SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

INSTITUTO TECNOLÓGICO DE MÉRIDA

TESIS

"RECONSTRUCCIÓN DE IMÁGENES MULTIESPECTRALES E HIPERESPECTRALES CON GPUs."

PARA OPTAR AL GRADO DE:

MAESTRO EN INGENIERÍA

PRESENTA:

ING. RODRIGO DE JESÚS RAMÍREZ ZAPATA

ASESOR:

M.C. MARIO RENÁN MORENO SABIDO

COASESOR:

DR. ALEJANDRO ARTURO CASTILLO ATOCHE

MÉRIDA, YUCATÁN, MÉXICO.

15 DE JUNIO DE 2015

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Mérida

"2015, Año del Generalísimo José María Morelos y Pavón"

DEPENDENCIA: DIV. DE EST. DE POSG. E INV.
No. DE OFICIO: X-089/2015

ASUNTO: AUTORIZACIÓN DE IMPRESIÓN

MERIDA, YUCATÁN A 02 DE JUNIO DE 2015

C. RODRIGO DE JESÚS RAMÍREZ ZAPATA
PASANTE DE LA MAESTRÍA EN INGENIERÍA
PRESENTE

De acuerdo al fallo emitido por su asesor el **M.C. Mario Renán Moreno Sabido** y su coasesor **el Dr. Alejandro Arturo Castillo Atoche**, y la comisión revisora integrada por el Dr. Carlos Alberto Lujan Ramírez y el Dr. José Ramón Atoche Enseñat, considerando que cubre los requisitos establecidos en el Reglamento de Titulación de los Institutos Tecnológicos le autorizamos la impresión de su trabajo profesional con la **TESIS:**

"RECONSTRUCCIÓN DE IMÁGENES MULTIESPECTRALES E HIPERESPECTRALES CON GPUS"

A T E N T A M E N T E
IN HOC SIGNO VINCES

M.C. MIRIAM H. SÁNCHEZ MONROY
JEFA DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

S. E. P.
INSTITUTO TECNOLOGICO
DE MERIDA
DIVISION DE ESTUDIOS DE
POSGRADO E INVESTIGACION

C.p. Archivo
C.p. Titulación
MHSM/fja

Dedicatoria

Dedico este trabajo a mi madre María Ramírez y a mi abuela Constancia Zapata que siempre creyeron en mí. Agradezco a mi novia Sheyla Rodríguez por apoyarme cada vez que estaba por darme por vencido y en aquellos momentos en el que el estudio y el trabajo ocuparon mi tiempo y esfuerzo. Agradezco a mis asesores al M.C. Mario Renán Moreno Sabido, al Dr. Alejandro Arturo Castillo Atoche por su apoyo y guía en el desarrollo de éste proyecto de tesis. Agradezco también a la maestra M.C. Larissa J. Peniche Ruiz por sus consejos y por los ánimos que siempre me dio y en especial agradezco al Ing. Pastor Enrique Góngora Cárdenas por su apoyo ya que siempre ha sido un ejemplo a seguir para mí.

Resumen

El presente trabajo de tesis se enfoca en la implementación de técnicas de regularización para la reconstrucción de imágenes multiespectrales e hiperespectrales mediante la aplicación de técnicas de cómputo en paralelo para su procesamiento. Las imágenes multiespectrales e hiperespectrales se conforman por un gran volumen de datos lo que dificulta su procesamiento mediante técnicas tradicionales y su implementación en aplicaciones en tiempo real. Por tal motivo es necesario buscar alternativas de procesamiento para resolver el problema de procesamiento en tiempo real y de esta manera permitir su uso en aplicaciones de percepción remota.

Las imágenes multiespectrales e hiperespectrales al momento de ser adquiridas por los diferentes tipos de sensores multiespectrales, por ejemplo, AVIRIS, Worldview 3, Landsat, entre otros, pueden sufrir de distorsión. La distorsión y el ruido se originan por una gran variedad de factores. Por ejemplo, factores ambientales, perturbaciones electromagnéticas e incluso por una mala calibración del sensor son algunos de los factores que pueden producir distorsión en las imágenes al momentos de ser adquiridas. A pesar del creciente interés de la comunidad científica en el procesamiento de imágenes percepción remota (RS, por sus siglas en inglés), muy pocas implementaciones en la literatura utilizan arquitecturas a nivel de *hardware* (HW) para alcanzar el máximo paralelismo posible, y así cumplir con los requerimientos de procesamiento en tiempo real.

Para ello se propone en esta tesis un análisis de las diferentes plataformas para el procesamiento digital de las imágenes entre las que se encontraron FPGAs (Field-Programmable Gate Array), DSPs (Digital Signal Processing), y arreglos de computadoras también conocidos como *Cluster* de computadoras. Algunos de los dispositivos anteriormente mencionados no cuentan con la flexibilidad necesaria para su aplicación (prototipado rápido) o el costo de la adquisición de los equipos es demasiado alto. Por ejemplo, los *Cluster* de computadoras que proporcionan una gran capacidad de procesamiento y se orientan muy bien para el procesamiento de imágenes multiespectrales e hiperespectrales, es complejo de estructurar y adicionalmente su costo es demasiado alto, por lo cual muy pocas or-

ganizaciones tienen acceso a este tipo de sistemas. Una alternativa según las tendencias actuales, son las unidades de procesamiento gráficas (GPUs, por sus siglas en inglés) que permiten procesar grandes volúmenes de datos gracias al paralelismo que estás pueden llegar alcanzar.

En este trabajo de tesis se propone el análisis e implementación de algoritmos de regularización para la reconstrucción de imágenes del satélite Landsat en específico del área de Yucatán (PATH 020, ROW 046) capturadas el año 2005 y de imágenes del sureste del condado de Tippecanoe, Indiana capturadas por el escáner Flighline C1 y una imagen hiperespectral del sensor AVIRIS. También se propone realizar el procesamiento con GPUs NVIDIA Tesla C2075 y Quadro 2000 utilizando CUDA como lenguaje de desarrollo el cual está basado en C/C++. Este lenguaje de alto nivel, permite acelerar la ejecución de las operaciones de reconstrucción propuestas con el paradigma de co-diseño HW/SW entre el CPU y GPU ejecutando las operaciones que se puedan paralelizar, logrando minimizar de esta manera el tiempo de procesamiento de los algoritmos de regularización propuestos en esta tesis.

Abstract

This thesis focuses on the implementation of regularization techniques for the reconstruction of multispectral and hyperspectral imaging by applying parallel computing techniques for processing. Multispectral and hyperspectral images are formed by a large volume of data makes it difficult using traditional processing techniques and their implementation in real-time applications. Therefore it is necessary to seek alternative processing to solve the problem of real-time processing and thus allow its use in remote sensing applications.

Multispectral and hyperspectral when being acquired by the different types of multispectral sensors, for example, AVIRIS, Worldview 3, Landsat, and other images, can suffer distortion. The distortion and noise are caused by a variety of factors. For instance, environmental factors, and even electromagnetic interference by a wrong sensor calibration are some of the factors that can cause distortion in the images at the moment of the acquisition. Despite the growing interest of the scientific community in processing of remote sensing (RS) imagery, very few implementations in the literature used to level architectures *hardware*(HW) to achieve maximum parallelism possible, and thus meet the requirements of real-time processing.

This thesis proposes an analysis of the different platforms for digital image processing including FPGAs (Field-Programmable Gate Array), DSPs (Digital Signal Processing) met and arrangements computers also known as computer *Cluster*. Some of the above devices do not have the necessary flexibility for application (rapid prototyping) or the cost of acquiring the equipment is too high. For enhance, the computer *Cluster* provides high processing capacity and is well geared for processing multispectral and hyperspectral imaging, but it is complex and further their cost structure is too high, so very few organizations have access to such systems. An alternative under current trends, are the graphics processing units (GPUs) that allow to process large amounts of data through parallelism that it may achieve. In this thesis the analysis and implementation of regularization algorithms for image reconstruction Landsat specific area of Yucatán (PATH 020, 046 ROW) acquired in 2005 and images from Southeast Tippecanoe County Indiana captured by the

scanner Flighline C1 and one hiperspectral image from the AVIRIS sensor. This document proposes the GPUs NVIDIA Tesla C2075 GPUs and CUDA and Quadro2000 using CUDA as development language which is based on C/C++. This high-level language, can accelerate the implementation of the proposed reconstruction operations with the paradigm of HW/SW co-design between the CPU and GPU running operations that can be parallelized, achieving thus minimizing the processing time.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La fotografía aérea, los sistemas satelitales, los telescopios espaciales, las radiografías, las ondas acústicas, entre otros, son formas de percepción remota; esta técnica ha evolucionado de tal manera que su aplicación en distintos campos del conocimiento se ha hecho más evidente. Los arreglos de sensores aéreos y satélites que orbitan alrededor del planeta a través de la percepción remota electromagnética permite la adquisición de imágenes multiespectrales e hiperespetrales de la superficie terrestre para su posterior análisis y tratamiento [1]. La percepción remota o teledetección no engloba sólo procesos de adquisición de las imágenes, sino también su posterior tratamiento (Figura 1.1) para interpretación y aplicación en los diferentes campos del conocimiento [2, 3]. La percepción remota es una técnica que permite obtener información a distancia de objetos ubicados en la superficie terrestre. Para que la percepción remota sea posible es necesario un interacción entre los objetos o área geográfica de donde se desea obtener una escena. Los principales componentes de la percepción remota son: un arreglo de sensores, la región geográfica a observar y el modelo espacio-temporal de las señales provenientes del arreglo de sensores. Mediante sensores hyperespectrales es posible recolectar cientos de imágenes, correspondientes a diferentes longitudes de onda, muy cercanas entre sí, de una misma escena representando un área geográfica de la superficie de la tierra. El término imagen multiespectral tiene su origen en el Laboratorio de Propulsión a Chorro de la Administración Nacional de la

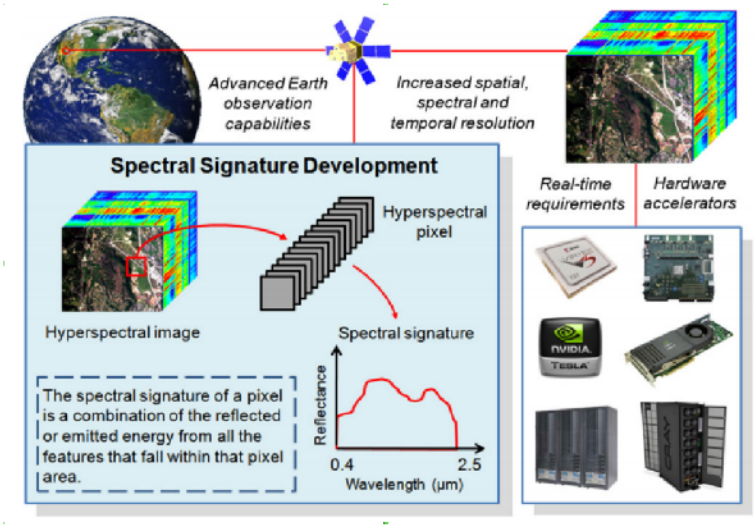


Figura 1.1: Ejemplo de la adquisición de imágenes de percepción y las diferentes técnicas para su procesamiento

Aeronáutica y del Espacio (NASA, por sus siglas en inglés) en California, en el cual se desarrollaron instrumentos tales como el Espectrómetro de Imágenes Aerotransportado (AVIRIS, por sus siglas en inglés) [4]. Este sistema tiene la habilidad de convertir una región de longitud de onda de 400 a 2500 nanómetros utilizando 224 canales, en una resolución nominal de 10 nanómetros.

1.1. Objetivos Generales y Específicos

El objetivo del presente tesis consiste en la implementación de técnicas de regularización para el mejoramiento/reconstrucción de imágenes multiespectrales e hiperespectrales de percepción remota, adquiridas por arreglos de sensores tales como microondas (radar/-SAR), láser (LIDAR) o acústica (SONAR). Particularmente, se propone la implementación en tiempo real del algoritmo WCLS (Weighted Constrain Least Square) utilizando técnicas de cómputo en paralelo con GPUs. A continuación se listan los objetivos específicos:

- Analizar las técnicas existentes para el mejoramiento/reconstrucción de imágenes.
- Analizar el estado del arte en relación al procesamiento de imágenes multiespectrales e hiperespectrales.

- Implementar los algoritmos CLS y WCLS en matlab como modelo de referencia.
- Implementar los algoritmos CLS y WCLS en C/C++, utilizando librerías tales como: CBLAS, LABPACK, entre otras utilizando únicamente CPUs.
- Analizar los algoritmos CLS y WCLS para la reconstrucción de imágenes multiespectrales e hiperespectrales con la perspectiva de diseño con técnicas de cómputo en paralelo con GPUs.
- Realizar pruebas de rendimiento de las diferentes implementaciones del los algoritmos CLS y WCLS utilizando imágenes multiespectrales e hiperespectrales.

1.2. Justificación

En la actualidad la utilización de imágenes de percepción remota esta tomando cada vez más importancia en los diferentes campos de conocimiento. Con la aparición de los nuevo sensores de super resolución como el *Worldview 3* el cual es capaz de generar imágenes multiespectrarles y cubrir un área geográfica de 680,000 km^2 por día, es necesario buscar alternativas para el procesamiento para la alta densidad de datos. Las imágenes de percepción tienen una amplia variedad de usos donde se requieren procesamiento en tiempo real. Por ejemplo, clasificación de áreas geográficas, detección de objetivos militares, estudios de áreas urbanas, estudios de vegetación, prevención de riesgos, entre otros. Debido a los grandes volúmenes de datos que conforman las imágenes de super resolución el tiempo de procesamiento con técnicas tradicionales con CPUs es costoso para su implementación en aplicaciones en tiempo real. En la actualidad existen alternativas para solucionar el problema anterior, un ejemplo de esto son los *cluster* de computadoras que permiten realizar el procesamiento de grandes volúmenes de datos mediante la distribución de la carga de procesamiento entre varios nodos que conforman el *cluster* permitiendo de esta manera reducir el tiempo de procesamiento. Una de las principales desventajas del uso de *clusters* de computadores es su costo ya que requiere de *hardware* especializado, como resultado el uso de *clusters* está limitado a muy pocas organizaciones e instituciones de investigación.

Las imágenes multiespectrales e hiperespectrales no se limitan únicamente al área de percepción remota, sino también se han utilizado en diferentes áreas de investigación. Por ejemplo, cada vez es más común el uso de imágenes multiespectrales en los museos para el análisis de las pinturas para determinar los tipos de tintas, y así poder realizar una planificación al momento de realizar una restauración. El uso de imágenes multiespectrales es un método no invasivo el cual permite extraer una gran cantidad de información de la obra de arte haciéndolo muy atractivo en los museos [5].

1.3. Aportación

La principal aportación de éste trabajo de tesis consiste en demostrar el potencial de las técnicas de cómputo en paralelo con GPUs, para el procesamiento de imágenes multiespectrales e hiperespectrales reduciendo el tiempo de procesamiento de manera drástica para luego poder ser utilizadas en aplicaciones en tiempo real. En éste documento de tesis se implementa los algoritmos CLS y WCLS utilizando cómputo en paralelo con GPUs para la mejoramiento/reconstrucción de las imágenes multiespectrales e hiperespectrales reduciendo el tiempo de procesamiento a diferencia si fueran procesados con técnicas tradicionales con CPUs.

1.4. Análisis del Estado del Arte

A continuación se analiza el estado del arte existente relacionados con las técnicas utilizadas para el mejoramiento/reconstrucción de imágenes y las diferentes técnicas de procesamiento para obtener un tiempo de procesamiento real. Por ejemplo, [6] se realiza el diseño de una aproximación descriptiva experimental (DEDR, por sus siglas en ingles), para tratar el problema de reconstrucción del patrón de espectro espacial (SSP, por sus siglas en inglés) en ambientes inciertos de percepción. En este artículo se busca resolver el problema inverso de la estimación de SSP en imágenes de percepción remota. En [7], se busca resolver los problemas de reconstrucción de SSP en el contexto de medio ambientes

inciertos desde el punto de vista de problemas inversos mal condicionados. En este artículo se propone el diseño de un aproximación descriptiva experimental. El DEDR propuesto incorpora un mínimo de riesgo (MR, por sus siglas en inglés) una estrategia no paramétrica de estimación. La función MR está limitado por la información WCSP, y la técnica DEDR para la reconstrucción de imágenes aplicable a escenarios con bajo rango de incertidumbre.

En [8] se implementa una versión mejorada de *wavelet-transform* (WT, por sus siglas en inglés) denominada *wavelet-lifting* para el procesamiento en tiempo real de vídeo e imágenes con DSPs. El WT tradicional se basa en la convolución y por tal motivo su implementación es compleja y requiere de más espacio de memoria para el almacenamiento de imágenes del medio ambiente y los cálculos, los cuales dificultan su implementación en aplicaciones en tiempo real. En WT (*wavelet-lifting*), se reduce la complejidad de procesamiento, reduciendo la necesidad de memoria, lo que se adecúa de mejor manera para su implementación en aplicaciones en tiempo real.

Otra alternativa de procesamiento es la utilización de un *hardware* híbrido. como se presenta en [9], donde se propone realizar el procesamiento mediante *software* (que se ejecuta en un DSP) en conjunto de *hardware* basado en FPGAs, con el objetivo de aliviar el cuello de botella al momento de realizar el procesamiento. Para lograr esto se aprovecha la arquitectura VLIW (*Very Long Instruction Word*) en el DSP permitiendo la ejecución de múltiples instrucciones al mismo tiempo (siempre y cuando no existe una dependencias entre las operaciones) logrando maximizar el paralelismo al momento de realizar el procesamiento en el DSP. Esta metodología se aplica a receptores inalámbricos 3.5G, así como en *software* basado decodificadores de vídeo H.263 para contenido multimedia.

En [10] se propone un esquema de procesamiento y mejoramiento de imágenes de percepción remota en tiempo real. El artículo plantea realizar un co-diseño *hardware*/*software* utilizando arreglos de procesadores (PAs, por sus siglas en inglés) en conjunto de técnicas de cómputo en paralelo para acelerar el procesamiento. A su vez se propuso un método de regularización descriptiva extendida que incorpora proyecciones en conjuntos de soluciones convexas implementándose de manera eficiente a nivel de *hardware* para el mejoramiento de imágenes de percepción remota.

Los FPGAs son dispositivos compactos y de bajo consumo de potencia, por lo tanto son altamente deseables para realizar el procesamiento abordo de los sensores aerotransportados, esto se presenta en [11]. Uno de los problemas críticos al momento de realizar un diseño basado en *hardware* con FPGAs para el procesamiento de imágenes de percepción remota es la complejidad algorítmica, esto trae como consecuencia una curva de aprendizaje larga para los desarrolladores del algoritmo. En este artículo se plantea una alternativa para mitigar el problema anterior, el cual consiste en realizar el diseño en lenguajes de alto nivel como: Matlab o C/C++ para su posterior conversión a su diseño equivalente en *hardware* (FPGAs), para realizar esto se utilizó la herramienta *Catapult* C^2 desarrollada por *Mentor Graphics* que permite la puesta en práctica de algoritmos realizados en lenguaje de alto nivel y generar con ayuda de las herramientas una síntesis de bajo nivel para su implemetación en *hardware* (FPGA).

En [12] se plantea la utilización de GPUs en conjunto de la biblioteca LAPACK basada en *Fortran* la cual cuenta con diferentes rutinas para la resolución de ecuaciones lineales y operaciones complejas como vector-vector, vector-matriz y matriz-matriz. En el artículo se plantea realizar el procesamiento híbrido entre procesadores tradicionales *socket* dual *quad-core Intel Xeon* 2.33 GHz con una tarjeta gráfica NVIDIA GTX 280, para aprovechar el máximo poder de cómputo al momento de procesar grandes volúmenes de datos a altas velocidades.

El procesamiento de datos hiperespectrales tiene una amplia variedad de aplicaciones entre ellas se encuentra la detección e identificación de objetos. En [13] se plantea la implementación del algoritmo *Automatic Target Detection and Classification Algorithm* (ATDCA, por sus siglas en inglés) para la detección y identificación de objetos. En este artículo se propone la utilización de GPUs para realizar el procesamiento en tiempo real, en específico GPUs NVIDIA: Tesla C1060 y GeForce GTX 580. En esté artículo se resalta la utilización de la memoria compartida (*shared memory*) del GPU para su uso al momento realizar el procesamiento. El principal motivo de esto es que la memoria compartida es una memoria de alta velocidad al momento de ser utilizada por los procesadores del GPU teniendo como resultado una latencia muy baja (al rededor de 100 veces en comparación

de la memoria global).

En [14] se plantea la utilización de *clusters* para el procesamiento de la alta densidad de datos proveniente de las imágenes hiperespectrales para permitir el procesamiento en tiempo real. En este artículo se implemento una nueva versión del algoritmo *automatic target generation process* el cual permite mejorar el paralelismo al momento de su procesamiento. El algoritmo se ejecutó en sesenta nodos, con dos CPUs y cuatro núcleos por nodo.

Debido a la alta densidad de datos de las imágenes percepción remota es necesario buscar una alternativa de bajo costo para su procesamiento. En [15] se plantea una arquitectura híbrida de cómputo en paralelo con GPUs y *clusters de computadoras*. En este artículo se maneja un conjunto de *mini-clusters* donde cada uno se conforma por varias computadoras personales (PC, por sus siglas en inglés) las cuales tienen múltiples procesadores y múltiples GPUs ínter-conectados a través de una red *Gigabit Ethernet* para garantizar una comunicación rápida entre los *mini-clusters*. El objetivo de dividir todo el *cluster* en un conjunto de *mini-clusters* consiste en reducir los cuellos de botellas al momento de comunicarse con el nodo de administración. El artículo también propone el manejo de PCs en lugar de servidores ya que permite bajar los costos que actualmente las PCs cuentan con una cantidad de recursos disponibles (las PCs actuales cuentan con varios Gigabytes de memoria RAM y procesadores multi-núcleos).

1.5. Discusión y tendencias del diseño multi-procesador

Las imágenes multiespectrales e hiperespectrales son utilizadas en diferentes áreas de investigación como se describió en el análisis del estado del arte. Se han implementado una variedad de técnicas de procesamiento para tratar la alta densidad de datos proveniente de las imágenes de percepción remota, y así acelerar su procesamiento para su implementación en aplicaciones en tiempo real. En la Figura 1.2 se presenta un diagrama a bloques el cual muestra de manera general todo el proceso por el que pasa las imánges multiespectrales e hiperespectrales desde el momento de su adquisición mediante los arreglos de

sensores hasta su aplicación en alguna de las áreas de investigación de percepción remota. En el análisis del estado del arte se presentan una variedad de trabajos referentes al

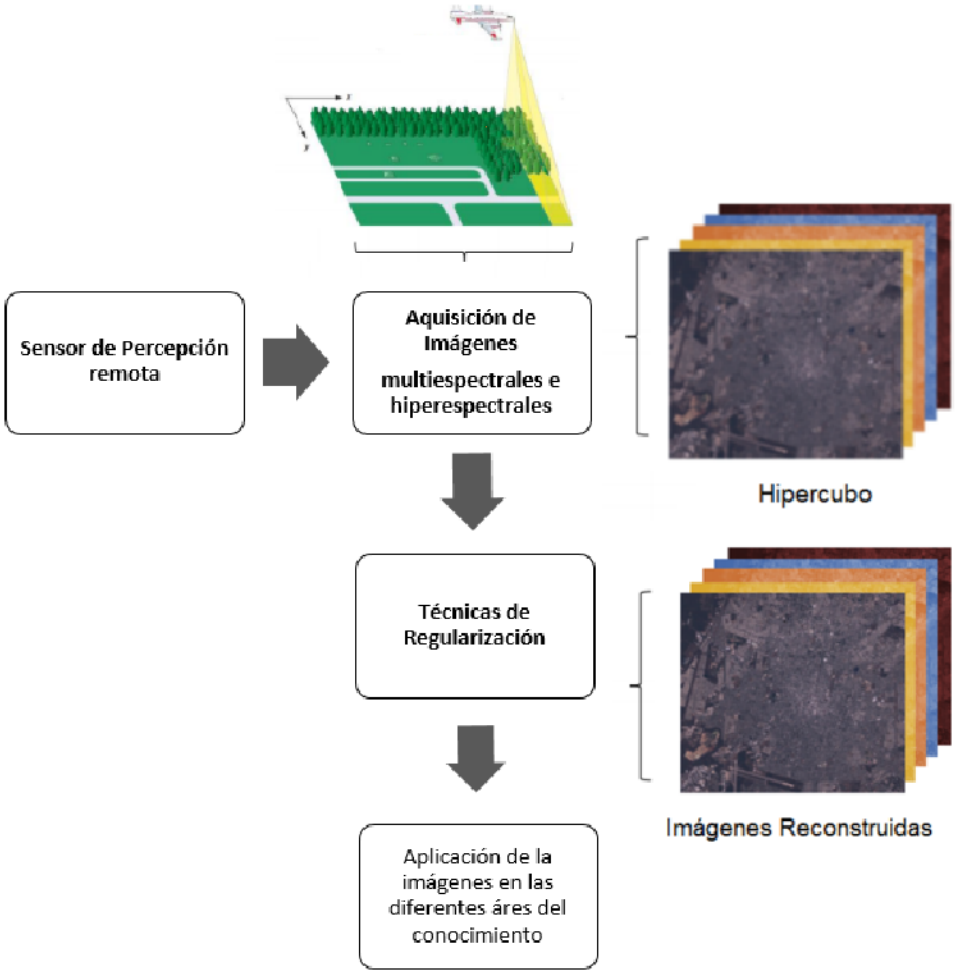


Figura 1.2: Diagrama de bloques representando la adquisición de imágenes de percepción remota y su aplicación en diferentes áres de investigación

tratamiento y procesamiento de imágenes multiespectrales e hiperespectrales, los trabajos presentados se pueden englobar en dos de las etapas en la Figura 1.2. Las técnicas de mejoramiento/reconstrucción se ubican en el tercer bloque de la Figura 1.2 y su posterior aplicación como la identificación de objetivos militares, se encuentran en el cuarto bloque. Aun cuando ya existen trabajos previos que proponen el manejo técnicas de regularización para el mejoramiento/reconstrucción de imágenes de percepción remota, lo que diferencia la propuesta realizada en esta tesis es la metodología de procesamiento utilizando técnicas

de cómputo paralelo. Las propuestas mostradas en el estado del arte muestran implementaciones en hardware (FPGAs), las cuales son complejas de implementar y la curva de aprendizaje llega a ser muy larga. En esta tesis se propone el uso de técnicas de cómputo en paralelo con GPUs para acelerar el procesamiento con el objetivo de mejorar/reconstruir las imágenes de percepción remota para su posterior aplicación en las diferentes áreas de investigación.

