UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

Análisis de imágenes multiespectrales con OpenCL para detección de zonas propensas a incendios en Quintana Roo

TESIS
PARA OBTENER EL GRADO DE

MAESTRO EN MECATRÓNICA

PRESENTA
Ing. Omar Alejandro Pérez Martínez

DIRECTOR
Dr. Jaime Silverio Ortegón Aguilar

ASESORES
Dr. Javier Vázquez Castillo
Dr. Iván Villalón Turrubiates
Dr. Gliserio Romeli Barbosa Pool
Dr. José Hernández Rodríguez

UNIVERSIDAD DE QUINTANA ROO
AREA DE TITULACION

Universidad de Quintana Roo
DCI DIVISIÓN DE CIENCIAS E INGENIERÍA

CHETUMAL QUINTANA ROO, MÉXICO, ENERO DE 2016

UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

TRABAJO DE TESIS BAJO LA SUPERVISIÓN DEL COMITÉ DEL PROGRAMA DE MAESTRÍA Y APROBADA COMO REQUISITO PARA OBTENER EL GRADO DE:

MAESTRO EN MECATRÓNICA

COMITÉ DE TESIS

DIRECTOR:

Dr. Jaime Silverio Ortegón Aguilar

ASESOR:

Dr. Javier Vázquez Castillo

ASESOR:

Iván F. Villalón Turrubiates

Dr. Iván Villalón Turrubiates

ASESOR:

Dr. Gliserio Romeli Barbosa Pool

ASESOR:

Dr. José Hernández Rodríguez

CHETUMAL, QUINTANA ROO, MÉXICO, ENERO DE 2016.

Universidad de Quintana Roo
DCI DIVISIÓN DE CIENCIAS E INGENIERÍA
UNIVERSIDAD DE QUINTANA ROO
AREA DE TITULACION

UNIVERSIDAD DE QUINTANA ROO
AREA DE TITULACION

Índice

Índice de Figuras

Índice de Tablas

Le dedico este trabajo a mi hija Isabella Alessandra Pérez Gómez, eres la luz que ilumina mi camino, este y todos mis logros son tanto tuyos como mios, gracias a ti puedo superar cada uno de los obstáculos que se me han presentado con una sonrisa. Espero que siempre seas tan risueña y nunca pierdqas esa gran imaginación que tienes, siempre se tan curiosa como ahora que la curiosidad es el origen del conocimiento.

Agradecimientos

Agradezco a Dios por la vida, por permitirme concluir este ciclo, por otorgarme la fortaleza y las capacidades necesarias para lograr concluir satisfactoriamente este nuevo nivel académico.

A mi madre por ser mi apoyo incondicional, por los consejos que siempre me ofrece y siempre estar ahí en los momentos difíciles, muchas gracias por todo sin ti esto no hubiera sido posible.

A mis compañeros de la maestría que han sido un apoyo para poder concluir este ciclo, complementando mis conocimientos con los suyos, y regalándome su amistad y compañerismo, espero poder volver a trabajar con ustedes en un futuro.

A mis profesores por brindarme sus conocimientos, aclarar mis dudas, darme todo su apoyo durante la maestria y ayudarme a ser mejor persona, espero poder volver a trabajar con ustedes en un futuro

Capítulo 1. Introducción

1.1. Antecedentes

El fuego es considerado de vital importancia en el ciclo vital de los ecosistemas, siendo éste el que regenera la tierra haciéndola fértil para la nueva vida; pero con la intervención del hombre y el crecimiento de las zonas urbanas, el fuego se ha vuelto un problema, dado que en algunas ocasiones los incendios llegan a ser demasiado grandes poniendo en riesgo vidas humanas, especies que habitan los ecosistemas y la misma fauna.

En México se ha venido desarrollando una cultura de prevención y control de incendios forestales a través del Programa Nacional de Prevención de Incendios Forestales, con el apoyo del gobierno, organismos civiles y voluntariados, a cargo de la Comisión Nacional Forestal. (CONAFOR, 2015)

Aun con todo ese trabajo apenas en el 2013 hubo 10,406 incendios en toda la república y se quemaron 413,215.95 hectáreas de bosques y selvas, por otra parte, el año 2014 fue el año con menos incendios registrados siendo que se registraron solo 5,325 incendios con una afectación de 155,534 hectáreas, en el 2015 se presentaron 3,809 incendios afectando a 88,538.14 hectáreas, y de enero a agosto del 2016 se presentaron 8,599 incendios afectando a 261,260.39 hectáreas. Hablando solo del estado de Quintana Roo en el 2013 se presentaron 71 incendios afectando 24,652.62 hectáreas colocando al estado como uno de los estados con mayor área afectada por incendios, por otra parte, en el 2014 se registraron 40 incendios afectando 1,409.25 hectáreas, en el 2015 se presentaron 81 incendios con una afectación de 5,572.17 (CONAFOR, 2015), y de enero a agosto del 2016 se presentaron 40 incendios afectando a 2,246.56 hectáreas. Siendo que en el 2015 Quintanaro llegó a ubicarse como el tercer Estado con mayor afectación de incendios forestales en mayo del mismo año (Rodríguez, 2015), pero el 2016 fue uno de los años con menos incendios para el estado de Quintana Roo llegando a estar en la posición 24 de los estados más afectados por incendios. (CONAFOR, 2016)

La gestión de los incendios forestales se puede dividir en tres etapas, las actividades antes del incendio o prevención, la etapa durante el incendio o combate, y el después del incendio. Siendo que la parte en donde se trata de atacar más los incendios es en la prevención de los mismos, ahí es donde la tecnología ha estado creciendo y se han estado buscando mejores maneras de prevenirlos. En un principio solo se tenía en cuenta las estadísticas de años pasados dando que estaciones secas y el clima eran los factores principales para la estimación de los incendios, pero en 1941 surge la idea de clasificar los tipos de combustibles mediante la fotointerpretación a través de la fotografía aérea, siendo que esta clasificación se llevó 30 años surgiendo de los mapas de vegetación los mapas de combustibles.

Hablando de los trabajos que existen con respecto a los incendios, podemos señalar el libro de Emilio Chuvieco titulado “Earth Observation of Wildland Fires in Mediterranean Ecosystems” (Chuvieco, 2009), en donde se realiza un análisis de las implicaciones globales cuando se presenta un incendio, considerando los efectos del incendio en la vegetación y su composición global atmosférica; al igual que los impactos del cambio climático esperado en los patrones de aparición de incendios. Este trabajo, aunque no se desarrolla en México nos sirve de pauta de lo que se ha estado haciendo en el mundo, para poder rescatar algunas ideas que podrían ayudar a prevenir y monitorear zonas propensas a incendios.

En 1972 con el lanzamiento del primer satélite del programa Landsat, se viene la llegada de las imágenes satelitales, pero con la escasa capacidad operacional que se tenía en ese entonces se abandona la idea de su utilización dado que los modelos de combustible son muy complejos. Y no es hasta 1977 con el rápido avance de las computadoras que se introducen las principales técnicas de clasificación digital que se usaron en los años venideros, como son la clasificación supervisada o algoritmo de máxima probabilidad, la clasificación no supervisada y el análisis de componentes principales. Y es en 1984 cuando se empiezan a considerar imágenes de múltiples espectros para las clasificaciones de los tipos de combustibles. (Chuvieco Salinero & Martín Isabel, 2004).

El programa Landsat ha ayudado a mejorar la manera en que se clasifica la superficie. En el trabajo “Land cover clasification using Landsat 8 Operational Land Imager data in Beijing, China” (Kun Jia, 2014) vemos como se ha desarrollado esta tecnología proveyendo una nueva fuente de información para poder monitorear las interacciones entre las actividades humanas y la naturaleza, siendo que la clasificación de los diferentes tipos de superficies nos permiten tener una idea de los cambios que se han dado en los diferentes tipos de superficies, no solo por actividades humanas, sino por la misma naturaleza. Este trabajo nos permite comparar la evolución de la tecnología, como es el programa Landsat 8 con respecto de sus predecesores.

El procesamiento de imágenes multiespectrales se refiere al análisis, interpretación y clasificación de los espectros adquiridos de un objeto específico. Siendo los espectros electromagnéticos más comunes, la corriente alterna que oscila en frecuencias entre 10^2HZ hasta 10^3Hz, las ondas de radio que oscilan en frecuencias entre 10^3Hz hasta 10^9Hz, las Microondas que oscilan en frecuencias de 10^9Hz hasta 10^{11}Hz, el infrarrojo que oscila en frecuencias de 10^{11}Hz hasta 10^{14}Hz, el espectro de luz visible que oscila en frecuencias de 10^{14}Hz hasta 10^{15}Hz, el ultravioleta que oscila en frecuencias de 10^{15}Hz hasta 10^{17}HZ los rayos x que oscilan en frecuencias de 10^{17}Hz hasta 10^{20}Hz y por último los rayos gama que oscilan en frecuencias de 10^{19}Hz hasta 10^{22}Hz (Fontal, 2005).

Existen muchos trabajos dedicados al procesamiento de las imágenes multiespectrales, tanto para conocer los avances tecnológicos que se han dado actualmente, como también la creación de nuevas técnicas que nos faciliten el poder clasificar y determinar las diferentes zonas y tipos de suelos en las imágenes que se están tomando. Podemos señalar algunos de esos trabajos como el de “Identification of Mangrove Areas by Remote Sensing: The ROC Curve Technique Applied to the Northwestern Mexico Coastal Zone Using Landsat Imagery” (Luis C. Alatorre, 2011), donde podemos ver el uso de estas tecnologías en el desarrollo de nuevas técnicas para la clasificación de las imágenes; “Advances in Hyperspectral Image Clasificaction” (Gustavo Camps-Valls, 2013), donde se analizan los diferentes avances en la clasificación de imágenes hiperespectrales en

comparación de las imágenes fotográficas comunes. Libros como el de Jian Guo Liu “Essential Image Processing and GIS for Remote Sensing” (Jian Guo Liu, 2009) que ve el crecimiento de las imágenes multiespectrales como una oportunidad para poder obtener nueva información que hasta le pueden permitir determinar de la composición de la tierra a través de los diferentes espectros electromagnéticos y la manera en que los compuestos químicos las reflejan.

Los sensores multiespectrales actuales son capaces de adquirir cientos o miles de imágenes simultáneamente, correspondientes a diferentes longitudes de onda muy cercanas entre sí, dando como resultado cada pixel de la escena con la firma espectral característica de los diferentes objetos que aparecen en ella, expresando cada longitud de onda en diferentes bandas. Hablando del conjunto de bandas de conforman una imagen multiespectral estas pueden ser representadas gráficamente como un cubo de datos cuyas primeras dos dimensiones representan la ubicación espacial de un píxel determinado de la imagen y una tercera dimensión que representa la singularidad espectral de cada píxel para las diferentes longitudes de onda.

Teniendo en cuenta los diferentes problemas que pueden conllevar estas imágenes, hay que considerar que no siempre se tendrán imágenes perfectas, por lo que será necesario implementar diferentes filtros de corrección de imágenes, que nos permitan aprovechar esas imágenes al mejorarlas. Algunos trabajos que tocan el tema son “Order-Statistic Filtering and Smoothing of Time-Series: Part II” (E. Barner & R. Arce), donde nos explican la implementación de filtros WOS para el mejoramiento de imágenes; “Risk Estimation Without Using Stein’s Lemma – Aplication to Image Denoising” (Sagar Venkatesh Gubbi, 2015), donde tratan de resolver el problema de ruido en imágenes a través de la estimación de riesgos sin el lema de Stein; “LLSURE: Local Linear SURE-Based Edge-Preserving Image Filtering” (Tianshuang Qiu, 2013), donde proponen un nuevo enfoque a la realización de filtros de alta calidad para la preservación de los bordes en las imágenes; “Parallel K-Means Clustering of Remote Sensing Images Based on MapReduce” (Zhenhua Lv, 2010), donde presentan un método básico para el análisis de imágenes, el cual genera una visión directa de los objetos.

Algunos trabajos ya han ido lo suficientemente lejos como lo que se espera lograr de este trabajo, al crear una correlación con las imágenes satelitales multiespectrales con respecto a los incendios, como son el trabajo “Mapping Forest and Peat Fires Using Hyperspectral Airbone Remote-Sensing Data” (V. V. Kozoderov, 2012), donde confirman algunos métodos que se encuentran en desarrollo que son lo suficientemente efectivos con la reducción dimensional de la característica espacial del espectro original y el decremento del volumen de muestras en procedimientos supervisados para la selección de clases de objetos; “Forest fire risk zone mapping from satellite images and GIS for Baihe Forestry Bureau, Jilin, China” (XU Dong, 2005), donde crean un método integrador del sensado remoto y GIS el cual fue usado para mapear el riesgo de incendio forestal en la oficina forestal Baihe; “Fire-prone areas delineated from a combination of the Nesterov Fire-risk Rating Index with multispectral satellite data” (Milan Onderka, 2010), donde presentan una síntesis del índice de riesgo de fuego de Nesterov y como se relaciona con el Índice de sequedad y temperatura de la vegetación (TDVI) derivado de las imágenes multiespectrales Landsat.

1.2. Definición del problema

Actualmente los índices de evaluación de prevención de incendios en zonas naturales, como son bosques, selvas, llanuras, sabanas, entre otras, están basados en métodos de relaciones estadísticas entre condiciones climáticas y el número de brotes de fuego observados en el área; estos índices de predicción de incendios no son del todo confiables, pues están basados en algo tan cambiante como es el clima, y no pueden generar la distribución espacial de los sitios susceptibles a incendios en un área grande de bosque. Lo anterior se debe a que los únicos datos que llegan son de estaciones de observación meteorológicas que están en lejanas entre sí. Debido a esto se están buscando nuevos y mejores métodos de predicción que permitan una mejor prevención de incendios, que no solo se enfoquen en el comportamiento del fuego según el tipo de combustible que se tiene en la zona y contemple las imágenes desde un satélite o un dron que se encuentra en movimiento. El programa presentado es este trabajo será un nuevo método para

poder determinar las zonas propensas a incendios utilizando las nuevas tecnologías de hoy en día, facilitando la labor de prevención.

1.3. Justificación

Dado que es necesario implementar una mejor manera de clasificar las zonas más propensas a incendios de una manera más exacta, que no esté basada en el comportamiento del clima, se realizó un sistema de clasificación de zonas propensas a incendios utilizando imágenes multiespectrales, las cuales dependiendo de las frecuencias que emitan las zonas puedan determinar la salud o humedad relativa de las zonas, siendo las zonas más secas las más propensas a incendios. Para esto se pueden utilizar índices de salud de la vegetación, que se obtienen a partir del análisis de las imágenes multiespectrales. Para ello nos apoyaremos con el lenguaje de programación OpenCL que permite una interacción directa entre unidades centrales de procesamiento (CPU) y unidades de procesamiento grafico (GPU) lo que permite cálculos más rápidos y complejos para tener un análisis más certero de las imágenes multiespectrales con menores costos computacionales. Esto es particularmente importante dada la gran cantidad de imágenes que se deben procesar.

1.4. Objetivo General

Desarrollar una aplicación que ayude a determinar qué áreas son más propensas a sufrir de un incendio para que se puedan tomar las medidas preventivas necesarias.

1.4.1. Objetivos Específicos

- Desarrollar un software que lea imágenes multiespectrales y las convierta en sus especificaciones matemáticas usando OpenCL y el método WOS[1].
- Delimitar una manera para determinar que colores refieren a una zona natural saludable o seca.

[1] Weighted Order Statistics (Orden Estadístico Ponderado)

- Implementar el método de mejoramiento LLSURE[2].
- Determinar las temperaturas de la zona usando la banda infrarroja de temperatura.
- Calcular los valores para el NDVI[3] y TDVI[4].
- Determinar las áreas más propensas a incendios.

1.5. Alcances y Restricciones

El proyecto se realizó con imágenes satelitales del programa Landsat 8, obtenidas de manera gratuita en la página del Servicio Geológico de los Estados Unidos USGS por sus siglas en ingles. Estas imágenes tienen un espectro térmico que puede convertirse en valores de temperatura por lo que no será necesario recolectar valores reales de temperatura de centros de información climática y se usarán imágenes con poca o nula nubosidad por lo que no se trabajara la parte de la corrección de imágenes. La zona que se estudió son los alrededores de la bahía de Chetumal y el tamaño de la zona es la del tamaño de la imagen disponible en la página de la USGS.

Se buscaron mapas e información en la página del INEGI, donde se encontraron diferentes tipos de mapas topográficos, urbanos y rurales, de uso de suelo y vegetación, climatológicos, hidrológicos, edafológicos, fisiográficos, geológicos, humedales potenciales, territorio insular, y uso potencial de suelo. Todos estos mapas ya están procesados y presentados como un producto final con un objetivo específico, que difiere de los que se requieren para el trabajo como el usado en el programa Landsat donde se tienen en sus diferentes componentes o espectros, como son la banda del rojo, el infrarrojo cercano y el infrarrojo de temperatura.

[2] Local Linear SURE-Based Edge-Preserving Image Filtering
[3] Normalized Difference Vegetation Index (Índice de vegetación de diferencia normalizada)
[4] Temperature Dryness Vegetation Index

Capítulo 2. Marco Teórico

2.1. Filtro WOS (Orden Estadístico Ponderado)

Los filtros WOS pertenecen a la rama de los filtros de media ponderada, estos tipos de filtros se han implementado desde que Tukey sugirió por primera vez el filtro de media como suavizador en 1974 (E. Barner & R. Arce).

El esquema de ponderamiento se ha usado en los filtros de media ponderada como un método para enfatizar en muestras ciertos lugares de la ventana de observación y desenfatizar otras. Sin embargo, la salida del filtro está restringida a ser la media del peso del conjunto expandido, esta falta de libertad en elegir el rango de la salida puede en algunos casos limitar el desempeño del filtro. Esta limitante puede ser eliminada permitiendo que el rango de salida sea un parámetro ajustable, lo que nos lleva a los filtros WOS, que incluyen a los filtros de media ponderada y todos los filtros del orden del rango como subconjunto.

La operación de una ventana de tamaño N para un filtro WOS está definida por el elemento N del vector de peso **w** y el parámetro del rango ω_0. Para valores enteros positivos de los parámetros peso y rango, la salida del filtro WOS se calcula como

$$y(n) = \omega_0^{\prime th} Largest[x(n) \diamond w] \qquad (1)$$

Nótese que si $\omega_0 = \frac{1}{2}(1 + \sum_{i-1}^{N} \omega_i)$ para pesos enteros, y para no enteros $\omega_0 = \frac{1}{2}\sum_{i-1}^{N} \omega_i$, entonces el filtro WOS se reduce a un filtro de media ponderada. Los filtros WOS también continen filtros de orden de rango como un caso especial al restringir cada uno de los pesos como la unidad, $\omega_i = 1 \;\; i = 1,2, \dots, N$ por lo que la salida del filtro WOS se convierte a $y(n) = \omega_0^{\prime th} Largest[x(n) \diamond w] - x(\omega_0)$ donde $x(1), x(2), \dots, x(N)$ son el orden estadístico. Existen muchas aplicaciones en donde la inclusión de los filtros de orden de rango es muy efectiva como es la demodulación de las señales AM donde la salida del rango es seleccionada, así como el seguimiento de la función envolvente de la señal AM.

2.2. SURE (Riesgo Estimado Imparcial de Stein)

Para conocer del riesgo estimado imparcial de Stein, primero hay que conocer el teorema de Stein que se divide en dos partes, el teorema univariable y el teorema multivariable publicados en 1981.

