vector propio. Finalmente, estos vectores propios normalizados se almacenan en la variable ***eigenvectors***.

Cálculo de matriz inversa de vectores propios - Líneas 254 – 256:

```
254. inv_eigenvectors = la.inv(eigenvectors)
255. Inv_list = inv_eigenvectors.tolist()
256. inv_eigenvector_cpu = np.array(inv_list).astype(np.float32)
```

En este fragmento de código se obtiene la matriz inversa de los vectores propios normalizados. Esto, a través de la función ***inv*** propia del módulo ***linalg*** de Numpy. Posterior a esto, se convierte a una lista y se pasa a un arreglo de numpy en CPU especificando ***float32*** como el tipo de dato de este arreglo. Es decir, finalmente la variable ***inv_eigenvector_cpu*** almacena la matriz inversa de los vectores propios.

División de bandas para cálculo de componentes principales - Líneas 257 – 259:

```
257. r_subs_split_cp = split(r_host,BLOCK_SIZE,BLOCK_SIZE)
258. g_subs_split_cp = split(g_host,BLOCK_SIZE,BLOCK_SIZE)
259. b_subs_split_cp = split(b_host,BLOCK_SIZE,BLOCK_SIZE)
```

Se vuelve a realizar el proceso de división las bandas de la imagen multiespectral en submatrices de 32x32, que se consolidan en arreglos de arreglos de Numpy. Este proceso, se lleva a cabo mediante la función ***split()***, expuesta durante esta implementación.

Cálculo de componentes principales iniciales y finales - Líneas 260 – 264:

```
260. pc_1,pc_2,pc_3 = componentes_principales_original
     (r_subs_split_cp,g_subs_split_cp,b_subs_split_cp,q_matrix_cpu,r_host.shape
     [0], BLOCK_SIZE)
261. p_subs_split_nb = split(panchromatic_host,BLOCK_SIZE, BLOCK_SIZE)
262.  pc_2_subs_split_nb = split(pc_2,BLOCK_SIZE,BLOCK_SIZE)
263.  pc_3_subs_split_nb = split(pc_3,BLOCK_SIZE,BLOCK_SIZE)
```

```
264. nb1,nb2,nb3 = componentes_principales_panchromartic
     (p_subs_split_nb,pc_2_subs_split_nb,pc_3_subs_split_nb,inv_eigenvector_cp
     u,r_host.shape[0], BLOCK_SIZE)
```

En estas líneas las variables ***pc_1, pc_2*** y ***pc_3*** almacenan los componentes principales iniciales. Esto es posible al hacer uso de la función ***componentes_principales_original()***, la cual invoca iterativamente el núcleo *('componentesPrincipalesOriginal')* en GPU. En cada una de estas iteraciones utiliza las submatrices de las bandas R, G, B conjuntamente con la matriz de vectores propios. Posteriormente, se realiza la división de submatrices de la imagen pancromática y del segundo y tercer componente principal obtenido anteriormente (pc2 y pc3). Lo anterior se lleva a cabo con el propósito de poder calcular los nuevos componentes principales a partir de la imagen pancromática, los componentes principales 2 y 3 y la matriz inversa de los vectores propios. Estos nuevos componentes se almacenan en las variables ***nb1, nb2*** y ***nb3***.

Ajuste espectral de componentes principales finales - Líneas 265 – 276:

```
265. nb11 = nb1.astype(np.float32)
266. nb22 = nb2.astype(np.float32)
267. nb33 = nb3.astype(np.float32)
268. nb11_gpu = gpuarray.to_gpu(nb11)
269. nb22_gpu = gpuarray.to_gpu(nb22)
270. nb33_gpu = gpuarray.to_gpu(nb33)
271. nb111_gpu = gpuarray.empty_like(nb11_gpu)
272. nb222_gpu = gpuarray.empty_like(nb22_gpu)
273. nb333_gpu = gpuarray.empty_like(nb33_gpu)
274. negative_adjustment(nb11_gpu,nb111_gpu)
275. negative_adjustment(nb22_gpu,nb222_gpu)
276. negative_adjustment(nb33_gpu,nb333_gpu)
```

Una vez se han calculado los componentes principales finales (nb1, nb2 y nb3), es necesario convertirlos a un tipo de dato flotante de 32 bits para mantener uniformidad en los cálculos realizados. Así mismo, se hace transferencia de estos componentes a variables en memoria global de GPU.

Por último, se realiza el ajuste de valores negativos, donde en algunos casos por computo se generan valores negativos que deberían ser valores en cero. De acuerdo a esto, se invoca el núcleo ***negative_adjustment*** y se realiza dicho ajuste. De otra manera, sin realizar este ajuste se tendrían píxeles erróneos dado que deberían estar en una escala entre 0 y 255.

Generación de la nueva imagen - Líneas 277 – 281:

```
277.  nb111_cpu = nb111_gpu.get().astype(np.uint8)
278.  nb222_cpu = nb222_gpu.get().astype(np.uint8)
279.  nb333_cpu = nb333_gpu.get().astype(np.uint8)
280.  fusioned_image=np.stack((nb111_cpu,nb222_cpu,nb333_cpu), axis=2)
281.  skimage.io.imsave('pcagpu_image.tif',fusioned_image, plugin='tifffile')
```

Para finalizar esta implementación, se realiza el proceso de concatenación de los componentes principales ajustados mediante la función ***stack*** de ***numpy***. Por último, mediante la función ***imsave*** de ***skimage*** se guarda localmente la imagen generada a partir de la fusión de estas imágenes. La Figura 18C, presenta la imagen resultado al realizar la fusión de la imagen multiespectral (Figura 18A) y pancromática (Figura 18B), ambas con dimensión de 1024 píxeles por 1024 píxeles. Lo anterior, mediante el análisis de componentes principales.

A) B) C)

Figura 18. Imagen Fusionada de 1024x1024 píxeles mediante análisis de componentes principales.

Capítulo 4

Métodos basados en Transformadas Wavelet Discretas (TWD)

Los métodos que utilizan el análisis multirresolución, y fundamentalmente la Transformada Wavelet Discreta (TWD), son los más populares para disminuir la distorsión espectral. En particular, para lograr fusiones de alta calidad, diversos esquemas basados en wavelets han sido propuestos por varios investigadores (Hong y Zhang, 2008; Amolins, Zhang y Dare, 2007; Lillo- Saavedra y Gonzalo, 2006) destacando los algoritmos de Mallat y À trous, cuya principal diferencia se refiere al sentido en el que se realiza la estrategia multirresolución, pues en el primer caso se diezma la imagen mientras que para À trous no se aplica ningún diezmado, se ha demostrado que en los resultados con el algoritmo À trous las imágenes son de mejor calidad espacial y degradan en menor valor la riqueza espectral.

4.1 Principios básicos de la transformada Wavelet

Cualquier transformada que se aplica a una señal se hace con la finalidad de obtener información de ella, información que no está disponible en el dominio del tiempo. Cuando se gráfica una señal en el

dominio del tiempo, se obtiene una representación de la amplitud de la señal, ésta no es una buena representación para el procesamiento de una señal. La información que interesa se encuentra oculta en la frecuencia. El espectro en frecuencia muestra cuáles son las frecuencias que existen en la señal. La forma en la que se puede encontrar la frecuencia contenida en una señal es mediante la Transformada de Fourier (TF). Es decir, al obtener la TF de una señal en el dominio del tiempo, se consigue la representación de la señal en la frecuencia (Nieto y Orozco, 2008).

Este capítulo se presenta una corta explicación de la teoría básica del análisis Wavelet y una de sus aplicaciones en la reconstrucción de señales. Inicialmente se hace una comparación con el análisis de Fourier y se justifica la importancia y necesidad de utilizar la transformada Wavelet. Luego se presenta matemáticamente la transformada Wavelet Continua, se discretizan los parámetros de tiempo y frecuencia obteniendo la Transformada Wavelet Discreta, por último, se explica la forma como se puede descomponer y representar los planos Wavelet en una señal bidimensional mediante el algoritmo À trous para fusionar imágenes satelitales.

En el procesamiento de señales se pueden encontrar diferentes tipos de señales estacionarias y no estacionarias. Las primeras son localizadas en el tiempo ya que su frecuencia no varía, este tipo de ondas son estudiadas por medio del análisis de Fourier, que permite su descomposición en términos de sus componentes sinusoidales, es decir, transforma la señal de la base de tiempo a la base de frecuencia y de igual manera permite el paso del dominio de la frecuencia al dominio del tiempo, sin embargo, en este último se pierde información necesaria, que, por ser de carácter estacionario no resulta relevante. En el caso de las señales con comportamiento no-estacionario, es decir, aquellas cuya frecuencia varía en el tiempo, al tener la señal producto de la transformada de Fourier resulta imposible realizar el paso al dominio del tiempo porque no permite determinar en qué momento se presenta un cambio en la frecuencia.

Surge entonces la necesidad de contar con una representación localizada en el tiempo y la frecuencia, como consecuencia de la desventaja presentada por el análisis de Fourier. De esta manera Gabor implementa el uso de la STFT (Short Time Fourier Transform) (Upegui y Medina, 2019), que consiste en analizar una pequeña sección de la señal a través de una ventana de longitud fija, llevando la información contenida en este pequeño intervalo del dominio del tiempo a la escala bidimensional de tiempo y frecuencia, donde se puede conocer cuándo y a qué frecuencia ocurre un suceso.

Al utilizar la STFT se presenta una nueva deficiencia, el tamaño fijo de la ventana temporal que impide analizar pequeños detalles en señales de frecuencia variable. Es así como se introduce el análisis Wavelet como herramienta que permite obtener una representación, descomposición y reconstrucción de señales, que presenten cambios abruptos en sus componentes de tiempo-frecuencia en forma instantánea, a través del análisis de multirresolución con ventanas de longitud variable adaptadas al cambio de frecuencia de la señal. Es decir, esta técnica permite el uso de intervalos grandes de tiempo en aquellos segmentos en los que se requiere mayor precisión en baja frecuencia, e intervalos más pequeños donde se requiere información en alta frecuencia (ver Figura 19).

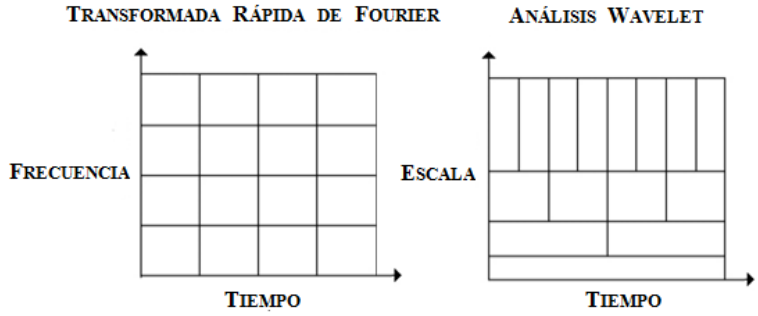


Figura 19. Comparación entre la STFT (tiempo-frecuencia) y el análisis Wavelet (tiempo-escala). Fuente: (Nieto & Orozco, 2008)

A diferencia de Fourier, en donde las funciones base son senos y cosenos de duración infinita, en el análisis Wavelet la base son funciones localizadas en frecuencia (dilatación) y en tiempo (traslación). Una Wavelet es una "pequeña onda" de duración limitada, es decir, su energía está concentrada en el tiempo alrededor de un punto, lo que proporciona una adecuada herramienta para el análisis de fenómenos transitorios, no estacionarios, variables en el tiempo y aquellos que presenten discontinuidades (ver Figura 20).

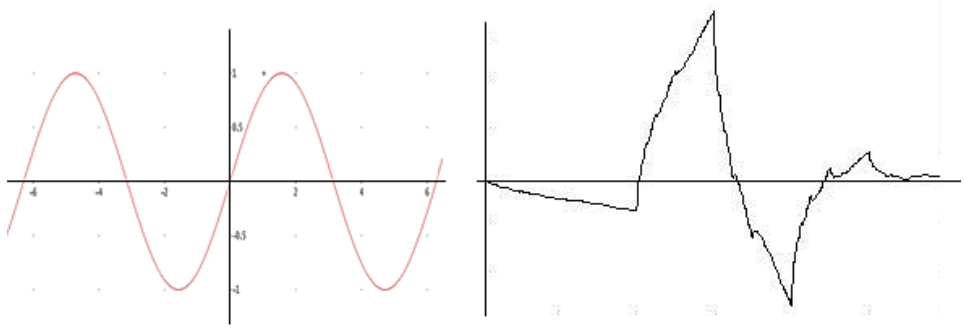


Figura 20. a) Señal seno. b) Wavelet Daubechies.