Como se puede observar en la Figura 1.2, la etapa de mejoramiento/reconstrucción es de vital importancia ya que afecta de manera directa los resultados en etapas subsiguientes la cual consiste en el procesamiento e interpretación de los datos multiespectrales e hiperespectrales.

Capítulo 2

Imágenes Multiespectrales e Hiperespectrales

2.1. Procesamiento Digital de Imágenes

Una imagen se puede definir como una función bidimensional $f(x, y)$, donde x y y son las coordenadas espaciales (del plano xy), y el valor de la amplitud f en cualquier punto coordenado (x, y) se le conoce como la intensidad o el *nivel de gris* de la imagen en ese punto. Cuando x, y, y los valores en amplitud f son cantidades discretas finitas el resultado es una *imagen digital*. El campo del *procesamiento digital de imágenes* se refiere al tratamiento de imágenes digitales mediante el uso de computadoras o sistemas digitales. Una imagen digital se compone de un número finito de elementos, cada uno de los cuales tiene una ubicación y valor específico. A estos elementos se les conoce como *píxeles*. Los tipos de imágenes son generados por la combinación de una fuente de *iluminación* y la reflexión o absorción de energía de esa fuente por los elementos de la escena que la captura. Cabe mencionar que el término de *iluminación* y *escena* abarcan un concepto más general del que se acostumbra a utilizar en situaciones cotidianas como las fuentes de iluminación de luz visible y las escenas clásicas tridimensionales. Por ejemplo, la iluminación puede originarse por fuentes de energía electromagnética como pueden ser los radares, energía

2.1.2. Muestreo y Cuantización de Imágenes

Existen muchas maneras de adquirir una imagen, radar, sensores CCD, rayos X, etc, pero el objetivo final en todos los casos es siempre el de generar una imagen digital a partir de los datos censados. Para crear una imagen digital es necesario convertir los datos, en el espacio continuo, a un formato digital. Esto involucra dos procesos: el muestro y la cuantización.

Una imagen puede ser continua respecto a las coordenadas espaciales x y y, así como en su amplitud. Para convertir la imagen a un formato digital, simplemente se tiene que muestrear la función tanto en amplitud como en ambas coordenadas. Al digitalizar los valores coordenados se le conoce como muestreo mientras que el digitalizar los valores en amplitud se le llama cuantización.

2.1.3. Tipos de Imágenes

Existen varios tipos de imágenes, las más importantes son:

(a) Imágenes a escala de grises

(b) Imágenes binarias

(c) Imágenes a color

(d) Imágenes multiespectrales e hiperespectrales

2.2. Imágenes a escala de grises

Las imágenes a escala de grises es una matriz de datos que representa la intensidad dentro un rango con 256 valores posibles, siendo 0 el valor negro y 255 el valor del blanco. Los colores entre el rango 0 y 255 representan diferentes niveles de intensidad (figura 2.1).

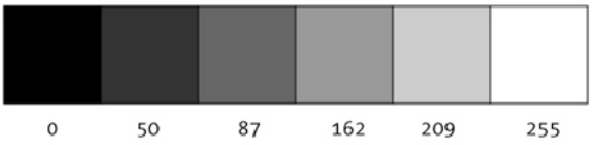


Figura 2.1: Ejemplo de diferentes niveles de intensidad

2.3. Imágenes binarias

Las imágenes binarias cada píxel asumen únicamente dos niveles de intensidad 0 (color negro) o 1 (color blanco). Una imagen binaria se puede visualizar como un arreglo binario (figura 2.2).

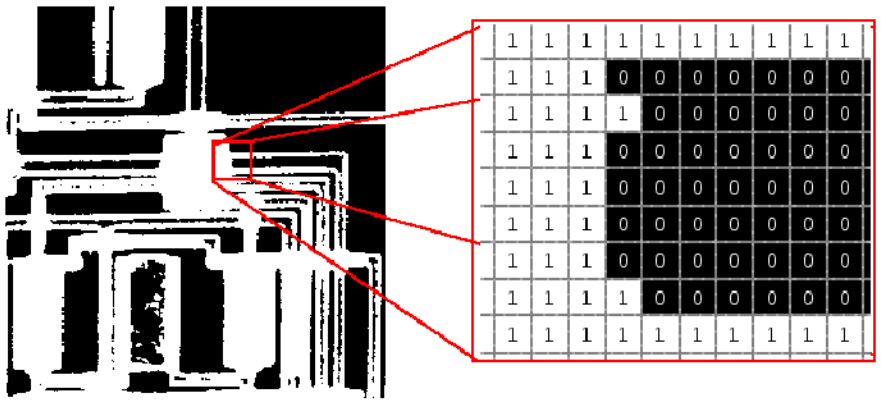


Figura 2.2: Ejemplo de una imagen binaria

2.4. Imágenes a color

Una imagen a color está formada por píxeles, cada píxel contiene tres números que corresponde a las longitudes de onda de los colores rojo, verde y azul (RGB); el valor del píxel puede tener 256 valores posibles (de 0 a 255), esto es debido a que cada valor está conformado por 3 bytes de memoria. Una imagen a color se puede visualizar como un arreglo de tres dimensiones donde x y y representan la posición del píxel y z representa alguna de las bandas Rojo, Verde o Azul.

Figura 2.3: Ejemplo de una imagen RGB

2.5. Imágenes multiespectrales e hiperespectrales

Una imagen hiperespectral o multiespectral se puede visualizar como una imagen de tres dimensiones: dos en el espacio (S_x y S_y) y una en el rango espectral (S_λ) (Figura 2.4). Los datos espectrales pueden visualizarse en forma de un *Cubo Espectral* [16].

El término *banda* es comúnmente utilizado para hacer referencia a una longitud de onda en específico (S_λ). Cada banda proporciona gran cantidad de información de un área geográfica en particular; ciertas longitudes de onda son fuertemente absorbidas/reflejadas por los materiales ubicados en la superficie terrestre. Un ejemplo de esto es la longitud infrarroja, que es altamente absorbida por el agua permitiendo identificar ríos, lagos, y humedales de manera fácil. Como resultado de este fenómeno se obtiene lo que se le denomina *firma espectral* (cada material tiene una firma particular producida por los materiales que lo conforman). Una imagen multiespectral esta conformada por varias docenas de bandas,

la combinación de *n bandas* (comúnmente 3 bandas) tiene como resultado un *false color* (la combinación de diferentes bandas tiene como resultado *false colors* con características diferentes).

A diferencia de las imágenes multiespectrales, una imagen hiperespectral se conforma por varios cientos de bandas. Esto incrementa dramáticamente el número de bandas (por ende su precisión al momento de clasificar) disponibles para el *false color*. El uso de las imágenes multiespectrales e hiperespectraleses tiene una gran variedad de usos tanto militares, como civiles, entre ellos se encuentra la identificación de humedales, bosques, desiertos, zonas urbanas, blancos militares, entre otros.

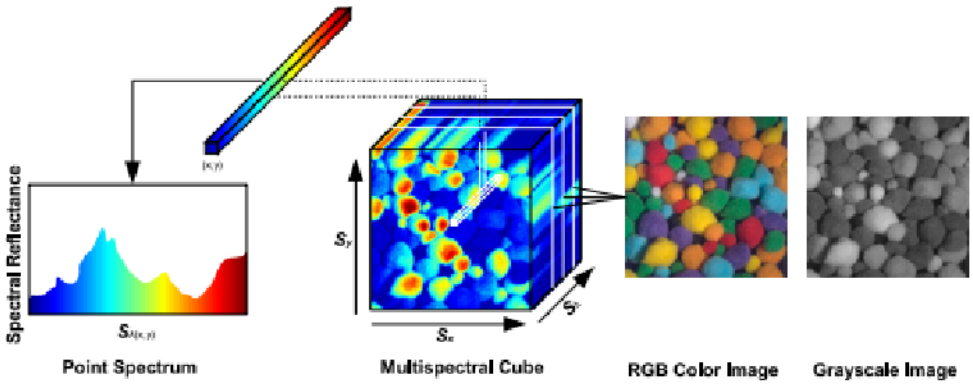


Figura 2.4: Representación de una imagen hiperespectral como un cubo y su firma espectral

2.6. Sensores satelitales y aerotrasnportados de percepción remota

En la actualidad existen una gran variedad de satélites (sensores) los más conocidos son Landsat, SPOT, IRS, AVHRR, Ikonos, Quickbird, FORMOSAT, CARTOSAT, Worldview, Alos, y Geoeye; Entre los sensores aéreos están Daedalus, AVIRIS, HYDICE, y DAIS 7915. Cada uno de los sensores mencionados cuentan con características específicas. A continuación se lista las características importantes de cada sensor.

2.6.1. Landsat

El primer satélite lanzado de esta familia fue el Landsat 1. Este satélite tenía dos sensores, el RBV (Return Beam Vidicon) y MSS (Multi Spectral Scanner). RBV era una cámara de televisión, que fue remplazada por un Thematic Mapper (TM) en Landsat 4 y 5. En los últimas dos satélites, se incorporaron sensores pancromática (pan) , Enhanced Thematic Mapper (ETM) y Enhanced Thematic Mapper Plus (ETM+). La resolución de estos sensores de alrededor de 15 metros con un intervalo de 16 días revisión (ver Tabla 2.1). A continuación se lista las características de la última familia Landsat, Landsat 7.

Tabla 2.1: Rango espectral de Landsat 7

Banda	Rango espectral (μm)	Resolución (m)
1	0.450 a 0.515	30
2	0.525 a 0.605	30
3	0.630 a 0.690	30
4	0.750 a 0.900	30
5	1.550 a 1.750	30
6	10.400 a 12.500	60
7	2.090 a 2.250	30
Pan	0.520 a 0.900	15

2.6.2. SPOT

SPOT es un satélite Francés, Belga, Sueco. En 1986 SPOT 1 fue lanzado (Tabla 2.3), ofreciendo imágenes pancromáticas (pan) de 10 metros y imágenes de 20 metros multiespectrales de resolución con un intervalo de 26 días de revisión. El SPOT 5 trabaja con el mismo intervalo de revisión con imágenes pancromáticas de 2.5 y 5 metros y imágenes multiespectrales (MS) de 10 metros de resolución (Tabla 2.2).

Tabla 2.2: Rango espectral de SPOT 5

Banda	Rango espectral (μm)	Resolución (m)
1	0.50 a 0.59	10
Pan	0.48 a 0.71	2.5 o 5

Tabla 2.3: Historia de la familia SPOT 5

Satélite	Fecha de lanzamiento	Resolución (m)	Intervalo de revisión(días)
SPOT1	22/02/1986	Pan 10; MS 20	26
SPOT2	22/01/1990	Pan 10; MS 20	26
SPOT3	26/09/1993	Pan 10; MS 20	26
SPOT4	04/03/1998	Pan 10; MS 20	26
SPOT5	04/05/2002	Pan 2.5; MS 10	26

2.6.3. IRS

Otra familia de satélites con sensores de percepción remota es la familia IRS (Indian Remote Sensing) de origen Indio. El primero fue lanzado en 1988 proporcionando imágenes con una resolución de 72.2 metros. Este satélite no fue el primero en ser lanzado al espacio, pero si el primero que se le dio un uso extensivo. El IRS 2A tiene la capacidad proporcionar imágenes pancromáticas de 5 a 10 metros de resolución y imágenes multiespectrales de 23.5 a 70 metros de resolución. En la Tabla 2.4 se puede ver las características espectrales del IRS 2A.

Tabla 2.4: Rango espectral de IRS 2A

Banda	Rango espectral (μm)	Resolución (m)
1	0.50 a 0.59	23.5
2	0.62 a 0.68	23.5
3	0.77 a 0.86	23.5
4	1.55 a 1.70	70.5
Pan	0.50 a 0.751	5.8

2.6.4. AVHRR

AVHRR (Advanced Very High Resolution Radiometer) satélite Estados Unidos de Amérida. Este sensor es de baja resolución (alrededor de 1.1 km) y su utilización está comúnmente limitada a estudios de vegetación y bosques. La última versión de este sensor denominado NOAA-16, cuenta con seis bandas abarcando el espectro visible, el cercano al infrarrojo, y infrarrojo térmico.

2.6.5. Ikonos

Ikonos es el primer satélite de percepción remota comercial de Estados Unidos de América. Es uno de los satélites que proporciona imágenes de alta resolución disponibles al público, con una resolución de un metro para imágenes pancromáticas y cuatro metros de resolución en imágenes multiespectrales (Tabla 2.5).

Tabla 2.5: Rango espectral de IRS 2A

Satélite	Fecha de lanzamiento	Resolución (m)	Intervalo de revisión(días)
Ikonos 1	27/04/1999	Pan 1; MS 4	3
Ikonos 2	24/09/1999	Pan 1; MS 4	3

2.6.6. Quickbird

Quickbird es un satélite de percepción remota de EE.UU. Quickbird fue lanzado el 18/10/2001, los sensores de este satélite permite capturar imágenes pancromáticas de 0.61 y 2.44 multiespectrales de resolución con tres días de intervalo de revisión. Este satélite es comúnmente utilizado para análisis de los cambios producidos por el uso de la tierra, agricultura, y climas forestales.

2.6.7. FORMOSAT

FORMOSAT-2 satélite de percepción remota de origen Chino lanzado el 21/05/2004. Ofrece imágenes pancromáticas y multiespectrales de dos metros y ocho metros con un intervalo de revisión de un día.

2.6.8. CARTOSAT

CARTOSAT desarrollado en la India lazado el 05/05/2005 proporciona únicamente imágenes pancromáticas con una resolución de 2.5 metros.

2.6.9. Worldview

Worldview desarrollado en EE.UU. es un satélite de percepción remota de uso comercial. El primer Worlview permite imágenes pancromáticas de 0.55 metros de resolución. Worldview-2 proporciona imágenes pancromáticas y multiespectrales de 0.46 y 1.8 metros de resolución. La versión más reciente hasta el 2014 es el Worldview-3 lanzado el 14 de agosto de 2014 proporcionando imágenes pancromáticas de 31 cm de resolución, 1.24 metros de resolución para imágenes multiespectrales, 3.7 metros para imágenes infrarrojas de onda corta, y 30 metros de resolución para imágenes CAVIS (Clouds, Aerosols, Vapors, Ice, and Snow) utilizado comúnmente para monitorear la atmósfera.

2.6.10. ALOS

ALOS es un satélite de origen Japones lanzado el 24/01/2006. ALOS proporciona imágenes pancromáticas y multiespectrales de 2.5 y 10 metros de resolución con un intervalo de revisión de dos días.

2.6.11. Geoeye

Geoeye fue lanzado el 06/09/2010, con un intervalo de revisión de tres días. La primer satélite ofrece imágenes pancromáticas y multiespectrales de 0.41 metros y 1.65 metros de resolución.

2.6.12. Sensores Aerotransportados

Existen varios sensores aerotransportados para complementar a los satélites de percepción remota. Estos sensores tienen resoluciones comparables a los satélites. Su superioridad se basa en el rango del espectro que pueden abarcar y el número de bandas espectrales que pueden detectar (Tabla 2.6) [17].

Tabla 2.6: Sensores Aerotransportados

Programa	Resolución (m)	Rango espectral	Número de bandas
Daedalus	25	0.42 a 14.00	12
DAIS 7915	3 a 20	0.40 a 12.60	79
HYDICE	1 a 4	0.40 a 2.45	210
AVIRIS	17	0.40 a 2.45	224

2.7. Dispositivos Programables para el Procesamiento Digital de Imágenes

El procesamiento de imágenes (RGB, Escala de grisis, multiespectrales, hiperespectrales, etc) requiere de hardware especializado. El hardware especializado permite procesar grandes cantidades de datos en tiempo real. La cantidad de datos limita al hardware en sí. Los tipos de hardware más comunes que se utilizan para el procesamiento de imágenes en tiempo real son:

2.7.1. Field Programmable Gate Array (FPGA)

Un FPGA es un dispositivo lógico el cual esta conformado por un arreglo de dos dimensiones. La estructura lógica del FPGA se puede observar en la Figura 2.5. El arreglo interno del FPGA puede ser reconfigurado (*programado*) para realizar funciones en específico utilizando el lenguaje de descripción de hardware VHDL o Verilog. Son dispositivos que pueden trabajar a altas frecuencias, pero su principal ventaja es el procesamiento en paralelo [18].

2.7.2. Digital Signal Processor (DSP)

Los DSP o procesadores digitales de señal son microprocesadores específicamente diseñados para el procesamiento de señales digitales. Algunos de sus características más básicas como el formato aritmético, la velocidad, la organización de la memoria o la arquitectura interna hacen que sean o no adecuados para una aplicación particular. Estrictamente hablando, el término DSP se aplica a cualquier chip que trabaje con señales

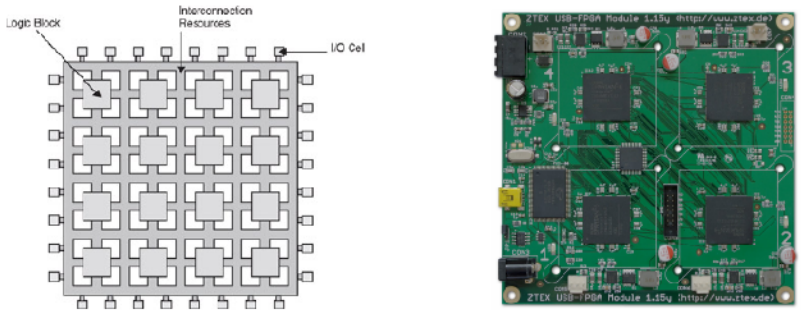


Figura 2.5: Estructura Interna de un FPGA y vista del FPGA XC6SLX150

representadas de forma digital. En esta tesis, el término se refiere a microprocesadores específicamente diseñados para realizar procesamiento digital de señales [19].

2.7.3. Graphics Processing Unit (GPU)

Las unidades de procesamiento gráfico son dispositivos que se pueden encontrar en las PCs modernas. Estos dispositivos proporcionan un número de operaciones básicas al CPU, como renderizar imágenes en memoria para ser desplegada en pantalla. Las GPUs pueden lanzar cientos incluso miles de hilos a la vez lo que permite tener un gran *throughput* de salida gracias a la gran cantidad de hilos con los que puede trabajar simultáneamente.

Arquitectura del GPU

Los GPUs tienen un diseño completamente diferente a las unidades de procesamiento o CPUs. Las GPUs están conformadas por los siguientes elementos clave:

- Memoria (Global, Constante, Compartida)
- Streaming multiprocessors (SMs)
- Streaming processors (SPs)

Una de las principales características de una GPU es que como se puede ver en la Figura 2.6 el GPU se conforma de un arreglo de SMs. El arreglo de SMs es una de las características más importantes de un GPU, esto se debe a que al agregar más SMs, el

GPU será capaz de procesar más tareas al mismo tiempo, o una única tarea de manera más rápida. Mientras con más SMs cuente el GPU más paralelismo se podrá alcanzar, es importante considerar que la afirmación realizada con anterioridad no siempre se cumple; esto se debe a que el nivel de paralelismo está limitado al algoritmo o los datos a procesar, ya que en ocasiones las operaciones del algoritmo y la dependencia entre ellas son las que limitan el nivel de paralelismo. A su vez cada SM está compuesto por múltiples SP; como se puede observar en la Figura 2.6, cada *Streaming Multiprocessor* está conformado por ocho SP [1].

Cada SM tiene acceso a lo que se le denomina registros de archivos; estos registros de archivos pueden visualizar como memoria que trabaja a la misma velocidad que los SP (Figura 2.7), es decir, el tiempo de acceso a este espacio de memoria es cero permitiendo realizar operaciones a gran velocidad. La memoria disponible por registros sólo es accesible por hilo, no es posible compartir información entre hilos utilizando registros, para compartir información entre hilos es necesario utilizar la memoria compartida o memoria global. Otro elemento importante es la memoria compartida (*shared memory*). La memoria compartida es accesible por SM; algunos desarrollados utilizan esta memoria como un tipo de cache (la memoria compartida trabaja a velocidades menores que los registros; pero a mayor velocidad que la memoria global), pero a diferencia de la cache de un CPU está completamente bajo control del programador.

Cada SM tiene separado el bus entre tres tipos de memorias las cuales son: memoria de texturas, memoria constante, y memoria global. La memoria de textura es un tipo de memoria especial ubicada dentro de la memoria global; esta memoria es útil donde los datos se tienen que interpolar, por ejemplo, en lo que se le denomina *lookup tables* en dos o tres dimensiones. La memoria constante se utiliza para los datos de sólo lectura y se almacena en caché. La memoria global está provista vía GDDR (Graphic Double Data Rate) en la tarjeta gráfica. Esta memoria es una versión del alto rendimiento de la memoria tradicional DDR (Double Data Rate). El tamaño del bus de memoria puede ser de hasta 512 bits, proporcionando un ancho de banda de 5 a 10 veces más que los CPUs, hasta 190

[1] Cada familia de GPU cuenta un número diferente de SM y SP.

GB/s con la arquitectura Fermi. Cada SM cuenta con dos o más unidades de uso especial (SPUs, por sus siglas en inglés), el cual realiza instrucciones especiales de hardware, tales como operaciones de alta velocidad como: seno, coseno y operaciones exponenciales.

Cada SM cuenta con dos o más unidades de propósito especial (SPUs), realizando instrucciones, como seno/coseno/exponentes; esta arquitectura se puede observar en la Figura 2.7 [20].

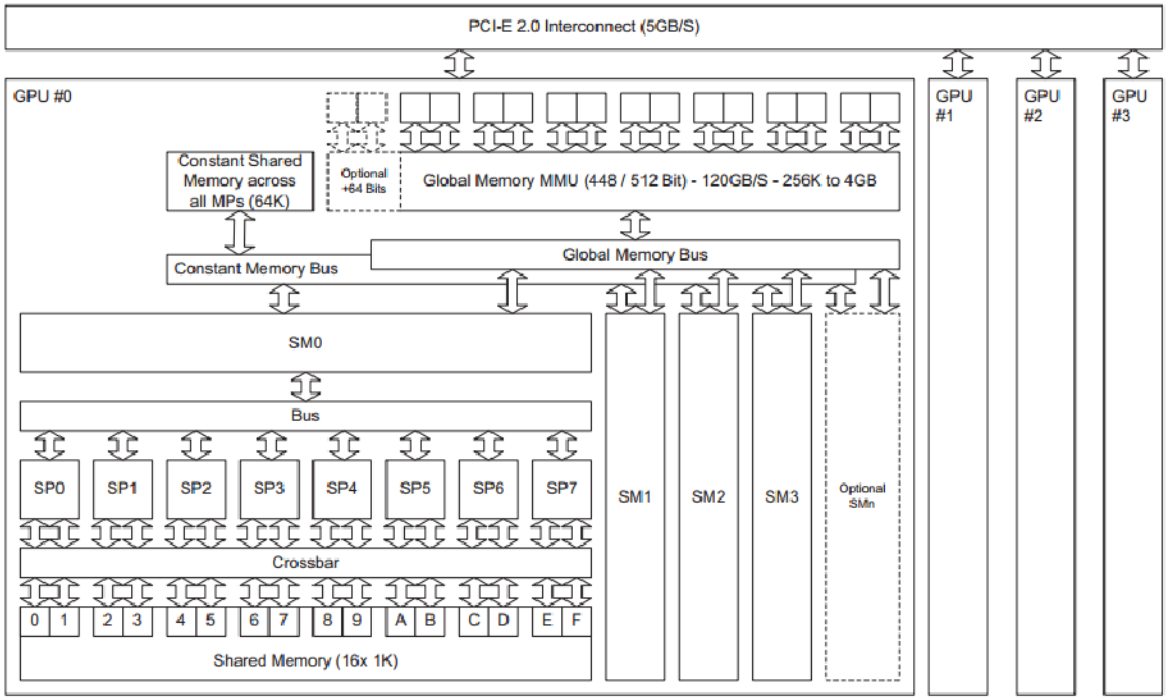


Figura 2.6: Estructura en bloques de un GPU

Cómputo en Paralelo

En la industria de los semiconductores han aparecidos dos ramas principales en el diseño de microprocesadores. La rama de *multicore* que busca mantener la velocidad de ejecución de un programa secuencial mediante la ejecución de los programa a través de múltiples núcleos. Un ejemplo de un microprocesador multi-núcleos es el Intel i7 con cuatro núcleos en su encapsulado llegando a ocho núcleos virtuales gracias al *hyperthreading*. En contraste, se encuentra la rama *multi-hilo* que se enfoca en obtener un mayor rendimiento

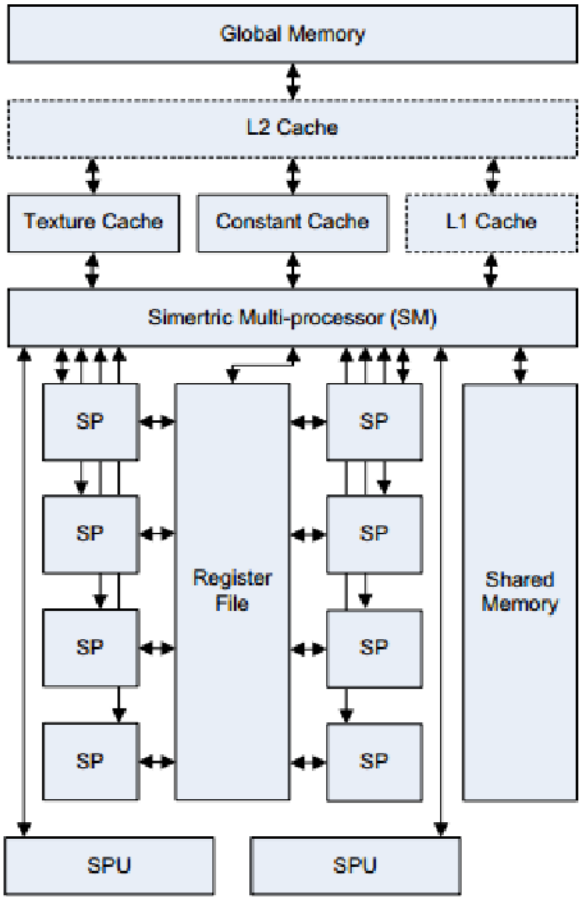


Figura 2.7: Estructura interna de SM

mediante la ejecución de aplicaciones en paralelo. Un ejemplo es la tarjeta de procesamiento gráfico NVIDIA GTX680 con 16,384 hilos. Los procesadores multi-hilo han liderado el rendimiento de operaciones de punto flotante desde el 2003. En el 2012, el rendimiento entre GPU milti-hilos y los CPU multi-núcleo fue de alrededor de 10 veces superior. La principal causa de la gran diferencia en rendimiento entre multi-hilo y multi-núcleo se debe las CPUs están diseñadas para ejecutar código secuencial. Los CPUs cuentan con un sofisticada unida lógica que permite a las instrucciones de un único hilo ejecutarse en paralelo incluso fuera de orden mientras da la apariencia de ejecutarse de manera secuencial. Una de las características de las CPUs, es la gran cantidad de memoria cache con la que cuenta la cual es utilizada para reducir las latencias producidas al momento de acceder a los datos en memoria. Un punto muy importante a considerar entre las GPU y

los CPUs es el ancho de banda. En los CPUs la velocidad de las aplicaciones está limitada a la tasa en la que los datos pueden ser movidos de la memoria del sistema al procesador.