Primero el teorema univariable, partiendo de que $Z \sim N(0,1)$ y que $f: \mathbb{R} \to \mathbb{R}$ es absolutamente continuo con la derivada f' y asumiendo que $\mathbb{E}|f'(Z)| < \infty$, se tiene que

$$\mathbb{E}[Zf(Z)] = \mathbb{E}[f'(Z)] \tag{2}$$

El cual se puede extender para cubrir una variable normal con una variancia y media arbitraria, $X \sim N(\mu, \sigma^2)$, para este caso se tiene

$$\frac{1}{\sigma^2}\mathbb{E}[(X-\mu)f(X)] = \mathbb{E}[f'(X)]. \tag{3}$$

Para el teorema multivariable se tiene que $X \sim N(\mu, \sigma^2 I)$, una variable normal de n dimensiones, con media $\mu \in \mathbb{R}^n$, y una matriz esférica de covariancia $\sigma^2 I \in \mathbb{R}^{n \times n}$. Siendo $f: \mathbb{R}^2 \to \mathbb{R}$ una función tal que para cada $i = 1, \dots, n$ y casi todo $x_{-1} \in \mathbb{R}^{n-1}$, se tiene la función

$$f(\cdot, x_{-1}): \mathbb{R} \to \mathbb{R} \tag{4}$$

Esta función es absolutamente continua y se le conoce como función f casi diferenciable. Cabe mencionar que esta función tiene derivadas parciales casi por todos lados las cuales se pueden denotar como $\nabla f = (\vartheta f/\vartheta x_1, \dots, \vartheta f/\vartheta x_n)$

Por lo que la resultante del teorema de multivariable de Stein teniendo una X, y la función f casi continúa satisfaciendo $\|f(x)\|_2 < \infty$ se tiene

$$\tfrac{1}{\sigma^2}\mathbb{E}[(X-\mu)f(X)] = \mathbb{E}[\nabla f(X)] \tag{5}$$

Por último, si se tiene que $f: \mathbb{R}^n \to \mathbb{R}^n$, escrita en función de coordenadas $f = (f_1, \dots, f_n)$ para cada $i = 1, \dots, n$, al usar todos los valores de i y sumarlos se tiene

$$\frac{1}{\sigma^2}\sum_{i=1}^{n} Cov\big(X_1, f_1(X)\big) = \frac{1}{\sigma^2}\sum_{i=1}^{n} \mathbb{E}\left[(X_i - \mu_i) f_1(X)\right] = \mathbb{E}\left[\sum_{i=1}^{n} \frac{\vartheta f_1}{\vartheta x_1}(X)\right] \tag{6}$$

Ahora que se conocen los teoremas de Stein, podemos hablar del SURE denotado por $\hat{\mu}$ el cual provee un estimado de la media subyacente μ. Esta estimación empieza al expandir

$$\mathbb{E}\|\mu - \hat{\mu}\| = \mathbb{E}\|\mu - y + y - \hat{\mu}\|_2^2 = n\sigma^2 + \mathbb{E}\|y - \hat{\mu}\|_2^2 + 2\mathbb{E}(\mu - y)^T(y - \hat{\mu})$$

$$= -n\sigma^2 + \mathbb{E}\|y - \hat{\mu}\|_2^2 + 2\sum_{i=1}^{n} Cov(y_1, \hat{\mu}_1) \tag{7}$$

Esta descomposición muestra que el riesgo $R = \mathbb{E}\|\mu - \hat{\mu}\|$ de $\hat{\mu}$ satisface

$$R = -n\sigma^2 + \mathbb{E}\|y - \hat{\mu}\|_2^2 + 2\sigma^2 df(\hat{\mu}) \tag{8}$$

Donde $\mathbb{E}\|y - \hat{\mu}\|_2^2$ es el error esperado del entrenamiento de $\hat{\mu}$, y sus grados de libertad están definidos como

$$df(\hat{\mu}) = \frac{1}{\sigma^2}\sum_{i=1}^{n} Cov(y_i, \hat{\mu}_i) \tag{9}$$

El teorema de Stein provee una estimado explícito de los grados de libertad del término $df(\hat{\mu})$, y también del riesgo de R. Siendo un poco específicos y sabiendo que $\hat{\mu}$ es casi diferenciable de la función y, tenemos que

$$\hat{R} = -n\sigma^2 + \|y - \hat{\mu}\|_2^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{\mu}_i}{\partial y_i}(y) \tag{10}$$

Esta ecuación es una estimación imparcial de R, i.e., $\mathbb{E}\big(\hat{R}\big) = R$. Esta estimación es conocida como la “Estimación imparcial del riesgo de Stein” o SURE por su acrónimo en inglés.

Esta estimación puede ser una herramienta extremadamente útil. Por un lado, se puede calcular es riesgo de un estimador, o incluso se puede usar para modelar diferentes propósitos. Si el estimador depende del parámetro de sintonía $\lambda \epsilon \Lambda$, denotado como $\hat{\mu}_\lambda$, entonces este parámetro se puede utilizar para minimizar el SURE dando como resultado

$$\hat{\lambda} = \underset{\lambda \epsilon \Lambda}{argmin} \|y - \hat{\mu}_\lambda\|_2^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{\mu}_{\lambda,i}}{\partial y_i}(y) \quad (11)$$

Para que esto sea útil es necesario calcular $\sum_{i=1}^{n} \frac{\partial \hat{\mu}_i}{\partial y_i}(y)$ para la estimación de $\hat{\mu}$ que es de nuestro interés y determinar que $\hat{\mu}$ es casi diferenciable para poder aplicar el teorema de Stein, esté cálculo es conocido como la divergencia de $\hat{\mu}$. Además, cuando se está minimizando SURE para elegir un parámetro de sintonía $\lambda \epsilon \Lambda$ es necesario algún tipo de argumento de concentración para demostrar que el parámetro λ posee buenas propiedades de riesgo.

2.3. LLSURE (Local Linear SURE)

El filtrado se puede considerar como la operación más importante del procesamiento de imágenes y visión computacional, y es utilizado en una gran gama de aplicaciones que van desde el suavizado y agudizamiento de imágenes, removimiento de ruido, mejoramiento y reducción de la resolución, extracción de características, y detección de bordes. Los filtros más comunes son los filtros lineales de transiciones invariantes (LTI por sus siglas en inglés), que tienen el problema de no solo reducir el ruido, sino que difuminan estructuras importantes con el ruido y valores típicos que ejercen una gran influencia en la salida filtrada.

Para reducir estos problemas se han propuesto una gran cantidad de técnicas desde la consideración de las estructuras locales y estadísticas durante el proceso de filtrado, hasta el filtrado por preservación de bordes que es un filtro no lineal y puede preservar los detalles de la imagen y las geometrías locales mientras se remueve el ruido no deseado.

Aunque los filtros de suavizado por preservación de bordes son muy usados para una gran variedad de tareas de edición y manipulación, originalmente fueron propuestos para remover el ruido mientras que se preservan los detalles finos y las estructuras geométricas de la imagen original. Se sabe que la eliminación de ruido es medida en términos del pico de señal con la proporción del ruido (PSNR por sus siglas en inglés), por lo que un alto PSNR indica que la reconstrucción es de alta

calidad. Para maximizar el PSNR, una alternativa es minimizar el error cuadrático medio (MSE por sus siglas en inglés), lo cual se puede estimar precisamente con el riesgo estimado imparcial de Stein. El SURE se ha convertido en una herramienta efectiva que puede ser aplicada directamente en la parametrización del estimador y encontrando los parámetros óptimos para minimizar el MSE estimado.

El filtro LLSURE está basado en el modelo local lineal y el principio del riesgo estimado imparcial de Stein. La salida del filtro en una ventana local es considerada como una simple transformada afín de la señal de entrada en la misma ventana, y los coeficientes de la transformada óptima son determinados al minimizar el SURE. El filtro LLSURE tiene las propiedades del suavizado por preservación de bordes que puede filtrar el ruido exterior al igual que preservar los bordes y los detalles a fina escala; además de ser muy simple y tener un algoritmo exacto de tiempo lineal que puede ser aplicado a varias tareas del procesamiento de imágenes.

La expresión de la estimada $\hat{x}$ juega un papel crucial en los métodos de eliminación de ruido basados en SURE, pero el LLSURE está basado en el modelo local lineal. Primero, en un vecindario local alrededor de cada posición, se asume que la salida filtrada de cada parte de la imagen es una transformada simple afín de cada parte de la imagen de salida en la misma posición. Segundo, para cada vecindario se determinan los coeficientes óptimos de la transformada al minimizar el MSE estimado con el SURE. Finalmente, todas las partes de la imagen de salida del filtro son promediadas juntas para obtener el resultado final filtrado. El proceso se describe a continuación.

Se tiene que ω_i es una ventana local alrededor de la posición i, y_ω y $\hat{x}_\omega$ son partes de la imagen de entrada y partes de la imagen ya filtrada respectivamente correspondientes a la ventana ω_i. Si se considera a ω_i como una ventana cuadrada de pixeles con un tamaño fijo se tiene que

$$\hat{x}_{\omega_i} = a_i y_{\omega_i} + b_i, \qquad a_i, b_i \in \mathbb{R}, \qquad a_i \geq 0 \tag{13}$$

Donde a_i y b_i son los coeficientes de las transformadas afines asumiendo que son constantes en la ventana ω_i. Por lo tanto, para cualquier $j \in \omega_i$, se tiene que $\hat{x}_j =$

$a_j y_j + b_j$. Es importante notar que se restringió a_i para que no sea negativo para prevenir que los resultados de la salida del filtro sean muy lejanos a la información de entrada. Si colocamos la ecuación anterior en la expresión del SURE se tiene.

$$SURE(a_i, b_i) = \frac{1}{N_\omega} ||y_{\omega_i} - (a_i y_{\omega_i} + b_i)||^2 + \frac{2\sigma^2}{N_\omega} div_y\left(a_i y_{\omega_i} + b_i\right) - \sigma^2$$

$$= \frac{1}{N_w} ||y_{\omega_i} - (a_i y_{\omega_i} + b_i)||^2 + \frac{2\sigma^2}{N_\omega} a_i N_\omega - \sigma^2$$

$$= \frac{1}{N_\omega} ||y_{\omega_i} - \left(a_i y_{\omega_i} + b_i\right)||^2 + 2\sigma^2 a_i - \sigma^2 \qquad (14)$$

Donde $N_\omega = (2r+1)\times(2r+1)$ es el número de pixeles en la ventana ω_i y r es el radio de la ventana.

Para determinar las transformadas optimas de los coeficientes a_i y b_i correspondientes a cada ventana local ω_i, se minimiza el $SURE(a_i, b_i)$.

En la Figura 1 se puede apreciar un ejemplo del uso del LLSURE correspondiente a la eliminación de ruido con una proporción pico de la señal de ruido de 31.93 dB.

Figura 1.- Ejemplo de LLSURE (Tianshuang Qiu, 2013)

2.3. NDVI (Normalized Difference Vegetation Index)

El índice normalizado diferencial de vegetación (NDVI) es el índice más usado y conocido que tiene como objetivo separar el brillo que reflejan las plantas al del suelo, mar y edificaciones. Este índice se basa en el comportamiento radiométrico de la vegetación, relacionado con la actividad fotosintética y la estructura foliar de las plantas, permitiendo determinar la vigorosidad de la planta, dando valores en función de la energía absorbida o reflejada por las plantas en diversos espectros electromagnéticos.

Los valores producidos por la vegetación sana muestran un claro contraste entre el espectro visible especialmente en la banda roja y el infrarrojo cercano. Mientras que el espectro visible es absorbido por los pigmentos de las hojas en el infrarrojo cercano, las células de las hojas saludables, que se encuentran llenas de agua, las cuales reflejan la mayor cantidad de energía. En el caso de las plantas no saludables que contienen menos agua disminuyen su reflectividad en el infrarrojo cercano y aumenta paralelamente en el rojo. Este hecho es el que permite separar la vegetación sana de las zonas con escasa o nula vegetación.

El cálculo del NDVI es la diferencia del infrarrojo cercano (NIR) con respecto a la banda del rojo divido por la suma del infrarrojo cercano con la banda del rojo.

$$NDVI = \frac{NIR - Rojo}{NIR + Rojo} \tag{15}$$

Dado que el NDVI es un índice no dimensional sus valores van de -1 a +1 donde los valores por debajo de 0.1 corresponden a los cuerpos de agua y a la tierra sin vegetación, mientras que los valores más altos son indicadores de la existencia de actividad fotosintética de los diferentes tipos de vegetación.

En la Figura 2 podemos observar la evolución del NDVI antes de un incendio (1990) y después de un incendio (1991, 1994, 1997) del Prepirineo, España, donde podemos observar la existencia de vegetación, su desaparición y como se va regenerando con el paso de los años.

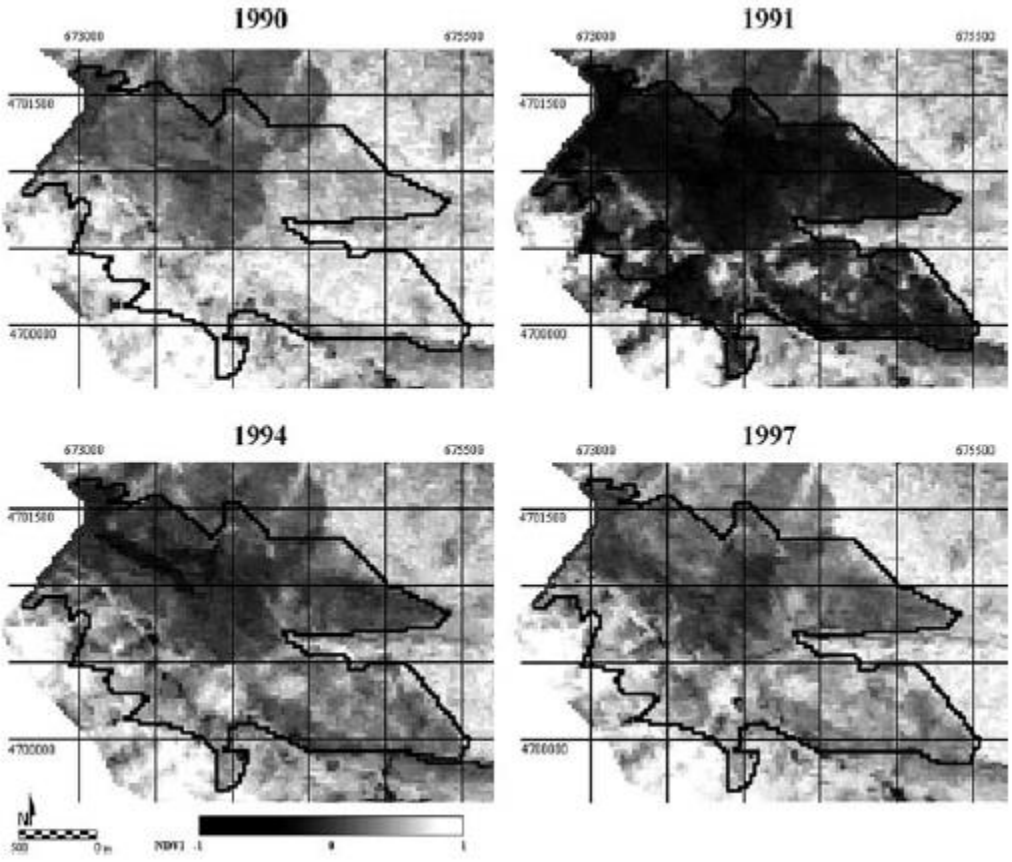

Figura 2.- Ejemplo de NDVI (Chuvieco, 2009)

2.4. TVDI (Temperature Vegetation Dryness Index)

El índice de sequedad en la vegetación y temperatura fue creado para poder establecer el estado de la humedad en las zonas de vegetación. Este índice demuestra una fuerte relación negativa entre la temperatura de la superficie de la zona y sus valores NDVI, lo cual puede ser explicado por la refrigeración evaporativa de la biomasa viva. Lo que quiere decir que cuando las condiciones en la superficie se vuelven más secas, la vegetación transpira menos agua y cuando hay largos periodos de sequía los valores del NDVI disminuyen por lo que la temperatura tiende a incrementarse debido al enfriamiento evaporativo impedido.

El TVDI se puede calcular con la diferencia de la temperatura en la superficie con respecto a la mínima temperatura de la superficie de la zona, dividido entre la diferencia de la temperatura mínimo de la superficie de la zona con respecto al NDVI que multiplica al parámetro b y la suma de a. Los parámetros a y b se calculan con

respecto a las variaciones de las temperaturas en la superficie y los valores del NDVI.

$$TVDI = \frac{T_S - T_{S\_min}}{a + b * NDVI - T_{S\_min}} \tag{16}$$

Para poder calcular los valores de a y b nos basamos en la dispersión de los pixeles con respecto a sus valores de NDVI, definiendo el límite seco y el límite húmedo formando así un triángulo como se puede apreciar en la Figura 3. Los valores de a y b son parámetros propios de los datos que definen el límite seco como una relación lineal entre los datos.

Para el cálculo de a y b se podría pensar en la ecuación de una pendiente para facilitar su comprensión. Y solo se tendrían que buscar los parámetros de la pendiente y el corrimiento correspondientes al valor de a y b para nuestro conjunto de datos de temperatura. Por lo que la ecuación de a y b quedan como.

$$b = \frac{(n * \sum(x * y)) - (\sum x * \sum y)}{(n * \sum x^2) - (\sum x * \sum x)} \tag{17}$$

$$a = \frac{\sum y - (b * \sum x)}{n} \tag{18}$$

Donde n es la cantidad de elementos máximos existentes en nuestro conjunto de datos de temperatura con respecto al NDVI, “y” son los valores máximos de la temperatura con respecto a la posición del NDVI, y “x” son los valores NDVI en las posiciones de la temperatura.

El modelo TDVI asume que la humedad del suelo es la principal fuente de variación de la temperatura en la superficie y que sus cambios son debidos a la inercia térmica, al control de la evaporación y transpiración de las plantas. Por lo que sus límites corresponden a los valores máximos de TDVI = 1 para el límite húmedo y TDVI = 0 para el límite seco.

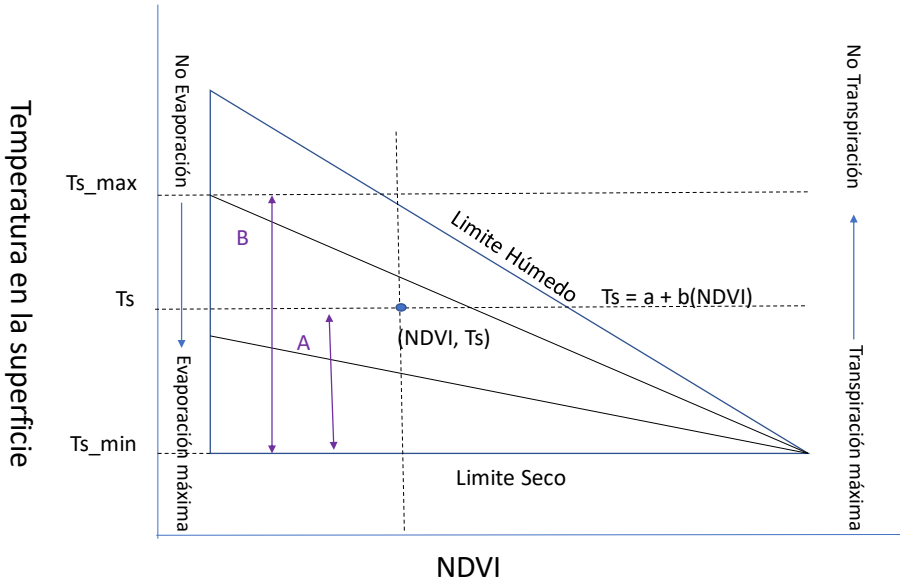

Figura 3.- Diagrama conceptual de TDVI

Un ejemplo de TDVI se puede apreciar en la Figura 4 donde se puede apreciar un mapa de Senegal que tiene aplicado el TDVI, donde las áreas más brillantes representan condiciones secas y las más oscuras zonas húmedas.

Figura 4.- Ejemplo de TDVI (Inge Sandholt, 2002)

Capítulo 3. OpenCL (Lenguaje de Cómputo Abierto)

Open Computing Language, OpenCL, es un estándar en la industria para programar computadoras compuestas de una combinación de CPUs, GPUs y otros procesadores. Estos sistemas heterogéneos se han convertido en una importante clase de plataformas, siendo OpenCL el primer estándar en la industria que atiende las necesidades de éstas. Su primera versión fue liberada en diciembre del 2008, con sus primeros productos disponibles en otoño del 2009.

Con OpenCL, se puede crear un programa que pueda funcionar en un vasto rango de sistemas, desde celulares, laptops, o nodos de supercomputadoras. Ningún otro tipo de programación en paralelo tiene ese tipo de alcance, siendo esta la razón de la importancia de OpenCL y el potencial que tiene para transformar la industria de software.

Actualmente OpenCL está disponible para diferentes equipos con diferentes capacidades, las diferentes versiones disponibles se pueden apreciar en la página principal de OpenCL que está a cargo de Khronos Group en su sitio web https://www.khronos.org/opencl/.

3.1. Modelos de OpenCL

Hablando de la manera en que OpenCL provee un sistema heterogéneo para explotar los recursos de computo en una plataforma que soporte una gran variedad de arquitecturas de hardware, se basa en modelos jerárquicos que describen la estructura de programación en OpenCL. Estos modelos son el modelo de Plataforma, el modelo de Memoria, el modelo de ejecución y el modelo de Programación.

3.1.1. Modelo de Plataforma

En el computo heterogéneo, es crítico conocer acerca de la arquitectura de los dispositivos con los que trabajamos para poder acceder a todos los beneficios del hardware.

El modelo de plataforma consiste en un equipo conectado a uno o más dispositivos de computo como CPUs o GPUS, donde cada dispositivo de OpenCL cuenta con

uno o más unidades de computo, los cuales están divididos en elementos de procesamiento. Los cálculos en el dispositivo, el cual es nuestro actual kernel, se ejecutan dentro de estos elementos de procesamiento, al conjunto de estos elementos de procesamiento podemos llamarlos objetos de trabajo o work ítems. Cabe mencionar que uno de los dispositivos de computo será el encargado de coordinar la ejecución de las tareas, mientras que los demás serán los que ayuden a la ejecución del código OpenCL. En la Figura 5 podemos apreciar cómo está distribuido el modelo de plataforma.