Transformada wavelet continua (CWT), permite el análisis de una señal en un segmento localizado de esta y consiste en expresar una señal continua como una expansión de términos o coeficientes del producto interno entre la señal y una Función Wavelet Madre $\psi(t)$. Una Wavelet Madre es una función localizada, perteneciente al espacio $L^2(R)$, que contiene todas las funciones con energía finita y funciones de cuadrado integrable definida.

$$f \in L^2 \Rightarrow \int |f(t)|^2 dt = E < \infty \qquad (6)$$

De esta manera se cuenta con una única ventana modulada y a partir de esta se genera una completa familia de funciones elementales mediante dilataciones o contracciones y traslaciones en el tiempo $\psi_{u,s}(t)$, denominados átomos wavelet o wavelet hijas que cumplen con todas las condiciones de la forma:

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right) \tag{7}$$

La Wavelet Madre debe cumplir con la condición de admisibilidad

$$C_\psi = \int_0^\infty \frac{\left|\overline{\psi(\omega)}\right|^2}{\omega}\, d\omega < \infty \tag{8}$$

Lo que quiere decir que la función $\psi(t)$ esta bien localizada en el tiempo, es decir, que la función oscile alrededor de un eje y su promedio sea cero, matemáticamente $\int_{-\infty}^{\infty} \psi(t)\, dt = 0$, y que la transformada de Fourier $\overline{\psi(t)}$ sea un filtro continuo pasa-banda, con rápido decrecimiento hacia el infinito y hacia $\omega = 0$ (Medina et al., 2004).

La transformada Wavelet de una función $f(t)$ a una escala s y una posición u, es calculada por la correlación de $f(t)$ con una $\psi_{u,s}(t)$ de la forma

$$CWTf(u,s) = \langle f, \psi_{u,s} \rangle = \int_{-\infty}^{\infty} f(t)\psi_{u,s}\,(t)dt \tag{9}$$

$$CWTf(u,s) = \int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right) dt \tag{10}$$

Para escalas pequeñas $(s < 1)$, con la CWT se obtiene información localizada en el dominio del tiempo de $f(t)$ y para escalas $(s > 1)$ la información de $f(\omega)$ se presenta localizada en el dominio de la frecuencia.

La transformada wavelet maneja un plano de tiempo-escala, pero también puede ser de tiempo-frecuencia, para esto se recurre al Teorema de Parseval y de esta manera es posible definir la transformada Wavelet en el dominio de la frecuencia ω.

$$CWTf(u,s) = \int_{-\infty}^{\infty} f(t)\sqrt{s}\overline{\psi}^*(s\omega)e^{jwu}\, d\omega \tag{11}$$

Para poder introducir el término de escala y frecuencia, es necesario ante todo definir una constante (c), que permite realizar un cambio de variable de una escala sa una frecuencia ω:

$$s \to \omega = \frac{c}{s} \tag{12}$$

Con este cambio de variable es posible observar que la CWT localiza de forma simultánea la señal $f(t)$ en el dominio del tiempo como su espectro $f(\omega)$ en el dominio de la frecuencia (Bracewell, 1978).

De igual manera es posible realizar una transformada Wavelet inversa, que permita reconstruir la señal a partir de la CWT (que preserva la energía de la señal) y las $\psi_{u,s}(t)$.

$$f(t) = C_\psi \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} CWTf(u,s)\psi_{u,s}(t) \frac{duds}{s^2} \tag{13}$$

4.2 Transformada Wavelet discreta (DWT)

Por la complejidad en el tratamiento numérico de la DWT, debido a la variabilidad en forma continua de los parámetros de escala como de traslación, es indispensable contar con una herramienta que permita la discretización de esta. Es así que se pasará de un mapeo continuo a un espectro o conjunto finito de valores, a través del cambio de la integral por una aproximación con sumatorias. La discretización permite representar una señal en términos de funciones elementales acompañadas de coeficientes.

$$f(t) = \sum_{\lambda} c_\lambda \phi_\lambda$$

En los sistemas Wavelet, las Wavelet madre $\psi(t)$ traen consigo unas funciones de escala $\varphi(t)$, las primeras son las encargadas de representar los detalles finos de la función mientras las funciones de escala realizan una aproximación. Es posible entonces representar una señal $f(t)$ como una sumatoria de funciones Wavelet y funciones de escala:

$$f(t) = \sum_k \sum_j c_{j,k}\varphi(t) + \sum_k \sum_j d_{j,k}\psi(t) \tag{14}$$

4.2.1 Función de escala y Función Wavelet

Una forma de discretizar los parámetros de escala y frecuencia es mediante un muestreo exponencial, para garantizar una mejor aproximación, con el cual se pueden redefinir los parámetros a valores discretos de la siguiente manera:

$$s = a^{-j}u = kna^{-j}$$

De esta manera y reemplazando en la ecuación (7), obtenemos la familia de funciones discretizadas que constituyen bases ortonormales de Wavelets en $L^2(R)$.

$$\psi_{u,s}(t) = \frac{1}{\sqrt{a^{-j}}}\psi\left(\frac{t - kna^{-j}}{a^{-j}}\right) = a^{\frac{j}{2}}\psi(a^j t - kn) \quad (15)$$

Para obtener una mejor aproximación de la señal en niveles de resolución muy finos, es necesario que las Wavelet sean dilatas por un factor de 2^{-j}, permitiendo tener una resolución de 2^j, estas funciones son denominadas Wavelets Diádicas.

$$\psi_{j,k}(t) = 2^{\frac{j}{2}}\psi(2^j t - kn) j, k \in Z \quad (16)$$

Teniendo en cuenta la ecuación (9) la transformada Discreta Wavelet tiene la forma:

$$CWTf(j, sk) = \langle f, \psi_{j,k} \rangle = \int_{-\infty}^{\infty} f(t)\psi_{j,k}(t)\, dt \quad (17)$$

$$DWTf(j, k) = \int_{-\infty}^{\infty} f(t)2^{\frac{j}{2}}\psi(2^j t - kn)\, dt \quad (18)$$

Teniendo en cuenta el anterior procedimiento es posible generar una familia de funciones de escala definidas:

$$\varphi_{j,k}(t) = 2^{\frac{j}{2}}\varphi(2^j t - kn) j, k \in Z \quad (19)$$

La representación general de la señal $f(t)$ será de la forma:

$$f(t) = \sum_k \sum_j c_{j,k} 2^{\frac{j}{2}} \varphi(2^j t - kn) + \sum_k \sum_j d_{j,k} 2^{\frac{j}{2}} \psi(2^j t - kn) \quad (20)$$

4.2.2 Coeficientes de escala $(c_{j,k})$ y Coeficientes Wavelet $(d_{j,k})$

Para representar una señal $f(t)$ y teniendo en cuenta la ecuación (20), es necesario encontrar los valores de los coeficientes $(c_{j,k})$ y $(d_{j,k})$ los cuales permiten finalmente hacer la aproximación de la señal. Estos son producto de una multiplicación vectorial entre la función $f(t)$ y la función de escala (φ) o wavelet (ψ). Para los coeficientes de escala tenemos:

$$c_{j,k} = \langle f(t), \varphi_{j,k}(t) \rangle = \int_{-\infty}^{\infty} |f(t)\varphi_{j,k}(t)| \, dt \quad (21)$$

$$\langle f(t), \varphi_{j,k}(t) \rangle = c_{j,-\infty} \langle \varphi_{j,-\infty}(t), \varphi_{(j,k)}(t) \rangle + \cdots + c_{j,k} \langle \varphi_{j,k}(t), \varphi_{(j,k)}(t) \rangle + \ldots c_{j,\infty} \langle \varphi_{j,\infty}(t), \varphi_{(j,k)}(t) \rangle \quad (22)$$

Ya que las funciones wavelet y de escala cumplen la propiedad de ortonormalidad, es posible asegurar que uno de los productos vectoriales sea diferente de cero, $\big(\langle \varphi_{j,k}(t), \varphi_{j,m}(t) \rangle = \delta(k-m)\big)$ o $\big(\langle \psi_{j,k}(t), \varphi_{j,m}(t) \rangle = \delta(k-m)\big)$ por lo tanto:

$$c_{j,k} = \langle f(t), \varphi_{j,m}(t) \rangle = \int_{t_1}^{t_2} f(t)\varphi_{j,k}(2^j t - k) \, dt \quad (23)$$

De igual manera para los coeficientes Wavelet:

$$d_{j,k} = \langle f(t), \psi_{j,m}(t) \rangle = \int_{t_1}^{t_2} f(t)\psi_{j,k}(2^j t - k) \, dt \quad (24)$$

4.2.3 Espacios vectoriales V_i y W_i

Las funciones de escala (φ) corresponden a la proyección ortogonal de $f(t)$ sobre un espacio $V_i \subset L^2(R)$. Dicho espacio agrupa todas las aproximaciones con resolución 2^{-j} y en él está contenida toda la información necesaria para realizar aproximaciones con menor resolución,

con lo que se puede afirmar que todos los espacios son versiones escaladas del espacio central V_0 (Espacios anidados).

$$\ldots \subset V_{-1} \subset V_0 \subset V_1 \subset \ldots \subset L^2$$
$$\forall j \in Z, f(t) \in V_j \Leftrightarrow f(2^j t) \in V_0 \quad (25)$$

Las funciones Wavelet ψ generan el espacio vectorial W_j (espacio de detalle) definido como el complemento ortogonal de V_j en V_{j-1}, donde

$$V_{j-1} = V_j \oplus W_j \quad (26)$$

Estos espacios presentan al igual que los espacios V_j , la propiedad de escalado, por lo cual:

$$\forall j \in Z, f(t) \in W_j \Leftrightarrow f(2^j t) \in W_0 \quad (27)$$

4.2.4 Aplicación de Transformada discreta de Wavelet para la fusión de imágenes

El mayor inconveniente de los métodos anteriores trabajados en esta investigación, es que modifican la información espectral de las bandas MS originales, lo que puede suponer un problema, por ejemplo, si las imágenes fusionadas resultantes se van a emplear para la obtención de información temática vía clasificación espectral.

El análisis multirresolución (MRA por sus siglas en inglés) se basa en la teoría según la cual el análisis de una imagen y la búsqueda de patrones son más eficientes si la imagen es analizada a diferentes niveles de resolución. El MRA permite descomponer datos bidimensionales en componentes de distinta frecuencia, para estudiar cada una de estas componentes a una resolución espacial acorde con su tamaño. De esta forma, en cada resolución la información de detalle (componentes de alta frecuencia) caracteriza distintas estructuras.

Este método se ha convertido en una herramienta de gran aplicación en el desarrollo de nuevos métodos de fusión. A lo largo de los años, se han

propuesto nuevos métodos de fusión empleando el MRA basado en las transformaciones Wavelet discretas (TWD), que permiten minimizar el problema anteriormente citado. La aproximación discreta de la transformada Wavelet puede realizarse a partir de distintos algoritmos. Dos de los más empleados en la fusión de imágenes de teledetección son los algoritmos de Mallat y À trous. Cada uno, con diferentes propiedades matemáticas conduce a distintas descomposiciones y, por lo tanto, a distintas imágenes fusionadas, dado las investigaciones realizadas los mejores resultados se obtienen usando el algoritmo À trous. (González-Audícana, et al., 2005).

4.3 Fusión de imágenes usando la Transformada Wavelet

En las últimas décadas, las estrategias de fusión de imágenes más utilizadas se han basado en técnicas de análisis multirresolución. El objetivo es encontrar una transformada discreta que mejore la respuesta espacial y que no degrade la resolución espectral, desde este punto de vista la transformada discreta de ondículas (Wavelet) (TDW) se puede considerar según los resultados de la evaluación de la fusión de imágenes, han demostrado que la fusión de imágenes satelitales usando la transformada de wavelet mejora la resolución espacial y degrada en menor valor la resolución espectral que los métodos tradicionales (Núñez et al., 1999).

La transformada discreta Wavelet, es una transformación lineal que tiene una gran utilidad en el área de procesamiento de señales. Una de sus principales aplicaciones consiste en separar un conjunto de datos en componentes de distinta frecuencia espacial representados en escalas comunes.