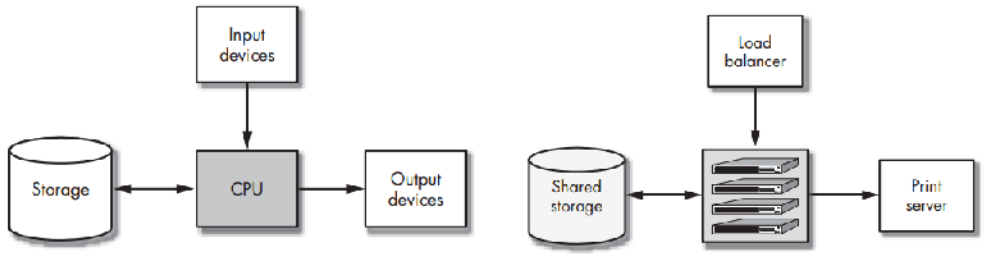
2.7.4. Cluster de Computadoras

El término *Cluster* es normalmente utilizado para describir un sistema de procesamiento distribuido. Los *Clusters* de computadoras son una combinación de hardware y software que permiten distribuir la carga de trabajo entre todos sus nodos; la arquitectura general desde el punto de procesamiento es muy similar a la de un CPU (Figura 2.8a), la única diferencia es que en lugar de tener un único CPU realizando los cálculos existen varios o incluso varios cientos realizando el procesamiento (Figura 2.8b).

Existe una gran variedad de *clusters* al rededor del mundo; algunos de ellos son *cluster* de alto desempeño (*Hight Performance*) que buscan obtener el máximo poder de procesamiento posible combinando varios equipos de cómputo más pequeños; estos tipos de *clusters* son utilizados por organizaciones que buscan procesar una grandes cantidades de información en el menor tiempo posible. Un ejemplo de este tipo de cluster es el Tianhe-2, una super computadora desarrollada por la Universidad Nacional de China de Defensa y Tecnología(NUDT) con la capacidad de realizar 33.86 petaflop/s (cuatrillones de cálculos por segundo) tomando como punto de referencia la biblioteca LAPACK. El Tianhe-2 esta conformado por 16,000 nodos, cada nodo cuenta con dos procesadores Intel Xeon E5-2692v2 12C 2.2GHz con 88 gigabytes de memoria.

Otro tipo de *clusters* conocido es el de alta disponibilidad (*High-availability*); este tipo de *clusters* busca mantener un servicio disponible el mayor tiempo posible evitando tiempos muertos en el servicio ante la falla de alguno de los nodos; su poder de cómputo puede ser muy superior al de un único equipo, pero mucho menor en comparación a un *cluster* de alto desempeño [21]. Los *cluster* de alta disponibilidad normalmente utilizan algún tipo de software de gestión (ej: pacemaker) que permite enmascarar el *cluster* de tal manera que para el usuario se final aparente ser un único *host*. El software de gestión igual tiene la función de monitor para identificar si alguna de las aplicaciones no está funcionando

de manera adecuada, reubicando dicha aplicación en otro *host* que tenga los recursos necesarios para esto, logrando de esta manera mantener el servicios activos.



(a) Arquitectura de una computadora (b) Arquitectura de un cluster

Figura 2.8: Comparativa general de una computadora y un cluster

Capítulo 3

Algoritmos CLS y WCLS

En este capítulo se describen los algoritmos de mejoramiento/reconstrucción implementados en esta tesis. Los algoritmos presentados son el de Mínimos Cuadrados Restringidos (CLS) y una variante mejorada del anterior denominado Mínimos Cuadrados Restringido Ponderado (WCLS).

3.1. Forma continua del modelo del problema

El planteamiento del problema de adquisición de imágenes de percepción remota se puede representar como la función de dispersión del objeto $e(\boldsymbol{x})$, la cual define la distribución de la dispersión (o radiación) en lo que corresponde al dominio de la imagen $X \ni \boldsymbol{x}$. La medición realizada del campo $u(\boldsymbol{y}) = s(\boldsymbol{y}) + n(\boldsymbol{y})$ (Figura 3.1) consiste en la señal de retorno s y ruido aditivo n que se encuentra disponible para su observación y almacenamiento dentro dentro del dominio espacio temporal $Y = T$ x P, donde $\boldsymbol{y} = (\boldsymbol{t}, \boldsymbol{\rho})^T$ define los puntos tiempo(t)-espacio($\boldsymbol{\rho}$) en Y; $t \in T$, $\boldsymbol{\rho} \in P$; $\boldsymbol{y} \in Y$. El modelo de datos $\boldsymbol{u}$ se define en una ecuación de observación (EO, por sus siglas en inglés) estocástica que en su

forma de integral se puede escribir como sigue:

$$\begin{aligned} u(\boldsymbol{y}) &= (\tilde{\boldsymbol{S}}e(\boldsymbol{x}))(\boldsymbol{y}) + n(\boldsymbol{y}) \\ &= \int_X \tilde{\boldsymbol{S}}(\boldsymbol{y},\boldsymbol{x})e(\boldsymbol{x})d\boldsymbol{x} + n(\boldsymbol{y}) \\ &= \int_X \boldsymbol{S}(\boldsymbol{x},\boldsymbol{y})e(\boldsymbol{x})d\boldsymbol{x} + \int_X \Delta\boldsymbol{S}(\boldsymbol{x},\boldsymbol{y})e(\boldsymbol{x})d\boldsymbol{x} + n(\boldsymbol{y}) \end{aligned} \tag{3.1}$$

La función $\tilde{\boldsymbol{S}}(\boldsymbol{y},\boldsymbol{x})$ de la perturbación aleatoria del operador $\tilde{\boldsymbol{S}}$ la cual es dada por la ecuación (3.1) define el modelo de formación de la señal. La media de $\tilde{\boldsymbol{S}}(\boldsymbol{y},\boldsymbol{x})$ hace referencia al ley de modulación nominal en la formación de los datos del canal definido por la modulación tiempo-espacio de la señal utilizadas en imágenes de radar/SAR, y la variación de la media $\Delta\boldsymbol{S}(\boldsymbol{y}) = \mu(\boldsymbol{y},\boldsymbol{x})S(\boldsymbol{y},\boldsymbol{x})$ modela las perturbaciones estocásticas del campo de ondas en diferentes rutas de propagación, donde $\mu(\boldsymbol{y},\boldsymbol{x})$ representa una media cero con ruido multiplicativo que modela las perturbaciones de propagación aleatorias en el medio. Los campos e, n, u en (3.1) se asumen con ruido aleatorio Gaussiano con media cero. En este planteamiento del problema se asume una naturaleza incoherente del campo de la retrodispersión $e(\boldsymbol{x})$. Esto naturalmente inherente para los experimentos de percepción remota y lo que conlleva a una función de correlación del campo del objeto, $R_e(\boldsymbol{x}_1,\boldsymbol{x}_2) = b(\boldsymbol{x}_1)\delta(\boldsymbol{x}_1-\boldsymbol{x}_2)$, donde $e(\boldsymbol{x})$ y $b(\boldsymbol{x}) = \langle\|e(\boldsymbol{x})\|^2\rangle$ se refieren como una función de dispersión aleatoria compleja con potencia media o patrón espectral espacial (SSP, por sus siglas en inglés). El principal problema en el mejoramiento de imágenes de percepción remota consiste en desarrollar métodos de procesamiento de señal para realizar una estimación eficiente de SSP $b(\boldsymbol{x})$ procesando la disposición de las mediciones del radar/SAR de la onda de campo de datos $u(\boldsymbol{y})$. Dicha estimación $\hat{b}(\boldsymbol{x})$ de la SSP $b(\boldsymbol{x})$ se refiere como la reconstrucción deseada de la imagen de percepción remota [22]. En la Figura 3.1 se ilustra el modelo de degradación y reconstrucción. El principal problema matemático para una ecuación de observación lineal consiste en

$$u = Sv + \eta, v \in V_N, u \in U_L, S : V_N \longrightarrow U_L \tag{3.2}$$

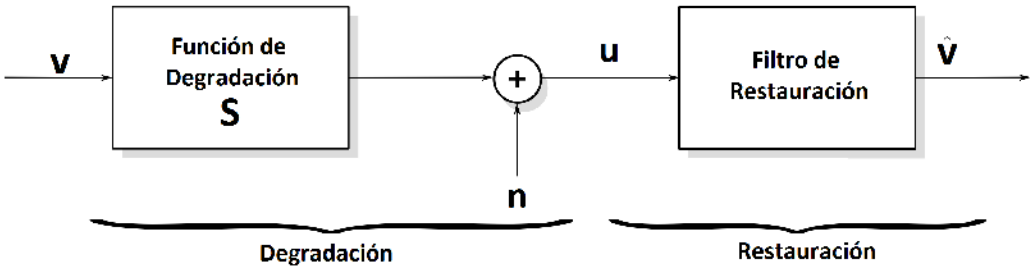


Figura 3.1: Modelo del degradación y restauración

el cual en su forma discreta vectorial es:

$$\boldsymbol{u} = \boldsymbol{S}\boldsymbol{v} + \boldsymbol{\eta} \tag{3.3}$$

En este documento se considera el problema del modelado de la función $\boldsymbol{S}$ que representa la matriz de dispersión (PSF, por sus siglas en inglés) en un planteamiento no-paramétrico más general; el problema inverso propuesto como solución en este trabajo de tesis consiste en: diseñar un generador de solución $\boldsymbol{W}$ o una matriz de solución $\boldsymbol{W}$ que se aplique al vector de datos $\boldsymbol{u}$, produciendo una solución para la ecuación (3.3), la cual será la estimación de la señal/imagen de interés. Esto es

$$\hat{\boldsymbol{v}} = \boldsymbol{W}\boldsymbol{u} = \boldsymbol{W}\boldsymbol{S}\boldsymbol{v} + \boldsymbol{n} \tag{3.4}$$

El planteamiento en esta tesis consistirá en encontrar una solución óptima en algún sentido estricto, bajo la solución de problemas inversos.

3.2. Información a priori

En el método se puede suponer un conocimiento a priori acerca del vector deseado $\mathbf{v}$ para un vector determinista $\boldsymbol{m}_v$ referido como el valor de la media conocida, o el vector esperado del vector deseado desconocido $\mathbf{v}$.

$$\boldsymbol{m}_v = \langle \boldsymbol{v} \rangle \tag{3.5}$$

En este documento se considera $m_v = 0$, como la medición del vector de ruido $\boldsymbol{\eta}$ como se muestra en (3.2), donde se modela como un vector desconocido aleatorio de media cero, esto es:

$$\boldsymbol{m}_n = \langle \boldsymbol{n} \rangle = 0 \tag{3.6}$$

y la matriz de correlación, $\boldsymbol{R}_n = \langle \boldsymbol{n}\boldsymbol{n}^T \rangle$; es importante notar que que no se específica del modelo del ruido $\boldsymbol{\eta}$ dentro de la regularización determinista aproximado del problema inverso dado en (3.2). En el método tradicional de *mínimos cuadrados (LS)* se busca obtener a una solución aproximada de **v** para la ecuación (3.2) y de esta manera mínimizar el error de ajuste de *LS*.

$$\begin{aligned} \widehat{\boldsymbol{v}} = argmin\{\boldsymbol{J}_1\} &= argmin\{\|\boldsymbol{\eta} - \boldsymbol{S}\boldsymbol{v}\|^2\} \\ &= argmin\{[(\boldsymbol{\eta} - \boldsymbol{S}\boldsymbol{v}), (\boldsymbol{\eta} - \boldsymbol{S}\boldsymbol{v})]\} \end{aligned} \tag{3.7}$$

La idea descriptiva de una regularización determinista (regularización de *Tikhonov's*) es remplazar el problema mal planteado originalmente como una "muy cercana"solución bien planteada del problema, la cual se aproxima a los requerimientos de la solución bien planteada. Esto puede ejecutarse por el reemplazamiento libre *LS* de la función costo J_1 por la función de costo aumentada *CLS* (ecuación 3.8). El termino J_1 representa el error de ajuste al cuadrado entre el vector de observación $\boldsymbol{n}$ y la señal del modelo *Sv*.

$$J(\boldsymbol{v}) = J_1(\boldsymbol{v}) + \alpha J_2(\boldsymbol{v}) \tag{3.8}$$

$$J_1(\boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}\|^2 = [(\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}), (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v})] \tag{3.9}$$

$$J_2(\boldsymbol{v}) = \|\boldsymbol{v} - \boldsymbol{m}_v\|^2 = [(\boldsymbol{v} - \boldsymbol{m}_v), (\boldsymbol{v} - \boldsymbol{m}_v)] \tag{3.10}$$

El segundo término J_2 impone el criterio de la norma mínima al cuadrado. El principal argumento en que se soporta el criterio de la norma al cuadrado (ecuación 3.10) es la

consistencia con la información previa $m_{\boldsymbol{v}} = \langle \boldsymbol{v} \rangle$ para alcanzar una solución $\widehat{\boldsymbol{v}}$ con la menor desviación de energía. Debido a que el mínimo costo de la norma cuadrada J_2 se le denomina como *stabilizer* en el sentido que se estabiliza la solución $\widehat{\boldsymbol{v}}$, haciendo el modelo previo más cercano a $\boldsymbol{m}_v$. La regularización del parámetro en α en la ecuación 3.8 provee la conmutación entre la fidelidad de la medida (criterio J_1) y la sensibilidad al ruido (criterio J_2) α es un número positivo real. En un sentido se puede ver a α como un indicador de suficiencia de los datos de la solución específica de la optimización del problema CLS:

$$\widehat{\boldsymbol{v}} = argmin\{\underbrace{J_1(\boldsymbol{v})}_{v \in V} + \alpha J_2(\boldsymbol{v})\} \tag{3.11}$$

$$= argmin\{\|\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}\|^2\} + \alpha\|\boldsymbol{v} - m_{\boldsymbol{v}}\|^2\} \tag{3.12}$$

En particular, el caso de que el limite $\alpha \to 0$ implica que el problema se reduce a una versión *unconstrained ill-posed* (3.7), con la solución de $\boldsymbol{v}$ completamente determinada por la forma de la medición del dato u (sin conocimiento previo acerca de la solución deseada involucrada). De otro modo, cuando el límite de $\alpha \to \infty$ implica que el conocimiento previo dado por el mismo es suficiente para especificar el mapeo (3.12). En la práctica la regularización del parámetro, α, es asignada a algún valor con dos límites de condiciones, entonces ambas medidas de datos, $\boldsymbol{u}$, y la información previa contribuye a la solución de (3.12). De esta manera el término $\alpha J_2(\boldsymbol{v})$ en la función de costo aumentada CLS representa un modelo de la función restringida penalizada, la cual representa una influencia sobre la solución final controlada por la regularización del parámetro α.

3.3. Algoritmo CLS

En la ecuación 3.8, la cual define la suma de las normas euclidianas al cuadrado específicamente por la función costo y el criterio de la norma mínima, es posible rescribir

estas expresiones en términos de productos internos como:

$$= const - \boldsymbol{v}^T\boldsymbol{S}^T\boldsymbol{u} - \boldsymbol{u}^T\boldsymbol{S}\boldsymbol{v} + \boldsymbol{v}^T\boldsymbol{S}^T\boldsymbol{S}\boldsymbol{v} + \alpha(\boldsymbol{v}^T\boldsymbol{v} - \boldsymbol{m}_v^T\boldsymbol{v} - \boldsymbol{v}^T\boldsymbol{m}_v) \tag{3.13}$$

donde *const*= $[\boldsymbol{u}, \boldsymbol{u}] + \alpha[\boldsymbol{m_v}, \boldsymbol{m_v}]$ factorizando los términos independientes del vector **v**. La ecuación 3.13 declara la función de costo $J(\boldsymbol{v})$ que es un escalar de segundo orden del vector funcional **v**. Consecuentemente se visualiza la dependencia del costo argumentado $J(\boldsymbol{v})$ sobre los elementos del vector **v** como una superficie en forma cónica y con un solo mínimo. Para determinar el optimo vector estimado $\widehat{\boldsymbol{v}}$, primero es necesario derivar el argumento del costo $J(\boldsymbol{v})$ dado en la ecuación 3.13 con recpecto al vector deseado **v** y igualando el resultado a cero.

$$g(\boldsymbol{v}) = \frac{\partial J(\boldsymbol{v})}{\partial \boldsymbol{v}} = 0 \tag{3.14}$$

La solución al campo de la ecuación para el óptimo estado estimado $\widehat{v}$. Para realizar la evaluación, es necesario conocer como diferenciar una función escalar funcional respecto a vectores de valores complejos.

Considerando la Tabla 3.1, se puede encontrar la gradiente del vector $g(\boldsymbol{v})$, definido como la derivada de el costo aumentado $J(\boldsymbol{v})$ dada en 3.13 con respecto al vector **v**.

Tabla 3.1: Reglas para derivar en primer orden y segundo orden

Vector con valores reales x	**Vector con valores complejos x**
$J(x) = a^T x \Rightarrow g(x) = a$	$J(x) = a^T x \Rightarrow g(x) = 0$
$J(x) = x_T \Rightarrow g(x) = a$	$J(x) = x^T a \Rightarrow g(x) = 2a$
$J(x) = x^T Ax \Rightarrow g(x) = 2Ax$	$J(x) = x^T Ax \Rightarrow g(x) = 2Ax$

Usando las reglas de la Tabla 3.1. Teniendo la expresión del vector gradiente $g(\boldsymbol{v})$, lo siguiente es igualarlo a cero:

$$g(\boldsymbol{v}) = \frac{\partial J(\boldsymbol{v})}{\partial \boldsymbol{v}} = -2\boldsymbol{S}^T(\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}) + 2\alpha(\boldsymbol{v} - \boldsymbol{m}_v) = 0 \tag{3.15}$$

Para obtener una solución para la ecuación 3.15 primero es necesario reacomodar la ecua-

ción.

$$(\boldsymbol{S}^T\boldsymbol{S} + \alpha\boldsymbol{I})\boldsymbol{v} = \alpha\boldsymbol{m}_v + \boldsymbol{S}^T\boldsymbol{u} \tag{3.16}$$

Sumando y restando el término $\boldsymbol{S}^T\boldsymbol{S}\boldsymbol{m}_v$ del lado derecho de la ecuación 3.16, se puede rescribir la ecuación de arriba como sigue:

$$(\boldsymbol{S}^T\boldsymbol{S} + \alpha\boldsymbol{I})\boldsymbol{v} = \alpha\boldsymbol{m}_v + \boldsymbol{S}^T\boldsymbol{u} + \boldsymbol{S}^T\boldsymbol{S}\boldsymbol{m}_v - \boldsymbol{S}^T\boldsymbol{S}\boldsymbol{m}_v \tag{3.17}$$

$$= (\boldsymbol{S}^T\boldsymbol{S} + \alpha\boldsymbol{I})\boldsymbol{m}_v + \boldsymbol{S}^T(\boldsymbol{u} - \boldsymbol{S}\boldsymbol{m}_v) \tag{3.18}$$

Para determinar la solución de la ecuación 3.18, es necesario pre-multiplicar ambos lados de la ecuación 3.18 por $(\boldsymbol{S}^T\boldsymbol{S} + \alpha\boldsymbol{I})^{-1}$, obteniendo el estimador óptimo deseado CLS:

$$\widehat{\boldsymbol{v}} = \boldsymbol{m}_v + (\boldsymbol{S}^T\boldsymbol{S} + \alpha\boldsymbol{I})^{-1}\boldsymbol{S}^T(\boldsymbol{u} - \boldsymbol{S}\boldsymbol{m}_v) = \boldsymbol{m}_v + \boldsymbol{W}(\boldsymbol{u} - \boldsymbol{m}_v) \tag{3.19}$$

donde la matriz $\mathbf{W}$ es el operador lineal solución deseado:

$$\boldsymbol{W} = (\boldsymbol{S}^T\boldsymbol{S} + \alpha\boldsymbol{I})^{-1}\boldsymbol{S}^T \tag{3.20}$$

3.4. Algoritmo WCLS

WCLS se deriva del algoritmo de mínimos cuadrados restringidos (CLS, por sus siglas en ingles), con la diferencia que cuenta con parámetros ponderados tales como la matriz $\boldsymbol{M}_u$ y $\boldsymbol{M}_v$. La matriz $\boldsymbol{M}_v$ se le conoce como estabilizador, en el sentido que estabiliza y pondera los cambios de la señal y como resultado suavizar. Dependiendo de la ponderación de la matriz $\boldsymbol{M}_v$, existen tres casos específicos de la norma ponderada de $\mathbf{v}$, la norma al cuadrado se definido como:

$$\|\boldsymbol{v}\|^2{}_v = \boldsymbol{v}^2\boldsymbol{M}_v\boldsymbol{v} \tag{3.21}$$

La matriz $\boldsymbol{M}_v$ es igual a la matriz identidad $\mathbf{I}$, para la cual la norma ecludiana se

$$D_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 1 & 0 & 0 & & & & \vdots \\ 0 & -1 & 1 & 0 & \ddots & & & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & -1 & 1 & 0 & \vdots \\ \vdots & & & & 0 & -1 & 1 & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 1 \end{pmatrix}$$

Figura 3.2: Matriz $\boldsymbol{D}_1$

obtiene como sigue:

$$\|\boldsymbol{v}\|^2{}_v = \|\boldsymbol{v}\|^2 = \boldsymbol{v}^T\boldsymbol{v}; \boldsymbol{M}_v = \boldsymbol{I} \tag{3.22}$$

La matriz $\boldsymbol{M}_v$ es una matriz diagonal, en la cual su diagonal tiene asignado una peso en especifico a cada elemento en su diagonal como se muestra en:

$$\|\boldsymbol{v}\|^2{}_v = \sum_{n=1}^{n} = \boldsymbol{M}_{\boldsymbol{V}_n}\boldsymbol{v}_n^2 \tag{3.23}$$

Donde $\boldsymbol{v}_n$ es el n elemento del vector $\boldsymbol{v}$ y $\boldsymbol{M}_{\boldsymbol{v}_n}$ es el n elemento de la matriz $\boldsymbol{M}_v$. La matriz $\boldsymbol{M}_v$ es una matriz no diagonal, en la cual la matriz $\boldsymbol{M}_v$ pude representarse como:

$$\boldsymbol{M}_v = \boldsymbol{I} + \boldsymbol{m}_1\boldsymbol{D}_1^T\boldsymbol{D}_1 \tag{3.24}$$

Donde el superíndice ^{T} representa la transpuesta de la matriz correspondiente, $\mathbf{I}$ es la matriz identidad N-por-N y $\boldsymbol{D}_1$ es el operador de matriz para el cálculo de la primera orden diferencias(Figura 3.2).