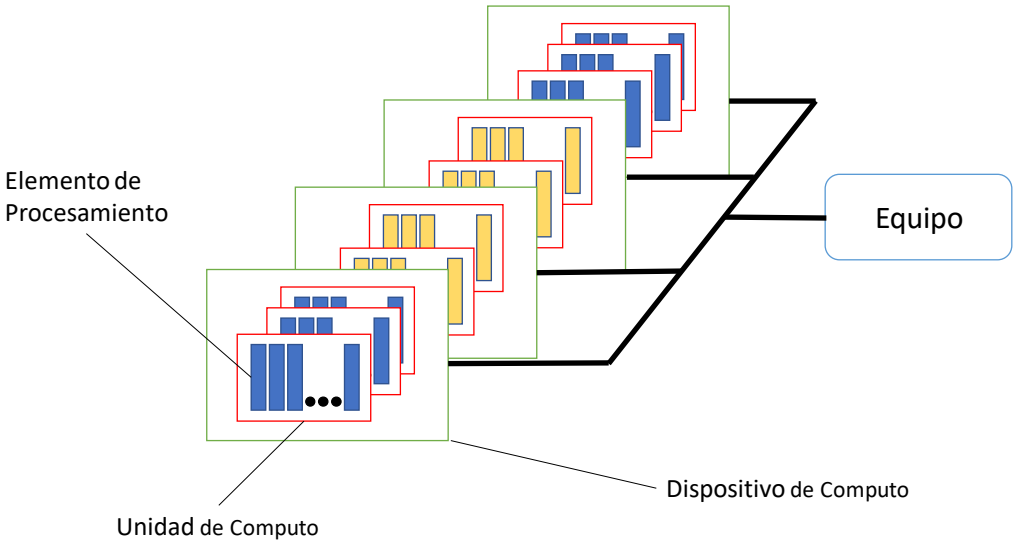

Figura 5.- Modelo de Plataforma de OpenCL

3.1.2. Modelo de Memoria

El modelo de memoria de OpenCL define una jerarquía abstracta de los tipos de memoria que se usan en el kernel, esta jerarquía puede apreciarse en la Figura 6, esta jerarquía es muy parecida a la jerarquía de memoria del GPU, con la diferencia de que este modelo no tiene límite de adaptabilidad por otros dispositivos de computo. El mapeo no es de 1 a 1 como en otras jerarquías de memoria. Sin embargo, la estructura básica de la memoria global es superior en contraparte a memoria local en cada grupo de trabajo, lo cual es consistente en ambas

plataformas. Avanzando más, los niveles más bajos de ejecución tienen pequeños espacios de memoria privada para los registros del programa.

Los grupos de trabajo se pueden comunicar a través de la memoria compartida y la sincronización primitiva, sin embargo, su acceso a la memoria es completamente independiente a otros grupos de trabajo, lo que es esencial para la ejecución del modelo en paralelo, donde el dominio de la independencia en las unidades de ejecución esta cercanamente ligado y definido por los patrones de acceso a la memoria.

Un importante problema que se debe tener en cuenta cuando se programan kernels es que el acceso en la memoria DRAM global y los bloques de memoria local no se protegen de ninguna manera. Esto significa que errores de segmentación no son reportados cuando los work ítems referencian a memoria fuera de su almacenamiento global. Como resultado la memoria del GPU separada para el sistema operativo puede ser dañada por accidente, lo cual puede resultar en comportamientos extraños de la pantalla o hasta poder colgarse el sistema.

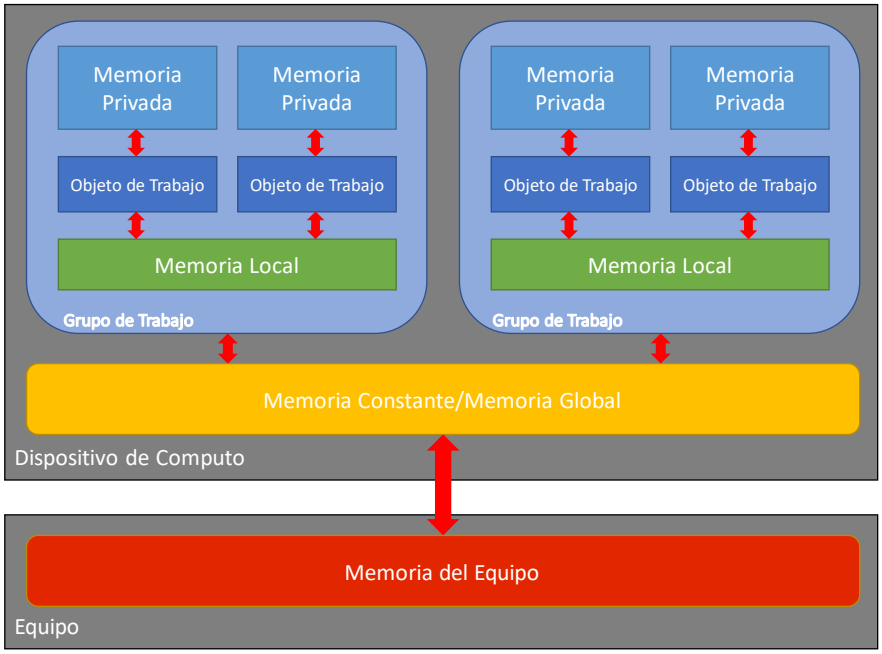

Figura 6.- Jerarquía de Memoria

3.1.3. Modelo de Ejecución

Define como el ambiente de OpenCL es configurado en el equipo y como el kernel será ejecutado en el dispositivo. Esto incluye crear un contexto en el equipo, proveer mecanismos para la interacción equipo-dispositivo, y definir un modelo concurrente para la ejecución del kernel en el dispositivo.

En el nivel más alto el equipo usa a la plataforma de OpenCL para llamar y seleccionar a los dispositivos, enviándoles trabajos a los dispositivos y manejando la carga de trabajo a través del contexto y las colas de trabajo. Por el otro lado, los niveles más bajos de la jerarquía son los kernels donde funcionan cada uno de los elementos de procesamiento. Estos kernels ejecutan las tareas sobre un dominio computacional de *n* dimensiones predefinido, donde cada elemento independiente del dominio de ejecución es llamado objeto de trabajo o work ítem. Estos objetos de trabajo se agrupan en grupos de trabajo independientes los cuales pueden son llamados por las diferentes tareas de manera separada o conjunta dependiendo de lo que se necesite.

3.1.4. Modelo de Programación

Define como el modelo de concurrencia es mapeado al hardware físico. Tradicionalmente se refiere al modelo donde la concurrencia es expresada como instrucciones desde un solo programa aplicado a los diferentes elementos dentro del conjunto de estructuras de información. En OpenCL el modelo es un conjunto de instrucciones de un solo programa que es aplicado al mismo tiempo a cada punto dentro del dominio abstracto de índices.

3.2. Tipos de variables en OpenCL

Al igual que cualquier lenguaje de programación OpenCL utiliza diferentes tipos de datos para poder realizar los cálculos necesarios o almacenar la información que se utiliza. Aquí se presenta una lista de los diferentes tipos de datos que utiliza OpenCL.

Tipos de Variables en OpenCL		
Tipo de dato OpenCL	**Tipo de data API**	**Descripción**
Bool	No cambia	1 bit, 0 o 1
Char	cl_char	8 bits con signo

Uchar	cl_uchar	8 bits sin signo
Short	cl_short	16 bits con signo
Ushort	cl_short	16 bits sin signo
Int	cl_int	32 bits con signo
Uint	cl_uint	32 bits sin signo
Long	cl_long	64 bits con signo
Ulong	cl_ulong	64 bits sin signo
Float	cl_float	32 bits de tipo flotante
size_t	No cambia	32 o 64 bits de tipo entero
Void	Void	Void
Variables para otros tipos de datos		
Variable	**Descripción**	
cl_event	Este tipo de variable se usa para declarar los eventos	
cl_context	Este tipo de variable se usa para declarar el contexto	
cl_program	Este tipo de variable se usa para declarar el programa en OpenCL que se ejecutara	
cl_kernel	Este tipo de variable se usa para declarar el Kernel	
cl_command_queue	Este tipo de variable se usa para el encolado de objetos	
cl_device_id	Este tipo de variable es para el dispositivo donde va a funcionar el programa y almacena su identificador	
cl_device_type	Este tipo de variable es para los dispositivos donde trabajará el programa y puede almacenar el tipo de dispositivo CPU, GPU, otros	
cl_mem	Este tipo de variable se usa para declarar los objetos de memoria con los que se estarán	
cl_platform_id	Este tipo de variable se utiliza para almacenar el tipo de plataforma que se va a usar	
cl_context_properties	Este tipo de variable se utiliza para almacenar las propiedades del contexto	

Tabla 1.- Variables de OpenCL (Khronos Group, 2011)

3.3. Funciones de OpenCL

OpenCL tiene diferentes funciones propietarias que se usan para la creación y manejo de los diferentes objetos, memorias, kernels, entre otros, estas diferentes funciones se pueden clasificar en 3 tipos, las funciones de inicialización, ejecución y finalización, una descripción de las operaciones que pueden realizar las funciones se puede apreciar en la Tabla 2.

Inicialización	Ejecución	Finalización
Obtener una lista de los dispositivos y plataformas disponibles	Encolar las diferentes tareas	Leer los objetos de memoria
Crear Contexto, Command Queue y objetos de memoria	Si es necesario esperar a que se los eventos anteriores se terminen	Liberar los diferentes objetos
Leer el archivo del kernel		
Crear el objeto tipo programa		
Compilar el kernel		
Crear los objetos tipo kernel		
Colocar los argumentos del kernel		

Tabla 2.- OpenCL procedimientos de ejecución dentro del host

3.3.1 Funciones de Inicialización

clCreateContext.- Crea un context OpenCL con uno o más dispositivos. Los Contextos son utilizados por el tiempo de ejecución del OpenCL para manejar los objetos como el command-queue, la memoria, el programa, los objetos del kernel y para ejecutar los kernels uno por uno o más dispositivos especificados en el contexto.

clCreateContextFromType.- Crea un contexto de OpenCL desde un tipo de dispositivo que identifica el dispositivo específico que se va a usar.

clCreateCommandQueue.- Crea el commanqueue que permite la realización de operaciones en los objetos OpenCL como la memoria, el programa y los objetos del kernel. El commandqueue puede crear una cola de operaciones en orden. Tener múltiples commandqueue permite a la aplicación encolar múltiples e independientes comandos sin la necesidad de requerir sincronización.

clCreateProgramWithSource.- Crea un objeto de tipo programa para un contexto, y carga el código fuente especificado por el texto tipo *string* en el arreglo de dentro del objeto tipo programa. Los dispositivos asociados con el objeto tipo programa son los dispositivos asociados con el contexto.

clCreateBuffer.- Crea un buffer de memoria diferente a cero y regresa CL_SUCCESS si es buffer es creado correctamente, en caso contrario se regresa un valor nulo seguido de su respectivo error.

clCreateKernel.- Crea un objeto de tipo kernel diferente a cero y regresa CL_SUCCESS si el kernel es creado satisfactoriamente, en caso contrario regresa un valor nulo seguido de su respectivo error.

clBuildProgram.- Construye un programa desde un código fuente o el binario del programa para todos los dispositivos o solo los dispositivos especificados en el contexto del OpenCL asociado al programa. OpenCL permite a programas ejecutables a ser creados desde un código fuente o binario. Esta función debe ser llamada para crear el programa usando ya sea clCreateProgramWithSource o clCreateProgramWithBinary para construir el programa ejecutable para uno o más dispositivos asociados al programa.

clGetContextInfo.- Regresa alguno de los parámetros del contexto como pueden ser su contador de referencia, los dispositivos, o sus propiedades del contexto.

clGetProgramBuildInfo.- Regresa la información de creación para cada dispositivo en el objeto de tipo programa.

clGetKernelWorkGroup.- Regresa información acerca del objeto de tipo kernel que puede estar asociado a un dispositivo en específico.

clGetPlatformInfo.-Regresa información acerca de la plataforma OpenCL, la información que puede regresar es el perfil de la plataforma, la versión de la plataforma, el nombre de la plataforma, el vendedor de la plataforma, y las extensiones que soporta la plataforma.

clGetPlatformIDs.- Regresa una lista de las plataformas disponibles en el sistema. Regresa CL_INVALID_VALUE si el número de entradas es igual a cero y las plataformas son no nulas o si tanto el número de plataformas y las plataformas es nula. Regresa CL_SUCCESS si la función es se ejecuta correctamente.

clGetDeviceIDs.- Regresa una lista de los dispositivos disponibles en la plataforma.

clGetEventProfilingInfo.- Regresa la información del perfil para el comando asociado al evento.

clSetKernelArg.- Establece el valor de los argumentos para un argumento específico de un Kernel.

3.3.2. Funciones de Ejecución

clEnqueueNDRangeKernel.- Encola un comando para ejecutar un kernel en un dispositivo.

clEnqueueMapBuffer.- Encola un comando para mapear la región del buffer de memoria a un buffer en la dirección del host y regresa un apuntador a dicha región mapeada.

clEnqueueReadBuffer.- Encola un comando para la lectura desde un objeto de tipo buffer a la memoria del host. Esto permite la utilización de los objetos de memoria en funciones dentro del host fuera de OpenCL.

clEnqueueUnmapMemObject.- Encola un comando para desmapear una región previamente mapeada de un objeto de memoria. La lectura y escritura desde el host usando el apuntador creado por clEnqueueMapBuffer o clEnqueueMapImage se consideran completados.

clFinish.- Bloquea la ejecución hasta que todos los comandos OpenCL previamente encolados en command_queue sean emitidos a su dispositivo asociado y se hayan completado. No tiene ningún tipo de retorno hasta que todos los comandos encolados en command_queue hayan sido procesados y completados. También es un punto de sincronización.

3.3.3. Funciones de Finalización

clReleaseProgram.- Disminuye el contador de referencia del programa. El objeto de tipo programa es eliminado después de que todos los objetos de tipo Kernel asociados al programa hayan sido eliminados y el contador de referencia del programa se convierta en cero. Regresa CL_SUCCESS si la función se ejecuta satisfactoriamente. Regresa CL_INVALID_PROGRAM si el programa es un objeto no valido de tipo programa.

clReleaseMemObject.- Disminuye el contador de referencia de los objetos de memoria. Después de que el contador de referencia de los objetos de memoria llega a cero y que los comandos encolados para su ejecución en el command_queue que usan los objetos de memoria han terminado, los objetos de memoria son eliminados. Si la función se ejecuta satisfactoriamente regresa CL_SUCCESS, en caso de que el objeto de memoria no es un objeto de memoria valido regresa CL_INVALID_MEM_OBJECT.

clReleaseEvent.- Disminuye el contador de referencia del evento. Si la función es ejecutada satisfactoriamente regresa CL_SUCCESS. Si el evento no es un objeto de tipo evento valido regresa CL_INVALID_EVENT. El objeto de tipo evento será eliminado una vez que el contador de referencia llegue a cero, el comando específico identificado por este evento sea completado o terminado y que no hayan comando en el command_queue de un contexto que requiera la espera de que este evento sea completado.

clReleaseContext.- Disminuye el contador de referencia del contexto. Si la función es ejecutada satisfactoriamente regresa CL_SUCCESS. Si el contexto no es un contexto valido de OpenCL regresa CL_INVALID_CONTEXT. Después de que el contador de referencia del contexto se vuelva cero y de que todos los objetos asignados al contexto sean liberados, se eliminara el contexto.

clReleaseCommandQueue.- Disminuye el contador de referencia del command_queue. Si la función es ejecutada satisfactoriamente regresa CL_SUCCESS. Si el command_queue no es un command-queue valido regresa CL_INVALID_COMMAND_QUEUE. Después de que el contador de referencia del command_queue se vuelva cero y todos los comandos encolados hayan terminado, el command_queue será eliminado.

clReleaseKernel.- Disminuye el contador de referencia del Kernel. Si la función es ejecutada satisfactoriamente regresa CL_SUCCSES. Si el Kernel no es un objeto valido regresa CL_INVALID_KERNEL. El kernel es eliminado una vez que el número de instancias que son retenidas al kernel son cero y que el kernel ya no es necesitado por ningún comando de encolado que usen el kernel.

Capítulo 4. Implementación

4.1. Equipo de pruebas

En nuestro caso se hicieron pruebas en dos equipos diferentes, el primer equipo marca Dell modelo Inspiron 5547 que se utilizó contaba con la versión de OpenCL 1.2, un procesador Intel® Core i7-4510U a 2.00 GHz, 8GB de memoria RAM y tarjeta de video integrada Intel® HD Graphics Family, por lo que se utilizó el Intel® Paralell Studio XE 2016 que está disponible en la página https://software.intel.com/en-us/intel-parallel-studio-xe, y el SDK de OpenCL para Intel® disponible en https://software.intel.com/en-us/intel-opencl. En este equipo se hicieron las primeras pruebas con OpenCL siguiendo el tutorial del Filtro Sobel de Malideveloper y el SDK proporcionado por ellos, del SKD de Malideveloper se utilizó el Platform y sus archivos de configuración de la carpeta common que incluía el SDK disponible en http://malideveloper.arm.com/resources/sdks/mali-opencl-sdk/ estas pruebas fueron la base para el procesamiento de las imágenes y su mejoramiento usando LLSURE.

Un segundo equipo de marca Apple se utilizó para probar la compatibilidad entre dispositivos de OpenCL, probando exitosamente que el programa LLSURE funcionaba sin importar el sistema operativo o la arquitectura, solo con unos pequeños cambios en el origen del compilador y el SDK. Cabe mencionar que Apple incluye su SDK para OpenCL en su sistema operativo, por lo que solo se necesitó que el compilador supiera donde están ubicados los archivos del marco de trabajo de OpenCL.

Posteriormente se utilizó un tercer equipo marca Dell modelo Inspiron 5559 que igual contaba con la versión de OpenCL 1.2, un procesador Intel® Core i7-6500U a 2.50 GHz, 8 de RAM y tarjeta de video AMD Radeon R5 M335 de 4GB, por lo que se requirió la instalación de la versión SDK de OpenCL proporcionada por AMD en la página http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.

4.2. Kernels

El programa está compuesto por 11 Kernels en total, los primeros tres kernels calculan los valores máximos, mínimos, la varianza y el promedio de las imágenes, estos valores son necesarios para poder calcular el LLSURE, el 4° Kernel normaliza los valores para tener rangos de 0 a 255 y no valores de 0 a 1, los kernels del 5 al 7 calculan el LLSURE para mejorar la imagen y resaltar los detalles. El 8° Kernel calcula el NDVI con el resultado de las imágenes del espectro rojo y cercano al infrarrojo que pasaron por el LLSURE, el 9° Kernel procesa los espectros de temperatura para calcular la temperatura en la superficie, el décimo Kernel calcula el TDVI que es el resultado que nos interesa para poder conocer que zonas son propensas a incendios debido a sus niveles de sequedad en la vegetación, y el último Kernel convierte los valores resultantes de tipo flotante en valores de tipo char para convertirlos en imágenes bmp.

El Kernel “reduce_max_min” es el encargado de calcular los valores máximos y mínimos de las imágenes que pasen por este Kernel, este cálculo es realizado en paralelo tanto dentro del Kernel como por fuera, y al final nos proporcionara el valor máximo y mínimo calculado para la imagen.

```
__kernel void reduce_max_min(__global float* buffer,
                         __local float* scratch,
                         __global float* result,
                         const int lengt)
{
```

El Buffer de este kernel es el encargado de almacenar los datos de la imagen, la variable scratch tiene la función de almacenar momentáneamente los valores para que puedan ser comparados y poder determinar el resultado final, la variable result almacenará el valor resultante de todas las operaciones, y por último la variable lengt nos ayuda a determinar el tamaño del arreglo de los valores que se calcularan para facilitar el procesamiento en paralelo. Este kernel está basado en una operación “reduce”.

Los kernel “reduce_mean_variance1” y “reduce_mean_variance2” en realidad son parte de la misma operación, el primero hace los primeros cálculos para obtener el promedio y la varianza de la imagen, pero debido a que el cálculo es en paralelo,

se tiene que esperar a que terminen de realizar los diferentes cálculos en las diferentes memorias para hacer el último cálculo de la varianza, el segundo Kernel es el encargado de juntar los diferentes valores que obtuvieron las memorias para proporcionar un valor único de varianza y promedio.

```
__kernel void reduce_mean_variance1(__global float* buffer,
                                    __local float* scratch,
                                    __global float* result)
{

__kernel void reduce_mean_variance2(__global float* buffer,
                                    __local float* scratch,
                                    __global float* result,
                                    const int num_groups)
{
```

En este caso las variables tienen funciones similares, solo que como es un proceso en dos fases, las variables tendrán valores diferentes. El primer buffer tendrá los valores de la imagen original que se está procesando, mientras que el buffer del segundo kernel contendrá el arreglo resultante de valores del primer kernel. El scratch en ambos casos es para almacenamiento temporal para poder hacer operaciones y comparaciones a los valores sin afectar a los originales. La variable result del primer kernel será un arreglo de valores resultante de las operaciones, pero este no es el valor final que se desea. La variable result del segundo kernel es el valor final que se desea. Y por último el num_groups va a ser el tamaño de los grupos para el procesamiento en paralelo. Ambas funciones realizan una operación “reduce” en dos etapas para calcular el promedio y la varianza “en líne”, es decir, en una sola corrida, agregando un dato a la vez.