Los algoritmos de Mallat y el 'À trous' son los algoritmos de transformación wavelet discreta más empleados en el ámbito de la fusión de imágenes. Cada uno, con distintas propiedades matemáticas, conduce a distintas descomposiciones y, por lo tanto, a distintas imágenes fusionadas. A pesar

de que desde el punto de vista teórico el algoritmo 'Á trous' es menos adecuado que el de Mallat para extraer detalle espacial en el ámbito del análisis multirresolución, este ha permitido obtener imágenes con una calidad global sensiblemente mayor que el de Mallat (González-Audícana, 2003).

4.4 Análisis multirresolución y las Transformaciones Wavelet

El análisis multirresolución, basado en la teoría Wavelet permite descomponer datos bidimensionales en componentes de distinta frecuencia y estudiar cada componente a una resolución acorde con su tamaño. A diferente resolución, el detalle de una imagen (componentes de alta frecuencia) caracteriza distintas estructuras físicas de la escena (Mallat, 1989). A resoluciones groseras, este detalle corresponde a las estructuras o elementos de mayor tamaño mientras que a resoluciones finas este detalle corresponde a las estructuras de menor tamaño. Las transformaciones Wavelet permiten en el ámbito del análisis multirresolución, extraer el detalle espacial que se pierde al pasar de una resolución espacial a otra menor. La aproximación discreta de la transformada Wavelet puede realizarse a partir de distintos algoritmos. Uno de los más empleados en la fusión de imágenes es el algoritmo de 'À trous'.

4.4.1 Método À trous para la fusión de imágenes

Dutilleux en (1987), propuso el algoritmo de À trous basado en la transformada de ondículas (Wavelet) calculadas mediante el algoritmo de cavidades (À trous). En 1987 Dutilleux propuso el algoritmo de Wavelet À trous ("con hoyos"). Presenta una independencia en la direccionalidad del proceso de filtrado y por otro lado es redundante, en el sentido de que, entre dos niveles de degradación consecutivos, no existe una compresión espacial diádica de la imagen original, si no que se mantiene el tamaño de

dicha imagen. Si bien esto se traduce en un mayor coste computacional (Chibani y Houacine, 2003), ha mostrado que tanto la calidad espacial como espectral de las imágenes fusionadas mediante el algoritmo À trous es superior a la proporcionada por otros algoritmos. En este método de fusión existe una amplia gama de estrategias para integrar la información espacial contenida en la imagen pancromática (PAN), dentro de cada una de las bandas de la imagen multiespectral (MULTI), ninguna de estas estrategias permite controlar de una forma objetiva el compromiso entre la calidad espectral y espacial de las imágenes fusionadas. Con objeto de paliar la limitación descrita en el párrafo anterior, en esta investigación se presenta la fusión de imágenes mediante el algoritmo Wavelet À trous, que establece objetivamente el grado de compromiso entre la calidad espectral y espacial de la imagen resultante mediante curvas características. Estas curvas representan conjuntamente índices de calidad espacial y espectral.

4.4.2 Algoritmos de À trous

Dutilleux propuso el algoritmo basado en la transformada de ondículas calculada mediante el algoritmo de (con agujeros) À trous. En la Figura 21 es posible observar una representación del proceso de degradación de una imagen, utilizando un del algoritmo de tipo no decimado (TDWM). El detalle espacial que se pierde al pasar de un nivel al nivel consecutivo se obtiene directamente restando las imágenes aproximadas de dichos niveles.

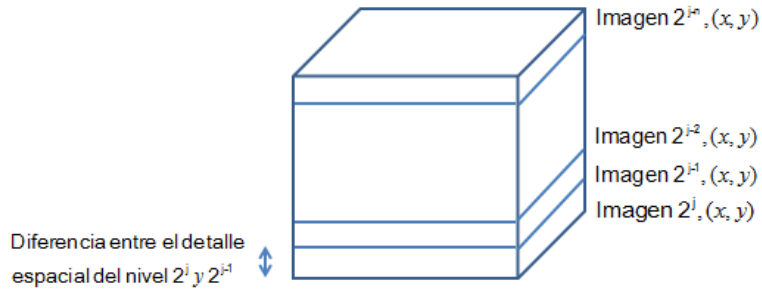


Figura 21. Algoritmo tipo decimado (TDWM). Fuente: (González-Audícana, 2004).

Diversos trabajos han demostrado que las transformadas de ondículas redundantes o no decimadas, proporcionan mejores resultados en determinadas aplicaciones de procesado de imágenes, como son eliminación de ruido (Mallat, 1996) o clasificación de texturas y más recientemente en el caso de la fusión de imágenes (Chibani, 2003) (Núñez, 1999).

Las aproximaciones discretas de la transformación Wavelet algoritmo 'À trous' (con agujeros) (Starck y Murtagh 1994), el esquema de descomposición de imágenes se representa con un paralelepípedo (Figura 7). La base de éste es también la imagen original A_{2^j} de resolución 2^j de C columnas y F filas. Cada nivel del paralelepípedo es una imagen aproximación de la imagen original. Conforme se asciende de nivel, las sucesivas aproximaciones presentan menor resolución, siendo ésta de 2^N en el nivel N del paralelepípedo ya que también en este caso el factor de degradación es diádico. Cada una de las imágenes aproximación se obtiene aplicando una función de escala. El detalle espacial que se pierde al pasar de la imagen A_{2^j} a $A_{2^{j-1}}$ se recoge en una única imagen de coeficientes wavelet, $w_{2^{j-1}}$, frecuentemente denominada plano wavelet y que se obtiene restando las imágenes original y aproximación. Cuando se aplica la transformación inversa, la imagen aproximación A_{2^j} puede reconstruirse sumando a la imagen aproximación $A_{2^{j-1}}$ el plano wavelet $w_{2^{j-1}}$., el algoritmo 'À trous' es invariante a la translación por lo que todas las imágenes aproximación y todos los planos wavelet resultantes de la descomposición tienen el mismo tamaño que la imagen original. La implementación práctica del algoritmo 'À trous' se realiza empleando un filtro bidimensional de paso bajo asociado a la función de escala, en este caso, una spline bi-cúbica. El algoritmo 'À trous', es no-ortogonal, lo implica que un determinado plano wavelet $w_{2^{j-1}}$ para una escala 2^{j-1}, puede retener información de la escala vecina 2^j.

El análisis multirresolución basado en la teoría de Wavelet, permite la presentación de los conceptos de detalle entre niveles sucesivos de escala o resolución. La descomposición de Wavelet es usada para la descomposición de imágenes. El método está basado en la descomposición de la imagen en múltiples canales basados en su frecuencia local. La transformación de la Wavelet provee un esquema para descomponer una imagen en un nuevo número de imágenes, cada una de ellas con un grado de resolución diferente.

4.5 Método de fusión usando el algoritmo de À trous

La transformada Wavelet Á trous para la fusión de imágenes satelitales permite generar mejores imágenes fusionadas gracias a la forma en que se obtienen los coeficientes resultantes de la transformación, obteniendo así los planos wavelet que tienen mayor información espacial y espectral de las imágenes originales.

4.5.1 Implementación de la Transformada Wavelet algoritmo de Á trous para la fusión de imágenes WorldView-2

Sintéticamente y como resultado de esta investigación se proponen los siguientes pasos para la implementación de la transformada Wavelet algoritmo de Á trous, generando dos planos Wavelet, para la fusión de imágenes satelitales (Upegui y Medina, 2019).

Paso 1. Registrar una composición a color RGB (verdadero color) de la imagen MS con la imagen PAN, usando el mismo tamaño de píxel de esta última. Transformar la imagen RGB en componentes HSV (Value, Tono y Saturación).

Paso 2. Ajustar la PAN a la componente Value (Pan-V), ajuste de histogramas. Aplicar el concepto de Transformada Wavelet algoritmo de Á trous al componente Pan-V, se resta Pan-V con la imagen resultante, de esta manera obteniendo el plano Wavelet w1, donde se almacena la información

espacial de Pan-V. se aplica Transformada Wavelet algoritmo de Á trous a la imagen resultante y al restarla con la anterior se obtiene el segundo plano Wavelet w2.

Paso 3. Generar una nueva componente Tono a partir de la suma de los planos Wavelet y la componente V, la matriz obtenida inmediatamente anterior para obtener la nueva componente Value (N-VAL), el cual corresponde N-Val=V+w1+w2.

Paso 4. Generar una nueva composición HSV (N-HSV), concatenando la N-VAL junto con las componentes originales H y S (obtenidas en el paso 1).

Paso 5. Realizar la transformación HSV a RGB, usando la nueva composición N-HSV. De esta manera se obtiene la nueva imagen multiespectral fusionada, que mantiene la resolución espectral ganando así la resolución espacial, (ver Figura 22).

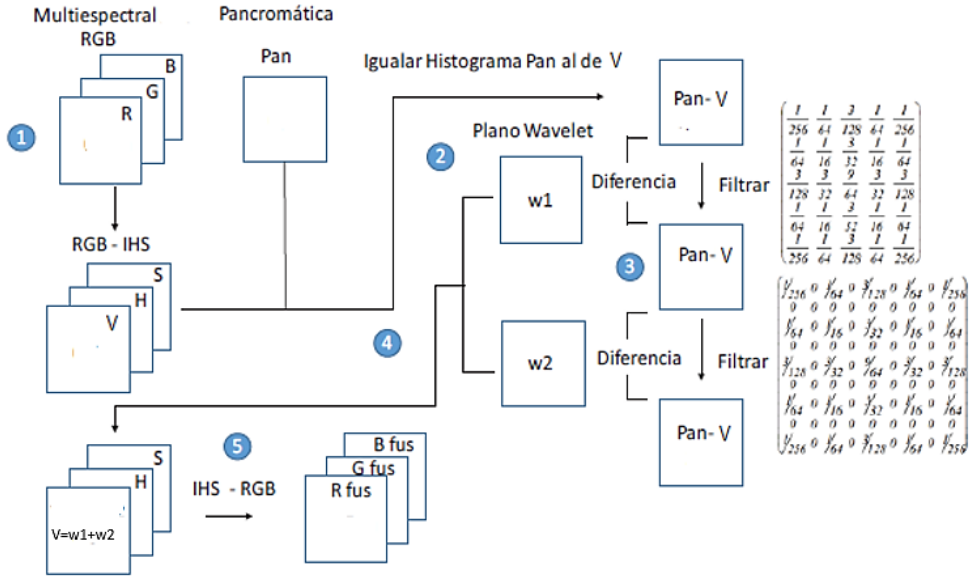


Figura 22. Diagrama del proceso de fusión de imágenes usando TWA.
Fuente: Adaptado de González-Audícana et al, 2005

De esta manera la transformada Wavelet Á trous implementada para la fusión de imágenes satelitales permite generar mejores imágenes fusionadas gracias a la forma en que se obtienen los planos Wavelet, estos planos Wavelet tienen mayor información espacial y espectral de las imágenes originales.

4.5.2 Modelo de procesamiento heterogéneo para la transformada Wavelet À trous

La Figura 23 presenta el modelo de procesamiento CPU/GPU usado para esta técnica, donde se inicia con la conversión de un espacio de color RGB a HSV. Después, se realiza el ajuste de la imagen pancromática a partir del histograma de *Value*, todo esto haciendo uso de la CPU. Acto seguido, se transfiere la pancromática ajustada a la memoria global de la GPU para realizar un proceso de filtrado. Este proceso de filtrado se obtiene al aplicar la operación de convolución entre la pancromática ajustada y el filtro *Bicubic Spline*. Así mismo, se repite este proceso, pero se utilizan la matriz resultante filtrada anteriormente y el filtro *Bicubic Spline* agregando columnas y filas en cero, todo esto en GPU. Posteriormente se obtienen los planos Wavelet a partir de la aplicación de estos filtros, para finalmente, generar la nueva componente Value a partir de la pancromática original y los dos planos wavelet obtenidos. Una vez se ha realizado esto, se hace un stack de las bandas originales de Hue y Saturation con la nueva Value. Por último, se realiza la conversión de HSV a RGB.