El valor $\boldsymbol{m}_1$ es un número real no negativo que indica la importancia relativa del segundo termino en (3.24) que pertenecen a las propiedades de control de *smoothness* en el operador $\boldsymbol{M}_v$. Para resolver el problema inverso del problema mal planteado se impone un criterio o función de costo (3.25).

$$J(\boldsymbol{v}) = J_1(\boldsymbol{v}) + \alpha J_2(\boldsymbol{v}) \tag{3.25}$$

Donde:

$$J_1(\boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}\|^2{}_u \tag{3.26}$$

$$= [(\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}), (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v})]_u \tag{3.27}$$

$$= (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v})^T \boldsymbol{M}_u (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}) \tag{3.28}$$

$$J_2(\boldsymbol{v}) = \|\boldsymbol{v} - \boldsymbol{m}_v\|^2 = [(\boldsymbol{v} - \boldsymbol{m}_v), (\boldsymbol{v} - \boldsymbol{m}_v)] \tag{3.29}$$

y:

$$J_2(\boldsymbol{v}) = \|\boldsymbol{v} - \boldsymbol{m}_v\|^2{}_v \tag{3.30}$$

$$= [(\boldsymbol{v} - \boldsymbol{m}_v), (\boldsymbol{v} - \boldsymbol{m}_v)]_v \tag{3.31}$$

$$= (\boldsymbol{v} - \boldsymbol{m}_v)^T \boldsymbol{M}_v (\boldsymbol{v} - \boldsymbol{m}_v) \tag{3.32}$$

El valor α es el parámetro de regularización, estableciendo una compensación entre la fidelidad de los datos de entrada (criterio J_1) y la sensibilidad del ruido (criterio J_2). La función de coste se puede expresar como sigue:

$$J(\boldsymbol{v}) = (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v})^T \boldsymbol{M}_u (\boldsymbol{u} - \boldsymbol{S}v) \tag{3.33}$$

$$= \alpha(\boldsymbol{v} - \boldsymbol{m}_v)^T \boldsymbol{M}_v (\boldsymbol{v} - \boldsymbol{m}_v)$$

La solución para el problema de optimización (3.33) es dada por:

$$\hat{\boldsymbol{v}} = \underbrace{argmin}_{v \in V} (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v})^T \boldsymbol{M}_u (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{v}) \tag{3.34}$$

$$= \alpha(\boldsymbol{v} - \boldsymbol{m}_v)^T \boldsymbol{M}_v (\boldsymbol{v} - \boldsymbol{m}_v)$$

El estimador $\hat{\boldsymbol{v}}$ se obtiene realizando el cálculo de la expresión (3.34), donde los parámetros

α, $\boldsymbol{M}_U$, y $\boldsymbol{M}_V$ son grados de libertad en el algoritmo, el cual en este caso tiene valores constantes en el método:

$$\begin{aligned} \hat{\boldsymbol{v}} &= \boldsymbol{m}_v + (\boldsymbol{S}^T \boldsymbol{M}_u \boldsymbol{S} + \alpha \boldsymbol{M}_v)^{-1} \boldsymbol{S}^T \boldsymbol{M}_u (\boldsymbol{u} - \boldsymbol{S}\boldsymbol{m}_v) \\ &= \boldsymbol{m}_v + \boldsymbol{W}(\boldsymbol{u} - \boldsymbol{S}\boldsymbol{m}_v) \end{aligned} \tag{3.35}$$

Donde la matriz $\boldsymbol{W}$ determina la solución lineal deseada $\boldsymbol{W}$ el cual permite estimar el sistema de la señal de entrada [23].

$$\boldsymbol{W} = (\boldsymbol{S}^T \boldsymbol{M}_u \boldsymbol{S} + \alpha \boldsymbol{M}_v)^{-1} \boldsymbol{S}^T \boldsymbol{M}_u \tag{3.36}$$

Capítulo 4

Implementación de los algoritmos CLS y WCLS

Es este capítulo se plantea la implementación propuesta para el procesamiento de los algoritmos de CLS y WCLS mencionados en el capítulo 3. De igual manera se realiza una descripción del hardware y software utilizado.

4.1. Bibliotecas

OpenCV

`OpenCV` es una biblioteca de uso académico y comercial multiplataforma con interfaces en C, C++, Python y Java, para el procesamiento de imágenes y vídeo en aplicaciones en tiempo real. `OpenCV` cuenta con un conjunto de rutinas que permite interactuar de manera fácil con los diferentes componentes de las imágenes y vídeo.

CBLAS

`CBLAS` permite utilizar las rutinas disponibles en BLAS(Basic Linear Algebra Subprograms) la cual esta desarrollada en Fortran y permite su utilización en lenguajes como C, C++. Las rutinas disponibles en BLAS se clasifican en tres bloques o niveles. El nivel

1 permite realizar operaciones vector-escalar y vector-vector, el nivel 2 permite realizar operaciones matriz-vector y el nivel 3 permite realizar operaciones matriz-matriz.

CUDA

`CUDA` es una extensión del lenguaje de programación C que permite escribir código que se ejecute en GPUs. `CUDA` fue desarrollado por NVIDIA con el objetivo de utilizar la alta capacidad de paralelismo con la que cuentan las GPUs actuales para su uso en cómputo científico.

cuBLAS

`cuBLAS` (CUDA Basic Lienear Algebra Subroutines) es la versión de BLAS desarrollada para ejecutarse en GPUs. cuBLAS permite la ejecución de rutinas estándares disponibles en BLAS en las GPUs, esta biblioteca cuenta con los tres niveles de operación de `blas`.

4.2. Flujo de procesamiento en paralelo

La Figura 4.1 se presenta un diagrama a bloques en la cual se visualiza el flujo de procesamiento propuesto en este documento de tesis. El diagrama engloba tanto el procesamiento en el *host* como en el GPU(*Device*). De un lado se utiliza las rutinas de la biblioteca `OpenCV` para la carga de las imágenes en la memoria RAM del *host*, posteriormente a la carga de las imágenes aplicando un conjunto de funciones de distorsión utilizando una matriz *toeplitz* para posteriormente agregar ruido aditivo a la imagen. Las matrices de distorsión *toeplitz* se construyeron mediante la Función de Densidad de Probabilidad Gaussiana (PDF, por sus siglas en inglés). Por otro lado se realiza el proceso de reconstructivo mediante la implementación de los algoritmo WCLS Y CLS en el GPU. El GPU realiza el procesamiento de las imágenes y cálculo del estimador $\boldsymbol{W}$ (ver ecuaciones 3.20, 3.36) con el cual se calcula la aproximación deseada de la señal $\boldsymbol{v}$ (imagen original sin

Tabla 4.1: Especificaciones técnicas del GPUs Quadro 200 y Tesla C2075

	Quadro 2000	**Tesla C2075**
CUDA Driver Version / Runtime Version	4.0 / 4.0	4.2 / 4.2
CUDA Capability Major/Minor version number	2.1	2.0
Total amount of global memory	1535 MBytes (1609760768 bytes)	5375 MBytes (5636554752 bytes)
Multiprocessors x CUDA Cores/MP	(3x48) 144 CUDA Cores	(14x32)448 CUDA Cores
GPU Clock Speed	1.58 GHz	1147 MHz (1.15 GHz)
Memory Clock rate	1804.00 Mhz	1566 Mhz
Memory Bus Width	192-bit	384-bit
L2 Cache Size	393216 bytes	786432 bytes
Max Texture Dimension Size (x,y,z)	1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)	1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers	1D=(16384)x2048, 2D=(16384,16384)x2048	1D=(16384)x2048, 2D=(16384,16384)x2048
Total amount of constant memory	65536 bytes	65536 bytes
Total amount of shared memory per block	49152 bytes	49152 bytes
Total number of registers available per block	32768	32768
Warp size	32	32
Maximum number of threads per block	1024	1536
Maximum sizes of each dimension of a block	1024 x 1024 x 64	1024 x 1024 x 64
Maximum sizes of each dimension of a grid	65535 x 65535 x 65535	65535 x 65535 x 65535
Maximum memory pitch	2147483647 bytes	2147483647 bytes
Texture alignment	512 bytes	512 bytes

distorsión). Como se puede observar en las ecuaciones 3.20, 3.36, el cálculo del estimador implica realizar el cálculo de la inversa de uno de sus componentes. El procesamiento de ésta operación es muy demandante computacionalmente hablando, por tal motivo se aprovecho la alta capacidad de procesamiento y paralelismo del GPU para efectuar el cálculo de la inversa y el proceso de reconstrucción.

4.3. Implementación Paralela de WCLS

Para la implementación del algoritmo WCLS se creó una función la cual permite especificar la imagen a tratar en el GPU, esta función se denomino `wcls` la cual es invocada desde el *host* ejecutando un conjunto de operaciones como la carga de elementos en la memoria del GPU. Esta rutina aún cuando se invoca en el *host*, realiza tareas que se ejecutan tanto en el *host* como en el GPU permitiendo realizar las operaciones necesarias para

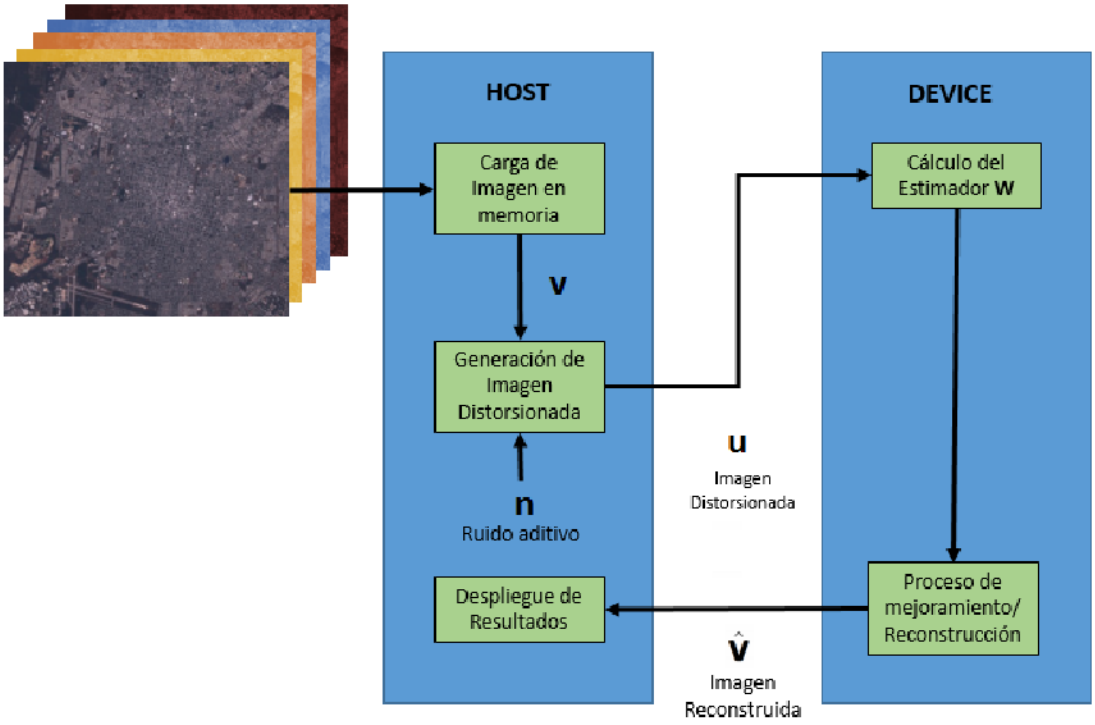


Figura 4.1: Descripción general del flujo de procesamiento

calcular el estimador $\boldsymbol{W}$ para el proceso de mejoramiento/reconstrucción de la imagen deseada (ver Ecuación 3.3). En la Figura 4.2 se muestra la rutina `wcls` y en la Tabla 4.3 se describen los parámetros requeridos para su ejecución. Como se describió en el capítulo

```
void wcls(float* h_mv, float* h_u, int Nx, float* h_Sx, int Ny, float*
h_Sy, float alpha, float No)
```

Figura 4.2: Rutina *wcls*

3, el cálculo del estimador $\boldsymbol{W}$ es considerar la incorporación de ruido Gausiano (AWGN). En la Figura 4.2 se observa que entre los parámetros de funcionamiento se encuentran dos matrices *toeplitz*($\boldsymbol{S}_x$ y $\boldsymbol{S}_y$); estas matrices son generadas en el *host* dentro la función `wcls` de manera automática. En la Figura 4.3 se muestra el encabezado de la función `toeplitz` y en la Tabla 4.2 los parámetros requeridos para su invocación. Esta función genera una matriz *toeplitz* en la memoria del *host* donde las diagonales de la matriz contienen valores de la Función de Densidad de Probabilidad Gaussiana que se encuentra almacenado en la

memoria (vector `diag`) del *host*. La dimensión de la matriz *toeplitz* es variable dependiendo del valor especificado en el parámetro `dim`, y éste puede definir la dimensión de la matriz cambiando su valor (La matriz *toeplitz* es de dimensión *dim x dim*). Dada una imagen de dimensión $m\ x\ n$ la función `wcls` requiere dos matrices *toeplitz* de dimensiones $m\ x\ m$ y $n\ x\ n$ las cuales se utilizarán para el cálculo de los estimadores $\boldsymbol{W}_x$ y $\boldsymbol{W}_y$.

```
void toeplitz (int dim, float *mat, const float *diag)
```

Figura 4.3: Rutina *toeplitz* y sus parámetros

Tabla 4.2: Parámetros de la función `toeplitz`

Parámetro	**Tipo**	**Descripción**
`dim`	Entrada	Número de filas/columnas de la matriz *toeplitz*
`mat`	Salida	Matriz *toeplitz* resultante
`diag`	Entrada	Ventor con los valores para poblar la diagonal de la matriz *toeplitz*

Una vez calculadas las matrices *toeplitz*, estas se utilizan para el proceso de distorsión; para realizar esto se utiliza la rutina `cblas_sgemm` de la biblioteca `CBLAS` la cual realiza el cómputo de $\boldsymbol{C} = \alpha\boldsymbol{A}\boldsymbol{B} + \beta\boldsymbol{C}$, donde $\boldsymbol{A}$ es la imagen cargada previamente en el *host* y $\boldsymbol{B}$ la matriz *toeplitz* de distorsión (En esta etapa la matriz $\boldsymbol{C}$ contiene valor del ruido aditivo y $\alpha = 1$ y $\beta = 0$). En la Figura 4.5 se muestra la cabecera con los parámetros requeridos para el funcionamiento de la rutina `cblas_sgemm`. El ruido aditivo fue generado previamente en Matlab y guardado en un archivo de texto plano para su uso en cada una de las bandas, esto para efectos de validación del sistema. Se realizó el cálculo del ruido aditivo por cada una de las bandas con la relación *SNR* (Signal Noise Ratio) de 1, 5, 10, 15, 20 y 25 guardando a su vez la varianza del ruido que será utilizado posteriormente para el proceso de reconstrucción. En la Figura 4.4 se ilustra la función utilizada en Matlab. Es importante recalcar que al compilar el código que se ejecuta en el *host* se utilizó la bandera de *-O3* esta bandera permite mejorar la ejecución del código secuencial en el *host*. Al utilizar esta

bandera de obtimización el compilador (g++) intenta mejorar el rendimiento a expensas de tiempo de compilación y, en ocasión inhabilita la capacidad de depurar el programa.

```
SNR =i;  % i=5,10,15,20,25
[N,M] = size(img);
med = mean(mean(img)) ;
    var2=0;
    for i=1:N
      for j=1:M
       var2= var2 + (img(i,j)-med)^2;
      end
    end
var2=((1/(N*M)))*var2;
NO=var2/(10^(SNR/10));
noise= randn(M*N,1).*sqrt(NO);
```

Figura 4.4: Función para el cálculo de ruido aditivo en Matlab

```
void cblas_sgemm(const enum CBLAS_ORDER Order, const enum
CBLAS_TRANSPOSE TransA, const enum CBLAS_TRANSPOSE TransB, const int
M, const int N, const int K, const float alpha, const float A, const int
lda, const float B, const int ldb, const float beta, float C, const int
ldc)
```

Figura 4.5: Rutina *cblas_sgemm* y sus parámetros

Posteriormente se ubican los componentes necesarios para el cálculo del estimador y así iniciar el proceso de reconstrucción. Para realizar el movimiento de los elementos que se encuentran en la memoria del *host* a la memoria del GPU, primero es necesario reservar el espacio de memoria para cada elemento en el GPU, para posteriormente ubicar los elementos que se desean copiar de la memoria del *host*. El lenguaje `CUDA` proporciona la rutina `cudaMalloc()` para reservar espacios de memoria, el funcionamiento de esta rutina es muy similar a su equivalente en C (`malloc()`), pero con la diferencia que esta reserva el espacio de memoria en el GPU. Una vez que el espacio de memoria se encuentra reservado se puede realizar el proceso de copiado de las matrices *toeplitz* y la imagen a procesar con

el algoritmo WCLS. Para realizar el copiado de elementos entre la memoria del *host* y el GPU, `CUDA` cuenta con la rutina `cudaMemcpy()` que copia los elementos alojados en la memoria del *host* o GPU de un punto a otro. La función `cudaMemcpy()` a diferencia de su equivalente en C (`memcpy()`) que copia los elementos dentro la misma memoria del *host*, `cudaMemcpy()` permite realizar la copia de elementos entre memorias, es decir, permite copiar un elemento de la memoria *host* a la memoria del GPU y viceversa. El sentido en el que se desee realizar el movimiento se puede especificar con el parámetro obligatorio `cudaMemcpyDeviceToHost` o `cudaMemcpyHostToDevice` de la rutina `cudaMemcpy()`.

Tabla 4.3: Parámetros de la rutina `wcls`

Parámetro	**Tipo**	**Descripción**
`h_mv`	Entrada/Salida	Como entrada se ingresa la imagen original sin distorsión. Como salida retorna la imagen reconstruida
`h_u`	Entrada	Imagen distorsionada
`Nx`	Entrada	Número de columnas de la imagen a reconstruir
`h_Sx`	Entrada	Matriz *toeplitz* de dimensión `n x n`
`Ny`	Entrada	Número de filas de la imagen a reconstruir
`h_Sy`	Entrada	Matriz *toeplitz* de dimensión `m x m`
`alpha`	Entrada	Valor α utilizado en el estimador
`No`	Entrada	Varianza del ruido

Una vez alojados los componentes necesarios (Imagen a tratar y matrices *toeplitz*) en la memoria del GPU es posible iniciar el proceso de reconstrucción. Para iniciar el proceso de reconstrucción primero se calcula el estimador $\mathbf{W}$. El cálculo del estimador se realiza en dos etapas; en la primera se calcula el elemento $\boldsymbol{C} = (\boldsymbol{S}^{+}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v)^{-1}$, y posteriormente se realiza la multiplicación de $\boldsymbol{C}S^{+}\boldsymbol{M}_u$. Como se puede observar para el cálculo de $\boldsymbol{C}$ es necesario contar con las matrices de distorsión (matrices *toeplitz*) y de las matrices $\boldsymbol{M}_u$ y $\boldsymbol{M}_v$, estas ultimas dos se generan directamente en el GPU. Para generar las matrices $\boldsymbol{M}_u$ y $\boldsymbol{M}S_v$ se implementaron dos *kernels* denominados `matrixMv` y `matrixMu` (Figura 4.8). Un

`kernel` es muy parecido a una función estándar en C/C++, pero con la diferencia que se le antepone la palabra reservada `__global__` en lugar del tipo de dato, la palabra reservada `__global__` le indica al compilador `nvcc` que esa sección del código será ejecutado en el GPU permitiendo la ejecución de código en paralelo. Aun cuando un *kernel* se invoca en el *host*, la ejecución de éste se efectúa en el GPU debido a que fue compilado con `nvcc`, cada *kernel* utiliza un conjunto de hilos también conocido como *threads* en inglés para realizar el procesamiento (ver Figura 4.9). La cantidad de *threads* a utilizar es definido por el desarrollador según las necesidades de procesamiento. Es importante mencionar que se tiene que contemplar cuantos *threads* puede invocar simultáneamente, ya que el hardware (GPU) solo puede ejecutar un número limitado de *threads* simultáneamente.

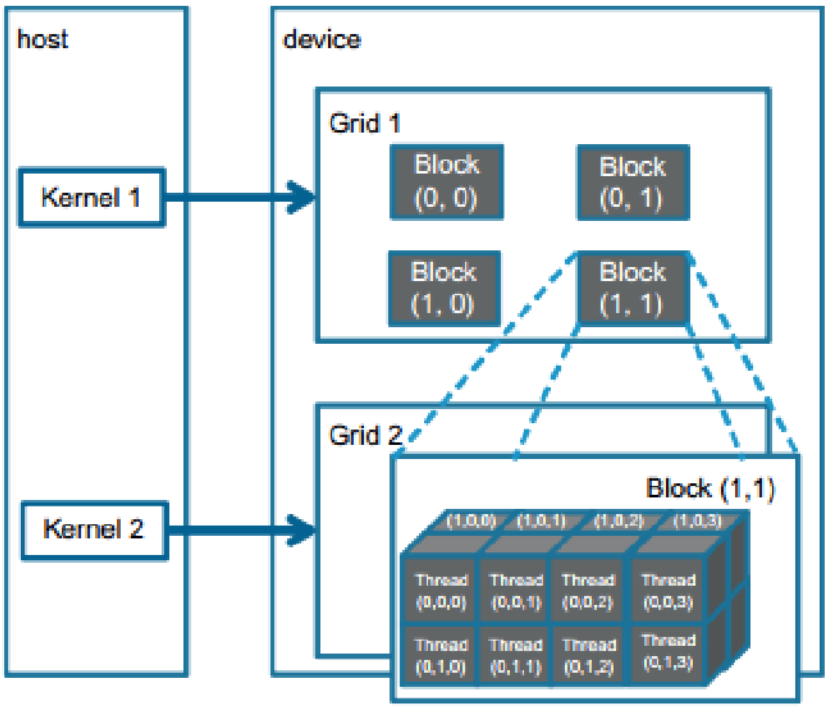


Figura 4.6: Distribución de los hilos en el GPU

Los *kernels* `matrixMu` y `matrixMv` generan las matrices $\boldsymbol{M}_u$ y $\boldsymbol{M}_v$ directamente en la memoria del GPU con el objetivo de disminuir la copia de los elementos entre la memoria del *host* y la memoria del GPU. El motivo principal de evitar lo anteriormente mencionado es que el movimiento de elementos entre la memoria del *host* y la memoria del GPU puede

llevar varios cientos incluso miles de ciclos de reloj antes de finalizar la transferencia de datos lo que se refleja en más tiempo de procesamiento. La disposición de los *threads* dentro de los *kernels* se organiza en dos niveles o jerarquías. En el primer nivel se encuentra lo que se le conoce como *grid* la cual se conforma por uno o más *blocks*, a su vez, cada *block* está constituido por uno o más *threads*. La disposición de los *blocks* dentro la *grid* puede ser unidimensional o bidimensional, es decir, una *grid* se puede conformar de un único *block* hasta *m x n blocks*. A diferencia de la disposición de los *blocks* dentro la *grid*, la disposición de los *threads* dentro de cada *block* puede ser unidimensional, bidimensional o tridimensional (ver Figura 4.6). Cada *block* dentro de la *grid* comparten un **ID** único o *index* interno, que es accesible únicamente dentro del *kernel* a través de la variable predefinida `blockIdx` de igual manera cada *thread* cuenta con un **ID** el cual permite identificar de manera única a cada *thread* y es accesible a través de la variable predefinida `threadIdx`. Otra variable que es accesible dentro el *kernel* son las variables `gridDim` y `blockDim`. El valor de `gridDim` especifica la cantidad de *blocks* con la que cuenta la *grid* y el valor `blockDim` indica la cantidad de *threads* con los que cuenta cada *block* (ver figura 4.7).

La cantidad de *blocks* requeridos para el cálculo de las matrices $\boldsymbol{M}_u$ y $\boldsymbol{M}_v$ se realiza de manera automática dependiendo de las dimensiones de las matrices a generar. La implementación emplea un *thread* por cada elemento de la matriz a generar, es decir, si la matriz es de dimensión *m x m* se utilizarán la misma cantidad de *threads*. La cantidad de *threads* por *block* depende del hardware, en este estudio la cantidad de *threads* por *block* es de 1024 (*blocks* de 32 x 32). Como se puede ver en la Tabla 4.1, la cantidad de *threads* por *block* en el GPU Tesla C2075 es de 1536; esto indica que se pueden utilizar más de 1024 *threads* por *block*, en esta implementación se manejaron 1024 *threads* por *block* como límite con el objetivo de mantener el tamaño mínimo recomendado de 32 *threads* por *block* [20]. En la Figura 4.8 la matriz $\boldsymbol{M}_u$ se calcula únicamente colocando el valor de uno entre `No` (varianza del ruido) en su diagonal principal y cero en todo el resto de los elementos. El cálculo de la matriz $\boldsymbol{M}_v$ requiere de un vector (parámetro `d_max`) de la misma dimensión principal *lda* del la matriz $\boldsymbol{M}_v$ este vector contiene puro ceros ha

excepción del primer elemento que con tiene el valor dos, y su segundo elemento menos uno. Utilizando la sentencia condicional `if` se evalúa la posición (i, j) para determinar si el valor a colocar en $d\_M_v(i, j)$ será: menos uno, dos o cero.