El Kernel “normaliza” es el encargado de cambiar el rango de valores de la imagen, en algunas ocasiones los valores que se reciben de la imagen tiene valores de 0 a 1 u otros rangos, lo que complica su utilización, por lo que es necesario cambiar ese rango de valores a otros que sigan representando su valor sin perder información, otra representación de estos valores es el rango de 0 a 255, el cual es el resultado del Kernel “normaliza”, y para ello es necesario conocer el valor máximo y mínimo de la imagen, que se calcula en los Kernel anteriormente mencionados.

```
__kernel void normaliza(const int ancho,
                        const int alto,
                        const float minimo,
                        const float maximo,
                        __global float *x)
{
```

Las variables ancho y alto corresponden a las dimensiones de la imagen que será procesada, las variables maximo y minimo corresponden al valor maximo y minimo de la imagen calculados en el kernel “reduce_max_min”, y x corresponde a los valores de la imagen, en esta misma variable se sobrescribirán los resultados de la normalización.

El Kernel donde se calcula el LLSURE se divide en dos partes, aun cuando son 3 Kernel que se usan en el programa, el tercer Kernel es igual al segundo Kernel de LLSURE, pero por el tamaño de las imágenes se requirió que la segunda parte del LLSURE se hiciera para cada imagen. En el primer Kernel del LLSURE se realizan la primera parte de los cálculos para obtener la varianza global, debido a que se requería que el cálculo para todos los pixeles ya hubiera concluido, se optó por realizar la segunda etapa de cálculos en un segundo Kernel, asegurando de esta manera que ya hubieran concluido los cálculos realizados por el primer Kernel del LLSURE.

```
__kernel void LLSURE_inicio(__global const float* restrict inputNIR,
                            const int ancho,
                            const int alto,
                            __global float *promNIR,
                            __global float *varianzaNIR,
                            __global float *aNIR,
                            __global const float* restrict inputRed,
                            __global float *promRed,
                            __global float *varianzaRed,
                            __global float *aRed,
                            const float varianza_globalNIR,
                            const float varianza_globalRED)
{
```

Las variables inputNIR e inputRed corresponden a los valores de las imágenes, las variables ancho y alto corresponden a las dimensiones de las imágenes, como las imágenes son del mismo conjunto de imágenes, estas tienen las mismas dimensiones. aNIR y aRed son variables para almacenar las operaciones

correspondientes a las varianzas respectivas de cada imagen. Cabe señalar que estas son operaciones entre varianzas. varianzaRed y varianzaNIR almacenan varianzas locales, estas varianzas son diferentes a la varianza global que se calcula con el kernel “reduce_mean_variance”. Las variables promNIR y promRed almacenan los promedios locales de la imagen. Y por último las variables varianza_globalNIR y varianza_globalRED almacenan las varianzas calculadas en el kernel “reduce_mean_variance”.

```
__kernel void LLSURE(__global const float* restrict inputNIR,
                          const int ancho,
                          const int alto,
                          __global float *promNIR,
                          __global float *varianzaNIR,
                          __global float *aNIR,
                          __global float *paNIR,
                          __global float *pbNIR,
                          __global float *NIR)
{
__kernel void LLSURE_final(__global const float* restrict inputRed,
                          const int ancho,
                          const int alto,
                          __global float *promRed,
                          __global float *varianzaRed,
                          __global float *aRed,
                          __global float *paRed,
                          __global float *pbRed,
                          __global float *RED)
{
```

Las variables inputNIR e inputRed almacenan los valore originales de las imágenes. Las variables ancho y alto corresponden a las dimensiones de las imágenes. promNIR y promRed almacena los valores de los promedios locales de las imágenes correspondientes calculados anteriormente en el kernel “LLSURE_inicio”. Las variables varianzaNIR y varianzaRed almacenan los valores de las varianzas locales que se calcularon en el kernel “LLSURE_inicio”. Las variables aNIR y aRed almacenan los resultados de las operaciones entre varianzas realizadas en el kernel “LLSURE_inicio”. Las variables paRed, paNIR, pbNIR y pbRed almacenarán operaciones entre las variables promRed, promNIR, varianzaNIR, varianzaRED, aRed y aNIR estas operaciones solo se realizan entre variables de la misma imagen. Las variables NIR y RED almacenaran el resultado final de la imagen procesada por LLSURE.

En el Kernel “NDVI” se realiza una operación para resaltar los detalles de la imagen usando el resultado del LLSURE y calculando las diferencias entre la imagen de LLSURE. Ya teniendo la nueva imagen con mejor calidad, se procede a calcular el NDVI que es el resultado de la diferencia entre el espectro del infrarrojo cercano con respecto al espectro del rojo dividido por la suma de estos mismos espectros.

```
__kernel void NDVI(const int ancho,
                   const int alto,
                   __global float *NIR,
                   __global float *entradaNIR,
                   __global float *entradaRED,
                   __global float *RED,
                   __global float *NDVI)
{
```

Las variables ancho y alto corresponden a las dimensiones de las imágenes. La variable NIR corresponde a la imagen del espectro del infrarrojo cercano procesada por el LLSURE, la variable RED corresponde a la imagen del espectro del rojo procesada por el LLSURE, las variables entradaNIR y entradaRED corresponden a las imágenes originales, y la variable NDVI será la que almacena el resultado de las operaciones que realiza el kernel.

El Kernel “Temperatura” toma las imágenes del infrarrojo de temperatura, y convierte sus valores en temperatura en Kelvin, dado que se tienen dos imágenes que representan los valores de la zona, se calculó el promedio de la temperatura para cada valor. Los valores de K1 y K2 se obtuvieron del archivo de información de los metadatos del satélite que capturo las imágenes y se usaron los cálculos proporcionados por (Centro de Investigación y Desarrollo CIAF, 2013).

```
__kernel void Temperatura(__global const float* restrict inputTIR1,
                          const int ancho,
                          const int alto,
                          __global float *Temperatura,
                          __global const float* restrict inputTIR2)
{
```

Las variables inputTIR1 e inputTIR2 almacenan los valores de las imágenes del espectro del infrarrojo de temperatura, Landsat 8 tiene dos que en conjunto sensan todo el espectro del infrarrojo de temperatura. Las variables ancho y alto almacenan las dimensiones de las imágenes. La variable Temperatura almacena el resultado

final del kernel, siendo este un conjunto de datos del tamaño de la imagen que almacena para cada pixel su valor de temperatura.

En el Kernel “TDVI” se realizan los cálculos para calcular el Índice de sequedad y temperatura de la vegetación que es el que nos interesa para poder saber si la zona esta saludable o no, lo que nos dice si la zona por estar muy seca es propensa a los incendios o por el contrario si está sana no ser propensa. En este Kernel se utilizan los valores calculados de la temperatura y el NDVI.

```
__kernel void TDVI(__global float *Temperatura,
                   __global float *NDVI,
                   __global float *TVDI,
                   const int ancho,
                   const float a,
                   const float minim,
                   const float b)
{
```

La variable Temperatura almacena los valores de temperatura para cada pixel calculados en el kernel “Temperatura”, la variable NDVI almacena los valores del NDVI de cada pixel calculados anteriormente con el kernel “NDVI”, la variable ancho almacena el valor del ancho de la imagen correspondiente a las dimensiones de las imágenes. Las variables a y b son valores únicos que representan la correlación entre el NDVI y la temperatura, estos valores se calculan dentro del programa antes de entrar a este kernel. Y la variable minim almacena el valor mínimo de temperatura calculado en conjunto con los valores de a y b.

El Kernel “Imagenes” es el último Kernel del programa, ya habiendo calculado los valores que necesitamos, nos faltaba una forma para poder convertir esos valores en una imagen, para ello nos apoyaremos en el código proporcionado por el SDK de malideveloper que tiene una instancia para crear imágenes bmp, solo hace falta convertir los valores flotantes que estamos usando en valores de tipo char y mandar el tamaño de la imagen.

```
__kernel void Imagenes (__global float *NDVI,
                        __global float *TDVI,
                        __global uchar *ImageTDVI,
                        __global uchar *ImageNDVI,
                        const int ancho,
                        const float mintdvi,
                        const float maxtdvi,
                        const float minndvi,
                        const float maxndvi,
                        __global float *RED,
                        __global float *NIR,
                        __global uchar *ImageRED,
                        __global uchar *ImageNIR)
{
```

Las variables NDVI, TDVI, RED y NIR corresponden a los valores flotantes de las imágenes que serán convertidas a formato uchar. Las variables ImageTDVI, ImageNDVI, ImageRED e ImageNIR contendrán el resultado de la conversión. La variable ancho corresponde al valor del ancho de las dimensiones de las imágenes. Las variables mintdvi y minndvi corresponden a los valores mínimos que tienen las imágenes correspondientes en formato flotante. Y las variables maxtdvi y maxndvi corresponden a los valores máximos que tienen las imágenes correspondientes en formato flotante.

Capítulo 5. Resultados

En el caso del LLSURE para probar la implementación se utilizó la imagen proporcionada por el ejemplo del filtro de Sobel del SDK de malideveloper. Esta imagen corresponde a dos planetas, el formato es bmp. Sin embargo, la versión final de LLSURE procesa imágenes de formato TIFF, el cual es usado por Landsat 8. Considerando que las imágenes finales serían de una sola banda, el resultado final del LLSURE sería en términos de una sola banda, que se visualiza en escala de grises.

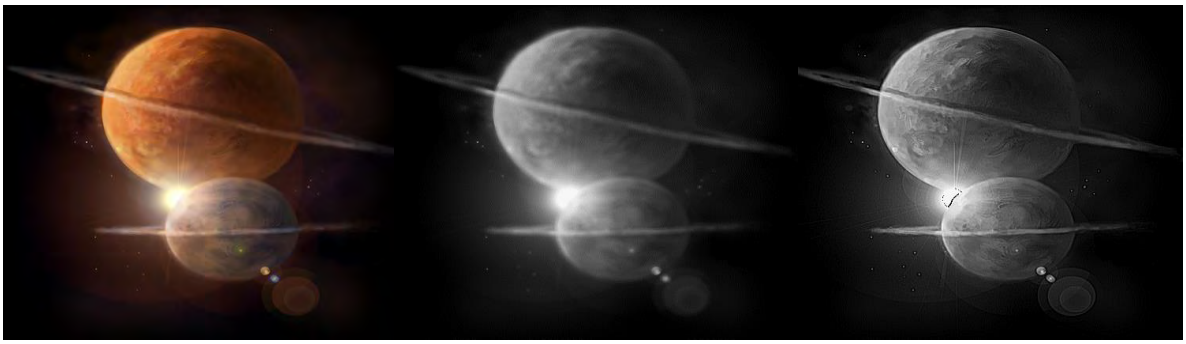

Figura 7.- Resultado LLSURE

En la Figura 7 se puede apreciar del lado izquierdo la imagen original que será procesada por el LLSURE, la imagen de en medio es el resultado del LLSURE, y la tercera imagen es el resultado al resaltar los detalles de la imagen original al calcular las diferencias existentes entre la imagen original y el resultado del LLSURE, y agregando esas diferencias en alguna de las imágenes utilizadas, en este caso el LLSURE. Esta tercera imagen es la que se usa para el programa final.

Las imágenes satelitales que se usaron para la implementación del programa final, fueron del programa Landsat 8, el cual consta de 9 bandas diferentes tomadas a una resolución de 30 m^2 por pixel y 2 bandas de temperatura tomadas a 100 m^2 por pixel. Todas las imágenes tienen las dimensiones de 7651 x 7811 pixeles. De todas estas imágenes solo se usaron las de la banda 4 correspondiente al rojo, la banda 5 correspondiente al infrarrojo cercano, y las dos bandas del infrarrojo térmico. Como se mencionó con anterioridad estas imágenes son del tipo TIR y pueden descargarse en “http://glovis.usgs.gov/”.

Es importante mencionar que la nubosidad de las imágenes es un factor importante a considerar en el momento de la selección de la imagen, mientras mayor sea la nubosidad menor es la información significante para nosotros, por lo que se buscó una imagen con mínimo de nubosidad, siendo que la mejor imagen al momento de la realización de este proyecto fue la del 14 de noviembre del 2014 que se puede apreciar en la Figura 8.

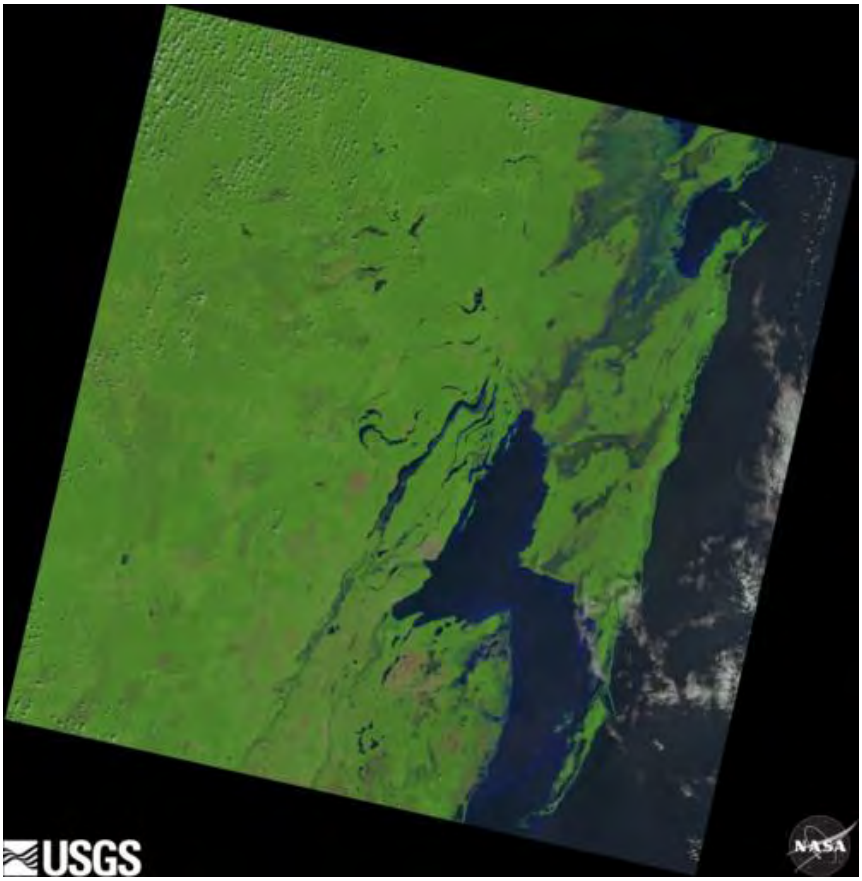

Figura 8.- Imagen Satelital Color Verdadero

Al calcular las temperaturas podemos ver los efectos de la nubosidad en el programa, siendo que hubo valores irreales en las zonas con nubosidad dando temperaturas de 148.523 grados kelvin. Aun así, estos valores no afectan los cálculos del TDVI debido a que este cálculo se realiza para cada pixel, teniendo cada pixel su propia temperatura, su propio valor NDVI y su propio TDVI resultante.

Por último, cabe mencionar que debido a falta de memoria física para realizar los cálculos en el equipo de pruebas y las grandez zonas negras en los bordes, se optó

por reducir el tamaño de la imagen que se estaría usando creando una sub-imagen dentro del mismo programa con dimensiones de 5000 x 5000 pixeles, como la que se puede apreciar en la Figura 9.

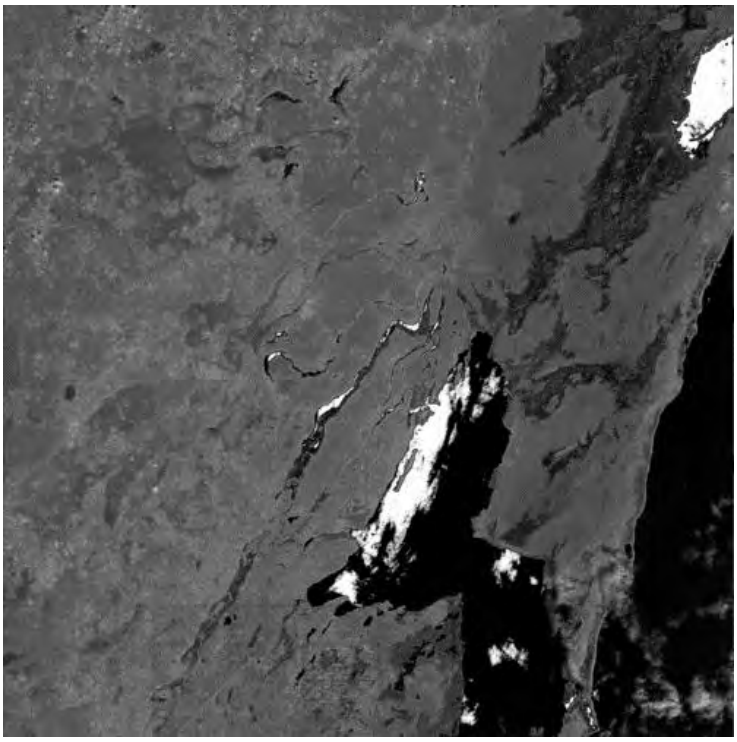

Figura 9.- Banda Infrarroja con LLSURE

Los resultados finales del NDVI y TDVI se pueden apreciar en la Figura 10 y la Figura 11 respectivamente, siendo que en el NDVI podemos apreciar la diferenciación de los cuerpos de agua y las ciudades con un color obscuro con respecto a la vegetación de colores claros. En la imagen del TDVI podemos ver algunos errores de identificación como son las nubes que son los objetos más obscuros, se pueden apreciar los lugares con nula vegetación como ciudades o con vegetación seca que es propensa a incendios, estos lugares se pueden apreciar de un color grisáceo oscuro, y los lugares más claros son los que cuentan con abundante vegetación y los cuerpos de agua que tienen altos índices de humedad.

En la Tabla 3 podemos ver los diferentes tiempos de ejecución del programa para los diferentes kernels, en tiempo de milisegundos, siendo que los kernels más rápidos tardan menos de un segundo, otros tardan 5 o 6 segundos y el kernel del LLSURE que es el que tarda más se toma 17.5 segundos en procesar las 2

imágenes que utiliza. El mayor tiempo que tarda el programa lo realiza al hacer cálculos fuera de las instancias de OpenCL y los kernels.