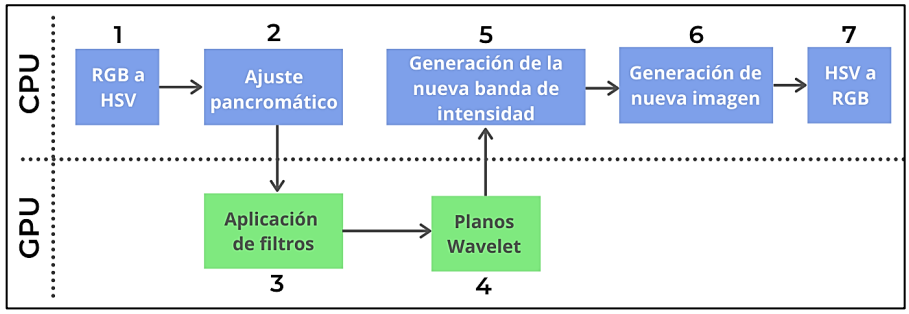


Figura 23. Modelo de procesamiento heterogéneo para la transformada Wavelet Á trous.

4.5.3 Implementación de la transformada Wavelet Á trous en Python

A continuación, se presentan fragmentos secuenciales de código en Python utilizados para poder llevar a cabo la fusión de imágenes satelitales mediante el método À Trous. En el repositorio del libro se encuentra el script completo con las imágenes de prueba: https://github.com/Parall-UD/libro_fusion_imagenes_satelitales_GPU.

Definición de dependencias - Líneas 1 – 9:

```
1. import skimage.io
2. from skimage.color import rgb2hsv, hsv2rgb
3. import numpy as np
4. import pycuda.autoinit
5. import pycuda.driver as drv
6. import pycuda.gpuarray as gpuarray
7. from pycuda.elementwise import ElementwiseKernel
8. import cupy as cp
9. from cupyx.scipy.ndimage import filters
```

De igual manera como en las implementaciones descritas a lo largo de este libro, lo primero que se debe realizar es la importación de librerías necesarias para la correcta ejecución del código. De acuerdo a esto, de nuevo se importan librerías como ***scikit-image***, ***numpy*** y ***pycuda***. Sin embargo, en esta ocasión se hace uso de un módulo extra de la librería ***scikit-image***, este módulo es ***color*** el cual nos proporciona funcionalidades para trabajar en distintos espacios de color. Adicionalmente, se importa ***cupy*** la cual es una librería matricial de código abierto acelerada mediante CUDA proporcionando computación acelerada por GPU con Python.

Núcleo para ajuste espectral - Líneas 10 – 13:

```
10. adjustment_values = ElementwiseKernel(
11.     "float *x, float *z",
12.     "if(x[i] < 0){z[i] = 0.0;}else{z[i] = x[i];}",
13.     "adjust_value")
```

En estas líneas de código, se establece un núcleo simple mediante la función ***ElementwiseKernel***. Este núcleo tiene como propósito tomar una matriz y evaluar cada uno de sus posiciones, si el valor de una posición dada resulta ser negativo se convertirá a un cero. Como se puede observar en estas líneas lo que se realiza es embeber código de C-CUDA en una variable netamente del lenguaje de Python.

Función para el ajuste de histogramas - Líneas 14 – 25:

```
14. def hist_match(source, template):
15.     oldshape = source.shape
16.     source = source.ravel()
17.     template = template.ravel()
18.     s_values,bin_idx,s_counts=np.unique(source, return_inverse=
                    True,return_counts=True)
19.     t_values, t_counts = np.unique(template, return_counts=True)
20.     s_quantiles = np.cumsum(s_counts).astype(np.float64)
21.     s_quantiles /= s_quantiles[-1]
22.     t_quantiles = np.cumsum(t_counts).astype(np.float64)
23.     t_quantiles /= t_quantiles[-1]
24.     interp_t_values = np.interp(s_quantiles, t_quantiles, t_values)
25.     return interp_t_values[bin_idx].reshape(oldshape)
```

En este fragmento de código, se define la función nombrada ***hist_match()*** la cual tiene como objetivo realizar un ajuste de histogramas entre dos imágenes mediante su representación matricial. En este proceso, como primera instancia se obtienen el conjunto de valores de píxeles únicos y sus índices, con su respectivo recuento. Acto seguido, se aplica la función de numpy ***cumsum*** a los recuentos y así poder realizar un proceso de normalización, haciendo uso del número de píxeles para obtener las funciones empíricas de distribución acumulativa para las imágenes denominada ***source*** y ***template***. Finalmente, se realiza una interpolación lineal para encontrar los valores de píxeles en la imagen ***template*** que se correspondan más con los cuartiles en la imagen ***source.***

Lectura y carga de imágenes - Líneas 26 – 27:

```
26. multispectral=skimage.io.imread('multispectral.tiff',plugin = 'tifffile')
27. panchromatic = skimage.io.imread('panchromatic.tiff',plugin = 'tifffile')
```

En estas líneas de código, se realiza la lectura de la imagen multiespectral y pancromática, esto, mediante la función ***imread*** perteneciente al módulo ***io*** de la librería ***scikit-image.*** Esta función convierte las imágenes que se desean leer, a un arreglo multidimensional de **numpy**; esto, con el propósito de poder ser utilizadas y manejadas mediante su representación matricial.

Conversión de espacio de color RGB a HSV - Líneas 28 – 32:

```
28. hsv = rgb2hsv(multispectral)
29. val = hsv[:,:,2]
30. sat = hsv[:,:,1]
31. mat = hsv[:,:,0]
32. pani = hist_match(panchromatic,val)
```

A partir de estas líneas, se realiza la conversión de la imagen multiespectral de un espacio de color RGB a Hue Saturation Value (HSV), esto mediante la función ***rgb2hsv()*** de la librería **scikit-image.** Acto seguido, se realiza la separación de bandas como valor, saturación y matiz. Lo anterior, haciendo uso de indexación de matrices de ***numpy.*** Por último, utilizando la función ***hist_match()*** se realiza el ajuste de histogramas entre la imagen pancromática y la banda de valor, extraída previamente almacenando su resultado en la variable ***pani***. Tanto la separación de bandas, como el ajuste de histogramas se realiza sobre la CPU.

Filtrado con Bicubi Spline - Líneas 33 – 36:

```
33. s = np.array([[1/256,1/64,3/128,1/64,1/256],[1/64,1/16,
        3/32,1/16,1/64],[3/128,3/32,9/64,3/32,3/128],[1/64,1/16,3/32,1/16,1/64],[1/256,1/64,
        3/128,1/64,1/256]])
34. s_gpu = cp.array(s)
35. p_gpu = cp.array(pani)
36. l1_gpu = filters.correlate(p_gpu, s_gpu, mode='constant')
```

Posteriormente, en estas líneas se crea la variable ***s***, la cual almacena un arreglo de ***numpy***, dicho arreglo representa el filtro ***Bicubic Spline.*** Después se transfiere a memoria global de la GPU este filtro junto con la pancromática ajustada. Lo anterior, mediante la función ***array()*** de la librería ***cupy***. Una vez se tiene estas variables en la GPU, se procede a aplicar un proceso de filtrado, al aplicar la operación de convolución entre la pancromática ajustada y el filtro *Bicubic Spline***.** Este filtrado se realiza mediante la función ***Correlate()*** propia del módulo ***filters*** de ***cupyx***, y se almacena en la variable ***I1_gpu.***

Filtrado con Bicubic Spline modificado - Líneas 37 – 39:

```
37. s1 = np.array([[1/256, 0, 1/64, 0, 3/128, 0, 1/64, 0, 1/256],[0, 0, 0, 0, 0, 0, 0, 0, 0],[1/64,
        0, 1/16, 0, 3/32, 0, 1/16, 0, 1/64],[0, 0, 0, 0, 0, 0, 0, 0, 0], [3/128, 0, 3/32, 0, 9/64, 0,
        3/32, 0, 3/128],[0, 0, 0, 0, 0, 0, 0, 0, 0],[1/64, 0, 1/16, 0, 3/32, 0, 1/16, 0, 1/64], [0,
        0, 0, 0, 0, 0, 0, 0, 0],[1/256, 0, 1/64, 0, 3/128, 0, 1/64, 0, 1/256]])
38. s1_gpu = cp.array(s1)
39. I2_gpu = filters.correlate(I1_gpu, s1_gpu, mode='constant')
```

Asimismo, se repite de nuevo el proceso anterior, pero se establece un nuevo filtro el cual es *Bicubic Spline* agregando columnas y filas en cero. Este filtro es almacenado en la variable ***s1*** en CPU. Acto seguido, se hace la transferencia de este nuevo filtro a memoria global de GPU mediante la librería ***cupy***. Una vez se ha realizado esto, se procede a utilizar la matriz guardada en la variable ***I1_gpu*** y ***s1_gpu*** para llevar a cabo el proceso de filtrado mediante convolución. Lo anterior, siendo ejecutado sobre la GPU.

Generación del primer plano Wavelet - Líneas 40 – 44:

```
40. W1=(pani-I1_gpu.get())
41. W1_gpu = gpuarray.to_gpu(W1)
42. W1_gpu_new = gpuarray.empty_like(W1_gpu)
43. adjustment_values(W1_gpu,W1_gpu_new)
44. W1 = W1_gpu_new.get().astype(np.uint8)
```

En estas líneas de código, se realiza el proceso para obtener el primer plano Wavelet. Este proceso consiste en tomar la pancromática ajustada y restarle en CPU la matriz consolidada en la variable ***I1_gpu***. Posteriormente, se realiza un salto a GPU del resultado de la resta anterior, para poder realizar de forma más rápida el ajuste de valores negativos, mediante el núcleo generado al inicio de esta implementación usando ***ElementwiseKernel.*** Por último, se realiza la transferencia a memoria de CPU del plano Wavelet siendo almacenado en la variable ***W1.***

Generación del segundo plano Wavelet - Líneas 45 – 49:

```
45. W2=(I1_gpu.get()-I2_gpu.get())
46. W2_gpu = gpuarray.to_gpu(W2)
47. W2_gpu_new = gpuarray.empty_like(W2_gpu)
48. adjustment_values(W2_gpu,W2_gpu_new)
49. W2 = W2_gpu_new.get().astype(np.uint8)
```

En este mismo orden de ideas, se debe obtener un segundo plano Wavelet. Sin embargo, aunque en este caso el proceso es el mismo, las variables utilizadas en este no lo son. Como primera instancia, se debe realiza la resta entre la variable ***I1_gpu*** y ***I2_gpu*** utilizando la función ***get()*** para ejecutar esta operación en CPU. Acto seguido, se transfiere a memoria global de GPU la matriz resultado de esta resta para realizar su ajuste de valores negativos. Para finalizar se trae a memoria de CPU dicha variable y se realiza su conversión a enteros de 8 bits. Finalmente, la variable que contiene el segundo plano Wavelet es nombrada ***W2***.

Generación del nuevo componente de intensidad - Líneas 50 – 51:

```
50. nint=(panchromatic+W1+W2).astype(np.uint8)
51. n_hsv = np.stack((mat, sat, nint),axis=2)
```

A partir de estas líneas, se genera un nuevo componente de intensidad al realizar la suma de la representación matricial de la imagen pancromática y los planos Wavelt ***W1*** y ***W2*** generados previamente. Después, se realiza el

proceso de concatenación de las bandas de matiz, saturación y el nuevo componente de intensidad mediante la función ***stack*** de ***numpy*** .

Generación de la nueva imagen - Líneas 52 – 53:

```
52. fusioned_image = hsv2rgb(n_hsv).astype(np.uint8)
53. skimage.io.imsave('atrousgpu_image.tif',fusioned_image, plugin = 'tifffile')
```

Sin embargo, al realizar la concatenación de estas bandas se sigue manteniendo el espacio de color HSV, pero se requiere realizar la conversión al espacio de color RGB. Debido a esto, mediante la función ***hsv2rgb()*** se realiza este proceso y se almacena en la variable ***fusioned_image***. Por último, mediante la función ***imsave*** de ***skimage*** se guarda localmente la imagen generada a partir de la fusión de estas imágenes. La Figura 24C presenta la imagen resultado al realizar la fusión de la imagen multiespectral (Figura 24A) y pancromática (Figura 24B), ambas con dimensión de 1024 píxeles por 1024 píxeles. Lo anterior, mediante la transformada Wavelet À trous.

A) B) C)

Figura 24. Imagen fusionada de 1024x1024 píxeles mediante Transformada Wavelet algoritmo de Á trous.