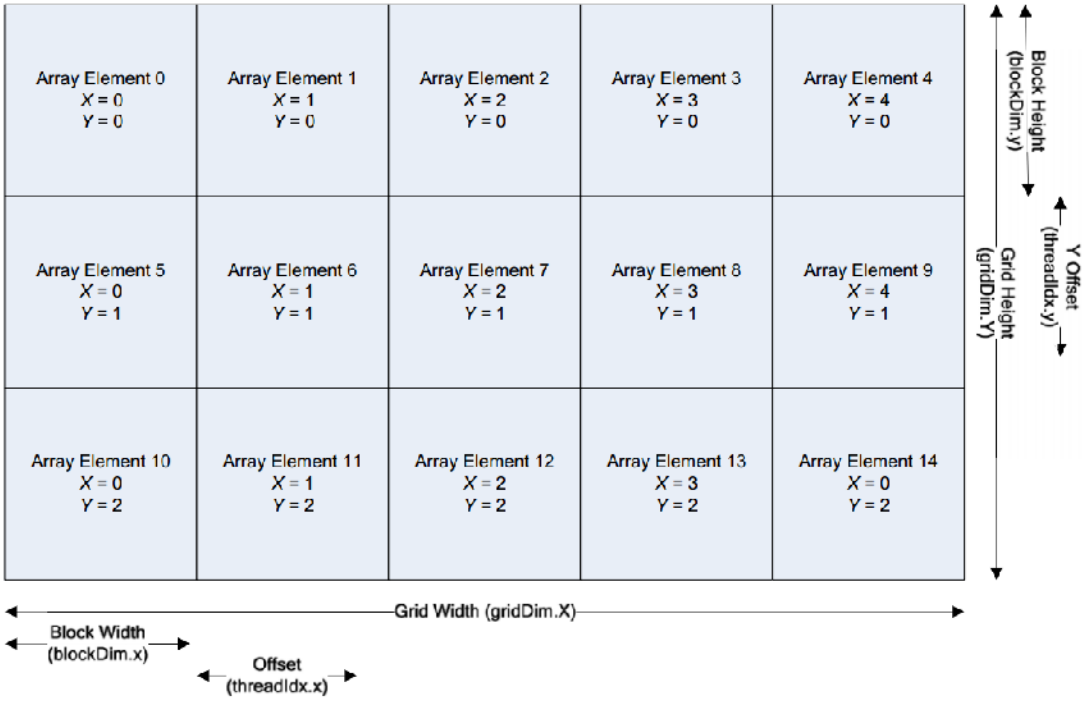


Figura 4.7: Representación de las dimensiones de una Grid, Block y Thread dentro de un kernel ejecutado en el GPU

Para el cálculo de $(\boldsymbol{S}^{+}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v)^{-1}$ primero se realizó el procesamiento de $(\boldsymbol{S}^{+}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v)$ para posteriormente calcular su inversa (ambas operaciones se ejecutan en el GPU). Para procesar la operación de $(\boldsymbol{S}^{+}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v)$ se utiliza la función `cublasSgemm` de la biblioteca `cuBLAS` la cual permite realizar la operación matricial $\boldsymbol{C} = \alpha op(\boldsymbol{A})op(\boldsymbol{B})+\beta\boldsymbol{C}$. Como se puede observar no es posible realizar la operación $(\boldsymbol{S}^{+}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v)$ de manera directa con la rutina `cublasSgemm`; para solucionar esto primero se efectuó la multiplicación de $(\boldsymbol{S}^{+}\boldsymbol{M}_u)$ utilizando la rutina `cublasSgemm` y posteriormente el resultado se utiliza para realizar $(\boldsymbol{X}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v)$, donde $\boldsymbol{X}$ es el resultado de $(\boldsymbol{S}^{+}\boldsymbol{M}_u)$, y de esta manera utilizar `cublasSgemm` para obtener $\boldsymbol{S}^{+}\boldsymbol{M}_u\boldsymbol{S}+\alpha\boldsymbol{M}_v$. Ya obtenido el resultado $\boldsymbol{C}$ se procede a realizar el cálculo de la inversa de $\boldsymbol{C}$. Para realizar el cálculo de la inversa se utilizó una variante de la descomposición de **LU** conocida como *block LU* para la descomposición

```
__global__ void matrixMv(int N, float* d_Mv, const float* d_max){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        d_Mv[i * N + j] = (i ≥ j)? d_max[i - j] : d_max[j - i];
}
__global__ void matrixMu(int N, float* Mu, float No){
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        int j = blockIdx.y * blockDim.y + threadIdx.y;
    if(i < N && j < N)
        Mu[i * N + j] = (i == j)? No : 0.0;
}
```

Figura 4.8: Kernels matrixMv y matrixMu para el cálculo de las matrices M_v y M_u

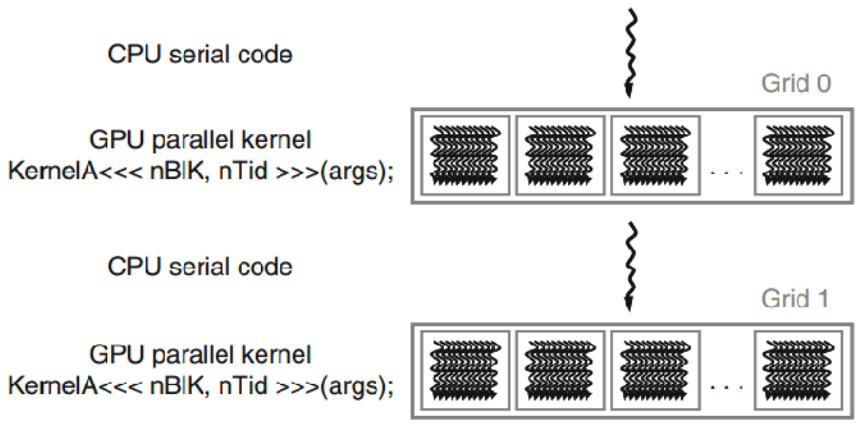


Figura 4.9: Ejemplo de la ejecución de un programa en CUDA ejecutando diferentes *kernels*

de $\boldsymbol{C}$ en dos matrices triangulares, una es una matriz triangular inferior $\mathbf{L}$, y la segunda una matriz triangular superior $\mathbf{U}$. El algoritmo *block LU* (ver Algoritmo 1) calcula la factorización $\boldsymbol{A} = \boldsymbol{LU} \in \mathbb{R}^{nxn}$ mediante una implementación de producto externo en bloques, utilizando un tamaño de bloque r. Una vez factorizado $\boldsymbol{C}$, donde $\boldsymbol{C}$ es igual a

Pseudocodigo 1 Algoritmo BlockLU

repeat
Factor $A_{11} = L_{11}U_{11}$
Resolver $L_{11}U_{12} = A_{12}$ *para* U_{12}
Resolver $L_{21}U_{11} = A_{21}$ *para* L_{21}
Para $S = A_{22} - L_{21}U_{12}$
until S para obtener L_{22} y $U_2 2$

$(\boldsymbol{S}^{+}\boldsymbol{M}_{u}\boldsymbol{S} + \alpha\boldsymbol{M}_{v})$ se realiza el cálculo de la inversa de $\boldsymbol{C}$. El cálculo de la inversa de $\boldsymbol{C}$ se puede plantear como la solución de un sistema de ecuaciones lineales utilizando los factores $\boldsymbol{L}$ y $\boldsymbol{U}$. El planteamiento del sistema de ecuaciones lineales a resolver queda de la siguiente manera $\boldsymbol{Ly} = \boldsymbol{b}$, donde $\boldsymbol{b}$ es una matriz identidad de la misma dimensión de $\boldsymbol{C}$, y posteriormente se resuelve el conjunto de ecuaciones lineas $\boldsymbol{Ux} = \boldsymbol{y}$, donde $\boldsymbol{x}$ es la la inversa del elemento deseado en este caso $(\boldsymbol{S}^{+}\boldsymbol{M}_{u}\boldsymbol{S} + \alpha\boldsymbol{M}_{v})$(a la solución del primer conjunto de ecuaciones se le conoce como *forward substitution* y al segundo como *back substitution*). La solución del sistema de ecuaciones lineales se computo en el GPU; para esto se utiliza la rutina `cublasStrsm` de la biblioteca `cuBLAS` la cual permite resolver sistemas lineales triangulares. La rutina `cublasStrsm` permite resolver sistemas triangulares como $op(\boldsymbol{A})\boldsymbol{X} = \alpha\boldsymbol{B}$ con el parámetro `CUBLAS_SIDE_LEFT` y $\boldsymbol{X}op(\boldsymbol{A}) = \alpha\boldsymbol{B}$ con `CUBLAS_SIDE_RIGHT`, donde $\boldsymbol{A}$ es una matriz triangular; $\boldsymbol{X}$, $\boldsymbol{B}$ son matrices de dimensión $m\ x\ n$ y α es un escalar. En la Figura 4.10 se muestra la cabecera de la rutina `cublasStrsm` y en la Tabla 4.4 se describen sus parámetros.

```
cublasStatus_t cublasStrsm(cublasHandle_t handle, cublasSideMode_t side,
cublasFillMode_t uplo,cublasOperation_t trans, cublasDiagType_t diag,int
m, int n,const float alpha, const float A, int lda, float B, int ldb)
```

Figura 4.10: Rutina *cublasStrsm*

Calculado la inversa ya es posible obtener el estimador $\boldsymbol{W}$, el cálculo del estimador se realiza con la operación de $\boldsymbol{W} = \boldsymbol{X}\boldsymbol{S}^{+}\boldsymbol{M}_{u}$, donde $\boldsymbol{X}$ es igual a $(\boldsymbol{S}^{+}\boldsymbol{M}_{u}\boldsymbol{S} + \alpha\boldsymbol{M}_{v})^{-1}$ con la rutina `cublasSgemm` de la biblioteca `cuBLAS`. Con el estimador calculado se puede realizar el proceso de mejoramiento/reconstrucción procesando $\hat{\boldsymbol{v}} = \boldsymbol{n}\boldsymbol{W}_{x}$ (para la reconstrucción en x) y posteriormente $\hat{\boldsymbol{v}} = \hat{\boldsymbol{v}}\boldsymbol{W}_{y}$ (para le reconstrucción en y), donde $\boldsymbol{n}$ es la imagen distorsionada con ruido aditivo generada en el *host*. El proceso de mejoramiento/-reconstrucción planteado en este documento de tesis se puede resumir en el pseudocódigo 2.

Tabla 4.4: Parámetros de la función `cublasStrsm`

Parámetro	Memoria	Tipo	Descripción
`handle`			Manejador para la biblioteca `cuBLAS`
`side`			Indica si la matriz **A** se encuentra a la izquierda o derecha de **X**
`uplo`			Indica si la matriz **A** los datos son almacenados en la parte inferior triangular o la parte superior triangular
`trans`		Entrada	Especifica si la matriz op(**A**) se tomará como transpuesta
`dig`		Entrada	Indica si los elementos en la diagonal principal de la matriz **A** deberán se utilizados o no
`m`		Entrada	Número de filas de la matriz **B**
`n`		Entrada	Número de columnas de la matriz **A**
`alpha`	host o device	Entrada	Escalar utilizado para la multiplicación, si `alpha==0` entonces **A** no se referencia y **B** no tiene que ser una entrada valida
`A`	device	Entrada	Matriz **A** de dimensión *lda x m* con `lda>=max(1,m)` si `side == CUBLAS_SIDE_LEFT` y `lda x n` con `lda>=max(1,n)`
`lda`		Entrada	Dimensión principal de la matriz bidimensional utilizado para almacenar la matriz **A**
`B`	device	Entrada/Salida	Matriz **B** de dimensión `ldb x n` con `ldb>=max(1,m)`
`ldb`		Entreda	Dimensión principal de la matriz bidimensional utilizado para almacenar la matriz **B**

Pseudocodigo 2 Algoritmo de Reconstrución

repeat
- $\boldsymbol{A}$ ←Cargar imagen multiespectral/hiperespectral
- Calcular $\boldsymbol{S}_x$ y $\boldsymbol{S}_y$
- $\boldsymbol{n}$ ← Distorsionar $\boldsymbol{A}$ aplicando las matrices de distorsión $\boldsymbol{S}_x$ y $\boldsymbol{S}_y$
- $\boldsymbol{u}$ ← Agregar ruido aditivo
- $\boldsymbol{W}$ ← Calcular estimador
- $\hat{\boldsymbol{v}}$ ← Proceso de reconstrucción para obtener la aproximación de $\boldsymbol{A}$

until N bandas

Tabla 4.5: Parámetros de la función `cublasSgemm`

Parámetro	Memoria	Tipo	Descripción
`handle`			Manejador para la biblioteca `cuBLAS`
`transa`			Especifica si la matriz op(**A**) se tomara como transpuesta
`transb`			Especifica si la matriz op(**B**) se tomara como transpuesta
`m`		Entrada	Número de filas de las matrices op(**A**) y **C**
`n`		Entrada	Número de columnas de las matrices op(**B**) y **C**
`k`		Entrada	Número de columnas de las matrices op(**A**) y filas de op(**B**)
`alpha`	host o device	Entrada	Escalar utilizado en la multiplicación
`A`		Entrada	Matriz **A** de dimensión `lda x k` con `lda>=max(1,m)` si `transa == CUBLAS_OP_N` y `lda x m` con `lda>=max(1,k)`
`lda`	device	Entrada	Dimensión principal de la matriz bidimensional utilizado para almacenar la matriz **A**
`B`		Entrada	Escalar utilizado en la multiplicación
`A`		Entrada	Matriz **A** de dimensión `ldb x n` con `ldb>=max(1,k)` si `transa == CUBLAS_OP_N` y `ldb x k` con `ldb>=max(1,n)`
`ldb`	device	Entrada	Dimensión principal de la matriz bidimensional utilizado para almacenar la matriz **B**
`beta`	device	Entrada	Escalar utilizado para la multiplicación, si `beta==0` entonces **C** no tiene que ser una entrada válida
`C`		Entrada/Salida	Matriz de dimension `ldc x n` con `ldc>=max(1,m)`
`ldc`		Entrada	Dimensión principal de la matriz bidimensional utilizado para almacenar la matriz **C**

4.4. Implementación Paralela de CLS

La propuesta de implementación en este documento de tesis del algoritmo CLS es muy similar a la propuesta anteriormente planteada para WCLS; la diferencia radica en

el estimador $\boldsymbol{W} = (\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})^{-1}\boldsymbol{S}^{+}$ que no requiere de las matrices de $\boldsymbol{M}_u$ y $\boldsymbol{M}_v$ para su cálculo por consiguiente el cálculo se procesa de manera más directa. Se planteó una función denominada `cls` (ver Figura 4.11) la cual carga en memoria los elementos necesarios para el cálculo del estimador y inicia el proceso de reconstrucción. En la Tabla 4.6 se describen los parámetros requeridos para convocar a la función `cls`. El procesamiento de algoritmo CLS inicia de manera similar a la implementación de WCLS planteado con anterioridad de esta manera se realiza el proceso de degradación sobre la banda de la imagen multiespectral o hiperespectral seleccionada en el *host*. Como resultado de la degradación se obtiene una imagen sintética la cual será tratada con el proceso de reconstrucción (algoritmo CLS). El estimador se calcula de manera similar al estimador utilizado en el algoritmo de WCLS, la primera etapa consiste en el cálculo de $(\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})$ donde $\boldsymbol{S}$ representa las matrices de distorsión *toeplitz* de dimensiones $m\ x\ m$ $(\boldsymbol{S}_x)$ y $n\ x\ n$ $(\boldsymbol{S}_y)$. Como se puede observar en el flujo de procesamiento en la Figura 4.1 las matrices de distorsión se generan con código que se ejecuta en el *host* con la función de densidad de probabilidad Gaussiana.

```
void cls(float* h_Sx, int Nx, float* h_Sy, int Ny, float* h_u, float alpha, float*
h_v)
```

Figura 4.11: Rutina `cls` para el proceso de reconstrucción en el CPU

Una vez cargada la imagen y generada las matrices de distorsión, éstas son copiadas a la memoria de el GPU utilizando la rutina `cudaMemcpy()`. Como se describió en el capítulo anterior, el estimador $\boldsymbol{W}$ requiere del cálculo de $(\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})^{-1}\boldsymbol{S}^{+}$. Para obtener el estimador primero se procesa $(\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})$ y posteriormente su inversa. El cálculo del estimador $\boldsymbol{W}$ se realiza en dos etapas. La primera etapa consiste en realizar las operaciones $(\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})$. Para esto se utilizó la rutina `cublasSgemm` de la biblioteca `cuBLAS` la cual permite realizar la operación matricial $\boldsymbol{C} = \alpha op(\boldsymbol{A})op(\boldsymbol{B}) + \beta\boldsymbol{C}$; esta rutina se invoca en el *host*, pero su ejecución se realiza en el GPU. Antes de utilizar la rutina `cublasSgemm` es necesario generar en la memoria del GPU una matriz identidad la cual utilizará en el cálculo del estimador $\boldsymbol{W}$. Para generar la matriz identidad se plantea un *kernel* que genera una matriz identidad directamente en la memora del GPU; si la matriz identidad se creara en el *host* se tendría que copiar de la memoria del *host* a la memoria del GPU

utilizando la rutina `cudaMemcpy()`. En la Figura 4.12 se muestra la cabecera del *kernel* `identityMatrix` donde se observa que entre sus parámetros sólo requiere el espacio de memoria en el GPU para almacenar la matriz identidad resultante y la dimensión principal de matriz identidad o `lda`. El *kernel* `identityMatrix` utiliza un *thread* por cada elemento de la matriz donde *thread* evalúa su *index* buscando aquellos donde tanto el *index* i y j sean iguales colocando es esa posición el valor uno y en el resto cero. Una vez generada en la memoria del GPU la matriz identidad ya es posible calcular $(\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})$ con la rutina `cublasSgemm`. El procesamiento se realiza en el GPU y como resultado de `cublasSgemm` se obtiene una matriz $\boldsymbol{C}$ la cual se factoriza con la descomposición de block LU ($\boldsymbol{C} = \boldsymbol{L}\boldsymbol{U}$). Ya realizado la factorización de la matriz $\boldsymbol{C}$ se plantea un conjunto de ecuaciones lineales para el cálculo de la inversa. El planteamiento queda de la siguiente manera $\boldsymbol{L}\boldsymbol{z} = \boldsymbol{b}$, donde $\boldsymbol{b}$ es una matriz identidad de la misma dimensión de $\boldsymbol{C}$, y posteriormente se resuelve el conjunto de ecuaciones lineas $\boldsymbol{U}x = \boldsymbol{z}$, donde $\boldsymbol{x}$ es la la inversa del elemento deseado en este caso $(\boldsymbol{S}^{+}\boldsymbol{S} + \alpha\boldsymbol{I})$. Con el estimador calculado se puede iniciar el proceso de mejoramiento/reconstrucción procesando $\hat{\boldsymbol{v}}_{tmp} = \boldsymbol{n}\boldsymbol{W}_x$ y posteriormente $\hat{\boldsymbol{v}} = \hat{\boldsymbol{v}}_{tmp}\boldsymbol{W}_y$ utilizando la rutina `cublasSgemm`, donde $\boldsymbol{n}$ es la imagen distorsionada con ruido aditivo generada en el *host* y $\hat{\boldsymbol{v}}$ es la imagen reconstruida o aproximación de $\boldsymbol{v}$.

```
__global__ void identityMatrix(int lda, float I)
```

Figura 4.12: Kernel identityMatrix

Para poder verificar el nivel de mejoramiento/reconstrucción tanto, para el algoritmo WCLS y CLS, se cálculo la relación de mejora de salida ruido señal (**IOSNR**, por sus siglas en inglés). El *IOSNR* es utilizado para describir el nivel de mejoramiento/reconstrucción que sufre una imagen se expresa en **dB** midiendo la sensibilidad de una señal (Imagen) ante el ruido.

El IOSNR se calcula comparando la relación que con respecto la imagen de entrada y salida, se evalúa *pixel* por *pixel* (ver Figura 4.13).

Tabla 4.6: Parámetros de la rutina `cls`

Parámetro	**Tipo**	**Descripción**
`h_Sx`	Entrada/Salida	Matriz *toeplitz* de dimensión $n\ x\ n$
`Nx`	Entrada	Dimensión principal de `h_Sx`
`h_Sy`	Entrada	Matriz *toeplitz* de dimensión $m\ x\ m$
`Ny`	Entrada	Dimensión principal de `h_Sy`
`h_u`	Entrada	Imagen distorsionada a reconstruir de dimensión $m\ x\ n$
`alpha`	Entrada	Valor α utilizado en el estimador
`h_v`	Salida	Imagen trata con el algoritmo CLS. Imagen reconstruida

$$IOSNR = 10 \log_{10} \left[\frac{\sum_{x=1}^{n} \sum_{y=1}^{m} (u(x,y) - f(x,y))^2}{\sum_{x=1}^{n} \sum_{y=1}^{m} (\hat{f}(x,y) - f(x,y))^2} \right]$$

Figura 4.13: Función IOSNR

Capítulo 5

Resultados experimentales

En este capítulo se describen los experimentos realizados y se presentan los resultados de mejoramiento/reconstrucción de los algoritmos CLS y su versión mejorada WCLS. Los resultados expuestos en este capítulo se dividen en dos categorías. En la primera se presentan los resultados experimentales desde el punto de vista del mejoramiento en la salida de la señal-ruido permitiendo comprar los algoritmos CLS y WCLS al momento de procesar imágenes multiespectrales e hiperespectrales. En la segunda se enfoca en el tiempo de procesamiento, es decir se calcula el tiempo requerido para procesar las imágenes multiespectrales y hiperespectrales utilizando técnicas de cómputo en paralelo con GPU en comparación de técnicas de procesamiento con CPUs.

5.1. Descripción de las Imágenes Multiespectrales

Las imágenes multiespectrales e hiperespectrales de percepción remota utilizadas en este documento de tesis son de los sensores AVIRIS (Airborne Visible/In-reflected spectrum), *Landsat 7*, y el escáner *Flightline C1* (FLC1, por sus siglas en inglés). Una de las imágenes a estudiar es la imagen multiespectral proveniente de uno de los satélites del programa *Landsat*; el programa *Landsat* ofrece el registro global continuo más largo de la superficie de la tierra y sus imágenes se encuentran disponibles en la página web http://landsat.org. Otra imagen se obtiene del escáner *Flightline C1* del condado

del Sur de Tippecanoe, Indiana de 1966, la cual se encuentra disponible en la página web https://engineering.purdue.edu/ biehl/MultiSpec/; por último, la imagen hyperespectral del sensor AVIRIS de la zona del este de Ontario USA disponibles en la página http://aviris.jpl.nasa.gov/. A continuación, se describen los tres casos de estudio para ilustrar el potencial de los GPUs en el proceso de mejoramiento/reconstrucción de imágenes de percepción remota en comparativa con el procesamiento tradicional con CPUs.

El primer caso de estudio pertenece a la imagen hyperespectral del sensor AVIRIS el cual fue el primer sensor en medir el espectro solar de 400 a 2500 nanómetros con un intervalo de 10 nanómetros proporcionado imágenes hiperespectrales de 224 bandas [24]. La imagen hiperespectral a utilizar en esta tesis es de la zona de Ontario de una dimensión de 677 x 2932 píxeles (1,984,964 píxeles por banda) para un tamaño de 1.94 GB.

Figura 5.1: *False color* extraído de la imagen hiperespectral del sensor AVIRIS

El segundo caso de estudio corresponde a la imagen multiespectral adquirida por el satélite *Landsat 7* de la NASA. La imagen a procesar corresponde a la ciudad de Mérida, Yucatán, México adquirida el año 2005. Se seleccionó esta imagen ya que permite visualizar zonas urbanas en conjunto de áreas verdes. Para las pruebas se utilizó una sección de la imagen de 847 x 1794 píxeles (1,519,258 píxeles por banda) y 8 bandas espectrales, en la Figura 5.2 se presenta el *false color* el cual es una representación lo más cercana posible al color verdadero de la escena combinando las bandas que se encuentran en las longitudes de onda 0.63-0.69, 0.52-0.60 y 0.45-0.52 micrómetros para desplegar los colores rojo, verde y azul respectivamente.

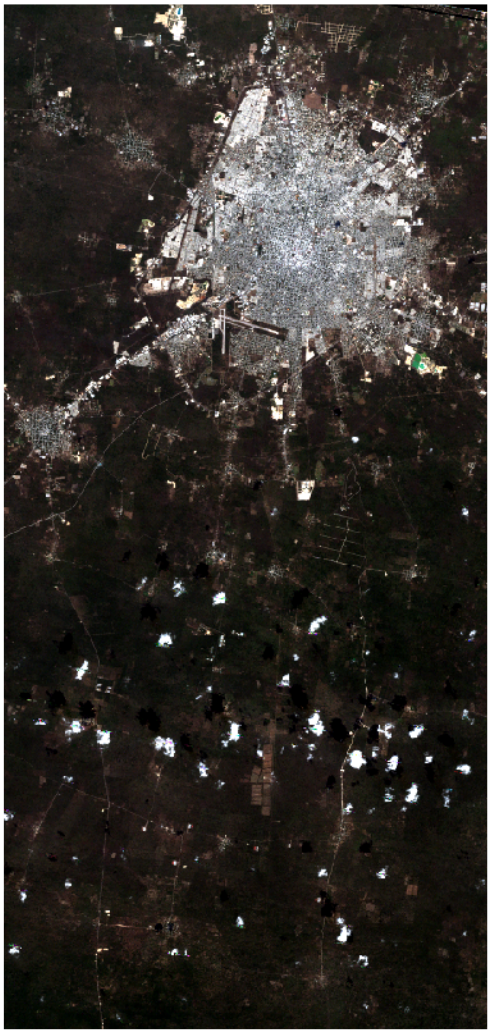


Figura 5.2: Escenas del área de Mérida, Yucatán, con *false color* con los colores casi real

El tercer caso de estudio es una imagen multiespectral de doce bandas cubriendo el rango de longitud de onda de los 0.40 a 1 micrómetros de una resolución de 220 x 949 (208,780 píxeles por banda) obtenida del sensor *Flightline C1*. En Figura 5.3 se presenta el *false color* de la imagen adquirida por el sensor *Flightline C1* siendo esta la imagen mas pequeña tratada en éste documento de tesis.

Figura 5.3: *False color* extraído de la de imagen multiespectral del sensor *Flightline C1*

5.1.1. Resultados Experimentales

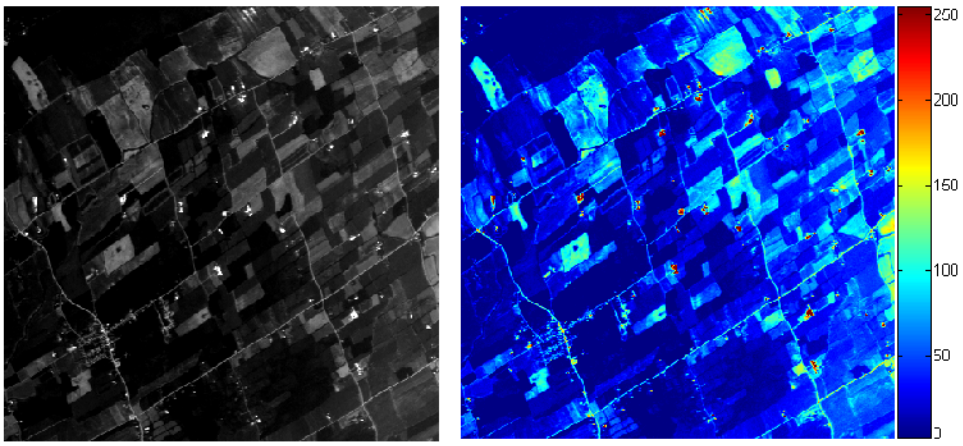
Del conjunto de escenas procesadas con técnicas de regularización para su reconstrucción/mejoramiento se analizaron los diferentes resultados desde dos perspectivas. En la primera, se analizan los resultados en cuestión de la calidad de la reconstrucción de los algoritmos CLS y WCLS, mostrando la relación IOSNR resultante. En la segunda perspectiva se compara el tiempo total de procesamiento de los algoritmos CLS y WCLS utilizando técnicas de cómputo en paralelo con GPUs y técnicas de procesamiento tradicional con

CPUs. El procesamiento se realizó en dos estaciones de trabajo con características de hardware diferentes (Tabla 5.1).