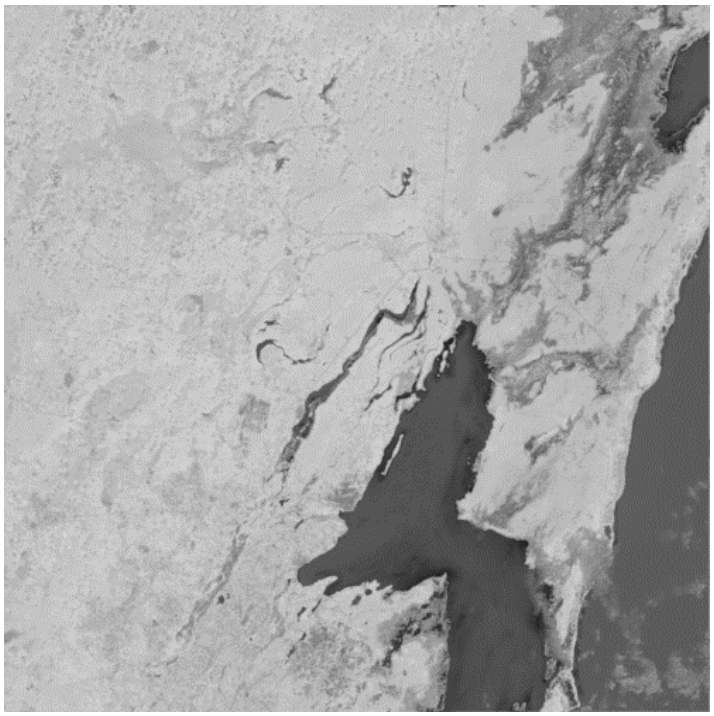

Figura 10.- NDVI

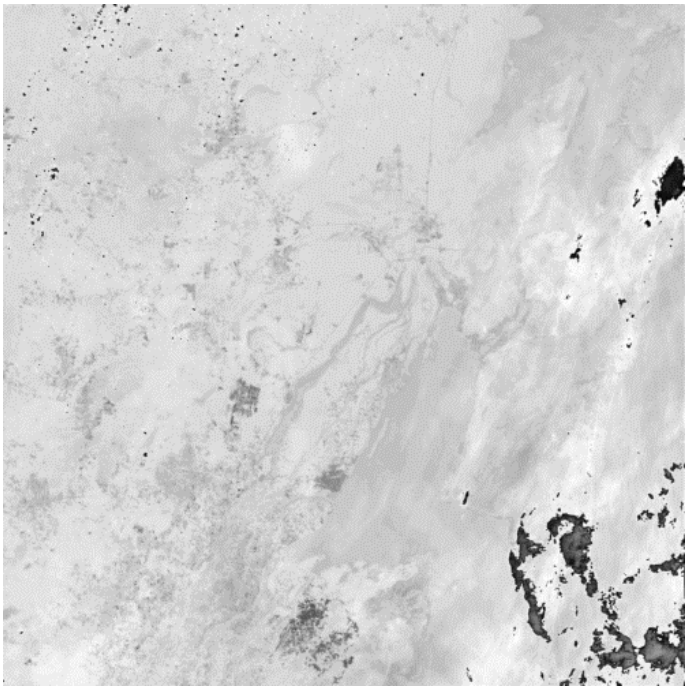

Figura 11.- TDVI

Tiempos de ejecución	
Información de los tiempos del Kernel LLSURE_inicio	
Tiempo de encolado	0.035756 ms
Tiempo de Espera	5.15994 ms
Tiempo de funcionamiento	17532.8 ms
Información de los tiempos del Kernel LLSURE	
Tiempo de encolado	0.026661 ms
Tiempo de Espera	5.08705 ms
Tiempo de funcionamiento	5552.59 ms
Información de los tiempos del Kernel LLSURE_final	
Tiempo de encolado	0.027394 ms
Tiempo de Espera	0.39258 ms
Tiempo de funcionamiento	6286.9 ms
Información de los tiempos del Kernel NDVI	
Tiempo de encolado	0.040124 ms
Tiempo de Espera	0.419914 ms
Tiempo de funcionamiento	785.065 ms
Información de los tiempos del Kernel Temperatura	
Tiempo de encolado	0.007886 ms
Tiempo de Espera	0.309456 ms
Tiempo de funcionamiento	414.054 ms
Información de los tiempos del Kernel TDVI	
Tiempo de encolado	0.008859 ms
Tiempo de Espera	0.398422 ms
Tiempo de funcionamiento	338.474 ms
Información de los tiempos del Kernel Imagen	
Tiempo de encolado	0.043973 ms
Tiempo de Espera	0.479145 ms
Tiempo de funcionamiento	660.49 ms

Tabla 3.- Tiempos de ejecución

Capítulo 6. Conclusiones y Trabajo Futuro

OpenCL es una opción para la aceleración de programas que procesan grandes cantidades de datos, contiene la opción de procesamiento en paralelo que reduce el tiempo requerido para realizar cálculos matemáticos. Disminuyendo el tiempo que normalmente tardaría el programa con otras implementaciones.

Nuestro programa es una buena opción para poder determinar zonas propensas a incendios a través del índice de sequedad de vegetación y temperatura. Con su única dependencia de las imágenes satelitales con nula nubosidad para evitar problemas por oclusión o la implementación de un método de corrección de imágenes que elimine la nubosidad sin afectar el contenido de la imagen.

Este programa está diseñado para funcionar con las imágenes del programa LANSAT 8 debido a que este programa ya incluye las bandas del infrarrojo de temperatura para poder hacer los cálculos necesarios. Dichas imágenes están disponibles en formato TIFF, por lo que la implementación propuesta está especializada en dicho formato.

Como se pudo ver en el capítulo de resultados, las imágenes finales del programa están en escalas de grises, lo que puede complicar su entendimiento, si se desconoce el significado de sus valores, por lo que se propone para seguir con el trabajo, una implementación que permita un entendimiento más sencillo al aplicar ciertos colores en las diferentes zonas dependiendo de los valores resultantes y su representación con respecto a una zona húmeda que es menos propensa a incendios y una zona seca que es propensa a incendios, un método para realizar esto puede ser el ParallelK basado en MapReduce explicado en (Zhenhua Lv, 2010).

Debido a que la zona por la que se optó trabajar es una zona normalmente húmeda y con bastante vegetación, los resultados que obtuvimos son casi uniformes, siendo que si se trabajara con una zona que cuente con una mayor variedad de ecosistemas que permitan hacer un contraste entre zonas secas y húmedas se podrían tener resultados más fáciles de apreciar. Por lo que la implementación del programa en diferentes lugares y en diferentes fechas podría ayudar a mejorar la

clasificación de las zonas propensas a incendios y tener una mejor predicción de estas para poder realizar la prevención necesaria.

Un problema que se tuvo fue la falta de imágenes gratuitas o a bajo costo con poca nubosidad en la zona, debido a que la solicitud de imágenes dedicadas al proyecto que cumplieran los requisitos para su implementación tenían costos que no se podían solventar con el presupuesto del proyecto, por lo que no se pudieron hacer las pruebas de campo con datos actuales, debido a que las imágenes disponibles eran de años anteriores, aun así el programa cumple con su objetivo dejando para trabajo futuro su implementación con imágenes más actuales para poder realizar los trabajos en campo que puedan sustentar los resultados logrados.

Hablando del programa y su implementación, es necesario que la persona que lo quiera usar tenga conocimientos básicos de programación para poder hacer las ediciones pertinentes para poder usar otras imágenes, y para poder realizar la migración del OpenCL al equipo que se quiera usar, siendo que debido a que OpenCL tiene diferentes implementaciones dependiendo de la plataforma en donde se encuentre, es necesario volver a compilar el programa para que se adapte al sistema en donde se quiere implementar, esto no limita los equipos donde se quiere implementar el programa, por el contrario permite que el programa pueda aprovechar las capacidades del equipo donde se quiere hacer funcionar el programa, lo que puede llevar a un mejoramiento en los tiempos de ejecución del programa dependiendo de las capacidades del nuevo equipo. Esto mismo puede llevar a un trabajo futuro donde se cree una implementación que permita un ambiente más grafico para usuarios con menos conocimientos en programación, que les permita desde un entorno grafico realizar los cambios pertinentes para la implementación del programa y les permita ver los resultados de una manera más sencilla o hasta su implementación en conjunto con el programa GIS y otros formatos de imágenes.

Con respecto a los tiempos de procesamiento, considerando el tamaño de los datos, el programa se lleva poco tiempo en procesar los datos, siendo lo más tardado, la parte de creación de las imágenes finales y los procesos que son fuera de los kernels. Esta es una de las ventajas que tiene el programa al utilizar OpenCL, y

tener partes del programa en procesamiento en paralelo, así puede procesar imágenes grandes en un tiempo relativamente corto, en comparación con otras opciones de procesamiento como OpenGL.

Bibliografía

Benedict R. Gaster, L. H. (2013). *Heterogeneous Computing with OpenCL.* Waltham, MA, Estados Unidos: Morgan Kaufmann.

Bhattacharyya, R. B. (2013). *OpenCL Programming by Example.* Birminham, Reino Unido: Packt Publishing LTD.

Centro de Investigación y Desarrollo CIAF. (2013). *Descripción y Corrección de Productos Landsat 8 LDCM.* Bogota, Colombia: Instituto Geográfico Agustín Codazzi.

Cervigón, J. J. (2015). *Estudio de Índices de vegetación a partir de imágenes aéreas tomadas desde UAS/RPAS y aplicaciones de estos a la agricultura de precisión.* Universidad Complutense de Madrid.

Chuvieco Salinero, E., & Martín Isabel, M. (2004). *Nuevas tecnologías para la estimación del riesgo de incendios forestales.* Madrid: Bouncopy S.A.

Chuvieco, E. (2009). *Earth Observation of Wildland Fires.* Alcala de Henares España: Springer Heidelberg Dordrecht.

CONAFOR. (2015). *Reporte Semanal de Resultados de Incendios Forestales 2015.* CONAFOR.

E. Barner, K., & R. Arce, G. (s.f.). Order-Statistic Filtering and Smoothing of Time-Series: Part II. *Applied Science and Engineering Laboratories*. Newark, Delaware: University of Delaware.

Fontal, B. (2005). *El Espectro Electromagnético y sus Aplicaciones.* Mérida, Venezuela: Escuela Venozolana para la Enseñanza de la Química.

Gasteratos, A., & Andreadis, I. (s.f.). A New Algorithm for Weighted Order Statistics Operations. En D. o. Engineering, *Electronics and Information Systems Tecnology.* Xanthi, Greece: Democritus University of Thrace.

Gustavo Camps-Valls, D. T. (2013). ADVANCES IN HYPERSPECTRAL IMAGE CLASSIFICATION [EARTH MONITORING WITH STATISTICAL LEARNING METHODS]. 1 - 10.

Inge Sandholt, K. R. (2002). A simple interpretation of the surface tecperature/vegetation index space for assessment of surface moisture status. *Remote Sensing of Enviroment 79*, 213 - 224.

Intel®. (22 de Julio de 2015). *Intel® Media Server Studio | Intel® Developer Zone*. Obtenido de Intel® Developer Zone: https://software.intel.com/en-us/intel-media-server-studio

Jian Guo Liu, &. P. (2009). *Essential Image Processing and GIS for Remote Sensing.* Singapore: WILEY-BLACKWELL.

Kenneth E. Barner, &. G. (1998). Order-Statistic Filtering and Smoothing of Time-Series: Part II. En E. Science, *handbook of statistics 17* (págs. 555 - 602). Elsevier Science.

Khronos Group. (2009). *The OpenCL Specification.* Aaftab Munshi.

Khronos Group. (2011). OpenCL API 1.2 Reference Card. www.khronos.org/opencl. Obtenido de www.khronos.org/opencl.

Khronos Group. (13 de Agosto de 2015). *OpenCL - The open standar for parallel programming of heterogeneous systems*. Obtenido de Open Standards for Media Autoring and Acceleration: https://www.khronos.org/opencl/

Khronos Group. (19 de Diciembre de 2016). *The Khronos Group Inc.* Obtenido de OpenCL-Details-Taiwan_June-2012: https://www.khronos.org/assets/uploads/developers/library/2012-pan-pacific-road-show-June/OpenCL-Details-Taiwan_June-2012.pdf

Kun Jia, X. W. (2014). Land cover classification using Landsat 8 Operational Land Imager data in Beijin, China. *Geocarto International*, 941 - 951.

Luis C. Alatorre, R. S.-A.-C. (2011). Identification of Mangrove Areas by Remote Sensing: The ROC Curve Technique Applied to the Northwestern Mexico Coastal Zone Using Landsat Imagery. *Remote Sensing*, 1568 - 1583.

Meneses-Tovar, C. L. (2011). El índice normalizado diferencial de la vegetación como indicador de la degradación del bosque. *Unasylva 238*, 39 - 46.

Milan Onderka, &. I. (2010). Fire-prone areas delineated from a combination of the Nesterov Fire-risk Rating Index with multispectral satellite data. *Appl Geomat*, 1 - 7.

Munshi, A., R. Gaster, B., G. Mattson, T., Fung, J., & Ginsburg, D. (2012). *OpenCL Programming Guide.* Boston, MA: Pearson Education, Inc.

Parinaz Rahimzadeh-Bajgiran, K. O. (2012). Comparative evaluation of the Vegetation Dryness Index (VDI), the Temperature Vegetation Dryness Index (TVDI) and the improved TVDI (iTVDI) for water stress detection in semi-arid regions of Iran. *ISPRS Journal of Photogrammetry and Remote Sensing*, 1 - 12.

Porter, R., Eads, D., Hush, D., & Theiler, J. (2003). Weighted Order Statistic Classifiers with Large Rank-Order Margin. *Proceedings of the Twentieth International Conference on Machine Learning*. Washington: Los Alamos National Laboratory.

Rodríguez, E. (27 de Mayo de 2015). Quintana Roo, tercer lugar en afectaciones por incencios forestales. *SIPSE Noticias*.

Sagar Venkatesh Gubbi, &. C. (2015). Risk Estimation Without Using Stein's Lemma - Application to Image Denoising.

Schlachter, J. T. (2013). *An Introduction to the OpenCL Programming Model.* Nueva York, Estados Unidos: Universidad de Nueva York.

Shulin Chen, Z. W. (2015). Temperature Vegetation Dryness Index Estimation of Soil Moisture under Different tree Species. *Sustainability*, 11401 - 11417.

Tianshuang Qiu, A. W. (2013). LLSURE: Local Linear SURE-Based Edge-Preserving Image Filtering. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 80 - 90.

V. V. Kozoderov, T. V. (2012). Mapping Forest and Peat Fires Using Hyperspectral Airborne Remote-Sensing Data. *Izvestiya, Atmospheric and Oceaninc Physics*, 941 - 948.

XU Dong, D. L.-m.-f. (2005). Forest fire risk zone mapping from satellite images and GIS for Baihe Forestry Bureau, Jilin, China. *Journal of Forestry Research*, 169 - 174.

Zhenhua Lv, Y. H. (2010). Parallel K-Means Clustering of Remote Sensing Images Based on MapReduce. *WISM*, 162 - 170.

ANEXOS

Anexo 1 Instalaciones previas

1. Instalación de librerías

Dependiendo del equipo se instalaron los diferentes SDK, en el caso del primer equipo dado que su tarjeta de video era integrada en la tarjeta madre, requería la instalación del SDK para Intel y para poder activar la compatibilidad con procesamiento en paralelo se instaló el Intel Parallel Studio XE, este último trae su asistente de instalación por lo que no requiere pasos significativos. En el caso de AMD, requiere la instalación del controlador de la tarjeta de video disponible en la página de AMD y su SDK compatible con AMD, el controlador y su SDK ya trae la compatibilidad con procesamiento en paralelo por lo que no es necesario otra instalación.

2. Instalación en Intel

Después de ya haber descargado los programas se procede con su descompresión, esta puede ser desde la línea de comandos con

```
$tar -xvf parallel_studio_xe_2016.tgz
$tar -xvf intel_sdk_for_opencl_2016_ubuntu_6.0.0.1049_x64.tgz
```

En este caso no tiene importancia cuál de los dos se instale primero, para la instalación solo hay que entrar en la carpeta que se acaba de descomprimir y en ella ejecutar como súper usuario el archivo “install_GUI.sh”, con ello se abrirá un instalador que nos guiará por la instalación.

```
$ cd parallel_studio_xe_2016
$sudo ./install_GUI.sh
```

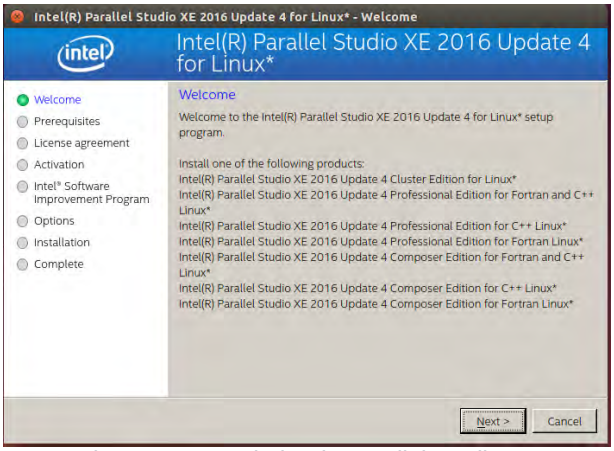

Figura 12.- Instalador de Parallel Studio XE

Solo hay que seguir el instalador, para concluir la instalación y en caso de que nos falten librerías para poder concluir la instalación este nos avisara e indicara que librerías faltan para una instalación exitosa. Lo mismo sucede con el SDK.

```
$ cd ../intel_sdk_for_opencl_2016_ubuntu_6.0.0.1049_x64.tgz
$sudo ./install_GUI.sh
```

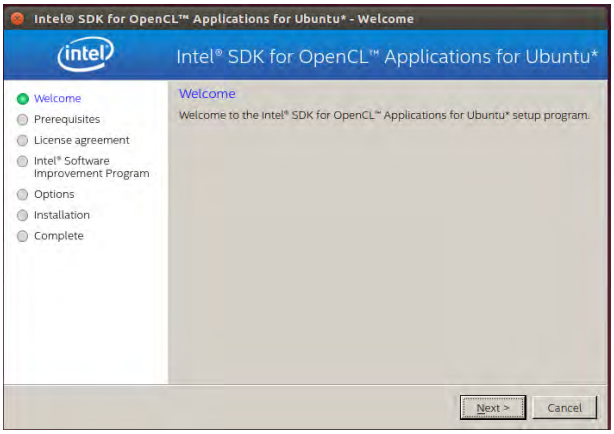

Figura 13.- Instalador de SDK Intel

Figura 14.- Licencia de SDK Intel

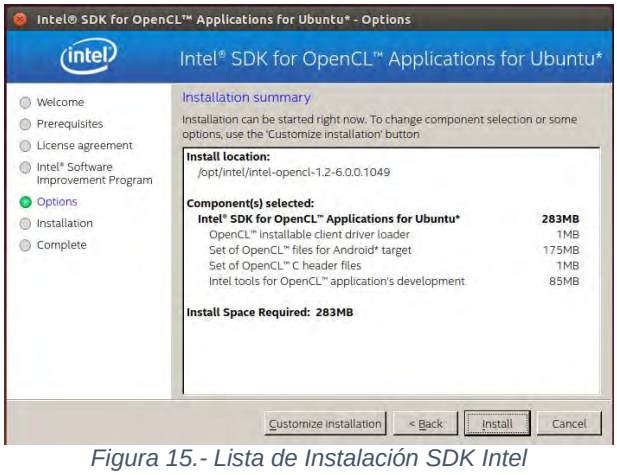

Figura 15.- Lista de Instalación SDK Intel

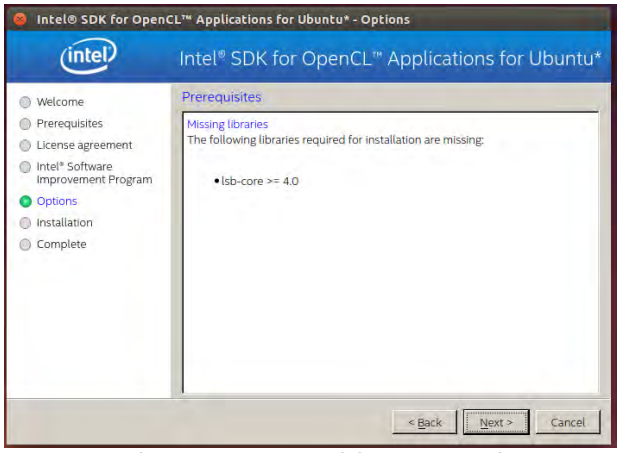

Figura 16.- Prerrequisitos SDK Intel

En este caso nos indica que nos falta una librería para su instalación correcta, la cual es "*lsb-core*", la instalación de la librería es como con cualquier otra con el comando "apt-get install" y de ahí se requiere que se vuelvan a comprobar los prerrequisitos del SDK.