Capítulo 5

Índices de evaluación de la calidad espacial y espectral de las imágenes fusionadas

El procesamiento de imágenes es una herramienta muy útil en muchos campos de las ciencias modernas, una de los procesos corresponde a la fusión de imágenes satelitales, el resultado de estos algoritmos matemáticos son imágenes, las cuales deben ser evaluadas para su interpretación. Las imágenes fusionadas a menudo deben correlacionarse con la imagen original para garantizar que la imagen resultante cumple algún propósito en específico, para la evaluación de la calidad de estas imágenes fusionadas se utilizan los siguientes índices: coeficiente de correlación, entropía, DIV, Bias, ERGAS, RASE, RMES, Qu, los cuales son muy útiles para decidir qué imagen fusionada degrada en menor valor la riqueza espectral con una ganancia significativa espacialmente.

5.1 Bias

Se basa en la división de los valores medios de la imagen procesada y original. El ideal teórico del valor de sesgo es 0. Un pequeño valor positivo

o negativo de sesgo significa una fuerte similitud entre x e y. (Vaiopoulos, 2011).

$$Bias = 1 - \frac{\bar{x}}{\bar{y}} \tag{28}$$

5.2 DIV (Difference In Variance)

DIV (diferencia en varianza): representa la varianza de la imagen procesada dividida por la varianza de la imagen original sustraído por uno. Los valores de interpretación son similares al sesgo (Vaiopoulos, 2011).

$$Div = 1 - \frac{\sigma_y^2}{\sigma_x^2} \tag{29}$$

5.3 Entropía

Imagen Entropía (*E*): este índice refleja la cantidad de información incluida en una determinada imagen. La entropía requiere análisis de histograma: *p* es el porcentaje de píxeles cuyo valor cae en una determinada clase *bin*, mientras que *bc* es el número total de clases *bin* (Vaiopoulos, 2011).

$$E = -\sum_{K=1}^{bc} p \cdot log_2^{(p)} \tag{30}$$

5.4 Coeficiente de correlación (corr)

La correlación entre las diferentes bandas de las imágenes fusionadas y las bandas de la imagen original se pueden calcular con la siguiente ecuación:

$$corr(A/B) = \frac{\sum_{j=1}^{npix}\left(A_j - \overline{A}\right)\left(B_J - \overline{B}\right)}{\sqrt{\sum_{j=1}^{npix}\left(A_j - \overline{A}\right)\sum_{j=1}^{npix}\left(B_j - \overline{B}\right)}} \tag{31}$$

Donde $\overline{A}$ y $\overline{B}$ son los valores de la media de las imágenes correspondientes, $corr(A/B)$ se llama coeficiente de correlación y varía entre –1 y +1. Se usan los signos + y – para las correlaciones positivas y negativas,

respectivamente. Nótese que $corr(A/B)$ es una cantidad adimensional, es decir no depende de las unidades empleadas. El valor ideal de la correlación, tanto espectral como espacial es 1.

5.5 Índice ERGAS

La evaluación de la calidad de las imágenes fusionadas se puede llevar a cabo mediante los índices ERGAS espectral y espacial. La definición de ERGAS espectral (del francés *Erreur Relative Globale Adimensionallede Synthèse*) (Wald, 2002; Ranchin et al., 2003) viene dada por la ecuación 32:

$$ERGAS_{Espectral} = 100\frac{h}{l}\sqrt{\frac{1}{N_{Bandas}}\sum_{i=1}^{N_{Bandas}}\left[\frac{(RMSE_{Espectral}(Banda^i))^2}{(MULTI^i)^2}\right]} \quad (32)$$

Donde h y l representan la resolución espacial de las imágenes PAN y $MULTI$; NBandas es el número de bandas de la imagen fusionada; $MULTI^i$ es el valor de la radiancia de la banda $i-ésima$de imagen MULTI (Wald, 2000) y RMSE será definida como sigue (33):

$$RMSE_{Espectral}(Banda^i) = \frac{1}{NP}\sqrt{\sum_{i=1}^{NP}(MULTI^i(j) - FUS^i(j))^2} \quad (33)$$

Siendo NP el número de píxeles de la imagen $FUS^i(x,y)$. Adicionalmente, Lillo y su equipo (2005) proponen otro índice, denominado $ERGAS_{Espacial}$ que está inspirado en el índice ERGAS espectral (Lillo-Saavedra et al., 2005). El objetivo del índice $ERGAS_{Espacial}$ es evaluar la calidad espacial de las imágenes fusionadas por lo que se define como (34):

$$ERGAS_{Espacial} = 100\frac{h}{l}\sqrt{\frac{1}{N_{Bandas}}\sum_{i=1}^{N_{Bandas}}\left[\frac{(RMSE_{Espacial}(Banda^i))^2}{(PAN^i)^2}\right]} \quad (34)$$

Donde $RMSE_{Espacial}$ es definido como sigue en la ecuación 35:

$$RMSE_{Espacial}(Banda^i) = \frac{1}{NP}\sqrt{\sum_{i=1}^{NP}(PAN^i(j) - FUS^i(j))^2} \quad (35)$$

Los mejores resultados de estos índices (ERGAS espacial y espectral) se obtienen cuando es más cercano a cero.

5.6 Índice RASE

El índice RASE se expresa como un porcentaje (ecuación 36):

$$RASE = 100\frac{h}{l}\sqrt{\frac{1}{N}\sum_{i=1}^{n}\left[\frac{(RMSE(B_i))^2}{M_i^2}\right]} \quad (36)$$

Donde h es la resolución de la imagen de alta resolución especial (PAN) y l es la resolución de la imagen de baja resolución espacial (MULTI) (Wald, 2000). Los mejores resultados se obtienen cuanto el porcentaje está más cerca a cero.

5.7 Índice de calidad universal Qu

Este modelo de índice de calidad identifica cualquier distorsión como una combinación de tres factores: pérdida de correlación, distorsión de luminancia y contraste de distorsión (Wang & Bovink, 2002). El índice se obtiene con la ecuación 37.

$$Qu = \frac{\sigma_{xy}}{\sigma_x\sigma_y}\cdot\frac{2\bar{x}\bar{y}}{(\bar{x})^2+(\bar{y})^2}\cdot\frac{2\sigma_x\sigma_y}{\sigma_x^2+\sigma_x^2} \quad (37)$$

Los mejores valores de este índice se obtienen cuando el valor es más cercano a uno.

5.8 Índice RMSE

RMSE (Root Mean Squared Error): quizás uno de los índices más populares y comúnmente utilizados. Es la raíz de la diferencia al cuadrado de dos conjuntos de datos (x, y) divididos por el número de elementos (o píxeles) n:

$$RMSE = \sqrt{\frac{\sum_{i1}^{n}(x_i-y_i)^2}{n}} \quad (38)$$

Capítulo 6

Resultados y análisis

En este capítulo se describe la metodología utilizada para llevar a cabo la evaluación tanto a nivel de tiempo de procesamiento (homogéneo vs heterogéneo) como a nivel de calidad de la imagen fusionada. Posterior a esto se presentan y analizan los resultados de dicha evaluación.

6.1 Metodología de la evaluación

Esta sección tiene como finalidad presentar la metodología que se tuvo en cuenta para realizar la evaluación de los modelos propuestos y sus implementaciones. Los aspectos de evaluación a tratar en esta sección son: el entorno de computación, las imágenes utilizadas y los criterios a analizar.

6.1.1 Entorno computacional

La Tabla 1, presenta las características del entorno de computación utilizado para llevar a cabo la evaluación de la librería Sallfus. Este entorno de computación fue acondicionado con la instalación de paquetes como *Scipy, Numpy, Pycuda* y *Cupy*.

Tabla 1. Entorno computacional.

Sigla	Procesador	GPU	Memoria
EC	Intel (R) Xeon (R) CPU E-52697 v3 @ 2.60GHZ	NVIDIA Tesla k80	128GB

6.1.2 Imágenes de prueba

Para realizar la evaluación se tomaron un total de cuatro pares de imágenes, es decir cada par es compuesto por su respectiva imagen multiespectral y pancromática. Estos pares de imágenes tienen distintos tamaños los cuales son: 1024x1024, 2048x2048, 4096x4096 y 8192x8192 píxeles. Además, las imágenes de 1024 y 2048 píxeles son subescenas de una imagen IKONOS y las otras dos imágenes fueron tomadas mediante el satélite Landsat. La Figura 25(A) presenta la imagen multiespectral de 2048x2048 píxeles y la Figura 25(B) la imagen pancromática.

A) B)

Figura 25. Imagen de prueba con tamaño 20148x2048 pixeles.
A) Multiespectral, B) Pancromática.

6.1.3 Proceso de evaluación y métricas

Este proceso de evaluación está orientado a probar cada uno de los métodos de fusión de imágenes satelitales con los distintos pares de imágenes presentados anteriormente. Esto con el propósito de calcular los tiempos de ejecución de cada método sobre la arquitectura homogénea y

heterogénea, para obtener el factor de aceleración o *speed-up*. Adicionalmente, al aplicar el proceso de fusión se determinará la calidad de la imagen, a partir de los índices matemático-estadísticos expuestos en el capítulo 5. Este proceso se realizará tanto a nivel espectral como a nivel espacial. Esto quiere decir que, se tomará la imagen fusionada y se obtendrán sus índices teniendo como referencia las imágenes de entrada: multiespectral y pancromática. Para esta evaluación se usa un script en lenguaje Matlab, que calcula automáticamente ocho índices (Vaiopoulos, 2011).

6.2 Tiempos de ejecución y factores de aceleración

La Tabla 2, presenta el tiempo de ejecución para cada una de las técnicas de fusión implementadas. Se realiza una discriminación por tamaño y tipo de arquitectura implementada.

Con base en la Tabla 2, se puede observar que para cada una de las técnicas implementadas tanto secuencial como paralelamente, a medida que incrementa el tamaño de la imagen aumenta su tiempo de ejecución. Sin embargo, para las técnicas de fusión que utilizan exclusivamente la CPU, se presentan incrementos de tiempo mucho más significativos que los presentados en las implementaciones en CPU/GPU. La Tabla 3 presenta la tasa de crecimiento en segundos por píxel de cada una de las técnicas. Esto se realizó, mediante una linealización para obtener la pendiente que representa la tasa de crecimiento del tiempo de ejecución en función de los píxeles. Lo anterior, dado que algunas de las técnicas presentan un comportamiento exponencial y otras aproximadamente lineal. Analizando esta tabla se evidencia que las tasas de crecimiento disminuyen sustancialmente al utilizar CPU/GPU (mucho más significativo para Brovey y Multiplicative), lo que indica que a mayor tamaño de las imágenes se sacará mayor beneficio de la plataforma heterogénea y se obtendrá mayor aceleración.

Tabla 2. Tiempo de ejecución.

Método	Arq.	Tiempo por tamaño			
		1024x 1024px	**2048x 2048px**	**4096x 4096px**	**8192x 8192px**
Brovey	CPU	25.39s	76.76s	311.85s	1437.85s
	CPU/GPU	1.43s	1.49s	1.72s	2.70s
Multiplicative	CPU	9.27s	36.66s	136.37s	534.57s
	CPU/GPU	0.98s	1.02s	1.22s	1.90s
PCA	CPU	23.29s	86.32s	342.03s	1360.40s
	CPU/GPU	3.05s	7.53s	24.36s	74.35s
À trous	CPU	1.94s	7.40s	30.86s	142.62s
	CPU/GPU	1.08s	2.11s	5.84s	22.93s

Tabla 3. Tasa de crecimiento del tiempo de ejecución por píxel.

Método	Tasa por arquitectura	
	CPU	**CPU/GPU**
Brovey	$2.13x10^{-5}$ s/píxel	$1.93x10^{-8}$ s/píxel
Multiplicative	$7.95x10^{-6}$ s/píxel	$1.39x10^{-8}$ s/píxel
PCA	$2.02x10^{-5}$ s/píxel	$1.08x10^{-6}$ s/píxel
À trous	$2.12x10^{-6}$ s/píxel	$3.30x10^{-7}$ s/píxel

A partir de la Tabla 4 y teniendo en cuenta el tamaño más alto de imagen, que en este caso corresponde a 8192x8192 píxeles, se puede observar que, la técnica que presenta la mayor aceleración en CPU/GPU respecto a CPU es Brovey con un total de 531.85x. Después se ubica Multiplicative con un total de 281.06x, posteriormente, se encuentra PCA con 18.30x y por último está À trous, evidenciando solo 6.22x.

Tabla 4. Speed-up.