Tabla 5.1: Hardware utilizado para la ejecución de las pruebas experimentales

	Hardware 1	Hardware 2
CPU	Intel(R) Xeon(R) CPU E5-2403 @ 1.80GHz	Intel(R) Xeon(R) CPU E5603 @ 1.60GHz
Memoria	8 GB	24 GB
GPU	Quadro 2000	Tesla C2075

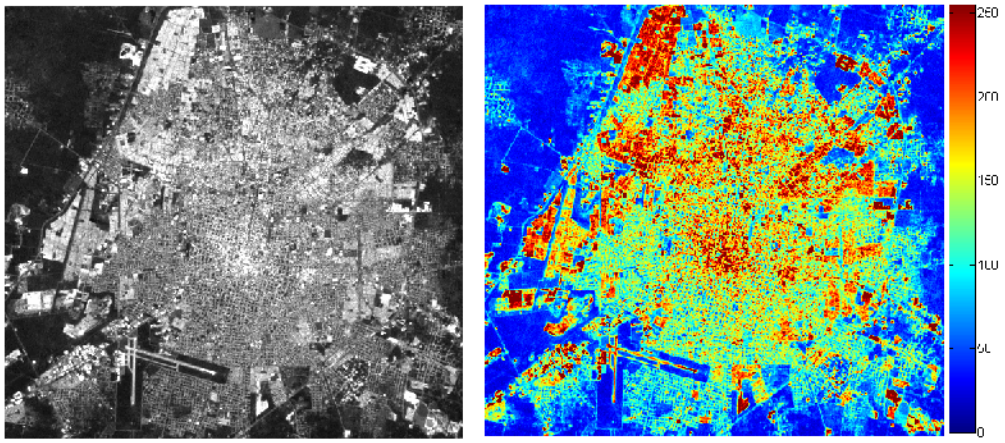
Debido a la alta resolución de las imágenes de los escenarios a estudiar se presenta únicamente una pequeña sección de cada una de las imágenes para mostrar de manera más clara los resultados[1]. Una vez aplicados los algoritmos CLS y WCLS a las imágenes multiespectrales e hiperespectrales se mostrará el resultado de la reconstrucción y su pseudo-color permitiendo mostrar de manera clara los resultados de reconstrucción después de aplicar las técnicas de regularización sobre la imágenes multiespectrales e hiperespectrales. En todos los casos de estudio se considero una matriz de distorsión con *azimuth=12*, *y=6* y un valor de $\alpha = 0.001$.



(a) (b)

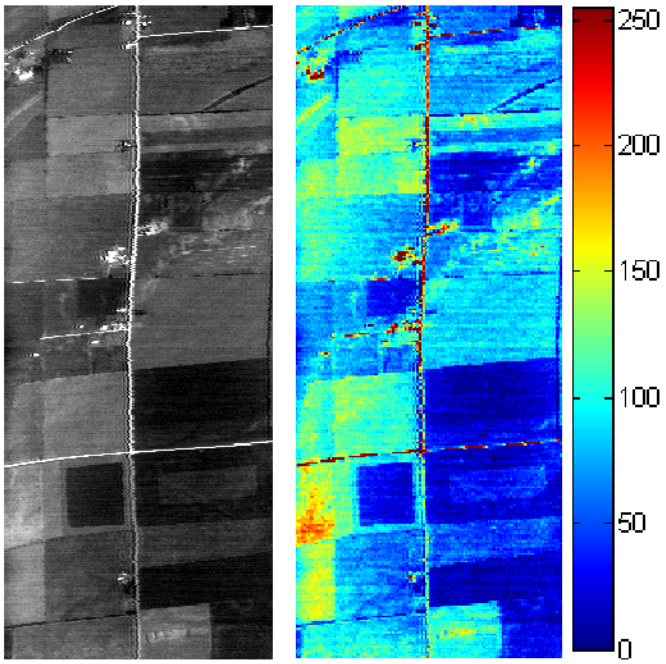
Figura 5.4: (a) sección de la escena original capturada por el sensor AVIRIS; (b) misma sección representado con pseudo-color en MATLAB.

[1]Se muestra una sección de la imagen por cuestión de claridad. El procesamiento se aplico en toda la imagen a tratar



(a) (b)

Figura 5.5: (a) sección de la escena original capturada por el sensor LANDSAT; (b) misma sección representado con pseudo-color en MATLAB.



(a) (b)

Figura 5.6: (a) sección de la escena original capturada por el sensor Flighline C1; (b) misma sección representado con pseudo-color en MATLAB.

5.1.2. Primer Caso de Estudio AVIRIS

En la Figura 5.4a se presenta una sección de la séptima banda del sensor AVIRIS y su pseudo-color en MATLAB (Figura 5.4b).

En las Figuras 5.7 a la 5.12, se presentan las diversas imágenes resultantes del proceso de reconstrucción de los algoritmos CLS y WCLS en conjunto de su pseudo-color en MATLAB. Como se observa en la Tabla 5.2, el nivel de reconstrucción del algoritmo WCLS alcanzó una relación IOSNR hasta 53.01 **dB** en comparativa del algoritmo CLS con 22.16 **dB** con ruido aditivo con una relación de SNR=1 siendo el peor de los casos presentado en éste documento de tesis. En la Figura 5.7c se puede observar bajo un ruido aditivo con una relación de SNR=1 el algoritmo CLS se satura de tal manera que distorsiona la imagen a reconstruir (5.7a) en lugar de reconstruirla/mejorarla. En la Tabla 5.2 es posible visualizar como el algoritmo CLS mejora gradualmente en condiciones de poco ruido (ej: SNR=25), pero sin superar al WCLS. En éste caso de estudio el algoritmo WCLS presentó mejores resultados ante todos los diferentes niveles de ruido utilizados para los experimentos.

Tabla 5.2: Comparativa de los resultados de IOSNR de los algoritmos CLS y WCLS con $azimuth = 12$, $y = 6$ y $\alpha = 0.001$ de la escena proveniente del sensor AVIRIS.

SNR (dB)	1^{er} CASO DE ESTUDIO ISONR (dB)	
	WCLS	CLS
1	53.01	22.16
5	52.80	26.28
10	52.29	31.06
15	51.89	36.24
20	52.20	41.22
25	52.95	46.22

5.1.3. Segundo Caso de Estudio Landsat 7

En la Figura 5.5a se presenta una sección de la imagen del satélite Landsat 7 proveniente de la banda 1 y su pseudo-color (Figura 5.5b). En las Figuras 5.13 a la 5.18 se presentan los resultados de la reconstrucción obtenidas aplicando los algoritmos CLS y su versión mejorada WCLS. Como se puede observar en la Tabla 5.3 en escenarios con ruido aditivo con una relación de SNR=1, el algoritmo WCLS presentó un resultado de IOSNR=49.18 **dB** en contraste de un IOSNR=29.79 **dB** del algoritmos CLS al reconstruir la Figura 5.13a. En este caso de estudio el algoritmo WCLS superó en la mayoría de los experimentos

realizados al algoritmo CLS, a excepción cuando la relación SNR del ruido es de 25; para este escenario CLS presenta mejores resultados que WCLS haciéndose evidente en las Figuras 5.18d (resultado del algoritmo CLS) y 5.18f (resultado del algoritmo WCLS) las cuales muestran el pseudo-color de las imágenes resultantes.

Tabla 5.3: Comparativa de los resultados de IOSNR de los algoritmos CLS y WCLS con $azimuth = 12$, $y = 6$ y $\alpha = 0.001$ de la escena proveniente del sensor Landsat 7.

SNR (dB)	2^o CASO DE ESTUDIO ISONR (dB)	
	WCLS	CLS
1	49.18	29.79
5	48.82	33.60
10	49.11	38.65
15	49.90	43.67
20	50.43	48.62
25	50.70	53.42

5.1.4. Tercer Caso de Estudio Flighline C1

En la Figura 5.6a se presenta una sección de la séptima utilizada en el *False Color* de la Figura 5.3. En las Figuras 5.7 a la 5.12 se presentan los resultados experimentales de la reconstrucción obtenida de los algoritmos CLS y WCLS. Analizando los resultados en la Tabla 5.4 se puede observar como el algoritmo CLS en escenarios con relación SNR altos el nivel de mejora/reconstrucción supera a su versión mejorada WCLS; esto se hace evidente en las Figuras 5.23d y 5.24d que muestran su pseudo-colores en MATLAB de la reconstrucción realizada por el algoritmo CLS.

Aún cuando en los últimos dos casos (con valores de SNR igual a 20 y 25) el algoritmo WCLS presento mejores resultados que con niveles de ruido mayores (ej: SNR=1). El resultado se puede observar el la Figura 5.19e donde el algoritmo WCLS tiene una relación entrada-salida de la señal de 50 **dB** a comparación de CLS con 34 **dB**. En la Figura 5.19c se presenta el pseudo-color de la imagen resultante de la reconstrucción de la Figura 5.19a.

Tabla 5.4: Comparativa de los resultados de IOSNR de los algoritmos CLS y WCLS con $azimuth = 12$, $y = 6$ y $\alpha = 0.001$ de la escena proveniente del sensor Flighline C1.

SNR (dB)	3^{er} CASO DE ESTUDIO ISONR (dB)	
	WCLS	CLS
1	50.62	34.45
5	50.46	38.58
10	50.75	43.40
15	51.65	48.59
20	52.45	53.20
25	50.87	57.98

5.1.5. Análisis de Rendimiento

Para medir el rendimiento de las implementaciones propuestas del algoritmo CLS y su versión mejorada WCLS se llevaron a cabo diferentes pruebas en dos estaciones de trabajo. La primera estación de trabajo cuenta con un CPU Intel Xen E5-2403 de cuatro núcleos con una frecuencia de reloj de 1.8 GHz con 8GB de memoria RAM. La segunda estación de trabajo cuenta con un procesador Inter Xeon E603 con cuatro cores con una frecuencia de reloj de 1.60 GHz con 24 GB de memoria RAM, ambas estaciones de trabajo utilizan el sistema operativo Ubuntu 12.04.4 LTS. La primera estación de trabajo cuenta con una tarjeta NVIDIA Tesla C2075, con 448 núcleos de CUDA, con 5.37 GB de memoria global y la segunda estación cuenta con una tarjeta NVIDIA Quadro 2000, con 144 núcleos de CUDA, con 1.5 GB de memoria global (ver Tabla 4.1). Los tiempos de procesamiento mostrados en éste documento de tesis no incluye el tiempo del procesamiento de $\boldsymbol{u} = \boldsymbol{S}\boldsymbol{v} + \boldsymbol{n}$, es decir únicamente se mide el tiempo de procesamiento requerido para realizar la reconstrucción de la imagen multiespectral e hiperespectral con los algoritmos CLS y WCLS.

Tabla 5.5: Tiempos de procesamiento (en milisegundos) del algoritmo WCLS

	1^{er} CASO DE ESTUDIO TIEMPO (ms)		2^{o} CASO DE ESTUDIO TIEMPO (ms)		3^{er} CASO DE ESTUDIO TIEMPO (ms)	
	GPU	CPU	GPU	CPU	GPU	CPU
Estación 1	4347.53	84348.78	1756.98	22843.41	828.78	2757.08
Estación 2	1890.52	74949.88	1303.61	20879.38	558.84	2595.96

Tabla 5.6: Tiempos de procesamiento (en milisegundos) del algoritmo CLS

	1^{er} CASO DE ESTUDIO TIEMPO (ms)		2^{o} CASO DE ESTUDIO TIEMPO (ms)		3^{er} CASO DE ESTUDIO TIEMPO (ms)	
	GPU	CPU	GPU	CPU	GPU	CPU
Estación 1	3911.80	65503.14	1594.32	17545.18	806.93	2169.40
Estación 2	1694.03	51467.23	857.09	13998.78	544.77	1775.54

Tras analizar los resultados registrados en las Tablas 5.5 y 5.6, es evidente que el tiempo de procesamiento utilizando el GPU es muy inferior a comparación de utilizar únicamente CPUs. Es importante mencionar que el código a ejecutar en el CPU se compiló con la bandera de optimización -O3 del compilador `g++` la cual permite optimizar el código al momento de compilarlo reduciendo el tiempo de procesamiento. Como se puede ver en la Tabla 5.5 el tiempo de procesamiento del algoritmo WCLS es de 4347.53 milisegundos con GPUs en comparación del tiempo de procesamiento de 84348.78 milesegundos utilizando CPUS, obteniendo así una aceleración de aproximadamente 19X ejecutándose en la primera estación de trabajo la cual cuenta con una tarjeta NVIDIA Quadro 2000. Utilizando el mismo caso de estudio y aplicando el mismo algoritmo, pero utilizando la segunda estación de trabajo la cual cuenta con una tarjeta NVIDIA Tesla C2075. El tiempo de procesamiento en el GPU es de 1890.52 milisegundos y de 74949.88 milisegundos en el CPU, obteniendo una aceleración aproximada de 39X duplicando la aceleración obtenida en comparativa de la primera estación de trabajo. Comparando los mismos datos, pero con la implementación del algoritmo CLS la aceleración obtenida entre el GPU y CPU fue de 16X utilizando la primera estación de trabajo de 30X para la segunda estación.

Es importante señalar la tendencia que hay de disminuir la diferencia del tiempo de procesamiento entre GPU y CPU. En la Tabla 5.5 se puede visualizar que en las pruebas realizadas la aceleración obtenida es de 19X, 13X, 3X para el primer, segundo y tercer caso de estudio en la primera estación de trabajo y 39X, 16X y 5X para la segunda estación de trabajo. Es evidente que el rendimiento del GPU decae dependiendo de la resolución de la imagen, es decir, a mayor resolución de la imagen mejor es el rendimiento de el GPU por tal motivo para imágenes de poca resolución es necesario evaluar si el tiempo

de procesamiento sería mayor en el GPU a comparación de realizar el procesamiento en el CPU.

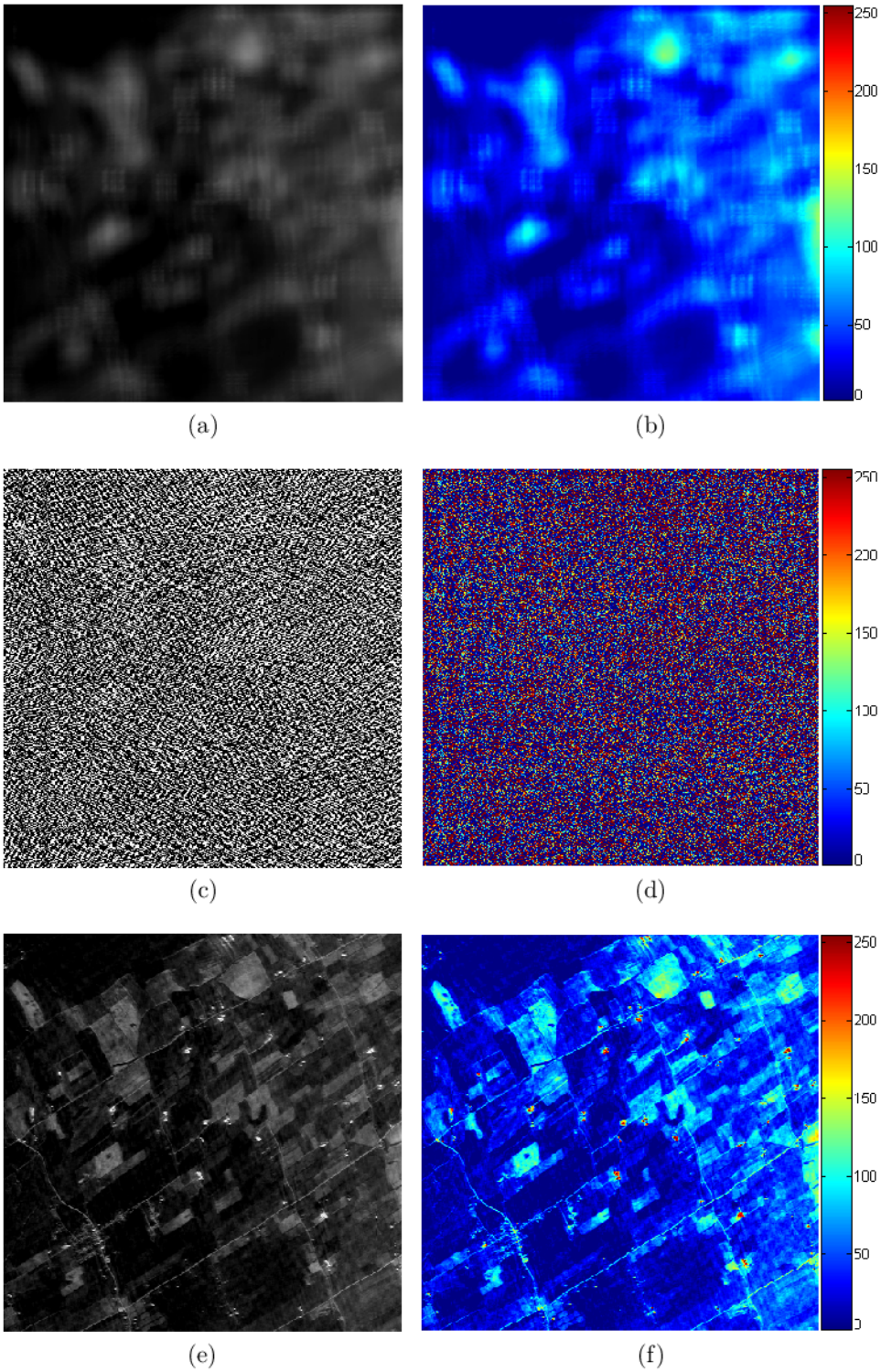


Figura 5.7: Resultados de reconstrucción de la imagen del sensor AVIRIS con un ruido aditivo con relación SNR=1; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

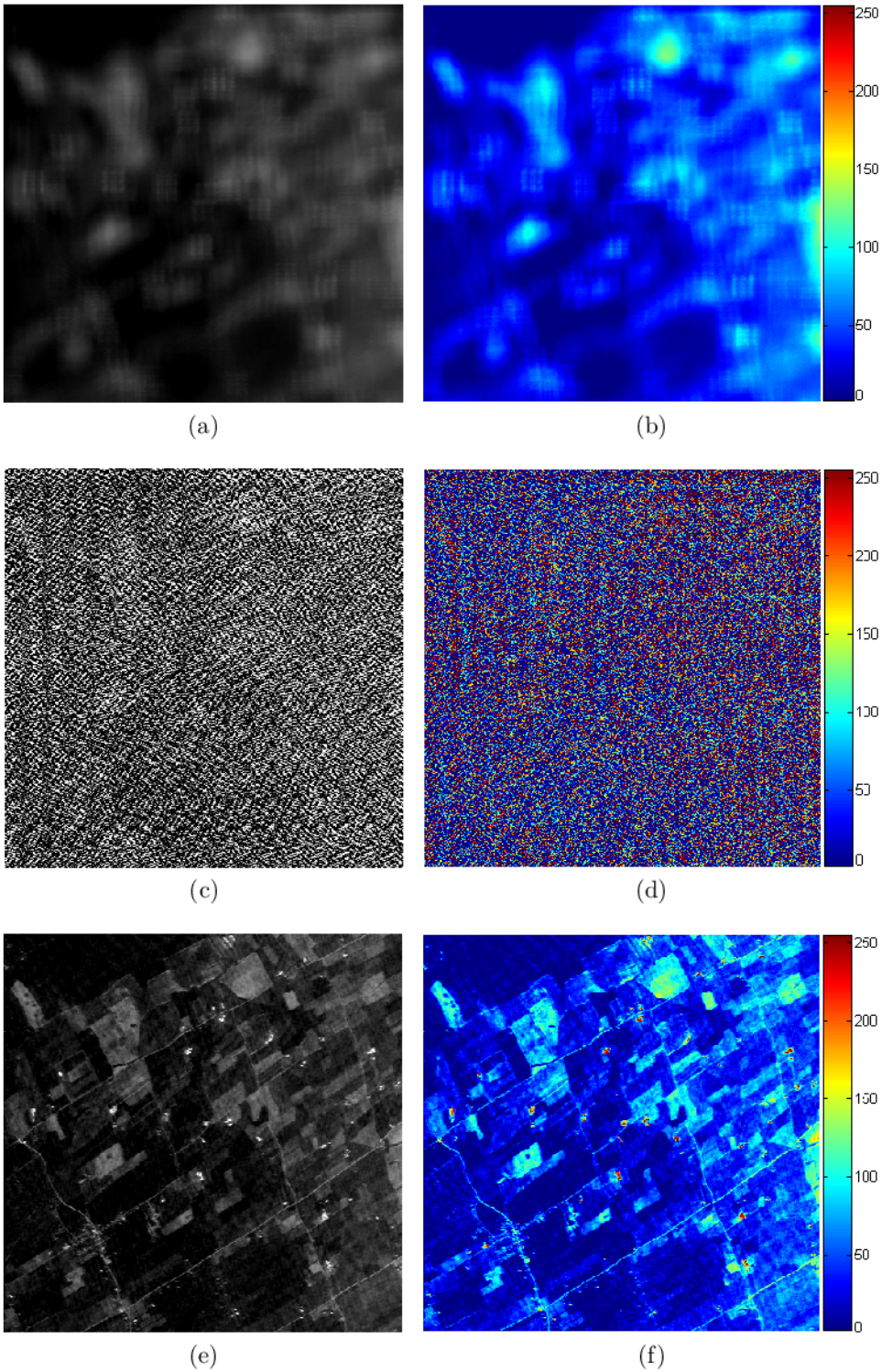


Figura 5.8: Resultados de reconstrucción de la imagen del sensor AVIRIS con un ruido aditivo con relación SNR=5; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

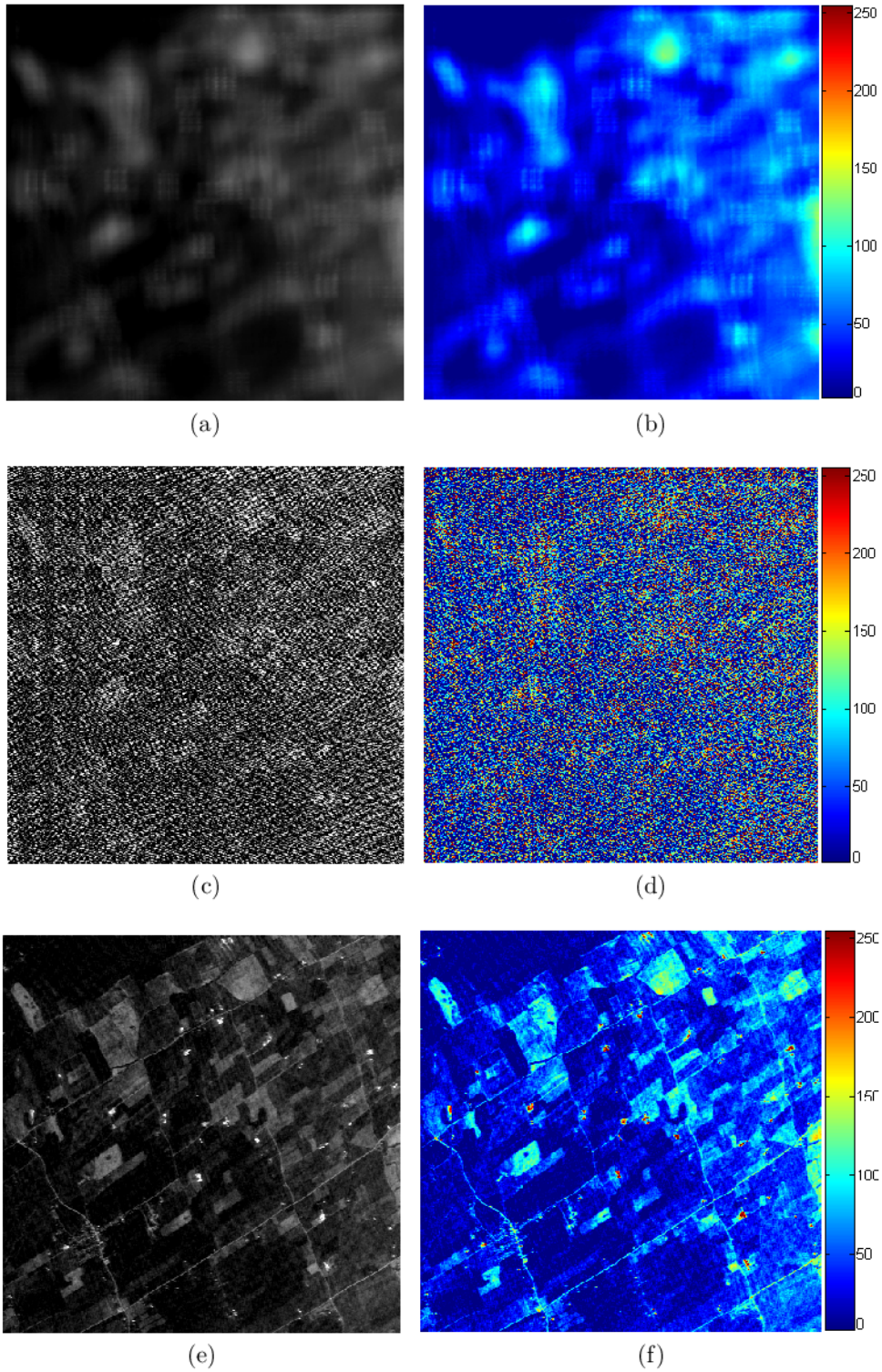


Figura 5.9: Resultados de reconstrucción de la imagen del sensor AVIRIS con un ruido aditivo con relación SNR=10; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