$sudo apt-get install lsb-core

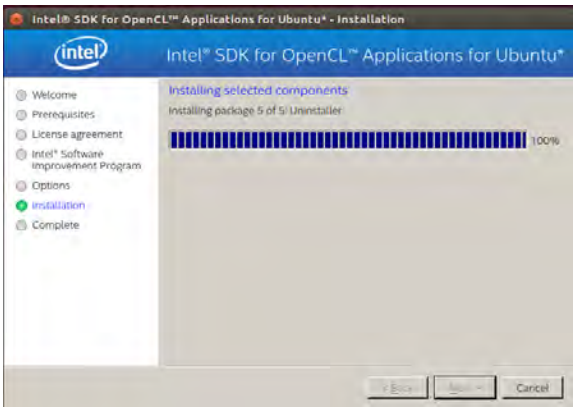

Figura 17.- Instalación de paquetes SDK Intel

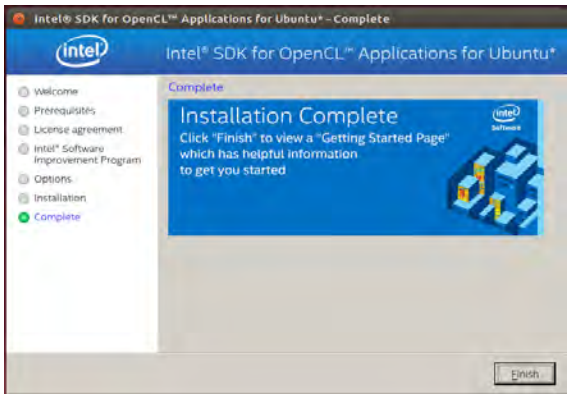

Figura 18.- Instalación Completa SDK Intel

3. Instalación en AMD

En el caso de AMD se requiere la instalación del controlador de la tarjeta de video, para la instalación se requieren descargar 4 archivos, el fglrx, el amdcccle, el core y el dev. La versión que nosotros utilizamos de los controladores es la 15.302 compatible con la tarjeta AMD Radeon R5 M335. Es importante mencionar que la instalación de los paquetes tiene un orden específico porque unos dependen de otros para su instalación correcta. Siendo ese orden el siguiente.

```
$sudo dpkg -i fglrx-core_15.302-0ubuntu1_amd64_ub_14.01.deb
$sudo dpkg -i fglrx_15.302-0ubuntu1_amd64_ub_14.01.deb
$sudo dpkg -i fglrx-dev_15.302-0ubuntu1_amd64_ub_14.01.deb
$sudo dpkg -i fglrx-amdcccle_15.302-0ubuntu1_amd64_ub_14.01.deb
```

Después de ya haber instalado el controlador será necesario reiniciar el equipo y por último para iniciar el controlador es necesario el comando

```
$sudo amdconfig -initial
```

Ahora para la instalación del SKD de AMD es necesario descomprimirlo y ejecutarlo, en el caso de AMD este solo trae un archivo de configuración a diferencia de Intel que contenía una carpeta con diferentes archivos para la instalación, se puede crear una carpeta para la instalación como se hizo en nuestro caso o solo correr el instalador, los comandos son los siguientes.

```
$ tar -xvf AMD-APP-SDKInstaller-v3.0.130.135-GA-linux64.tar.bz2
$sudo ./AMD-APP-SDK-v3.0.130.135-GA-linux64.sh
```

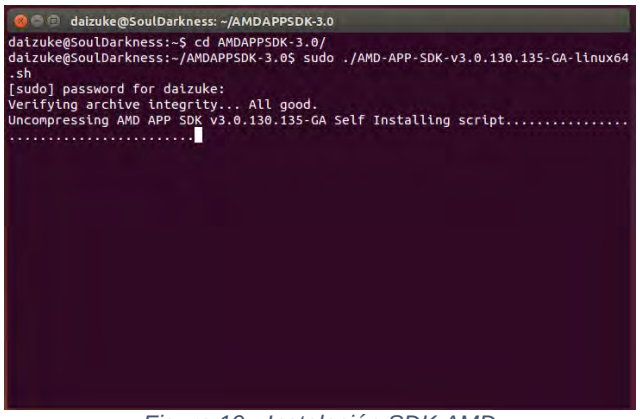

Figura 19.- Instalación SDK AMD

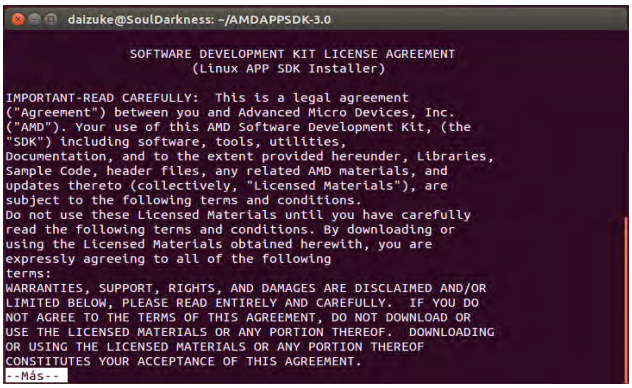

Figura 20.- Licencia SKD AMD

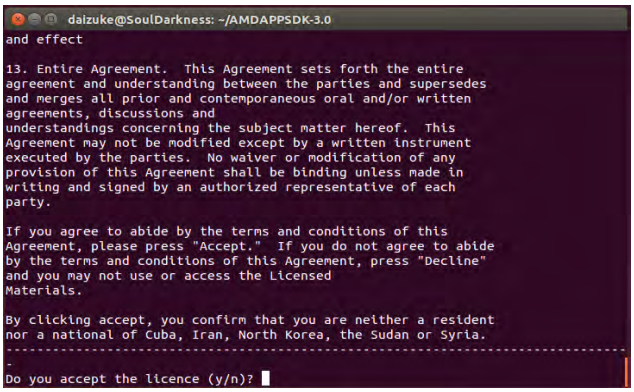

Figura 21.- Aceptación de Licencia SDK AMD

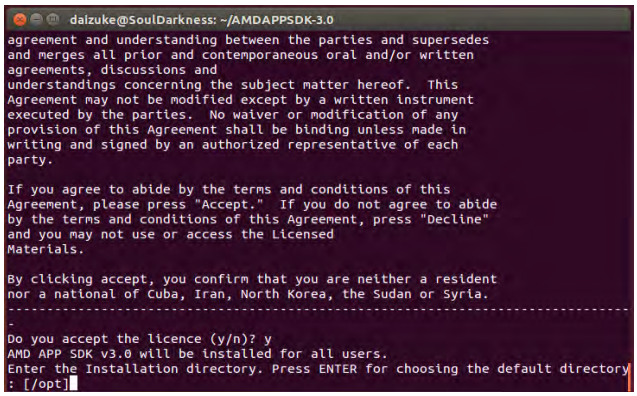

Figura 22.- Carpeta de instalación SDK AMD

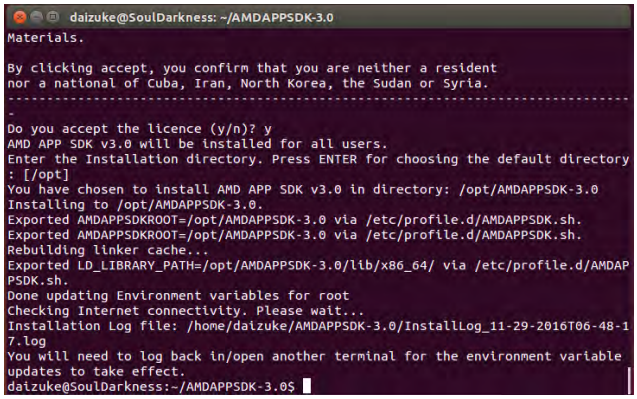

Figura 23.- Instalación completa SDK AMD

Para que podamos usar la instalación será necesario abrir una nueva terminal para que carguen las variables de entorno.

En algunos casos la instalación del SDK no agrega el “OpenCL.so” y “OpenCL.so.1” a la carpeta de las librerías “usr/lib” esta carpeta puede cambiar por “usr/libx86_64” o “usr/lib/x86_64” dependiendo de la arquitectura, por lo que es necesario copiar los archivos a la carpeta o crear una liga al archivo con los comandos “cp” o “ln”, esto puede suceder tanto en el caso de Intel como para AMD, esto no quiere decir que no los cree la instalación sino que solo los agrego a las librerías activas, los archivos los podrán encontrar en la carpeta que creo la instalación en “/opt”.