Método	Speed- up por tamaño			
	1024x 1024px	**2048x 2048px**	**4096x 4096px**	**8192x 8192px**
Brovey	17,80x	51,47x	180,83x	531,85x
Multiplicative	9,44x	35,95x	112,02x	281,06x
PCA	7,63x	11,47x	14,04x	18,30x
À trous	1,79x	3,51x	5,29x	6,22x

6.3 Calidad de la imagen fusionada

En la tabla 5 se realiza el análisis espectral de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Ikonos de tamaño 1024x1024, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con el método de multiplicación (promedio de 82.6% de dependencia lineal), sin embargo con los 7 índices BIAS, DIV, Entropía, ERGAS, Qu,RASE y RMSE los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 5. Análisis Espectral imagen Ikonos 1024 líneas por 1024 columnas.

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 5.99	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.67	0.53	0.46	0.37	0.59	5.59	11.8	0.67	46.8	26.9
RGB/Multi	0.84	0.83	0.81	0.76	0.69	4.47	19.9	0.56	79.6	45.1
RGB/PCA	0.58	0.56	0.58	0.41	0.64	5.45	12.5	0.50	49.7	27.9
RGB/À trous	0.69	0.56	0.51	0.11	0.21	5.86	7.61	0.58	30.52	17.2

En la Tabla 6 se realiza el análisis espacial de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Ikonos de tamaño 1024x1024 con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con el método de multiplicación (promedio de 97% de dependencia lineal) y con la transformada Wavelet con el algoritmo À trous (promedio de 96.3% de dependencia lineal). Cuando se analizan los 7 índices: BIAS, DIV, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 6. Análisis Espacial Ikonos 1024 líneas por 1024 columna.s

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 5.92	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.96	0.98	0.97	0.38	0.56	5.59	10.12	0.76	40.4	23.4
RGB/Multi	0.83	0.85	0.84	0.77	0.68	4.47	19.93	0.31	79.7	46.2
RGB/ PCA	0.94	0.97	0.91	0.42	0.63	5.54	11.32	0.70	45.2	26.2
RGB/À trous	0.95	0.98	0.96	0.13	0.18	5.86	4.11	0.94	16.3	9.53

En la tabla 7 se realiza el análisis espectral, de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Ikonos de tamaño 2048x2048, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con el método de multiplicación (promedio de 82.3% de dependencia lineal) y la transformada Wavelet con el algoritmo À trous (promedio de 53.3% de dependencia lineal). Cuando se analizan los 7 índices: BIAS, DIV, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 7. Análisis Espectral Ikonos 2048 líneas por 2048 columnas.

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 6.33	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.66	0.51	0.41	0.43	0.61	5.84	12.8	0.39	51.0	19.3
RGB/Multi	0.84	0.83	0.80	0.74	0.70	5.01	19.3	0.33	77.6	22.4
RGB/ PCA	0.55	0.54	0.57	0.42	0.61	5.73	12.5	0.43	50.6	18.5
RGB/À trous	0.68	0.53	0.45	0.12	0.11	6.23	7.9	0.55	31.9	20.1

En la Tabla 8 se realiza el análisis espacial de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Ikonos de tamaño 2048x2048, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con la transformada Brovey (promedio de 96.3% de dependencia lineal) y la

transformada Wavelet con el algoritmo À trous (promedio de 96% de dependencia lineal). Cuando se analizan los 7 índices: BIAS, DIV, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 8. Análisis Espacial Ikonos 2048 líneas por 2048 columnas.

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 6.31	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.95	0.98	0.96	0.42	0.62	5.84	11.20	0.70	47.1	28.3
RGB/Multi	0.98	0.85	0.85	0.74	0.72	5.01	19.34	0.33	77.4	48.9
RGB/ PCA	0.94	0.97	0.89	0.42	0.62	5.73	11.27	0.70	45.7	28.5
RGB/À trous	0.94	0.99	0.95	0.12	0.15	6.23	3.97	0.94	17.8	10.0

En la Tabla 9 se realiza el análisis espectral de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Landsat 8 OLI TIRS de tamaño 4096x4096, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con la transformada Brovey, con PCA y con la transformada Wavelet con el algoritmo À trous (97% de dependencia lineal). Cuando se analiza el índice DIV el mejor es el método de multiplicación, cuando se analizan los 6 índices: BIAS, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 9. Análisis Espectral Landsat 8 OLI TIRS 4096 líneas por 4096 columnas.

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 4.96	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.97	0.97	0.97	0.25	-2.70	5.67	23.1	0.76	46.2	17.3
RGB/Multi	0.95	0.96	0.97	0.80	-0.33	3.65	40.8	0.35	81.7	30.7
RGB/ PCA	0.97	0.97	0.97	0.45	-1.07	5.45	24.7	0.76	49.5	18.4
RGB/À trous	0.97	0.97	0.97	0.10	-4.51	5.94	27.5	0.69	55.1	20.6

En la Tabla 10 se realiza el análisis espacial de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada

Wavelet con el algoritmo À trous, con una subescena Landsat 8 OLI TIRS de tamaño 4096x4096, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con la transformada Brovey, con PCA y con la transformada Wavelet con el algoritmo À trous (99% de dependencia lineal). Cuando se analizan los 7 índices: BIAS, DIV, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 10. Análisis Espacial Landsat 8 OLI TIRS 4096 líneas por 4096 columnas

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 5.98	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.99	0.99	0.99	0.24	0.40	5.67	16.9	0.92	33.9	12.2
RGB/Multi	0.94	0.94	0.94	0.80	0.78	3.65	49.4	0.27	98.8	36.2
RGB/ PCA	0.99	0.99	0.99	0.43	0.66	5.45	30.4	0.73	61.2	22.3
RGB/À trous	0.99	0.99	0.99	0.08	0.10	5.92	5.79	0.98	14.3	4.2

En la Tabla 11 se realiza el análisis espectral de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Landsat 8 OLI TIRS de tamaño 8192x8192, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con la transformada Brovey, con PCA y con la transformada Wavelet con el algoritmo À trous (promedio de 97.3% de dependencia lineal). Cuando se analiza el índice DIV, se observa que el mejor es el método de multiplicación, cuando se analizan los 6 índices: BIAS, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 11. Análisis Espectral Landsat 8 OLI TIRS 8192 líneas por 8192 columnas.

Imagen fusionada	R	G	B	BIAS	DIV	Entropía 4.69	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.98	0.97	0.97	0.33	-2.42	5.21	24.3	0.75	48.7	17.5
RGB/Multi	0.96	0.97	0.97	0.83	-0.28	3.15	42.3	0.30	85.0	30.4
RGB/ PCA	0.98	0.97	0.97	0.50	-0.98	5.13	26.9.0	0.73	53.3	19.2
RGB/À trous	0.98	0.97	0.97	0.18	-4.30	5.54	27.3	0.70	54.5	19.6

En la Tabla 12 se realiza el análisis espacial de las imágenes fusionadas con la Transformada Brovey, Multiplicación (Multi), PCA y la Transformada Wavelet con el algoritmo À trous, con una subescena Landsat 8 OLI TIRS de tamaño 8192x8192, con los índices de Correlación, BIAS, DIV, Entropía, ERGAS, índice de calidad Universal Qu, RASE y RMSE, donde se puede observar que los mejores resultados con el índice de correlación se obtienen con la transformada Brovey, con PCA y con la transformada Wavelet con el algoritmo À trous (99% de dependencia lineal). Cuando se analizan los 7 índices: BIAS, DIV, Entropía, ERGAS, Qu,RASE y RMSE, se observa que los mejores resultados espectralmente se obtienen con la transformada Wavelet usando el algoritmo À trous.

Tabla 12. Análisis Espacial Landsat 8 OLI TIRS 8192 líneas por 8192 columnas.

Imagen fusionada	R	G	B	BAI	DIV	Entropía 5.55	ERGAS	Qu	RASE	RMSE
RGB/Brovey	0.99	0.99	0.99	0.25	0.42	5.21	18.2	0.96	36.5	7.74
RGB/Multi	0.94	0.95	0.95	0.81	0.78	3.15	50.8	0.99	101.7	0.58
RGB/ PCA	0.99	0.99	0.99	0.43	0.66	5.13	31.6	0.99	63.3	3.00
RGB/À trous	0.99	0.99	0.99	0.08	0.10	5.54	7.51	0.98	15.0	3.77

La comparación visual de las imágenes fusionadas usando la Transformada de Brovey, Multiplicación, Análisis de Componentes principales y la transformada Wavelet À trous con los diferentes tamaños, se pueden ver en el anexo.

Conclusiones

A partir de la comparación de tiempos de ejecución realizado, se demuestra que todos los métodos implementados presentan una disminución significativa en su tiempo de ejecución. Sin embargo, Brovey es el método que expone el mejor esquema de paralelización, dado que llega a ser aproximadamente 532 veces más rápido que en CPU. Adicionalmente, analizando la tasa de crecimiento del tiempo de ejecución por pixel, se evidencia que el método PCA presenta un comportamiento atípico frente a los otros métodos sobre una arquitectura heterogénea, esto podría significar que el costo de transferencia entre unidades de procesamiento es más alto que en las otras técnicas y que si de igual manera se presenta una mejora significativa en el tiempo de ejecución, las operaciones realizadas en PCA siguen representando un alto costo computacional en dispositivos many-core.

En cuanto a calidad espectral y espacial de la imagen fusionada, las evaluaciones realizadas anteriormente han demostrado que los métodos de fusión de imágenes basados en la transformada de Wavelet usando el algoritmo de "Á trous" son más adecuados para la fusión de imágenes que los métodos convencionales.

Los resultados obtenidos del análisis cuantitativo demuestran que los mejores resultados de la imagen satelitales Ikonos de 2048 por 2048 fusionada de imágenes usando la TWA implementada en Python ofrece

mejores resultados con valores de los índices BIAS, DIV, ERGAS, RASE, Qu, RMSE son mejores que los obtenidos con las imágenes Ikonos de tamaño 1024x1024. Con el índice de correlación los mejores resultados se obtienen con los métodos convencionales. Lo que significa que la mejor dependencia lineal tanto espectral como espacial se obtiene con los métodos tradicionales.

Los resultados obtenidos del análisis cuantitativo demuestran que los mejores resultados de la imagen satelitales Landsat 8 OLI TIRS de 8192x8192 fusionada de imágenes usando la TWA implementada en Python ofrece mejores resultados con valores de los índices BIAS, DIV, ERGAS, RASE, Qu, RMSE son mejores que los obtenidos con las imágenes Landsat 8 OLI TIRS de tamaño 4096x4096. Lo que significa que la mejor dependencia lineal tanto espectral como espacial se obtienen con los métodos tradicionales.

La metodología propuesta permite implementar de forma eficiente las principales metodologías de fusión de imágenes sobre plataformas computacionales heterogéneas (CPU/GPU), permitiendo obtener de forma rápida, imágenes fusionadas que ofrecen a los usuarios información detallada sobre los entornos urbanos y rurales, lo cual es útil para aplicaciones como la planificación y la gestión urbana. Su utilidad se extiende al desarrollo de proyectos en diversos campos como agricultura, hidrología, medioambiente y gestión de emergencias producidas por catástrofes naturales (inundaciones, incendios forestales), entre otros.

Anexo

A continuación, se presenta el conjunto de datos y las imágenes resultantes de la evaluación bajo cada una de las metodologías de fusión: Transformada de Brovey, multiplicación, PCA y transformada Wavelet À trous. El conjunto de datos corresponde a 4 pares (Multiespectral y Pancromática) de imágenes satelitales:

- Ikonos 1024x1024
- Ikonos tamaño 2048x2048
- Landsat 8 OLI TIRS 4096x4096
- Landsat 8 OLI TIRS 8192x8192

Las imágenes se presentan en arreglos de 2 filas por dos columnas que comprenden las 2 imágenes originales de entrada y las 4 imágenes resultantes de la fusión por cada uno de los métodos. Esto facilita la inspección visual de la calidad de la imagen resultante y la comparación de los métodos.