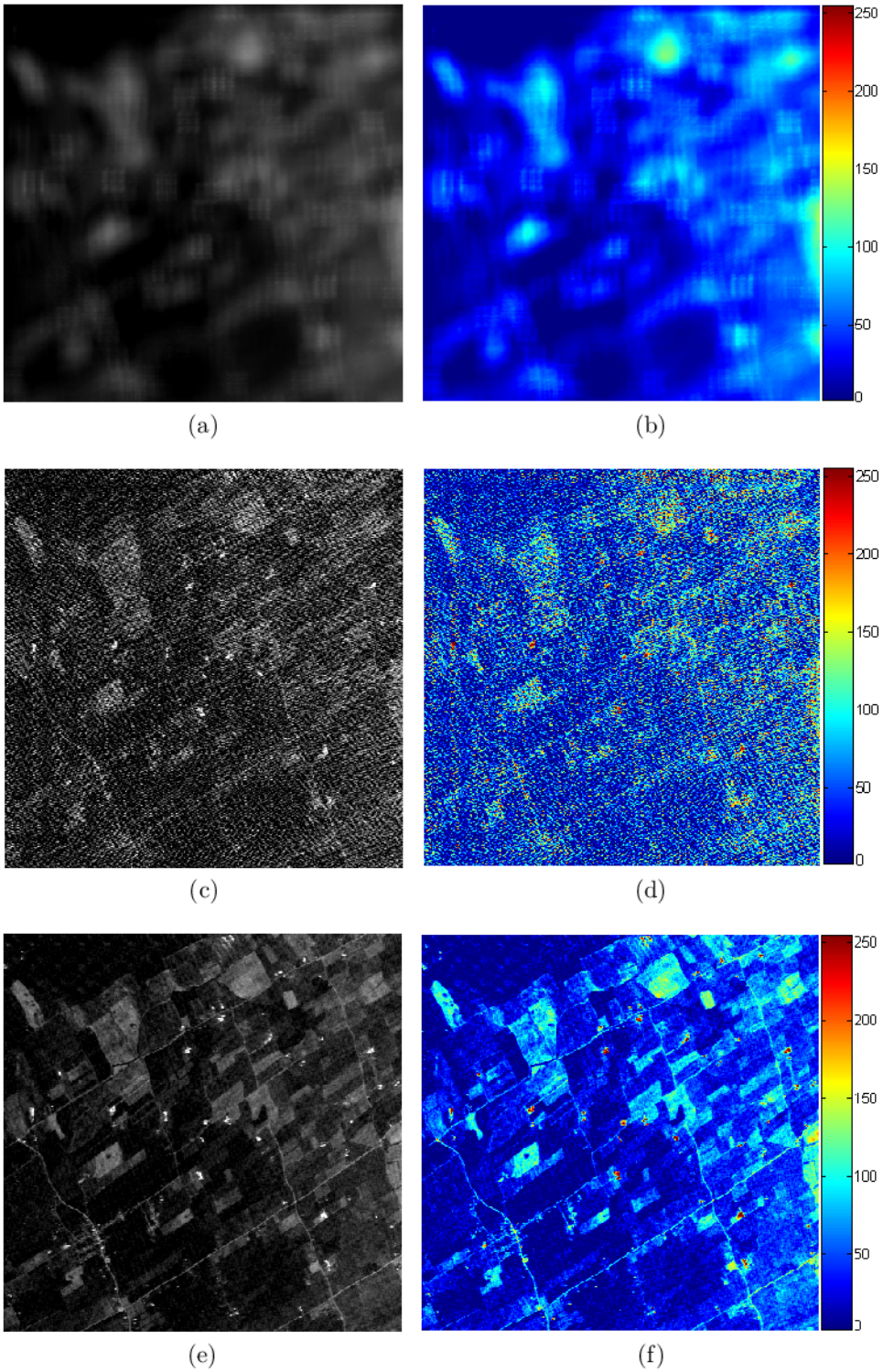


Figura 5.10: Resultados de reconstrucción de la imagen del sensor AVIRIS con un ruido aditivo con relación SNR=15; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

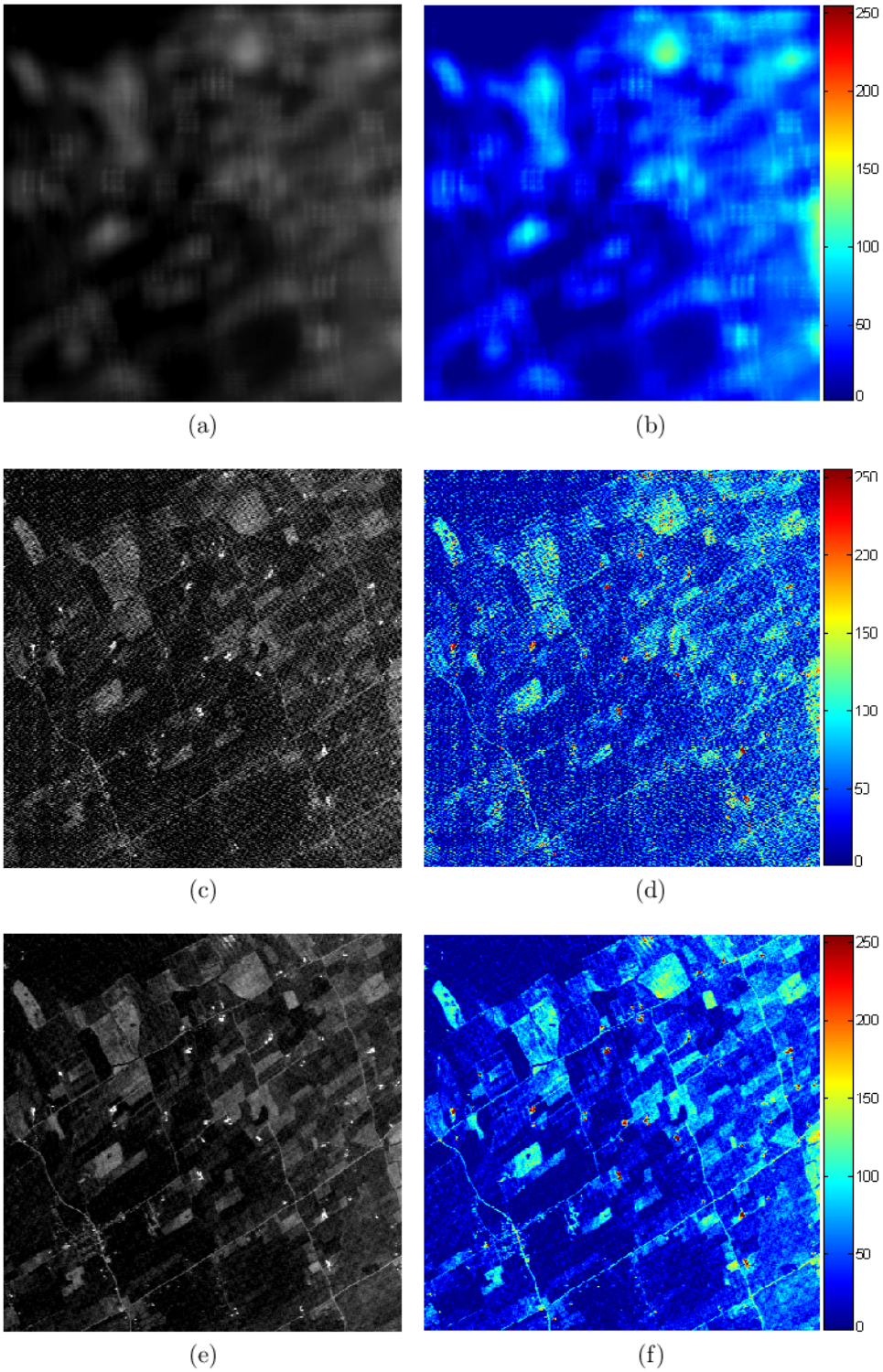


Figura 5.11: Resultados de reconstrucción de la imagen del sensor AVIRIS con un ruido aditivo con relación SNR=20; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

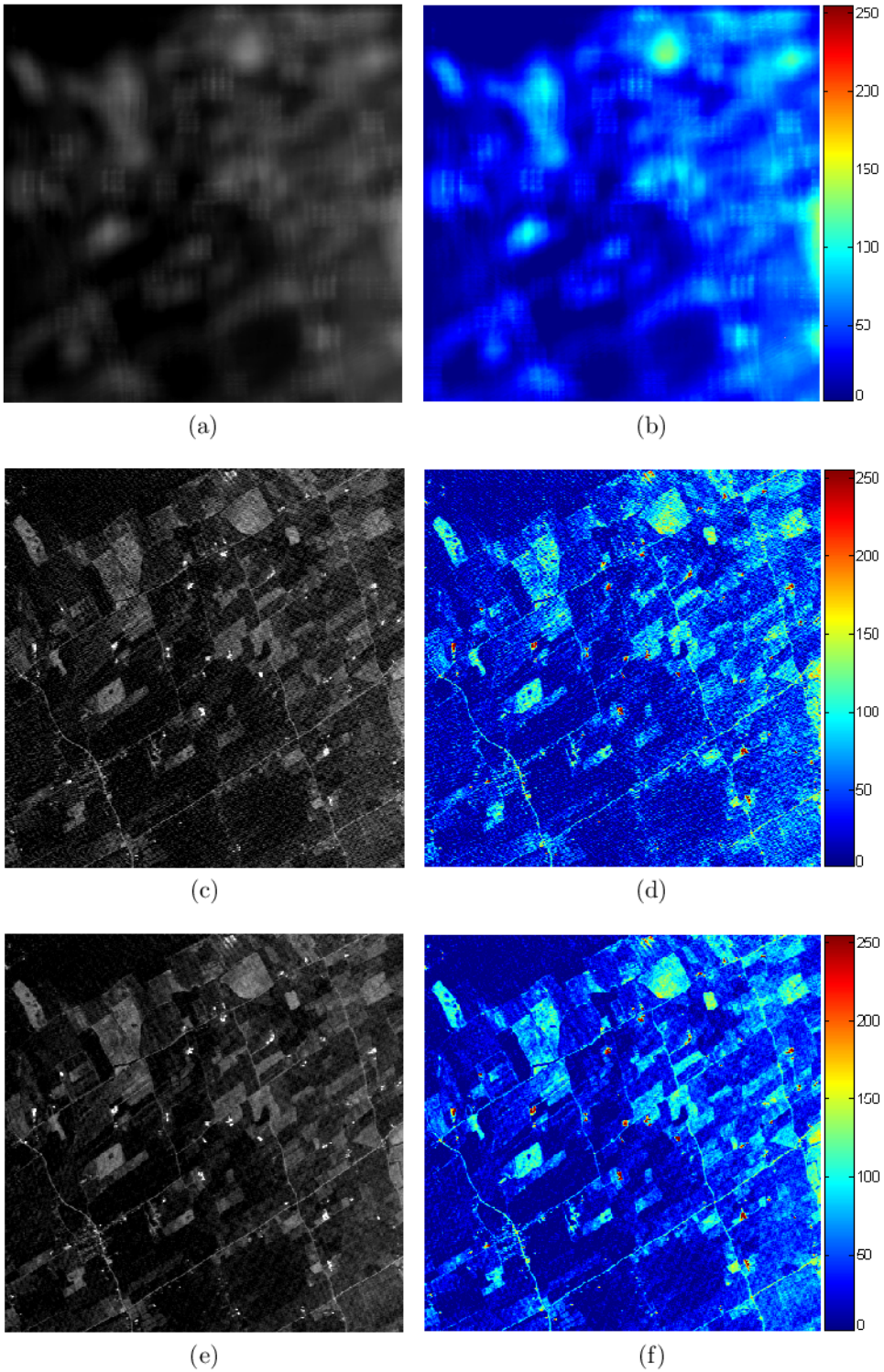


Figura 5.12: Resultados de reconstrucción de la imagen del sensor AVIRIS con un ruido aditivo con relación SNR=25; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

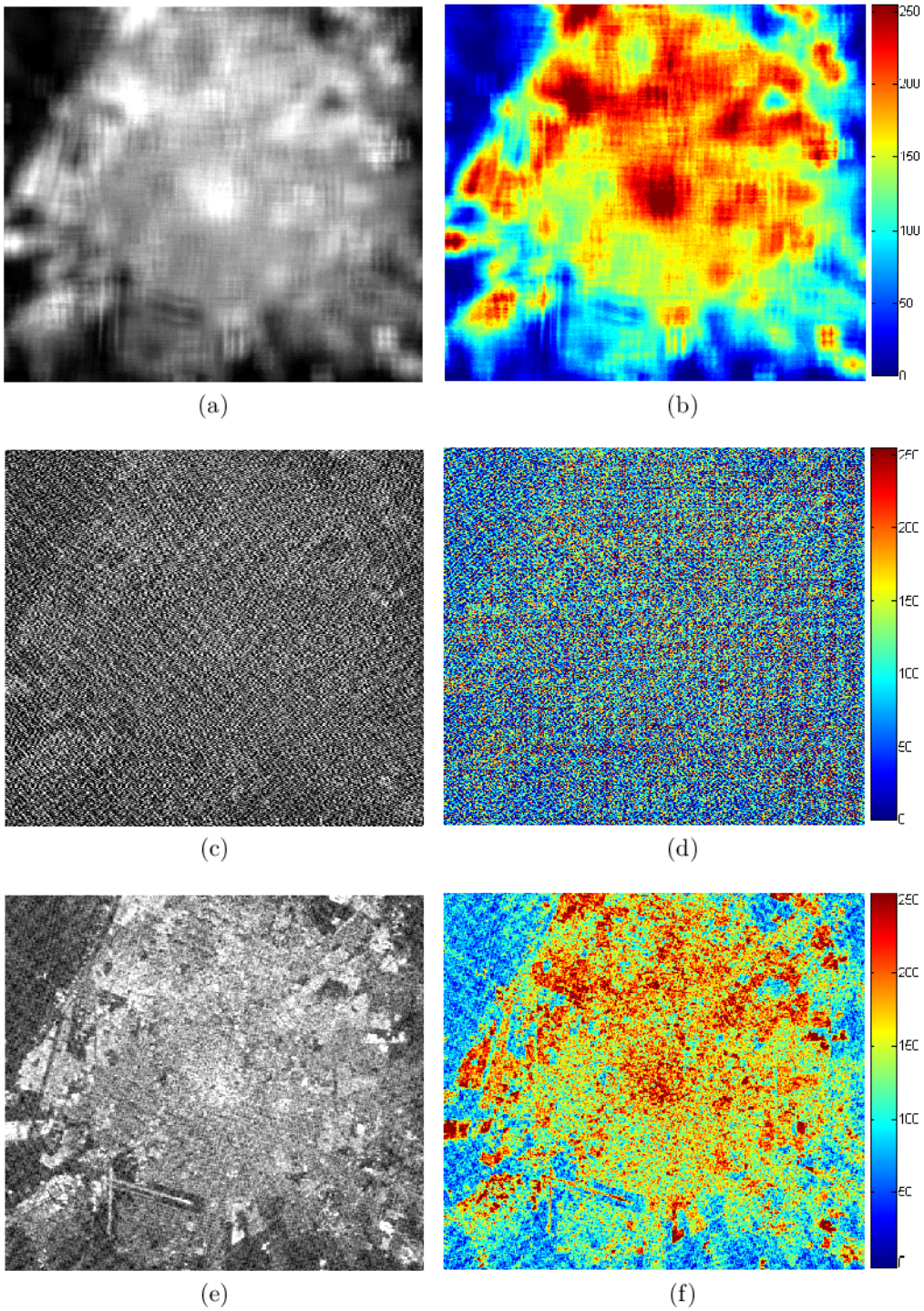


Figura 5.13: Resultados de reconstrucción de la imagen del sensor LANDSAT con un ruido aditivo con relación SNR=1; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

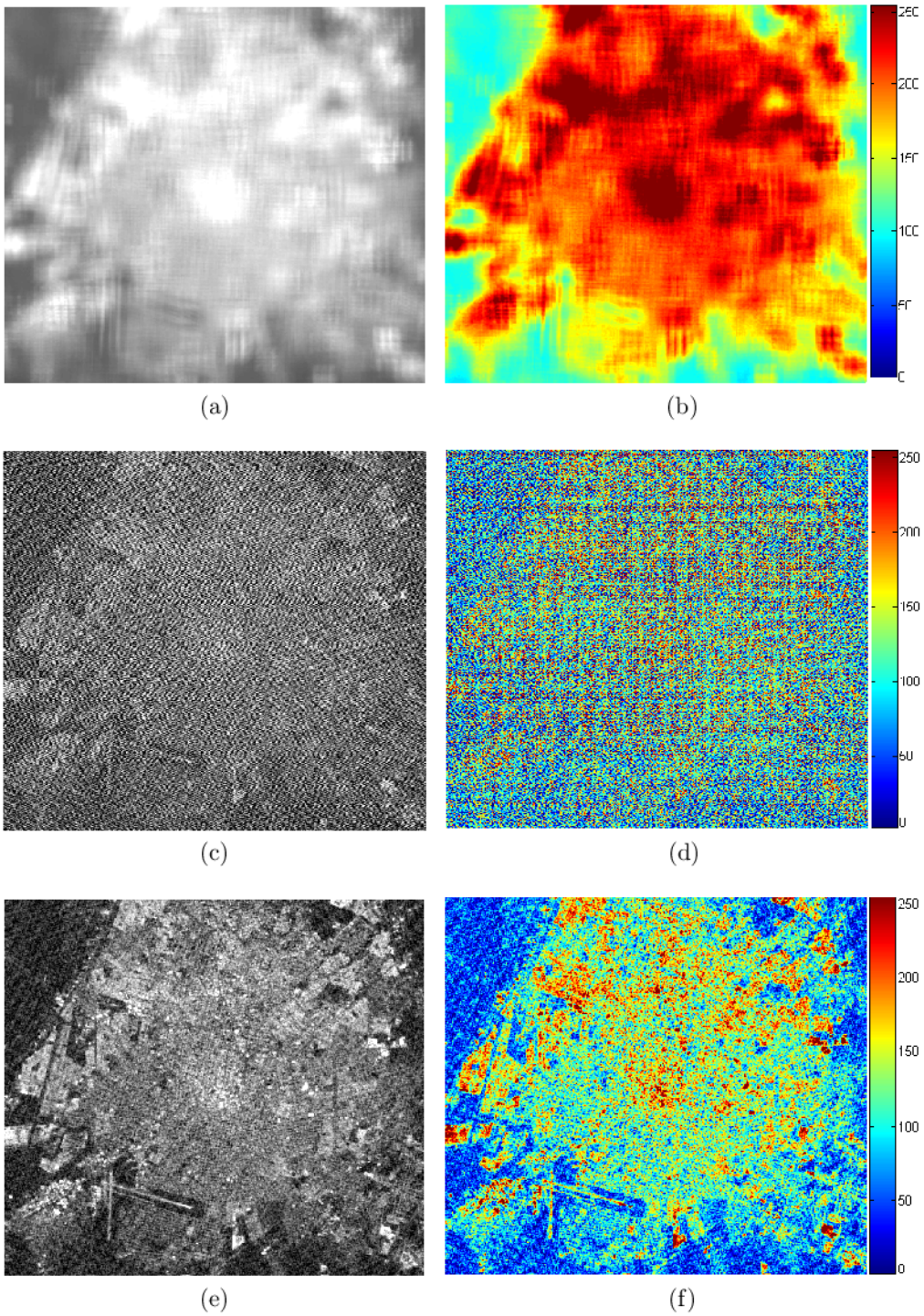


Figura 5.14: Resultados de reconstrucción de la imagen del sensor LANDSAT con un ruido aditivo con relación SNR=5; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB;(c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

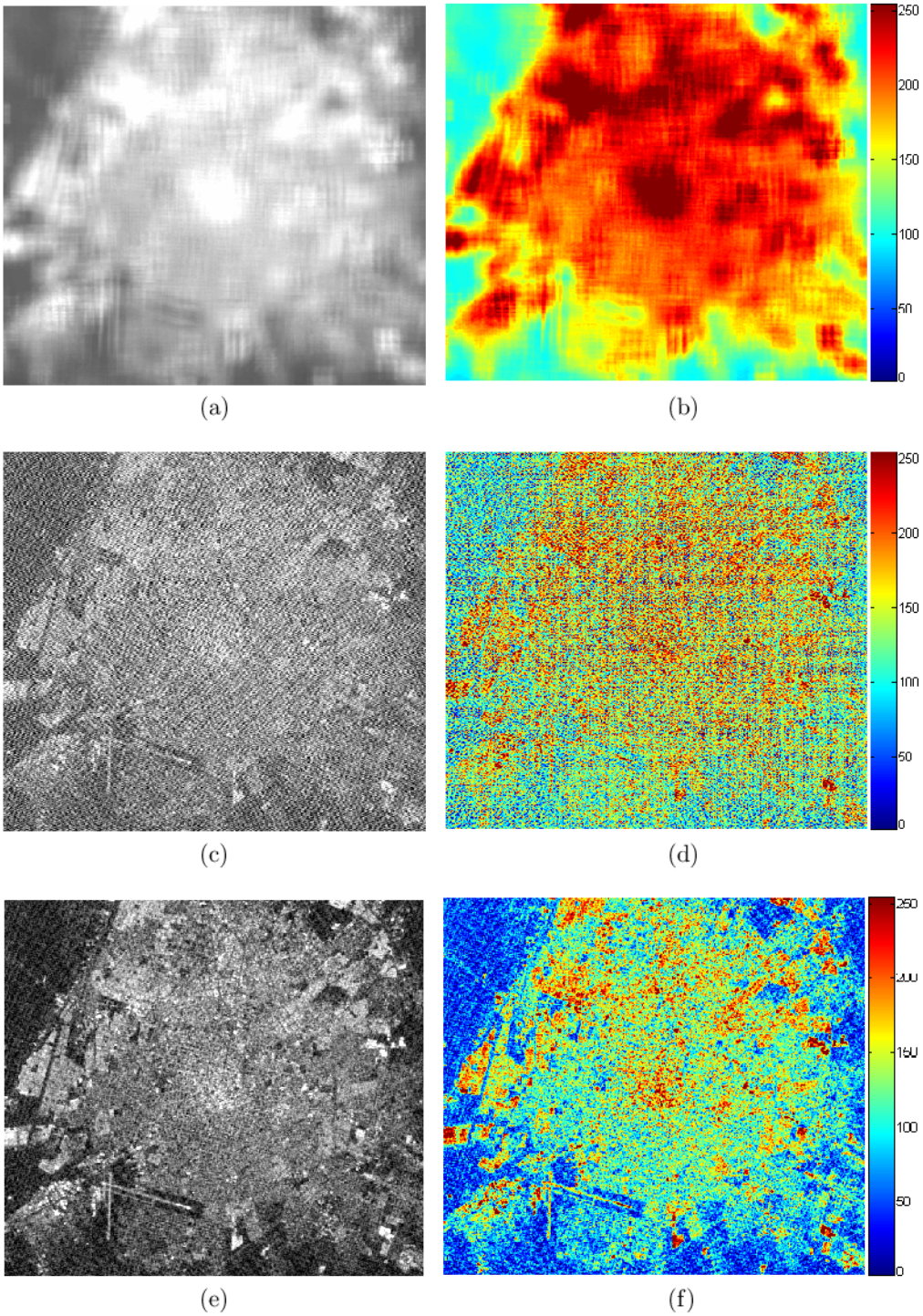


Figura 5.15: Resultados de reconstrucción de la imagen del sensor LANDSAT con un ruido aditivo con relación SNR=10; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

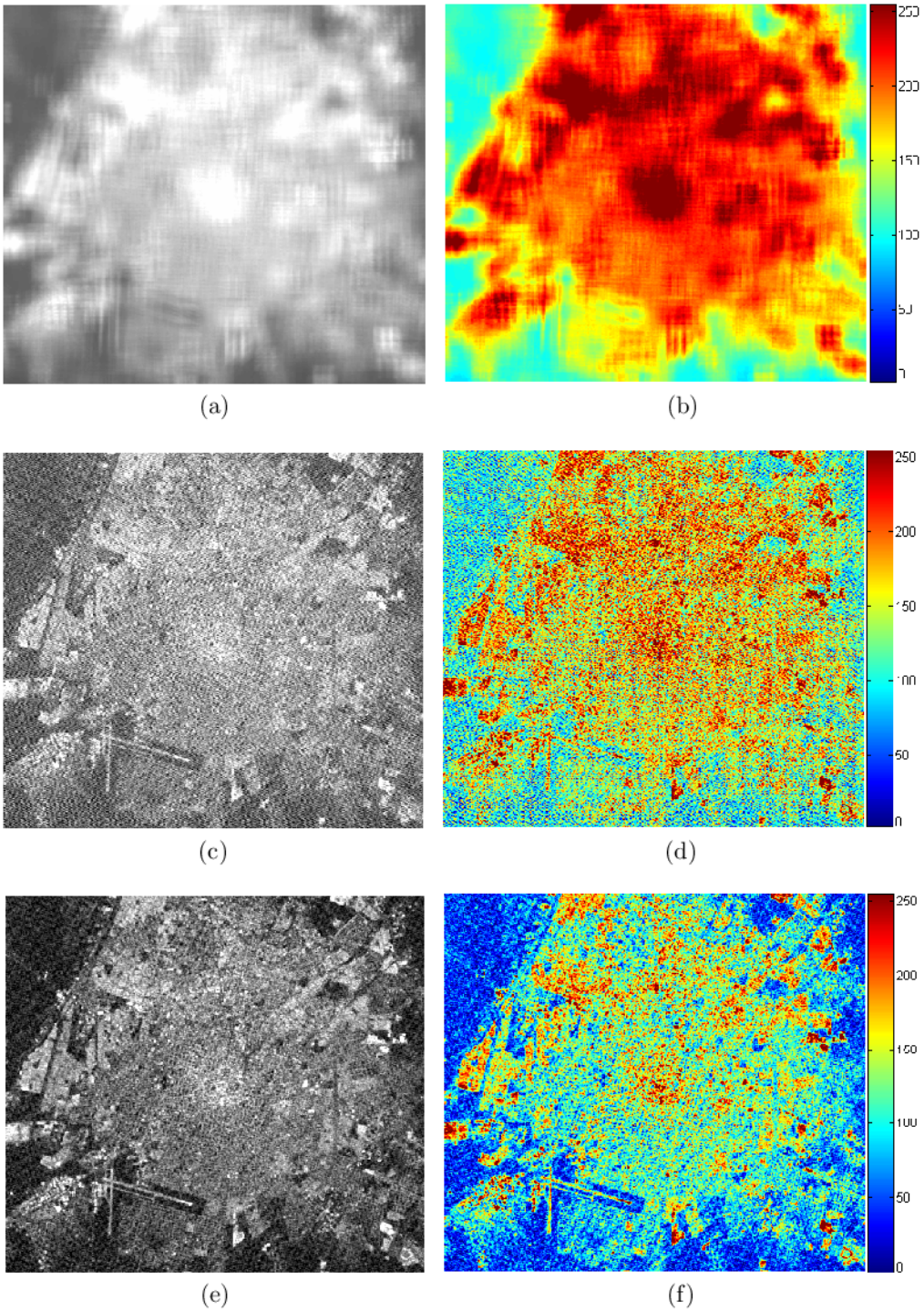


Figura 5.16: Resultados de reconstrucción de la imagen del sensor LANDSAT con un ruido aditivo con relación SNR=15; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