Anexo 2 Código del programa

CIAPI.cpp

```
#include "common.h"
#include "image.h"
#include "tiffio.h"
#include "tiff.h"

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstddef>
#include <cstdlib>
#include <cmath>
#include <cstring>

using namespace std;

int main(int argc, char** argv)
{
    const char * imagenRED = "LC80190472014318LGN00/LC80190472014318LGN00_B4.TIF";
    const char * imagenNIR = "LC80190472014318LGN00/LC80190472014318LGN00_B5.TIF";
    const char * imagenTIR1 = "LC80190472014318LGN00/LC80190472014318LGN00_B10.TIF";
    const char * imagenTIR2 = "LC80190472014318LGN00/LC80190472014318LGN00_B11.TIF";
    const char * imagen;

cl_context context = 0;
    cl_command_queue commandQueue = 0;
    cl_program program = 0;
    cl_device_id device = 0;
    cl_int typeDevice = atoi(argv[1]);
    cl_int numDevice = 0;
    cl_kernel kernel = 0;
    cl_kernel kernel2 = 0;
    cl_kernel kernel3 = 0;
    cl_kernel kernel4 = 0;
    cl_kernel kernel_temp = 0;
    cl_kernel kernel_TDVI = 0;
    cl_kernel kernel_IMG = 0;
    const unsigned int numberOfMemoryObjects = 24;
    cl_mem memoryObjects[numberOfMemoryObjects] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    cl_int errorNumber;
    size_t local_size, num_groups, global_size;
    // ON createContext, send 1 for CPU or 0 for GPU
    if (!createContext(&context, &numDevice, typeDevice))
    {
        cleanUpOpenCL(context,   commandQueue,   program,   kernel,   memoryObjects,
numberOfMemoryObjects);
```

```
        cerr << "Falló la creación del OpenCL context. " << __FILE__ << ":"<< __LINE__
<< endl;
        return 1;
    }
    for (int i = 0; i < 1; i++)
    {
       int j = 0; //Usa el primer dispositivo
       if (numDevice > 1 && i > 1) j = 1; //Usa el segundo dispositivo
       if (!createCommandQueue(context, &commandQueue, &device, j))
       {
           cleanUpOpenCL(context,  commandQueue,  program,  kernel,  memoryObjects,
numberOfMemoryObjects);
           cerr << "Falló la creación del OpenCL command queue. " << __FILE__ << ":"<<
__LINE__ << endl;
           return 1;
       }

if (!createProgram(context, device, "ClAPI.cl", &program))
       {
           cleanUpOpenCL(context,  commandQueue,  program,  kernel,  memoryObjects,
numberOfMemoryObjects);
           cerr  <<  "Falló  la  creación  del  programa  OpenCL."  <<  __FILE__  <<  ":"<<
__LINE__ << endl;
           return 1;
       }
/* Carga información de la imagen */
       cl_int ancho;
    cl_int alto;
       size_t pixelSize;
    float* imageDataR = NULL;
       float* imageDataG = NULL;
       float* imageDataTIR1 = NULL;
       float* imageDataTIR2 = NULL;
       float* imageData = NULL;
       for(int l = 0; l < 4; l++){
              if (l == 0)
              {
                     imagen = imagenRED;
              }
              if (l == 1)
              {
                     imagen = imagenNIR;
              }
              if (l == 2)
              {
                     imagen = imagenTIR1;
              }
              if (l == 3)
              {
                     imagen = imagenTIR2;
              }
              TIFF* tif = TIFFOpen(imagen, "r");
              if ((tif))
              {
                     short config;
                     unsigned short *raster;
                     raster = (unsigned short *)_TIFFmalloc(TIFFScanlineSize(tif));
                     TIFFGetField(tif, TIFFTAG_PLANARCONFIG, &config);
```

```
                        TIFFGetField(tif, TIFFTAG_IMAGEWIDTH, &ancho);
                        TIFFGetField(tif, TIFFTAG_IMAGELENGTH, &alto);
                        pixelSize = ancho * alto;
                        cerr << "alto " << alto << endl;
                        cerr << "ancho " << ancho << endl;
                        cerr << "config " << config << endl;
                        imageData = (float *) malloc(pixelSize * sizeof (float));
                        if (config == 1)
                        {
                                int n = 0;
                                alto = 5000;
                                ancho = 5000;
                                pixelSize = alto * ancho;
                                for(int row = 1400; row < 6400; row++)
                                {
                                        TIFFReadScanline(tif, raster, row);
                                        for(int i = 1400; i < 6400; i++)
                                        {
                                                *(imageData + n) = (float)(*(raster + i));
                                                n++;
                                        }
                                }
                                if(l == 1){
                                        imageDataG  =  (float*)  malloc(pixelSize  *  sizeof
(float));
                                        memcpy(imageDataG,    imageData,    pixelSize    *
sizeof(float));
                                }
                                if(l == 0){
                                        imageDataR  =  (float*)  malloc(pixelSize  *  sizeof
(float));
                                        memcpy(imageDataR,  imageData,  (ancho  *  alto)  *
sizeof(float));
                                }
                                if(l == 2){
                                        imageDataTIR1 = (float*) malloc(pixelSize * sizeof
(float));
                                        memcpy(imageDataTIR1, imageData, (ancho * alto) *
sizeof(float));
                                }
                                if(l == 3){
                                        imageDataTIR2 = (float*) malloc(pixelSize * sizeof
(float));
                                        memcpy(imageDataTIR2, imageData, (ancho * alto) *
sizeof(float));
                                }
                        }
                        else
                        {
                                cleanUpOpenCL(context,  commandQueue,  program,  kernel,
memoryObjects, numberOfMemoryObjects);
                        cerr << "Fallo la carga de la imagen. " << __FILE__ << ":"<<
__LINE__ << endl;
                        return 1;
                        }
                        _TIFFfree(raster);
                }
        }
```

```
/* Tamaño del Buffer */
	cerr << "Finalizacion de carga de las imagenes" << endl;
	size_t bufferSize = (ancho * alto) * sizeof(cl_uchar);
	size_t bufferFloat = (ancho * alto) * sizeof(cl_float);

/* Crea el buffer de entrada de la imagen y los buffer de salida de lo gradientes X y
Y */
	bool createMemoryObjectsSuccess = true;
	memoryObjects[0]	=	clCreateBuffer(context,	CL_MEM_READ_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber); //Imagen NIR
	memoryObjects[1]	=	clCreateBuffer(context,	CL_MEM_READ_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber); //Imagen RED
	createMemoryObjectsSuccess &= checkSuccess(errorNumber);
	memoryObjects[2]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferSize, NULL, &errorNumber); //Imagen NDVI
	memoryObjects[3]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//PromNIR
	memoryObjects[4]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//VarNIR
	memoryObjects[5]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//aNIR
	memoryObjects[6]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//PromRED
	memoryObjects[7]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//VarRED
	memoryObjects[8]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//aRED
	memoryObjects[9]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//paNIR
	memoryObjects[10]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//pbNIR
	memoryObjects[11]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//NIR
	memoryObjects[12]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//paRED
	memoryObjects[13]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//pbRED
	memoryObjects[14]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//RED
	memoryObjects[15]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//NDVI
	memoryObjects[16]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferSize, NULL, &errorNumber); // Final NIR
	memoryObjects[17]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferSize, NULL, &errorNumber); // Final RED
	memoryObjects[18]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//Pinned
	memoryObjects[19]	=	clCreateBuffer(context,	CL_MEM_READ_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//Imagen TIR1
	memoryObjects[20]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//Temperatura
	memoryObjects[21]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//TDVI
	memoryObjects[22]	=	clCreateBuffer(context,	CL_MEM_WRITE_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferSize, NULL, &errorNumber);//Imagen TDVI
	memoryObjects[23]	=	clCreateBuffer(context,	CL_MEM_READ_ONLY	|
CL_MEM_ALLOC_HOST_PTR, bufferFloat, NULL, &errorNumber);//Imagen TIR2
	createMemoryObjectsSuccess &= checkSuccess(errorNumber);
```

```
	if (!createMemoryObjectsSuccess)
	{
		cleanUpOpenCL(context, commandQueue, program, kernel, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo la creación de los OpenCL buffers. " << __FILE__ << ":"<<
__LINE__ << endl;
		return 1;
	}
/* Mapeo de la entrada de luminicencia del objeto de memoria al host desde el lado
del apuntador. */
	cl_float* NIR = (cl_float*)clEnqueueMapBuffer(commandQueue, memoryObjects[0],
CL_TRUE, CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL, &errorNumber);
	cl_float* RED = (cl_float*)clEnqueueMapBuffer(commandQueue, memoryObjects[1],
CL_TRUE, CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL, &errorNumber);
      cl_float* TIR1 = (cl_float*)clEnqueueMapBuffer(commandQueue, memoryObjects[19],
CL_TRUE, CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL, &errorNumber);
     cl_float* TIR2 = (cl_float*)clEnqueueMapBuffer(commandQueue, memoryObjects[23],
CL_TRUE, CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL, &errorNumber);
	if (!checkSuccess(errorNumber))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel,
memoryObjects, numberOfMemoryObjects);
		cerr << "Fallo el Mapeo de los objetos de memoria " << __FILE__
<< ":"<< __LINE__ <<	endl;
		return 1;
	}
	memcpy(NIR, imageDataG, bufferFloat);
	memcpy(RED, imageDataR, bufferFloat);
	memcpy(TIR1, imageDataTIR1, bufferFloat);
	memcpy(TIR2, imageDataTIR2, bufferFloat);
	delete [] imageData;
	delete [] imageDataG;
	delete [] imageDataR;
	delete [] imageDataTIR1;
	delete [] imageDataTIR2;

//Allocate and initialize output arrays
	float varianzaNIR;
	float varianzaRED;
	float maxi, mini;
	float *scalar_sum;
	cl_mem scalar_sum_buffer;
	cl_mem pinned_scalar_sum;
	cl_int err = CL_SUCCESS;

//Kernel MEAN
	for (int s = 0; s < 2; s ++)
	{
		maxi = 0, mini = 1;
		cerr << "Inicia el Kernel reduce_mean_variance1" << endl;
		cl_kernel kernelMean = clCreateKernel(program, "reduce_mean_variance1",
& errorNumber);
		if(!checkSuccess(errorNumber))
		{
			cleanUpOpenCL(context, commandQueue, program, kernelMean,
memoryObjects, numberOfMemoryObjects);
			cerr << "Fallo la creación del OpenCL Kernel Mean. " << __FILE__
<< ":" << __LINE__ << endl;
```

```
                return 1;
        }
//LOCAL_SIZE & GLOBAL_SIZE
        errorNumber     =     clGetContextInfo(context,     CL_CONTEXT_DEVICES,
sizeof(cl_device_id), &device, NULL);
        clGetKernelWorkGroupInfo(kernelMean, device, CL_KERNEL_WORK_GROUP_SIZE,
sizeof(size_t), &local_size, NULL);
        cout << "KERNEL_WORK_GROUP_SIZE " << local_size << endl;
        num_groups = pixelSize/local_size;
        global_size = num_groups * local_size;
        if(errorNumber < 0)
        {
                perror("No se pudo obtener la información del Dispositivo");
                cleanUpOpenCL(context,    commandQueue,    program,    kernelMean,
memoryObjects, numberOfMemoryObjects);
                return 1;
        }

// Create data buffer
        scalar_sum_buffer  =  clCreateBuffer(context,  CL_MEM_READ_WRITE,  2  *
num_groups * sizeof(float), NULL, &errorNumber);
        if(!createMemoryObjectsSuccess)
        {
                cleanUpOpenCL(context,    commandQueue,    program,    kernelMean,
memoryObjects, numberOfMemoryObjects);
                cerr << "Fallo la creación de los OpenCL buffers. " << __FILE__
<< ":" << __LINE__ << endl;
                return 1;
        }

//Argumentos del kernelMean
        if(  s  ==  0)  err  =  clSetKernelArg(kernelMean,  0,  sizeof(cl_mem),
&memoryObjects[0]);
        if(  s  ==  1)  err  =  clSetKernelArg(kernelMean,  0,  sizeof(cl_mem),
&memoryObjects[1]);
        err  =  clSetKernelArg(kernelMean,  1,  2  *  local_size  *  sizeof(float),
NULL);
        err = clSetKernelArg(kernelMean, 2, sizeof(cl_mem), &scalar_sum_buffer);
        if(err < 0)
        {
                perror("No se pudo crear los argumentos del kernel");
                exit(1);
        }

//Enqueue kernelMean
        err   =   clEnqueueNDRangeKernel(commandQueue,   kernelMean,   1,   NULL,
&global_size, &local_size, 0, NULL, NULL);
        if(err < 0)
        {
                cerr << "Error : " << err << endl;
                perror("No se pudo encolar el kernelMean");
                exit(1);
        }

//Finish processing the queue and get profilin information
        clFinish(commandQueue);
        cerr << "Termina el kernel reduce_mean_variance1" << endl;
```

```
//KernelMean2
		cerr << "Empieza el kernel reduce_mean_variance2" << endl;
		cl_kernel kernelMean2 = clCreateKernel(program, "reduce_mean_variance2",
&errorNumber);
		if (!checkSuccess(errorNumber))
		{
			cleanUpOpenCL(context,	commandQueue,	program,	kernelMean2,
memoryObjects, numberOfMemoryObjects);
			cerr << "Fallo la creación del OpenCL kernelMean2. " << __FILE__
<< ":" << __LINE__ << endl;
			return 1;
		}

//LOCAL_SIZE & GLOBAL_SIZE
		clGetKernelWorkGroupInfo(kernelMean2, device, CL_KERNEL_WORK_GROUP_SIZE,
sizeof(size_t), &local_size, NULL);
		cout << "KERNEL_WORK_GROUP_SIZE " << local_size << endl;
		if(errorNumber < 0)
		{
			perror("No se puede obtener la información del dispositivo");
			cleanUpOpenCL(context,	commandQueue,	program,	kernelMean2,
memoryObjects, numberOfMemoryObjects);
			return 1;
		}
		size_t num_groups2 = num_groups/local_size;
		size_t dif_groups = num_groups2 * local_size;
		cl_mem scalar_result_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE,
2 * num_groups2 * sizeof(float), NULL, &errorNumber);
		createMemoryObjectsSuccess &= checkSuccess(errorNumber);
//Argumentos del kernelMean2
		err	=	clSetKernelArg(kernelMean2,	0,	sizeof(cl_mem),
&scalar_sum_buffer);
		err |= clSetKernelArg(kernelMean2, 1, 2 * local_size * sizeof(float),
NULL);
		err	|=	clSetKernelArg(kernelMean2,	2,	sizeof(cl_mem),
&scalar_result_buffer);
		err |= clSetKernelArg(kernelMean2, 3, sizeof(cl_int), &num_groups);
		if(err < 0)
		{
			perror("No se pudieron crear los argumentos del KernelMean2");
			exit(1);
		}

//Encolado del KernelMean2
		err	=	clEnqueueNDRangeKernel(commandQueue,	kernelMean2,	1,	NULL,
&dif_groups, &local_size, 0, NULL, NULL);
		if(err < 0)
		{
			perror("No se pudo encolar el kernelMean2");
			exit(1);
		}

//Finish processing the queue and get profiling information
		clFinish(commandQueue);
		cerr << "Termina el kernel reduce_mean_variance2" << endl;

//Creación de la información del buffer
```

```
		pinned_scalar_sum = clCreateBuffer(context, CL_MEM_READ_WRITE|CL_MEM_ALLOC_HOST_PTR, 2 * num_groups2 * sizeof(float), NULL, &errorNumber);
		createMemoryObjectsSuccess &= checkSuccess(errorNumber);
		scalar_sum = (float *)clEnqueueMapBuffer(commandQueue, pinned_scalar_sum, CL_FALSE, CL_MAP_READ, 0, 2 * num_groups2 * sizeof(float), 0, NULL, NULL, &errorNumber);

//Leer el resultado
		err = clEnqueueReadBuffer(commandQueue, scalar_result_buffer, CL_TRUE, 0, 2 * num_groups2 * sizeof(float), scalar_sum, 0, NULL, NULL);
		if(err < 0)
		{
			perror("No se puede leer el buffer");
			exit(1);
		}
		maxi = scalar_sum[0];
		mini = scalar_sum[num_groups2];
		float delta, nb = local_size * local_size, na = nb;
		for(j = 1; j < num_groups2; j++)
		{
			delta = scalar_sum[j] - maxi;
			maxi += (delta*nb)/(na+nb); // reduce el promedio
			mini += (scalar_sum[j + num_groups2] + (delta * delta) * (na*nb)/(na+nb)); // ((na*na)/(na+na))); // reduce la varianza
			na += nb;
		}
		mini = mini/na;
		cout << "Promedio " << maxi << endl;
		cout << "Varianza " << mini << endl;
		clEnqueueUnmapMemObject(commandQueue, pinned_scalar_sum, scalar_sum, 0, NULL, NULL);
		if (!checkSuccess(clFinish(commandQueue)))
		{
			cleanUpOpenCL(context, commandQueue, program, kernel, memoryObjects, numberOfMemoryObjects);
			cerr << "Falló la ejecución de la finaliación del kernel. " << __FILE__ << ":" << __LINE__ << endl;
			return 1;
		}
		clReleaseMemObject(scalar_sum_buffer);
		clReleaseMemObject(scalar_result_buffer);
		clReleaseMemObject(pinned_scalar_sum);
		if(s == 0) varianzaNIR = mini;
		if(s == 1) varianzaRED = mini;
	}
/******************************** KERNEL LLSURE ********************************/
	cerr << "Inicio del Kernel LLSURE_inicio" << endl;
	kernel = clCreateKernel(program, "LLSURE_inicio", &errorNumber);
	if (!checkSuccess(errorNumber))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel, memoryObjects, numberOfMemoryObjects);
		cerr << "Fallo la creación del OpenCL kernel. " << __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
```

```
/* Argumentos del kernel */
	bool setKernelArgumentsSuccess = true;
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	0,
sizeof(cl_mem), &memoryObjects[0]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	1,
sizeof(cl_int), &ancho));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	2,
sizeof(cl_int), &alto));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	3,
sizeof(cl_mem), &memoryObjects[3]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	4,
sizeof(cl_mem), &memoryObjects[4]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	5,
sizeof(cl_mem), &memoryObjects[5]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	6,
sizeof(cl_mem), &memoryObjects[1]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	7,
sizeof(cl_mem), &memoryObjects[6]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	8,
sizeof(cl_mem), &memoryObjects[7]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	9,
sizeof(cl_mem), &memoryObjects[8]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	10,
sizeof(cl_float), &varianzaNIR));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel,	11,
sizeof(cl_float), &varianzaRED));
	if (!setKernelArgumentsSuccess)
	{
		cleanUpOpenCL(context,	commandQueue,	program,	kernel,	memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la colocación de los argumentos del OpenCL kernel. " <<
__FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}

/* Creacion del evento */
	cl_event event = 0;

/* Encolado del kernel */
	size_t	globalWorksize[2]	=	{static_cast<size_t>(ancho),
static_cast<size_t>(alto)};
	if	(!checkSuccess(clEnqueueNDRangeKernel(commandQueue,	kernel,	2,	NULL,
globalWorksize, NULL, 0, NULL, &event)))
	{
		cleanUpOpenCL(context,	commandQueue,	program,	kernel,	memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo el encolado del kernel. " << __FILE__ << ":"<< __LINE__
<< endl;
		return 1;
	}
	if (!checkSuccess(clFinish(commandQueue)))
	{
		cleanUpOpenCL(context,	commandQueue,	program,	kernel,	memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la ejecución de la finalización del kernel. " << __FILE__
<< ":"<< __LINE__ << endl;
		return 1;
	}
```

```
	cerr << "Termina el kernel LLSURE" << endl;

/************************* Se declara el segundo kernel *************************/
	cerr << "Inicia el kernel LLSURE" << endl;
	kernel2 = clCreateKernel(program, "LLSURE", &errorNumber);
	if (!checkSuccess(errorNumber))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo la creación del OpenCL kernel2. " << __FILE__ << ":"<<
__LINE__ << endl;
		return 1;
	}
/* Argumentos del kernel */
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	0,
sizeof(cl_mem), &memoryObjects[0]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	1,
sizeof(cl_int), &ancho));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	2,
sizeof(cl_int), &alto));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	3,
sizeof(cl_mem), &memoryObjects[3]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	4,
sizeof(cl_mem), &memoryObjects[4]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	5,
sizeof(cl_mem), &memoryObjects[5]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	6,
sizeof(cl_mem), &memoryObjects[9]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	7,
sizeof(cl_mem), &memoryObjects[10]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel2,	8,
sizeof(cl_mem), &memoryObjects[11]));
	if (!setKernelArgumentsSuccess)
	{
		cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la colocación de los argumentos del OpenCL kernel2. " <<
__FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
/* Creacion del evento */
	cl_event event2 = 0;

/* Encolado del kernel */
	if  (!checkSuccess(clEnqueueNDRangeKernel(commandQueue,  kernel2,  2,  NULL,
globalWorksize, NULL, 1, &event, &event2)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo el encolado del kernel2. " << __FILE__ << ":"<< __LINE__
<< endl;
		return 1;
	}
	if (!checkSuccess(clFinish(commandQueue)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
```

```
		cerr << "Falló la ejecución de la finalización del kernel. " << __FILE__
<< ":"<< __LINE__ << endl;
		return 1;
	}

/* Finalización del tipo evento objeto */
	printProfilingInfo(event);
	if (!checkSuccess(clReleaseEvent(event)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
			return 1;
	}

	printProfilingInfo(event2);
	if (!checkSuccess(clReleaseEvent(event2)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
			return 1;
	}
	cerr << "Termina el kernel Final" << endl;

/************************** Se declara tercer kernel ****************************/
	cerr << "Inicia el kernel LLSURE_final" << endl;
	kernel4 = clCreateKernel(program, "LLSURE_final", &errorNumber);
	if (!checkSuccess(errorNumber))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel4, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo la creación del OpenCL kernel4. " << __FILE__ << ":"<<
__LINE__ << endl;
		return 1;
	}

/* Argumentos del kernel */
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	0,
sizeof(cl_mem), &memoryObjects[1]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	1,
sizeof(cl_int), &ancho));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	2,
sizeof(cl_int), &alto));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	3,
sizeof(cl_mem), &memoryObjects[6]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	4,
sizeof(cl_mem), &memoryObjects[7]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	5,
sizeof(cl_mem), &memoryObjects[8]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	6,
sizeof(cl_mem), &memoryObjects[12]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	7,
sizeof(cl_mem), &memoryObjects[13]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel4,	8,
sizeof(cl_mem), &memoryObjects[14]));
```

```
	if (!setKernelArgumentsSuccess)
	{
		cleanUpOpenCL(context, commandQueue, program, kernel4, memoryObjects, numberOfMemoryObjects);
		cerr << "Falló la colocación de los argumentos del OpenCL kernel4. " << __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}

/* Creacion del evento */
	cl_event event4 = 0;

/* Encolado del kernel */
	if (!checkSuccess(clEnqueueNDRangeKernel(commandQueue, kernel4, 2, NULL, globalWorksize, NULL, 0, NULL, &event4)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel4, memoryObjects, numberOfMemoryObjects);
		cerr << "Fallo el encolado del kernel4. " << __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
	if (!checkSuccess(clFinish(commandQueue)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel4, memoryObjects, numberOfMemoryObjects);
		cerr << "Falló la ejecución de la finalización del kernel4. " << __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
	cerr << "Termina el kernel Final2" << endl;

/****************************** Max - Min Red Nir ******************************/
	bool mapMemoryObjectsSuccess = true;
	cl_float*	redfloat	=	(cl_float*)clEnqueueMapBuffer(commandQueue, memoryObjects[14], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL, &errorNumber);
	mapMemoryObjectsSuccess &= checkSuccess(errorNumber);
	cl_float*	nirfloat	=	(cl_float*)clEnqueueMapBuffer(commandQueue, memoryObjects[11], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL, &errorNumber);
	mapMemoryObjectsSuccess &= checkSuccess(errorNumber);
	if (!mapMemoryObjectsSuccess)
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_temp, memoryObjects, numberOfMemoryObjects);
		cerr << "Fallo el mapeo de los objetos de memoria " << __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
	float maxred = 0;
	float minred = INFINITY;
	float maxnir = 0;
	float minnir = INFINITY;
	for(int h = 0; h < ancho * alto; h++)
	{
		maxred = (maxred > redfloat[h]) ? maxred : redfloat[h];
		minred = (minred < redfloat[h]) ? minred : redfloat[h];
```

```
                maxnir = (maxnir > nirfloat[h]) ? maxnir : nirfloat[h];
                minnir = (minnir < nirfloat[h]) ? minnir : nirfloat[h];
        }
        cerr << "Maximo Red " << maxred << endl;
        cerr << "Minimo Red " << minred << endl;
        cerr << "Maximo Nir " << maxnir << endl;
        cerr << "Minimo Nir " << minnir << endl;
//Finish processing the queue and get profiling information
        clFinish(commandQueue);
        printProfilingInfo(event4);
        if (!checkSuccess(clReleaseEvent(event4)))
        {
                cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
                cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
                        return 1;
        }

/************************* Se declara el tercer kernel *************************/
        cerr << "Inicia el kernel NDVI" << endl;
        kernel3 = clCreateKernel(program, "NDVI", &errorNumber);
        if (!checkSuccess(errorNumber))
        {
                cleanUpOpenCL(context, commandQueue, program, kernel3, memoryObjects,
numberOfMemoryObjects);
                cerr << "Fallo la creación del OpenCL kernel3. " << __FILE__ << ":"<<
__LINE__ << endl;
                return 1;
        }
/* Argumentos del kernel */
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    0,
sizeof(cl_int), &ancho));
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    1,
sizeof(cl_int), &alto));
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    2,
sizeof(cl_mem), &memoryObjects[11]));
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    3,
sizeof(cl_mem), &memoryObjects[0]));
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    4,
sizeof(cl_mem), &memoryObjects[1]));
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    5,
sizeof(cl_mem), &memoryObjects[14]));
        setKernelArgumentsSuccess    &=    checkSuccess(clSetKernelArg(kernel3,    6,
sizeof(cl_mem), &memoryObjects[15]));
        if (!setKernelArgumentsSuccess)
        {
                cleanUpOpenCL(context, commandQueue, program, kernel3, memoryObjects,
numberOfMemoryObjects);
                cerr << "Falló la colocación de los argumentos del OpenCL kernel3. " <<
__FILE__ << ":"<< __LINE__ << endl;
                return 1;
        }
/* Creacion del evento */
        cl_event event3 = 0;

/* Encolado del kernel */
```

```
	if (!checkSuccess(clEnqueueNDRangeKernel(commandQueue, kernel3, 2, NULL,
globalWorksize, NULL, 0, NULL, &event3)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel3, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo el encolado del kernel3. " << __FILE__ << ":"<< __LINE__
<< endl;
		return 1;
	}
	if (!checkSuccess(clFinish(commandQueue)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel3, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la ejecución de la finalización del kernel. " << __FILE__
<< ":"<< __LINE__ << endl;
		return 1;
	}
	cerr << "Termina el kernel NDVI" << endl;

//Finish processing the queue and get profiling information
	clFinish(commandQueue);
	printProfilingInfo(event3);
	if (!checkSuccess(clReleaseEvent(event3)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel2, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
			return 1;
	}
	cerr << "Termina el Kernel normaliza" << endl;

/****************************** Kernel TEMPERATURA ******************************/
	cerr << "Inicia Kernel Temperatura" << endl;
	kernel_temp = clCreateKernel(program, "Temperatura", &errorNumber);
	if (!checkSuccess(errorNumber))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_temp, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo la creación del OpenCL kernel de Temperatura. " << __FILE__
<< ":"<< __LINE__ << endl;
		return 1;
	}
/* Argumentos del kernel */
	setKernelArgumentsSuccess &= checkSuccess(clSetKernelArg(kernel_temp, 0,
sizeof(cl_mem), &memoryObjects[19]));
	setKernelArgumentsSuccess &= checkSuccess(clSetKernelArg(kernel_temp, 1,
sizeof(cl_int), &ancho));
	setKernelArgumentsSuccess &= checkSuccess(clSetKernelArg(kernel_temp, 2,
sizeof(cl_int), &alto));
	setKernelArgumentsSuccess &= checkSuccess(clSetKernelArg(kernel_temp, 3,
sizeof(cl_mem), &memoryObjects[20]));
	setKernelArgumentsSuccess &= checkSuccess(clSetKernelArg(kernel_temp, 4,
sizeof(cl_mem), &memoryObjects[23]));
	if (!setKernelArgumentsSuccess)
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_temp, memoryObjects,
numberOfMemoryObjects);
```

```
            cerr << "Falló la colocación de los argumentos del OpenCL kernel de
Temperatura. " << __FILE__ << ":"<< __LINE__ << endl;
            return 1;
      }
/* Creacion del evento */
      cl_event event_temp = 0;

/* Encolado del kernel */
      if (!checkSuccess(clEnqueueNDRangeKernel(commandQueue, kernel_temp, 2, NULL,
globalWorksize, NULL, 0, NULL, &event_temp)))
      {
            cleanUpOpenCL(context, commandQueue, program, kernel_temp, memoryObjects,
numberOfMemoryObjects);
            cerr << "Fallo el encolado del kernel de Temperatura. " << __FILE__ <<
":"<< __LINE__ << endl;
            return 1;
      }
      if (!checkSuccess(clFinish(commandQueue)))
      {
            cleanUpOpenCL(context, commandQueue, program, kernel_temp, memoryObjects,
numberOfMemoryObjects);
            cerr << "Falló la ejecución de la finalización del kernel de Temperatura.
" << __FILE__ << ":"<< __LINE__ << endl;
            return 1;
      }

/* Finalización del evento tipo objeto */
      printProfilingInfo(event_temp);
      if (!checkSuccess(clReleaseEvent(event_temp)))
      {
            cleanUpOpenCL(context, commandQueue, program, kernel_temp, memoryObjects,
numberOfMemoryObjects);
            cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
                  return 1;
      }
      cerr << "Termina Kernel Temperatura" << endl;

/***************************** Normalizacion NDVI *****************************/
      bool mapMemoryObjectsSuccess0 = true;
      cl_float*      NDVI1      =      (cl_float*)clEnqueueMapBuffer(commandQueue,
memoryObjects[15], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL,
&errorNumber);
      mapMemoryObjectsSuccess0 &= checkSuccess(errorNumber);
      cl_float*    temperatura    =    (cl_float*)clEnqueueMapBuffer(commandQueue,
memoryObjects[20], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL,
&errorNumber);
      mapMemoryObjectsSuccess0 &= checkSuccess(errorNumber);
      if (!mapMemoryObjectsSuccess0)
      {
                  cleanUpOpenCL(context,   commandQueue,   program,   kernel_temp,
memoryObjects, numberOfMemoryObjects);
                  cerr << "Fallo el mapeo de los objetos de memoria " << __FILE__
<< ":"<< __LINE__ << endl;
                  return 1;
      }
      float maxndvi = 0;
      float minndvi = INFINITY;
```

```
	for(int h = 0; h < ancho * alto; h++)
	{
		maxndvi = (NDVI1[h] > maxndvi) ? NDVI1[h] : maxndvi;
		minndvi = (NDVI1[h] < minndvi) ? NDVI1[h] : minndvi;
	}
	cerr << "Maximo NDVI " << maxndvi << endl;
	cerr << "Minimo NDVI " << minndvi << endl;
	cerr << "Inicia calculo de a y b" << endl;
	float histograma[256] = {0} ;
	float temp_min = INFINITY;
	float temp_max = 0;
	float histNDVI[256];
	float histNDVI2[256];
	for(int f = 0; f < 256; f++)
	{
		histNDVI[f] = f;
		histograma[f] = 0;
		histNDVI2[f] = 0;
	}
	for(int g = 0; g < ancho * alto; g++)
	{
		for(int f = 1; f < 256; f++)
		{
			if (floor ((255 * ((NDVI1[g] - minndvi)/ (maxndvi - minndvi))) +
0.5) == histNDVI[f])
			{
				histograma[f] = (histograma[f] > temperatura[g]) ?
histograma[f] : temperatura[g];
				histNDVI2[f] = (histNDVI2[f] > NDVI1[g]) ? histNDVI2[f] :
NDVI1[g];
				break;
			}
		}
		if (temperatura[g] != 0)
		temp_min = (temp_min < temperatura[g]) ? temp_min : temperatura[g];
		temp_max = (temp_max > temperatura[g]) ? temp_max : temperatura[g];
	}
	int n = 0;
	for(int f = 1; f < 256; f++)
	{
		if (histograma[f] != 0)
		{
			cerr << "Temperatura Maxima en la Posicion " << histNDVI[f] <<
" : " << histograma[f] << endl;
			n++;
		}
		else histNDVI[f] = 0;
	}
	cerr << "Temperatura Minimia Kelvin " << temp_min << endl;
	cerr << "Temperatura Maxima Kelvin " << temp_max << endl;
	cerr << "Numero de elementos " << n << endl;
	float a = 0;
	float b = 0;
	float x = 0;
	float y = 0;
	float x2 = 0;
	float xy = 0;
	for (int k = 1; k < 256; k++)
```

```
	{
		x += histNDVI2[k];
		y += histograma[k];
		x2 += histNDVI2[k] * histNDVI2[k];
		xy += histNDVI2[k] * histograma[k];
	}
	cerr << "X " << x << " Y " << y << " X2 " << x2 << " XY " << xy << endl;
	b = ((n * xy) - (x * y))/((n * x2) - (x * x));
	a = (y - (b * x))/n;
	cerr << "El valor de a es: " << a << endl;
	cerr << "El valor de b es: " << b << endl;
	bool unmapMemoryObjectsSuccess = true;
      unmapMemoryObjectsSuccess &= checkSuccess(clEnqueueUnmapMemObject(commandQueue,
memoryObjects[0], NIR, 0, NULL, NULL));
      unmapMemoryObjectsSuccess &= checkSuccess(clEnqueueUnmapMemObject(commandQueue,
memoryObjects[1], RED, 0, NULL, NULL));
      unmapMemoryObjectsSuccess &= checkSuccess(clEnqueueUnmapMemObject(commandQueue,
memoryObjects[15], NDVI1, 0, NULL, NULL));
      unmapMemoryObjectsSuccess &= checkSuccess(clEnqueueUnmapMemObject(commandQueue,
memoryObjects[20], temperatura, 0, NULL, NULL));
	 if (!unmapMemoryObjectsSuccess)
	 {
			cleanUpOpenCL(context,   commandQueue,   program,   kernel_temp,
memoryObjects, numberOfMemoryObjects);
			cerr << "Fallo el desmapeo de los objetos de memoria " << __FILE__
<< ":"<< __LINE__ << endl;
			return 1;
	}
	cerr << "Termina calculo de a y b" << endl;

/******************************** Kernel TDVI *********************************/
	cerr << "Inicia Kernel TDVI" << endl;
	kernel_TDVI = clCreateKernel(program, "TDVI", &errorNumber);
	if (!checkSuccess(errorNumber))
	{
	      cleanUpOpenCL(context, commandQueue, program, kernel_TDVI, memoryObjects,
numberOfMemoryObjects);
	       cerr << "Fallo la creación del OpenCL kernel TDVI. " << __FILE__ << ":"<<
__LINE__ << endl;
	       return 1;
	}

/* Argumentos del kernel */
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   0,
sizeof(cl_mem), &memoryObjects[20]));
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   1,
sizeof(cl_mem), &memoryObjects[15]));
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   2,
sizeof(cl_mem), &memoryObjects[21]));
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   3,
sizeof(cl_int), &ancho));
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   4,
sizeof(cl_float), &a));
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   5,
sizeof(cl_float), &temp_min));
	setKernelArgumentsSuccess   &=   checkSuccess(clSetKernelArg(kernel_TDVI,   6,
sizeof(cl_float), &b));
	if (!setKernelArgumentsSuccess)
```

```
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_TDVI, memoryObjects,
numberOfMemoryObjects);
		 cerr << "Falló la colocación de los argumentos del OpenCL kernel TDVI.
" << __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}

/* Creacion del evento */
	cl_event event_TDVI = 0;

/* Encolado del kernel */
	if (!checkSuccess(clEnqueueNDRangeKernel(commandQueue, kernel_TDVI, 2, NULL,
globalWorksize, NULL, 0, NULL, &event_TDVI)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_TDVI, memoryObjects,
numberOfMemoryObjects);
		 cerr << "Fallo el encolado del kernel TDVI. " << __FILE__ << ":"<<
__LINE__ << endl;
		return 1;
	}
	if (!checkSuccess(clFinish(commandQueue)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_TDVI, memoryObjects,
numberOfMemoryObjects);
		 cerr << "Falló la ejecución de la finalización del kernel TDVI. " <<
__FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}

/* Finalización del evento tipo objeto */
	printProfilingInfo(event_TDVI);
	if (!checkSuccess(clReleaseEvent(event_TDVI)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_TDVI, memoryObjects,
numberOfMemoryObjects);
		 cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
			return 1;
	}
	cerr << "Termina Kernel TDVI" << endl;

	bool mapMemoryObjectsSuccess2 = true;
	cl_float*	TDVI1	=	(cl_float*)clEnqueueMapBuffer(commandQueue,
memoryObjects[21], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL,
&errorNumber);
	mapMemoryObjectsSuccess2 &= checkSuccess(errorNumber);
	cl_float*	NDVI0	=	(cl_float*)clEnqueueMapBuffer(commandQueue,
memoryObjects[15], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL,
&errorNumber);
	cl_float*	redfloat0	=	(cl_float*)clEnqueueMapBuffer(commandQueue,
memoryObjects[14], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL,
&errorNumber);
	cl_float*	nirfloat0	=	(cl_float*)clEnqueueMapBuffer(commandQueue,
memoryObjects[11], CL_TRUE, CL_MAP_READ|CL_MAP_WRITE, 0, bufferFloat, 0, NULL, NULL,
&errorNumber);
	mapMemoryObjectsSuccess2 &= checkSuccess(errorNumber);
	if (!mapMemoryObjectsSuccess0)
```

```
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_temp,
memoryObjects, numberOfMemoryObjects);
		cerr << "Fallo el mapeo de los objetos de memoria " << __FILE__
<< ":"<< __LINE__ << endl;
		return 1;
	}
	float maxtdvi = 0;
	float mintdvi = INFINITY;
	for(int h = 0; h < ancho * alto; h++)
	{
		maxtdvi = (TDVI1[h] > maxtdvi) ? TDVI1[h] : maxtdvi;
		mintdvi = (TDVI1[h] < mintdvi) ? TDVI1[h] : mintdvi;
	}
	for(int g = 0; g < ancho * alto; g++)
	{
		NDVI0[g] = floor ((255 * ((NDVI0[g] - minndvi)/ (maxndvi - minndvi))) +
0.5);
		TDVI1[g] = floor ((255 * ((TDVI1[g] - mintdvi)/ (maxtdvi - mintdvi))) +
0.5);
		redfloat0[g] = floor ((255 * ((redfloat[g] - minred)/ (maxred - minred)))
+ 0.5);
		nirfloat0[g] = floor ((255 * ((nirfloat[g] - minred)/ (maxnir - minnir)))
+ 0.5);
	}
	cerr << "Maximo TDVI " << maxtdvi << endl;
	cerr << "Minimo TDVI " << mintdvi << endl;
/****************************** Kernel Imagenes ********************************/
	cerr << "Inicia Kernel Imagenes" << endl;
	kernel_IMG = clCreateKernel(program, "Imagenes", &errorNumber);
	if (!checkSuccess(errorNumber))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_IMG, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo la creación del OpenCL kernel IMG. " << __FILE__ << ":"<<
__LINE__ << endl;
		return 1;
	}
/* Argumentos del kernel */
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	0,
sizeof(cl_mem), &memoryObjects[15]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	1,
sizeof(cl_mem), &memoryObjects[21]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	2,
sizeof(cl_mem), &memoryObjects[22]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	3,
sizeof(cl_mem), &memoryObjects[2]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	4,
sizeof(cl_int), &ancho));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	5,
sizeof(cl_float), &mintdvi));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	6,
sizeof(cl_float), &maxtdvi));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	7,
sizeof(cl_float), &minndvi));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	8,
sizeof(cl_float), &maxndvi));
```

```
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	9,
sizeof(cl_mem), &memoryObjects[14]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	10,
sizeof(cl_mem), &memoryObjects[11]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	11,
sizeof(cl_mem), &memoryObjects[17]));
	setKernelArgumentsSuccess	&=	checkSuccess(clSetKernelArg(kernel_IMG,	12,
sizeof(cl_mem), &memoryObjects[16]));
	if (!setKernelArgumentsSuccess)
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_IMG, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la colocación de los argumentos del OpenCL kernel IMG. "
<< __FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
/* Creacion del evento */
	cl_event event_IMG = 0;
/* Encolado del kernel */
	if (!checkSuccess(clEnqueueNDRangeKernel(commandQueue, kernel_IMG, 2, NULL,
globalWorksize, NULL, 0, NULL, &event_IMG)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_IMG, memoryObjects,
numberOfMemoryObjects);
		cerr << "Fallo el encolado del kernel IMG. " << __FILE__ << ":"<< __LINE__
<< endl;
		return 1;
	}
	if (!checkSuccess(clFinish(commandQueue)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_IMG, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la ejecución de la finalización del kernel IMG. " <<
__FILE__ << ":"<< __LINE__ << endl;
		return 1;
	}
/* Finalización del evento tipo objeto */
	printProfilingInfo(event_IMG);
	if (!checkSuccess(clReleaseEvent(event_IMG)))
	{
		cleanUpOpenCL(context, commandQueue, program, kernel_IMG, memoryObjects,
numberOfMemoryObjects);
		cerr << "Falló la liberación del evento objeto. " << __FILE__ << ":"<<
__LINE__ << endl;
			return 1;
	}
/********************** Creacion de Imagegenes Finales **********************/
	bool mapMemoryObjectsSuccess1 = true;
	cl_uchar* NDVI = (cl_uchar*)clEnqueueMapBuffer(commandQueue, memoryObjects[2],
CL_TRUE, CL_MAP_READ, 0, bufferSize, 0, NULL, NULL, &errorNumber);
	mapMemoryObjectsSuccess1 &= checkSuccess(errorNumber);
	cl_uchar*	FinalNIR	=	(cl_uchar*)clEnqueueMapBuffer(commandQueue,
memoryObjects[16], CL_TRUE, CL_MAP_READ, 0, bufferSize, 0, NULL, NULL, &errorNumber);
	mapMemoryObjectsSuccess1 &= checkSuccess(errorNumber);
	cl_uchar*	FinalRED	=	(cl_uchar*)clEnqueueMapBuffer(commandQueue,
memoryObjects[17], CL_TRUE, CL_MAP_READ, 0, bufferSize, 0, NULL, NULL, &errorNumber);
	mapMemoryObjectsSuccess1 &= checkSuccess(errorNumber);
```

```
      cl_uchar* TDVI = (cl_uchar*)clEnqueueMapBuffer(commandQueue, memoryObjects[22],
CL_TRUE, CL_MAP_READ, 0, bufferSize, 0, NULL, NULL, &errorNumber);
       mapMemoryObjectsSuccess1 &= checkSuccess(errorNumber);
       if (!mapMemoryObjectsSuccess1)
       {
                     cleanUpOpenCL(context,    commandQueue,    program,    kernel3,
memoryObjects, numberOfMemoryObjects);
                     cerr << "Fallo el mapeo de los objetos de memoria " << __FILE__
<< ":"<< __LINE__ << endl;
                     return 1;
       }
    /* Convert the two output luminance arrays to RGB and save them out to files.*/
       saveToBitmapL("FinalNIR.bmp", ancho, alto, FinalNIR);
       saveToBitmapL("FinalRED.bmp", ancho, alto, FinalRED);
       saveToBitmapL("NDVI.bmp", ancho, alto, NDVI);
       saveToBitmapL("TDVI.bmp", ancho, alto, TDVI);
    /* Liberacion de los objets OpenCL.*/
       bool unmapMemoryObjectsSuccess1 = true;
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[2], NDVI, 0, NULL,
NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[16], FinalNIR, 0,
NULL, NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[17], FinalRED, 0,
NULL, NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[22], TDVI, 0, NULL,
NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[21], TDVI1, 0, NULL,
NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[15], NDVI0, 0, NULL,
NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[14], redfloat0, 0,
NULL, NULL));
       unmapMemoryObjectsSuccess1                                                    &=
checkSuccess(clEnqueueUnmapMemObject(commandQueue, memoryObjects[11], nirfloat0, 0,
NULL, NULL));
       if (!unmapMemoryObjectsSuccess1)
       {
                     cleanUpOpenCL(context,    commandQueue,    program,    kernel_IMG,
memoryObjects, numberOfMemoryObjects);
                     cerr << "Fallo el desmapeo de los objetos de memoria " << __FILE__
<< ":"<< __LINE__ << endl;
                     return 1;
       }
       cleanUpOpenCL(context,  commandQueue,  program,  kernel_IMG,  memoryObjects,
numberOfMemoryObjects);
       cerr << "Termina Kernel IMG" << endl;
       delete [] NDVI;
       delete [] FinalNIR;
       delete [] FinalRED;
       delete [] TDVI;
    }
```

```
    return 0;
}
```