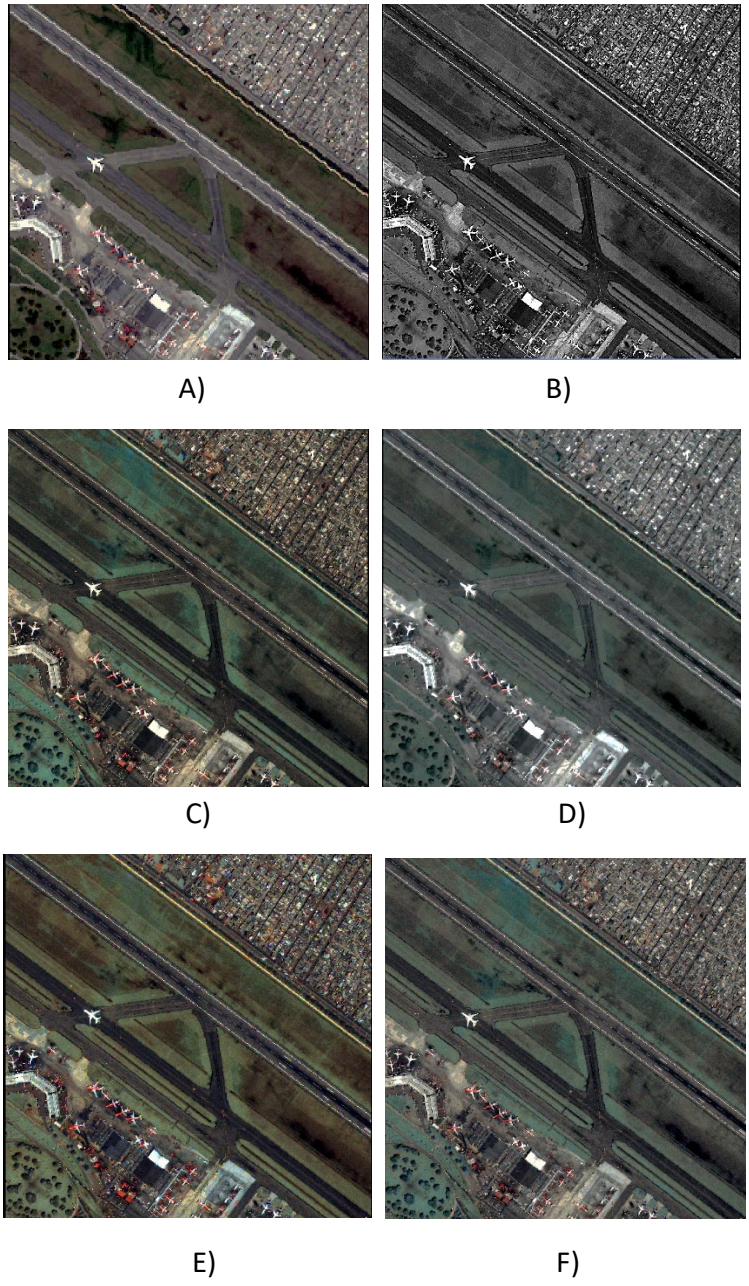


Figura 26. Imagen Ikonos 1024x1024.. Entrada: A)Multiespectral; B)Pancromática. Salida: C)Transformada Brovey; D)Multiplicación; E)PCA; F)Transformada Wavelet À trous

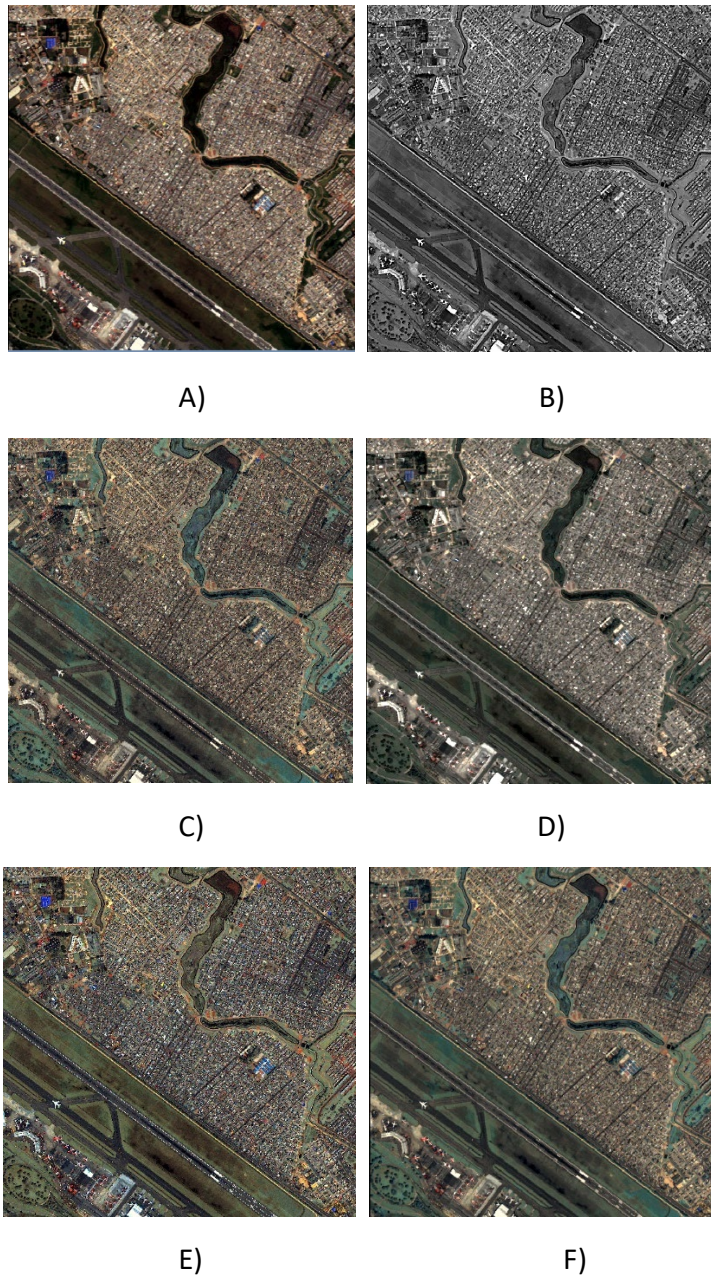


Figura 27. Imagen Ikonos tamaño 2048x2048. Entrada: A)Multiespec.; B)Pancromática. Salida: C)Transformada Brovey; D)Multiplicación; E)PCA; F)Transformada Wavelet À trous

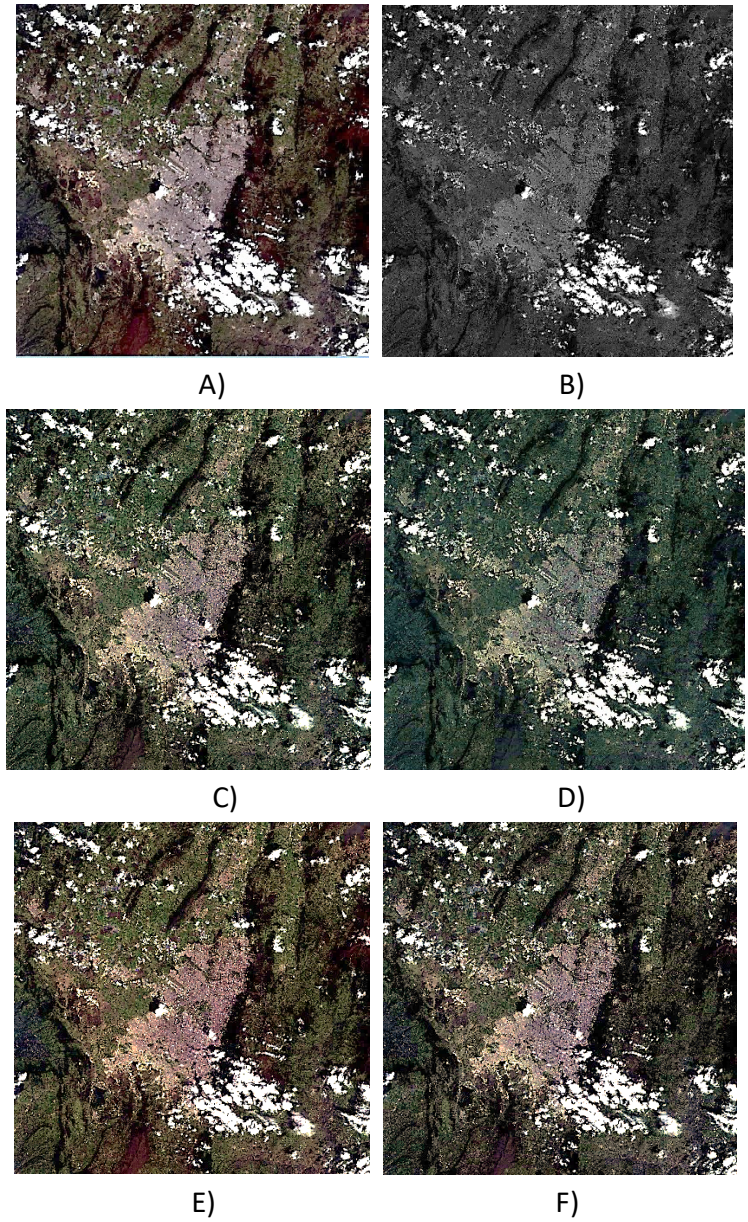


Figura 28. Imagen Landsat 8 OLI TIRS 4096x4096. Entrada: A)Multiespec..; B)Pancromática. Salida: C)Transformada Brovey; D)Multiplicación; E)PCA; F)Transformada Wavelet À trous

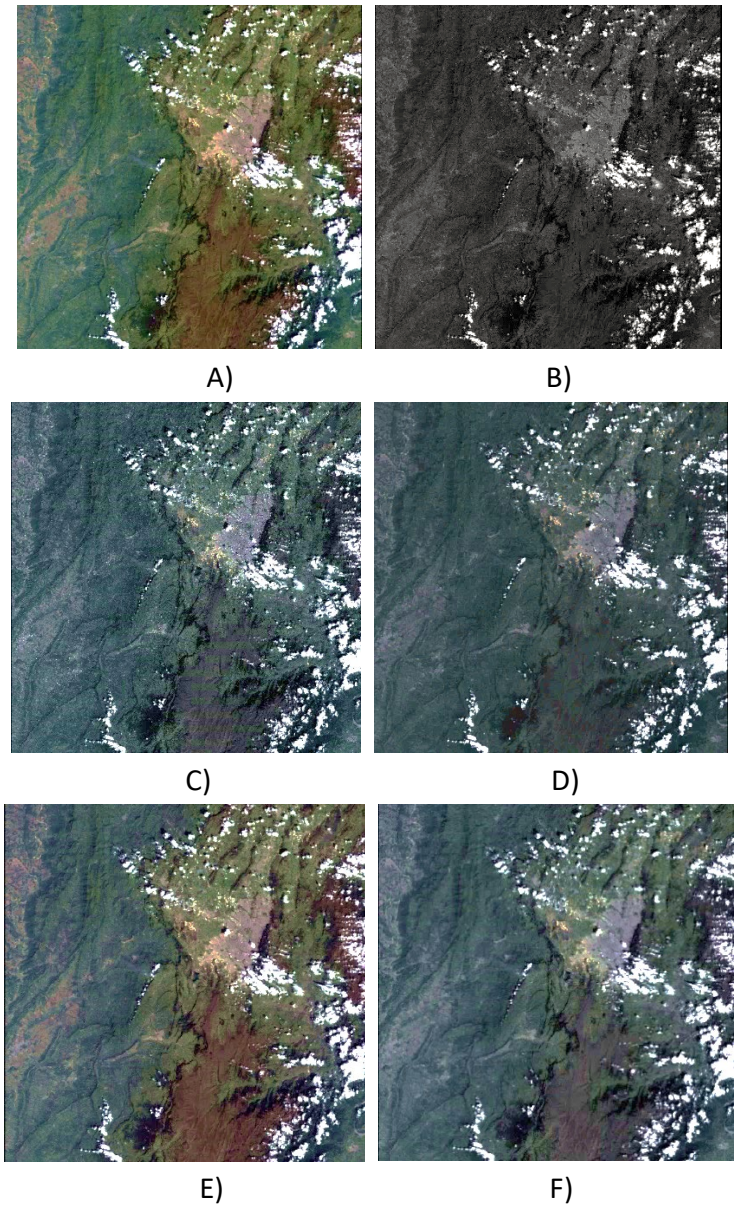


Figura 29. Imagen Landsat 8 OLI TIRS 8192x8192. Entrada: A)Multiespec..; B)Pancromática. Salida: C)Transformada Brovey; D)Multiplicación; E)PCA; F)Transformada Wavelet À trou

Referencias bibliográficas

Alba, E. (2005). Parallel metaheuristics: a new class of algorithms (47). John Wiley & Sons.