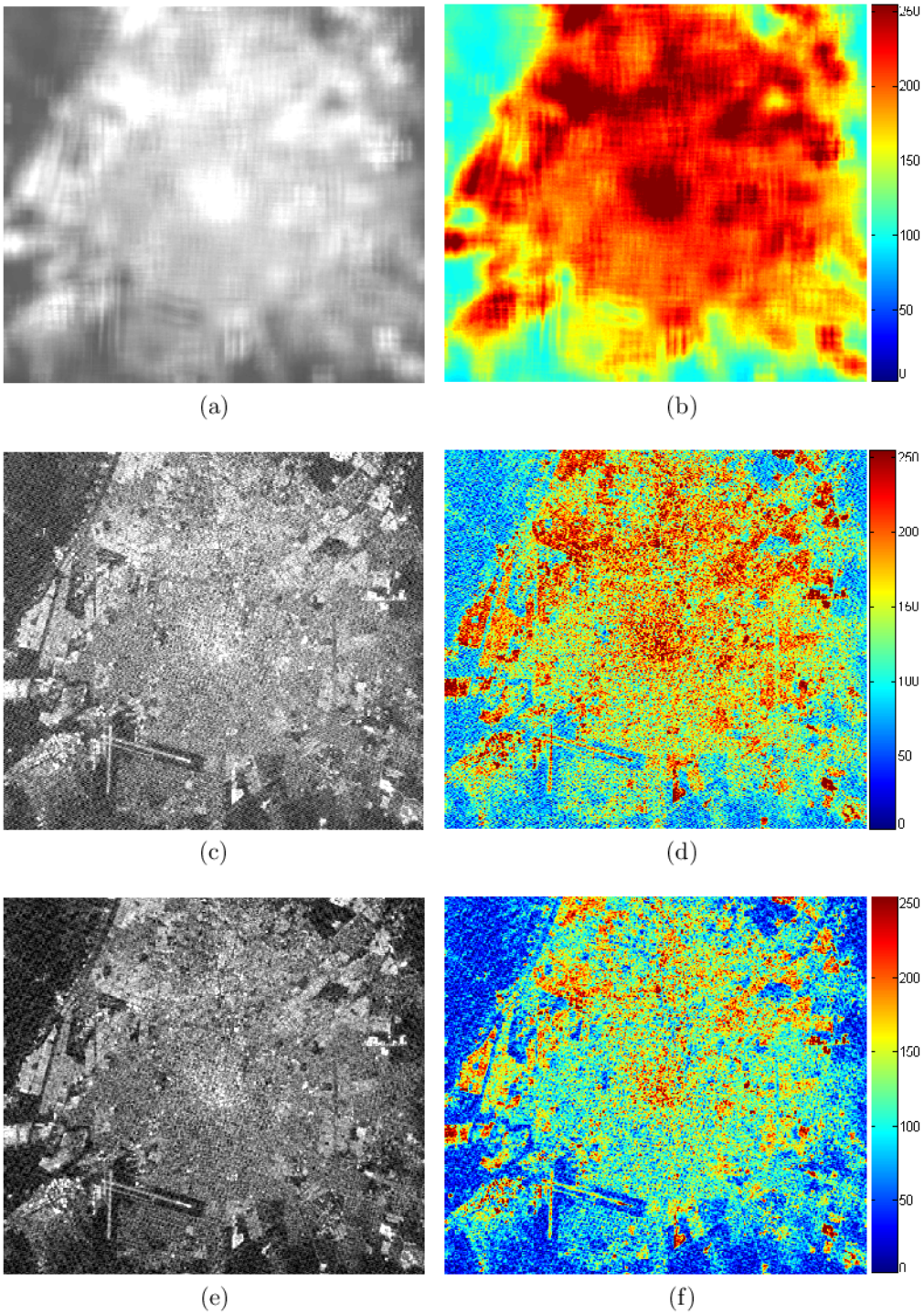


Figura 5.17: Resultados de reconstrucción de la imagen del sensor LANDSAT con un ruido aditivo con relación SNR=20; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

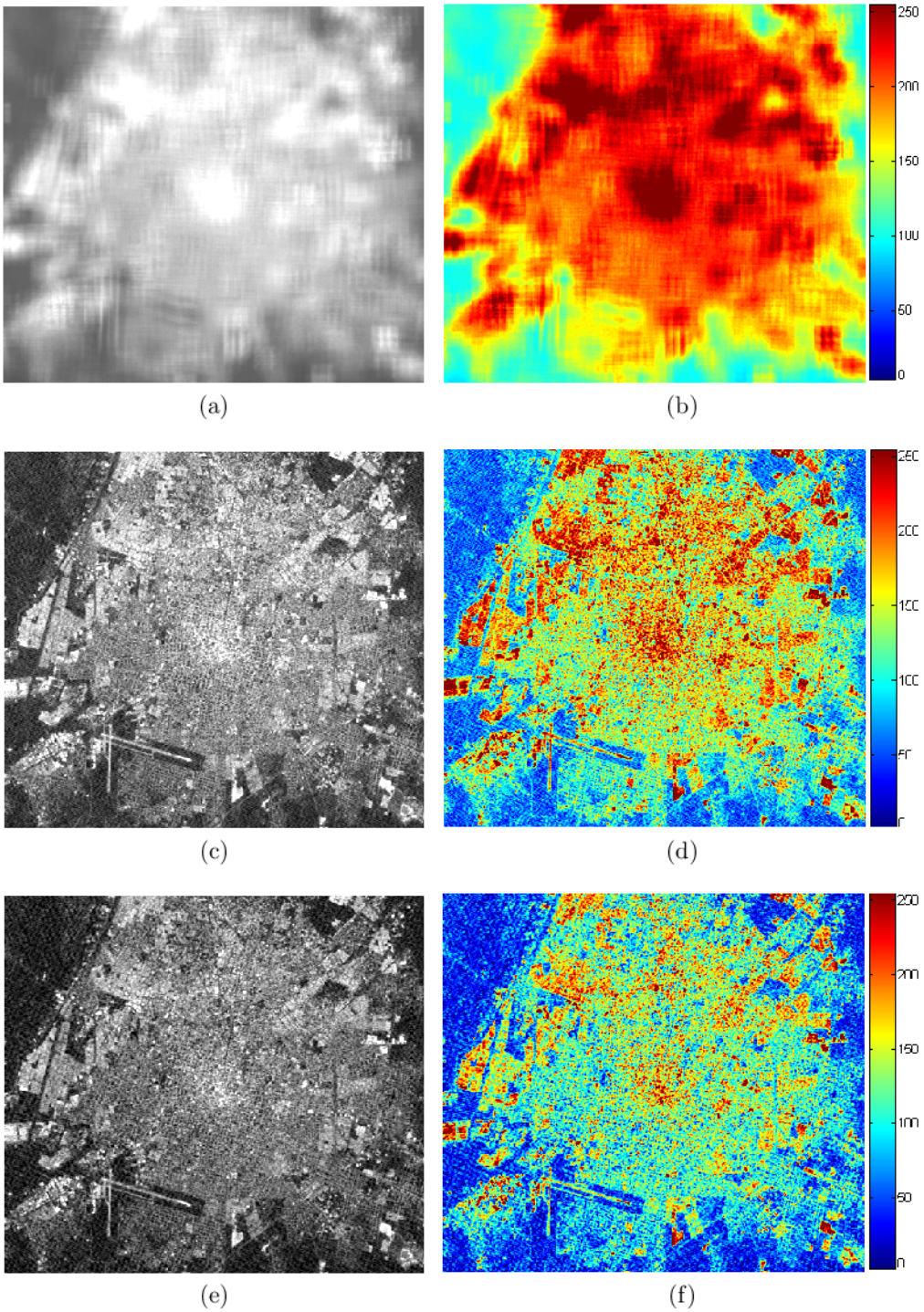


Figura 5.18: Resultados de reconstrucción de la imagen del sensor LANDSAT con un ruido aditivo con relación SNR=25; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

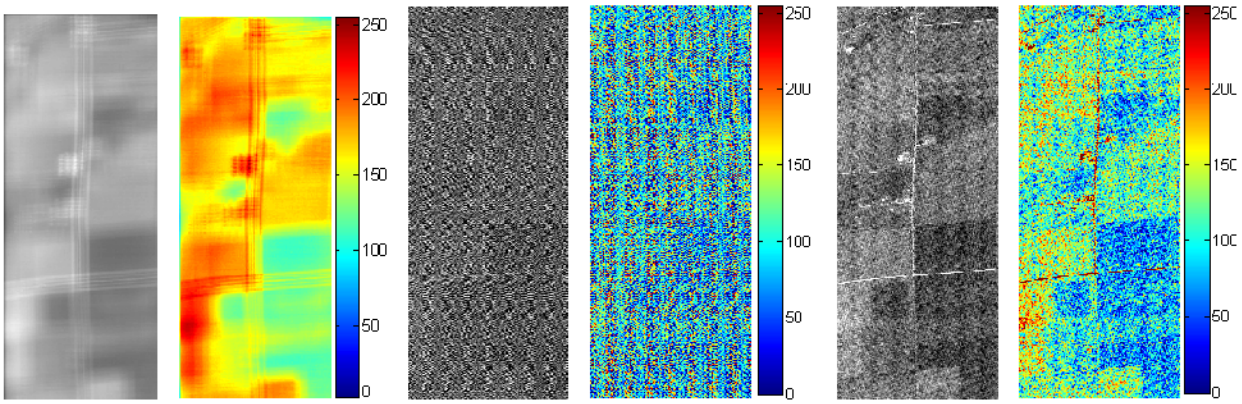


Figura 5.19: Resultados de reconstrucción de la imagen del sensor Flighline C1 con un ruido aditivo con relación SNR=1; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

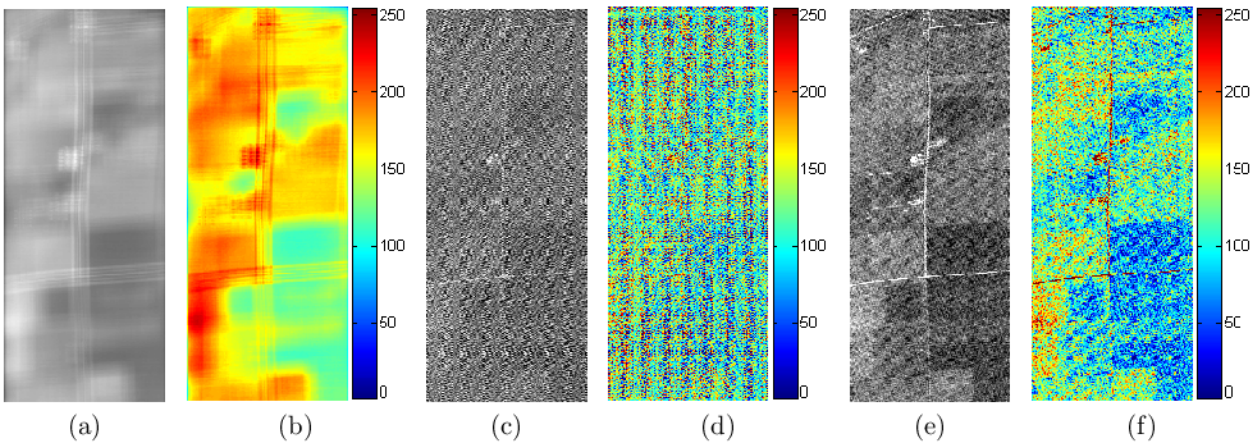


Figura 5.20: Resultados de reconstrucción de la imagen del sensor Flighline C1 con un ruido aditivo con relación SNR=5; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

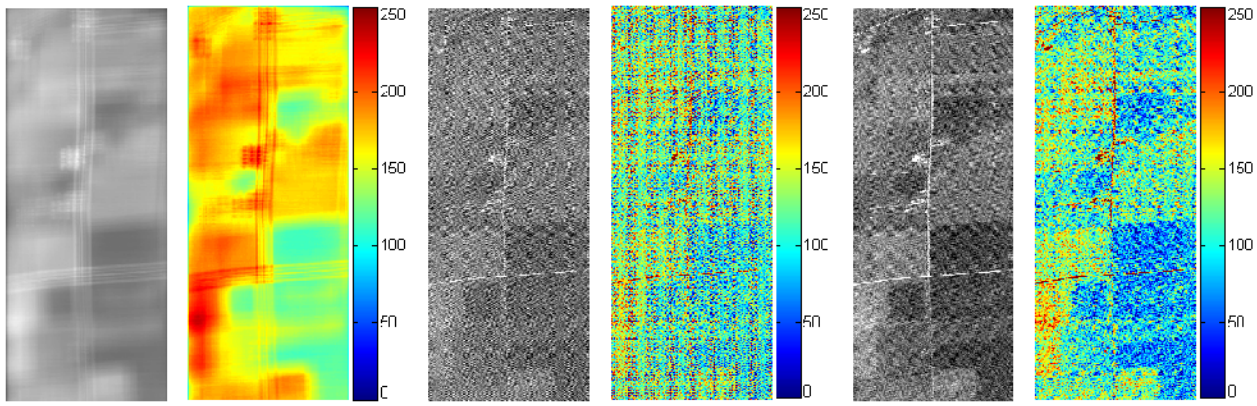


Figura 5.21: Resultados de reconstrucción de la imagen del sensor Flighline C1 con un ruido aditivo con relación SNR=10; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

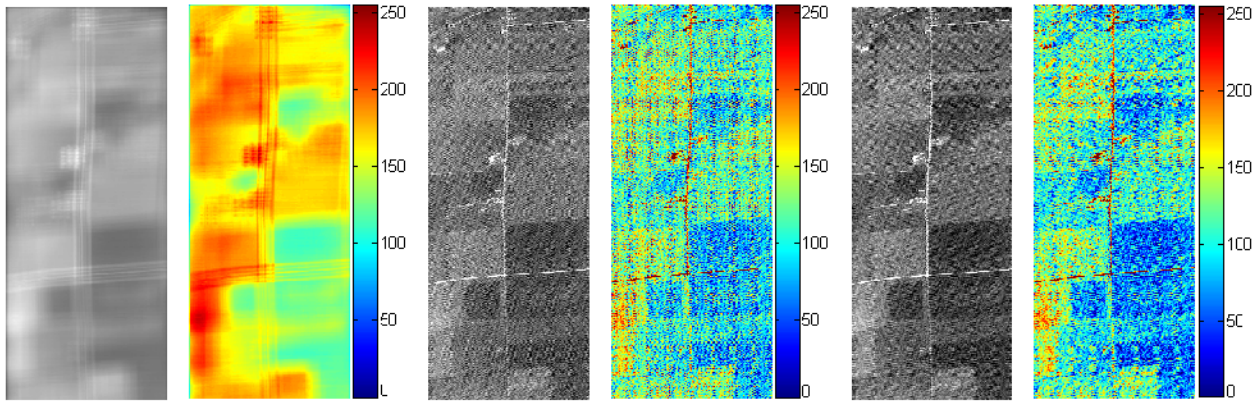


Figura 5.22: Resultados de reconstrucción de la imagen del sensor Flighline C1 con un ruido aditivo con relación SNR=15; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

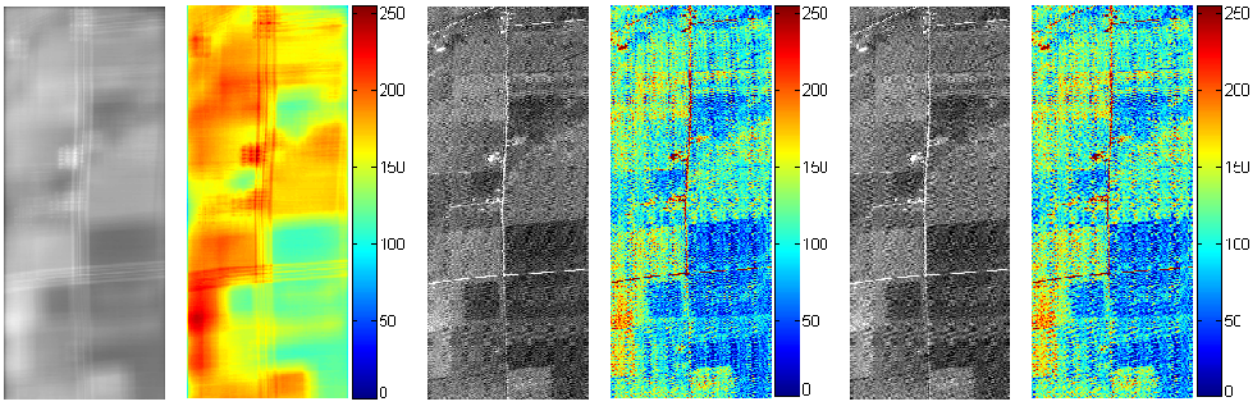


Figura 5.23: Resultados de reconstrucción de la imagen del sensor Flighline C1 con un ruido aditivo con relación SNR=20; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

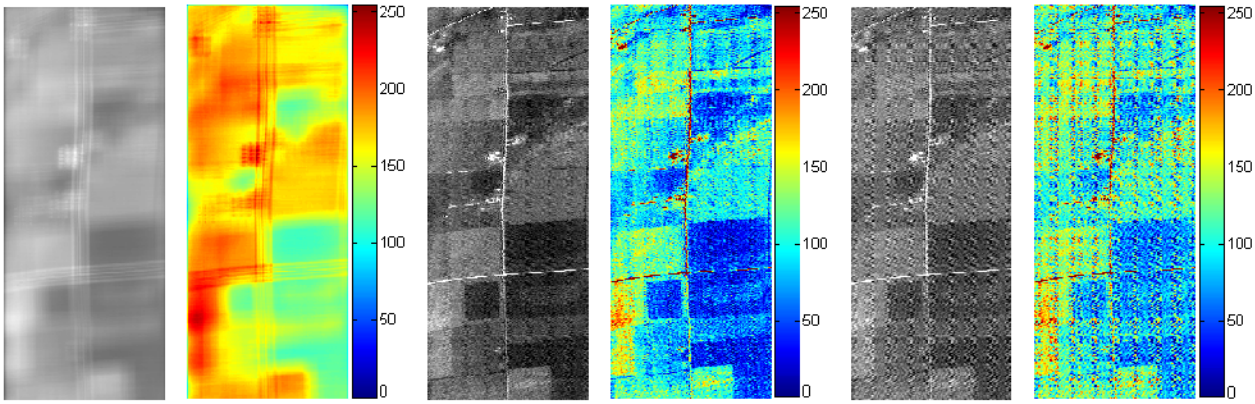


Figura 5.24: Resultados de reconstrucción de la imagen del sensor Flighline C1 con un ruido aditivo con relación SNR=25; (a) escena distorsionada; (b) misma escena con su pseudo-color en MATLAB; (c) escena reconstruida con CLS; (d) misma escena con su pseudo-color en MATLAB; (e) escena reconstruida con WCLS; (f) misma escena con su pseudo-color en MATLAB.

Capítulo 6

Conclusiones y Trabajos a Futuro

En conclusión, se cumplieron con los objetivos propuestos en el presente trabajo de tesis. En específico, se demostró que las técnicas de regularización en conjunto de técnicas de cómputo en paralelo con GPUs permiten reducir el tiempo de procesamiento de manera drástica llegando a tener una aceleración aproximada de 39X. Los resultados obtenidos de la reconstrucción de los algoritmos CLS y WCLS presentado en el capítulo 5 deja de manera clara que el algoritmo WCLS presenta mejores resultados que CLS en situaciones con demasiado ruido. Como se presentó en el capítulo 5, en ciertos escenarios los resultados del algoritmo CLS son superiores al WCLS, es necesario recalcar que para todos los experimentos se utilizó el valor $\alpha = 0.001$. Variando éste valor se puede lograr mejorar el resultado del algoritmo WCLS, el grado de libertad de α se encuentra entre el rango de 0.01 a 0.001. La principal aportación de esta tesis es presentar la alta capacidad de procesamiento en paralelo con la que cuenta un GPU, a diferencia del procesamiento con CPUs. La flexibilidad con la que cuenta el procesamiento con GPU y el lenguaje CUDA la hace una herramienta muy potente al momento de procesar grandes volúmenes de datos. Otra aportación son los resultados de mejora/reconstrucción obtenidos en la aplicación del algoritmo CLS y WCLS, dejando claro que el algoritmo WCLS es una excelente alternativa para la reconstrucción de imágenes multiespectrales e hiperespectrales en ambientes inciertos. El mejoramiento/reconstrucción de imágenes multiespectrales e hiperespectrales

con técnicas de regularización es de vital importancia en la percepción remota ya que la calidad de las imágenes afecta directamente los resultados en las diferentes áreas de aplicación de la percepción. Las principales áreas de aplicación de la percepción remota son: detección de anomalías, reconocimiento de objetivos y caracterización de superficies.

Entre los posibles trabajos futuros se destacan los siguientes:

- Plantear una alternativa de implementación del algoritmo WCLS en más de dos GPUs para el acelerar el procesamiento de los algoritmos y así procesar de manera simultanea los componentes del estimador $\boldsymbol{W}$.

- Realizar la implementación del algoritmo WCLS en un *cluster* de servidores o estaciones de trabajo, donde cada estación contará con su GPU para realizar cómputo en paralelo y así aprovechar el poder de cómputo del *cluster* y la alta capacidad de paralelismo de los GPUs. Una propuesta es el *cluster* conocido como *beowulf* que requiere un mínimo de dos nodos para conformar el *cluster*.

- Determinar la dimensión mínima de la matrices o imágenes que se procesarán en el GPU para obtener su máximo rendimiento y así reducir el tiempo de procesamiento.

Bibliografía

[1] Instituto Nacional de Estadística Geográfica e Informática, "Boletín de los sistemas nacionales estadístico y de información geográfica," 7 2007.

[2] J. C. Trinder, "The Evolution of ISPRS Activities in the Spatial Information Sciences."

[3] G. A. Shaw and H.-h. K. Burke, "Spectral imaging for remote sensing," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 3–28, 2003.

[4] National Aeronautics and Space Administration, "Airborne visible/infrared imaging spectrometer," http://aviris.jpl.nasa.gov/aviris/index.html, [Online; accessed 16-April-2015].

[5] A. Pelagotti, A. Del Mastio, A. De Rosa, and A. Piva, "Multispectral imaging of paintings," *Signal Processing Magazine, IEEE*, vol. 25, no. 4, pp. 27–36, July 2008.

[6] Y. V. Shkvarko, "Unifying experiment design and convex regularization techniques for enhanced imaging with uncertain remote sensing data: Part i: Theory," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 48, no. 1, pp. 82–95, Jan 2010.

[7] H. P.-M. Yuriy Shkvarko and A. Castillo-Atoche, "Enhanced radar imaging in uncertain environment: A descriptive experiment design regularization approach," *International Journal of Navigation and Observation*, vol. 2008, pp. 1–3, may 2008.

[8] Q. Chang, S. Fuxiong, and H. Tianshu, "The real-time image processing based on dsp," in *Cellular Neural Networks and Their Applications, 2005 9th International Workshop on*, May 2005, pp. 40–43.

[9] M. Brogioli, P. Radosavljevic, and J. Cavallaro, "A general hardware/software co-design methodology for embedded signal processing and multimedia workloads," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, Oct 2006, pp. 1486–1490.

[10] A. Castillo Atoche, O. Palma Marrufo, and L. Ricalde Castellanos, "Aggregation of parallel computing and hardware/software co-design techniques for high-performance remote sensing applications," in *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, July 2011, pp. 217–220.

[11] S. Bernabe, S. Lopez, A. Plaza, R. Sarmiento, and P. Rodriguez, "Fpga design of an automatic target generation process for hyperspectral image analysis," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, Dec 2011, pp. 1010–1015.

[12] C. Vömel, S. Tomov, and J. Dongarra, "Divide and conquer on hybrid gpu-accelerated multicore systems," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. C70–C82, 2012. [Online]. Available: http://dx.doi.org/10.1137/100806783

[13] S. Bernabe, S. Lopez, A. Plaza, and R. Sarmiento, "Gpu implementation of an automatic target detection and classification algorithm for hyperspectral image analysis," *Geoscience and Remote Sensing Letters, IEEE*, vol. 10, no. 2, pp. 221–225, March 2013.

[14] S. Bernabe and A. Plaza, "Commodity cluster-based parallel implementation of an automatic target generation process for hyperspectral image analysis," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, Dec 2011, pp. 1038–1043.

[15] H. Liu, Y. Fan, X. Deng, and S. Ji, "Parallel processing architecture of remotely sensed image processing system based on cluster," in *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on*, Oct 2009, pp. 1–4.

[16] A. M. Zohaib Khan, Faisal Shafait, "Towards Automated Hyperspectral Document Image Analysis."

[17] C. Ünsalan and K. Boyer, *Multispectral Satellite Image Understanding: From Land Classification to Building and Road Detection*, ser. Advances in Computer Vision and Pattern Recognition. Springer London, 2011. [Online]. Available: https://books.google.com.mx/books?id=m9_CNAEACAAJ

[18] P. P. Chu, *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version.* John Wiley & Sons, 2008.

[19] J. Salazar, "Procesadores digitales de señal (DSP) Arquitecturas y criterios de selección."

[20] S. cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of GPU Computing).* MORGAN KAUFMANN, 2013.

[21] K. Kopper, *The Linux Enterprise Cluster: Build a Highly Available Cluster with Commodity Hardware and Free Software*, ser. No Starch Press Series. No Starch Press, 2005. [Online]. Available: https://books.google.com.mx/books?id=wiCGoCq8n4oC

[22] A. Castillo Atoche, D. Torres Roman, and Y. Shkvarko, "Experiment design regularization-based hardware/software codesign for real-time enhanced imaging in uncertain remote sensing environment," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, no. 1, p. 254040, 2010. [Online]. Available: http://asp.eurasipjournals.com/content/2010/1/254040

[23] L. Guerrero, J. Gutierrez, and C. Jalisco, "Simulation study of the sar imaging with the virtual remote sensing laboratory," in *Electronics, Robotics and Automotive Mechanics Conference, 2007. CERMA 2007*, Sept 2007, pp. 563–567.

[24] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris)," *Remote Sensing of Environment*, vol. 65, no. 3, pp. 227–248, 1998.