Amolins, K., Zhang, Y. and Dare, P., (2007). Wavelet based image fusion techniques — An introduction, review and comparison. ISPRS Journal of Photogrammetry and remote Sensing, 62(4), 249-263.

Amro, I., Mateos, J., Vega, M., Molina, R. and Katsaggelos, A., (2011). A survey of classical methods and new trends in pansharpening of multispectral images. EURASIP Journal on Advances in Signal Processing, (1), 1-22.

Bracewell, R. N. (1978). The Fourier Transform and ist Applications, MacGraw-Hill.

Chibani, Y., Houacine, (2003). A. Redundant versus ortogonal Wavelet descomposition for multisensor image fusion, Pattern Recognition. (36), 879-889.

Chuvieco, E. (2002). Teledetección Ambiental. La Observación de la Tierra Desde el Espacio. Barcelona: Ariel, 2002. ISBN 84-344-8047-6

Chuvieco, E., (2008). Teledetección Ambiental Espacial., 3ª Edición. Ed., Ariel Ciencia. ISBN: 978-84-344-8073-3.

CUDA C. (September 22, 2017) Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide.

De Antonio, M., y Marina, L. (2005). Computación paralela y entornos heterogéneos.

Dutilleux, P. (1987). An implementation of the algorithm a trous to compute the Wavelet transform. In Compt-rendus du congres ondulttes et methods temp-fréquence et espace des phase, Marseille, Springer Verlag, 298-304.

Ehlers, M., Klonus, S., Johan Åstrand, P. and Rosso, P., (2010). Multi-sensor image fusion for pansharpening in remote sensing. International Journal of Image and Data Fusion, 1(1), 25- 45.

González-Audícana, M. (2003). Bondad de los Algoritmos de descomposición Wavelet de Mallat y 'à trous' Para la fusión de imágenes Quickbird. Teledetección y Desarrollo Regional. X Congreso de Teledetección. Cáceres, España. 295-300.

González-Audícana, M., (2007). Métodos clásicos de fusión de Imágenes de satélite, I Jornadas de Fusión. Asociación Española de Teledetección.

González-Audicana, M., X. Otazu, O. Fors y A. Seco (2005). Comparison Between the Mallat's and the à trous Discrete Wavelet Transform Based Algorithms for the Fusion of multispectral and Panchromatic Images, International Journal of Remote Sensing, (26), 597-616.

González-Audícana, X. Otazu, O. Fors, A. Seco y R. García. (2003). Bondad de los Algoritmos de descomposición Wavelet de Mallat y 'à trous' Para la fusión de imágenes Quickbird. Teledetección y Desarrollo Regional. X Congreso de Teledetección. Cáceres, España. 295-300.

Hallada, W.A. and Cox, S., (1983). Image sharpening for mixed spatial and spectral resolution satellite systems. International Symposium on Remote Sensing of Environment, 17 th, Ann Arbor, 1023-1032.

He, C., Liu, Q., Li, H., Wang, H: Multimodal Medical Image Fusion Base don IHS and PCA, IN: Symposium on Security Detection and Information Processing, Vol 7, pp. 280-285, Elsevier (2010).

Hong, G. and Zhang, Y., (2008). Comparison and improvement of wavelet-based image fusion. International Journal of Remote Sensing, 29(3), 673-691.

Kirk, D. B., & Wen-mei, W. H. (2012). Programming massively parallel processors: a hands-on approach. Newnes.

Kpalma, K., El-Mezouar, M.C. and Taleb, N., (2013). Recent Trends in Satellite Image Pansharpening techniques, 1st International Conference on Electrical, Electronic and Computing Engineering.

L. Alparone, L.Wald, J.Chanussoat, C. Thomas, P.Gamba, L. Bruce, (2007). "Comparison of Pansharpening Algorithms: Outcome of the 2006 GRS-S Data Fusion Contest", IEEE Transactions on Geoscience and Remote Sensing,Vol. 45, No. 10, pp 3012-3021, doi: 10.1109/TGRS.2007.904923.

Li, X., Lixin, L. and Mingyi, H., (2012). A Novel Pansharpening Algorithm for WorldView-2 Satellite Images, International Conference on Industrial and Intelligent Information (ICIII 2012), 17-18.

Lillo-Saavedra M. y C., Gonzalo. (2006). Spectral or Spatial Quality for Fused Satellite Imagery? A Trade-Off Solution Using Wavelet à trous Algorithm. International Journal of Remote Sensing, 27(7), 1453-1464.

Lillo-Saavedra, M. Gonzalo, C. Arquero, A. and Martínez, E. (2005). Fusion of multispectral and panchromatic satellite sensor imagery based on tailored filtering in the Fourier domain. International Journal of Remote Sensing. (26), 1263-1268.

Lu, J., Zhang, B., He, H., & Zhang, H. (2011). The high-pass filtering fusion based on GPU. In 2011 International Symposium on Computer Science and Society, 122-125). IEEE.

Mallat, Stéphane. (1989). A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence. (II), 7.

Mallat, Stéphane. (1996). Wavelet for Vision. Proceedings of the IEEE. 4(84).

Medina, J., Lizarazo, I. (2004) Fusión de Imágenes Satelitales usando la Transformada de Wavelet. Universidad Distrital Francisco José de Caldas. ISBN: 958-8175-97-6. 2004.

Nieto N., Orozco D., (Junio 2008). El uso de la Transformada Wavelet Discreta en la Construcción de Señales Senosoidales. Scientia et Technica, (38). Universidad Tecnológica de Pereira. ISSN 0122-1701

Nugteren, C. (2018, May). Clblast: A tuned opencl blas library. In Proceedings of the International Workshop on OpenCL, 1-10.

Nuñez, J., Otazu, X., Fors, O., Prades, A., Palá, V., Arbiol, R. (1999). Multiresolution-Based Image fusion whit Additive Wavelet Decomposition. IEEE Transactions on Geoscience and Remote Sensing. 3(37), 1204 -1211.

Otazu, X., González-A. M., Fors, O. and Ñuñez, J. (2005). Introduction of sensor spectral response into image fusion methods. application to wavelet-based methods. IEEE Trans. on geoscience and rem. sensing, 43(10).

Padwick, C., M. Deskevich, F. Pacifici, and S. Smallwood. (2010). "WorldView-2 Pan-Sharping". Paper presented at the 2010 Conference of American Society for Photogrammetry and Remote Sensing. San Diego, CA, April 26–30.

Pohl, C. and Van Genderen, J. L. (1998). Multisensor image fusion in sensing: concepts methods and application. int. J. Remote Sensing. 5(19), 823-854.

Ranchin T., Aiazzi B., Alparone L., Baronti S., Wald L., (2003). Image fusion. The ARSIS concept and some successful implementation schemes. ISPRS Journal of Photogrammetry & Remote Sensing, 58, 4-18.

Ruiz, Marcello., Rodríguez-Esparragón, J., Rodríguez-Esparragón, D. y Eugenio-González, F. (2011). Identificación y análisis de técnicas de fusión en imágenes de satélites de muy alta resolución. 525-528.

Shan, A. (2006). Heterogeneous processing: a strategy for augmenting moore's law. Linux Journal, 2006(142), 7.

Shettigara, V. K., (1992). A Generalized Component Substitution Technique for Spatial Enhancement of Multispectral Images Using a Higher Resolution Data Set, Photogrammetric Engineering & Remote Sensing, 5 (58), 561-567.

Starck, Jean-Luc & Murtagh, Fionn. (1994). Image Restoration with Noise Suppression Using the Wavelet Transform. Astronomy and Astrophysics. 288. 342-348.

Stathaki, T., (2008). Image fusion: algorithms and applications. London [etc.]: (xxk): Academic Press,

Toolkit, C. U. D. A. (2011). 4.0 cublas library. Nvidia Corporation, 2701, 59-60.

Upegui E. Medina, J. (2019). Análisis de imágenes usando las transformadas de Fourier y Wavelet. Editorial Universidad Distrital Francisco José de Caldas, Bogotá-Colombia.

Vaiopoulos, A. D. (2011). Developing Matlab scripts for image analysis and quality assessment Developing Matlab scripts for image analysis and quality assessment. Earth Resources and Environmental Remote Sensing/GIS Applications II, Proc. of SPIE Vol. 8181, 81810B.

Vivone, G., Alparone, L., Chanussot, J., Dalla Mura, M., Garzelli, A., Licciardi, G.A., Restaino, R. and Wald, L., (2015). A critical comparison among pansharpening algorithms. Geoscience and Remote Sensing, IEEE Transactions on, 53(5), pp. 2565-2586.

Wald, L. (1999). Some terms of reference in data fusion. IEEE Transactions on Geoscience and Remote Sensing, 37(3), 1190–1193. doi:10.1109/36.763269

Wald, L. (2000). Quality of high resolution synthesized images: is there a simple criterion? Proceedings of the third conference Fusion of Earth data: merging point measurements, raster maps and remotely sensed image, Sophia Antipolis, T Ranchin and L. Wald Editors, published by SEE/URISCA, Nice, France, 99-105.

Wald, L., (2002). Data fusion definitions and architectures, fusion of images of different spatial resolutions, Les Presses de l'École des Mines, Paris.

Wald, L., Ranchin, T. & Mangolini, M., (1997). Fusion of Satellite Images of Different SpatialResolutions: Assessing the Quality of Resulting Images, Photogrammetric Engineering & Remote Sensing, 6(63), 691-699.

Wang, Z., Ziou, D., Armenakis, C., Li, D., and Li, Q. (2005). A comparative analysis of image fusion methods. IEEE Trans. on geoscience and rem. sensing, 43(6).

Yoo, S. H., Park, J. H., & Jeong, C. S. (2009, December). Accelerating multi-scale image fusion algorithms using CUDA. In 2009 International Conference of Soft Computing and Pattern Recognition (pp. 278-282). IEEE.

Zhang, J., (2010). Multi-source remote sensing data fusion: status and trends. International Journal of Image & Data Fusion, 1(1), pp. 5-24.

Zhou Wang, Alan C. Bovik. (2002). A Universal Image Quality Index. IEEE Signal Processing Letter, Vol. XX, No. Y March.

Este libro presenta los resultados de una investigación sobre una forma eficiente de acelerar la implementación de los principales métodos de fusión de imágenes mediante procesamiento heterogéneo, segmentando y distribuyendo tareas convenientemente entre cómputo secuencial sobre CPU y cómputo paralelo masivo sobre GPU (Graphics Processing Unit).

La fusión de imágenes al igual que la gran mayoría de operaciones con imágenes presentan una exigencia computacional dependiente del tamaño de la imagen, debido a la granularidad pixel a pixel presente en estas operaciones. Esta granularidad que aparentemente es un inconveniente, termina convirtiéndose en una ventaja porque habilita la posibilidad de paralelización masiva sobre arquitecturas computacionales que ofrecen un alto número de núcleos de procesamiento, tales como las GPU (Graphics Processing Unit).

ISBN: 978-958-49-4957-8

Otros títulos de la colección

Doctorado en Ingeniería
UNIVERSIDAD DISTRITAL "FRANCISCO JOSÉ DE CALDAS"

ARQUITECTURAS DE RED NEURO-CONVOLUCIONAL PARA APLICACIONES DE ROBÓTICA ASISTENCIAL

DETECCIÓN Y CORRECCIÓN DE PROPAGACIONES ANÓMALAS EN RADARES METEREOLÓGICOS

GESTIÓN DE LA ENERGÍA: EL USUARIO DE ENERGÍA COMO PARTE ACTIVA DEL SISTEMA

GESTIÓN Y CIBERSEGURIDAD PARA MICRORREDES ELÉCTRICAS RESIDENCIALES

INTRODUCCION A LA INVESTIGACIÓN SOBRE DESASTRES NATURALES Y CIUDADES INTELIGENTES

RADIACIÓN-MATERIA: GEANT4 HANDS ON!

INVESTIGACIÓN EN INGENIERÍA FUNDAMENTADA EN LA GESTIÓN DEL CONOCIMIENTO

LOS RECURSOS DISTRIBUIDOS DE BIOENERGÍA EN COLOMBIA

REDES NEURONALES CONVOLUCIONALES USANDO KERAS Y ACELERANDO CON GPU