Anexo 3 Código de los Kernel

CIAPI.cl

```
#define max(x,y) ((x)>(y)?(x):(y))

__kernel void reduce_max_min(__global float* buffer,
                         __local float* scratch,
                         __global float* result,
                         const int lengt)
{
	int global_index = get_global_id(0);
	float accumulator = 0;
	float accumulator2 = INFINITY;
	int global_size = get_global_size(0);
	int group_size = get_local_size(0);

//Loop sequentially over chunks of input vector
	while (global_index < lengt)
	{
		float element = buffer[global_index];
		accumulator = (accumulator > element) ? accumulator : element;
		accumulator2 = (accumulator2 < element) ? accumulator2 : element;
		global_index += global_size;
	}

//Perform parallel reduction
	int local_index = get_local_id(0);
	int local_index2 = local_index + group_size;
	scratch[local_index] = accumulator;
	scratch[local_index2] = accumulator2;
	barrier(CLK_LOCAL_MEM_FENCE);
	for(int offset = group_size / 2; offset > 0; offset >>= 1)
	{
		if(local_index < offset)
		{
			float other = scratch[local_index + offset];
			float mine = scratch[local_index];
			float otro = scratch[local_index2 + offset];
			float mini = scratch[local_index2];
			scratch[local_index] = (mine > other) ? mine : other;
			scratch[local_index2] = (mini < otro) ? mini : otro;
		}
		barrier(CLK_LOCAL_MEM_FENCE);
	}
	if (local_index == 0)
	{
		result[get_group_id(0)] = scratch[0];
		result[get_group_id(0) + get_num_groups(0)] = scratch[0 + group_size];
	}
}

__kernel void reduce_mean_variance1(__global float* buffer,
                                __local float* scratch,
                                __global float* result)
{
	int global_index = get_global_id(0);
```

```
	int group_size = get_local_size(0);

//Perform parallel reduction to compute mean
	int local_index = get_local_id(0);
	scratch[local_index] = buffer[global_index];
	barrier(CLK_LOCAL_MEM_FENCE);
	for(int offset = group_size/2; offset > 0; offset >>= 1)
	{
		if(local_index < offset)
		{
			scratch[local_index] += scratch[local_index + offset];
		}
		barrier(CLK_LOCAL_MEM_FENCE);
	}
	if(local_index == 0)
	{
		result[get_group_id(0)] = scratch[0] / group_size; //mean value
	}

//Perform parallel reduction to compute variance after mean
	barrier(CLK_LOCAL_MEM_FENCE);
	float other = buffer[global_index] - result[get_group_id(0)];
	scratch[local_index] = other * other;
	barrier(CLK_LOCAL_MEM_FENCE);
	for(int offset = group_size / 2; offset > 0; offset >>= 1)
	{
		if(local_index < offset)
		{
			scratch[local_index] += scratch[local_index + offset];
		}
		barrier(CLK_LOCAL_MEM_FENCE);
	}
	if(local_index == 0)
	{
		result[get_group_id(0) + get_num_groups(0)] = scratch[0];
	}
}

__kernel void reduce_mean_variance2(__global float* buffer,
                                    __local float* scratch,
                                    __global float* result,
                                    const int num_groups)
{
    //perform the reduction of mean and variance of groups
    //Chan et al.[4] note that the above "On-line" algorithm is a special case of an
algorithm that works for any partition of the sample X into sets X_A, X_B:
    //Chan, Tony F.; Golub, Gene H.; LeVeque, Randall J. (1979), "Updating Formulae
and a Pairwise Algorithm for Computing Sample Variances." (PDF), Technical Report
STAN-CS-79-773, Department of Computer Science, Stanford University.
	int global_index = get_global_id(0);
	int group_size = get_local_size(0);
	int local_index = get_local_id(0);
	int local_index2 = local_index + group_size;
	scratch[local_index] = buffer[global_index]; //mean_a
	scratch[local_index2] = buffer[global_index + num_groups]; //var_a
	float na = group_size;
	float nb = group_size;
	barrier(CLK_LOCAL_MEM_FENCE);
```

```
	for(int offset = group_size / 2; offset > 0; offset >>= 1)
	{
		if(local_index < offset)
		{
			float delta = scratch[local_index + offset] -
scratch[local_index];
			float var_b = scratch[local_index2 + offset];
			scratch[local_index] += (delta * nb) / (nb + na); //
(na/(na+na)));
			scratch[local_index2] += (var_b + (delta * delta) * (na * nb) /
(na + nb)); ///((na*na)/(na+na)));
			na += nb;
		}
		barrier(CLK_LOCAL_MEM_FENCE);
	}
	if(local_index == 0)
	{
		result[get_group_id(0)] = scratch[0];
		result[get_group_id(0) + get_num_groups(0)] = scratch[0 + group_size];
	}
}
__kernel void normaliza(const int ancho,
			const int alto,
			const float minimo,
			const float maximo,
			__global float *x)
{
	const int columna = get_global_id(0);
	const int fila = get_global_id(1);
	int offset = fila * ancho + columna;
	if (minimo < 0)
	{
		x[offset] = 255 * ((x[offset] - minimo) / maximo);
	}
	else
	{
		x[offset] = 255 * ((x[offset] + minimo) / maximo);
	}
}

__kernel void LLSURE_inicio(__global const float* restrict inputNIR,
				const int ancho,
				const int alto,
				__global float *promNIR,
				__global float *varianzaNIR,
				__global float *aNIR,
				__global const float* restrict inputRed,
				__global float *promRed,
				__global float *varianzaRed,
				__global float *aRed,
				const float varianza_globalNIR,
				const float varianza_globalRED)
{
     float epsilon = 0.0000001f;
     int radio = 3;
     const int columna = get_global_id(0);
     const int fila = get_global_id(1);
     int offset = fila * ancho + columna;
```

```
     radio = ((fila & columna) > radio) ? radio : 0;
     promNIR[offset] = 0;
     varianzaNIR[offset] = 0;
     aNIR[offset] = 0;
     promRed[offset] = 0;
     varianzaRed[offset] = 0;
     aRed[offset] = 0;
     for(int r = -radio; r <= radio; r++)
     {
       for(int c = -radio ; c <= radio ; c++)
       {
           int newOffset = offset + /*moviminto en columnas*/ (ancho * r) +
/*movimiento en filas*/ c;
           float Imagen = *(inputNIR + newOffset);
           float Imagen2 = *(inputRed + newOffset);
           promNIR[offset] += Imagen;
           promRed[offset] += Imagen2;
           varianzaNIR[offset] += Imagen * Imagen;
           varianzaRed[offset] += Imagen2 * Imagen2;
       }
     }
     promNIR[offset] = promNIR[offset] / ((2*radio + 1)*(2*radio + 1));
     promRed[offset] = promRed[offset] / ((2*radio + 1)*(2*radio + 1));
     varianzaNIR[offset] = varianzaNIR[offset] / ((2*radio + 1) * (2*radio + 1));
     varianzaRed[offset] = varianzaRed[offset] / ((2*radio + 1) * (2*radio + 1));
     varianzaNIR[offset] = varianzaNIR[offset] - (promNIR[offset] *
promNIR[offset]);
     varianzaRed[offset] = varianzaRed[offset] - (promRed[offset] *
promRed[offset]);
     aNIR[offset] = max(varianzaNIR[offset] - varianza_globalNIR, 0) /
(varianzaNIR[offset] + epsilon);
     aRed[offset] = max(varianzaRed[offset] - varianza_globalRED, 0) /
(varianzaRed[offset] + epsilon);
     promNIR[offset] = (1 - aNIR[offset]) * promNIR[offset];
     promRed[offset] = (1 - aRed[offset]) * promRed[offset];

/*Espera a que se terminen de calcular los procesos*/
     barrier(CLK_GLOBAL_MEM_FENCE);
}
__kernel void LLSURE(__global const float* restrict inputNIR,
                          const int ancho,
                          const int alto,
                          __global float *promNIR,
                          __global float *varianzaNIR,
                          __global float *aNIR,
                          __global float *paNIR,
                          __global float *pbNIR,
                          __global float *NIR)
{
     float epsilon = 0.0000001f;
     int radio = 3;
     const int columna = get_global_id(0);
     const int fila = get_global_id(1);
     int offset = fila * ancho + columna;
     float sum_NIR = 0;
     radio = ((fila & columna) > radio) ? radio : 0;
     paNIR[offset] = 0;
     pbNIR[offset] = 0;
```

```
     for(int q = -radio; q <= radio; q++)
     {
       for(int k = -radio; k<= radio; k++)
       {
           int newoffset = offset + /*moviminto en columnas*/ (ancho * q) +
/*movimiento en filas*/ k;
           if(varianzaNIR[newoffset] > epsilon)
           {
              paNIR[offset] += (aNIR[newoffset] * varianzaNIR[newoffset]);
              pbNIR[offset] += (promNIR[newoffset] * varianzaNIR[newoffset]);
              sum_NIR += varianzaNIR[newoffset];
           }
       }
     }
     if (sum_NIR > epsilon)
     {
       sum_NIR = 1 / sum_NIR;
       paNIR[offset] = paNIR[offset] * sum_NIR;
       pbNIR[offset] = pbNIR[offset] * sum_NIR;
     }
     else
     {
       paNIR[offset] = 1;
       pbNIR[offset] = 0;
     }
     float Imagen = *(inputNIR + offset);
     NIR[offset] = (paNIR[offset] * Imagen + pbNIR[offset]);
/*Espera a que se terminen de calcular los procesos*/
     barrier(CLK_GLOBAL_MEM_FENCE);
}
__kernel void LLSURE_final(__global const float* restrict inputRed,
                           const int ancho,
                           const int alto,
                           __global float *promRed,
                           __global float *varianzaRed,
                           __global float *aRed,
                           __global float *paRed,
                           __global float *pbRed,
                           __global float *RED)
{
     float epsilon = 0.0000001f;
     int radio = 3;
     const int columna = get_global_id(0);
     const int fila = get_global_id(1);
     int offset = fila * ancho + columna;
     float sum_Red = 0;
     radio = ((fila & columna) > radio) ? radio : 0;
     paRed[offset] = 0;
     pbRed[offset] = 0;
     for(int q = -radio; q <= radio; q++)
     {
       for(int k = -radio; k<= radio; k++)
       {
           int newoffset = offset + /*moviminto en columnas*/ (ancho * q) +
/*movimiento en filas*/ k;
           if(varianzaRed[newoffset] > epsilon)
           {
              paRed[offset] += (aRed[newoffset] * varianzaRed[newoffset]);
```

```
            pbRed[offset] += (promRed[newoffset] * varianzaRed[newoffset]);
            sum_Red += varianzaRed[newoffset];
          }
      }
    }
    if (sum_Red > epsilon)
    {
      sum_Red = 1 / sum_Red;
      paRed[offset] = paRed[offset] * sum_Red;
      pbRed[offset] = pbRed[offset] * sum_Red;
    }
    else
    {
      paRed[offset] = 1;
      pbRed[offset] = 0;
    }
    float Imagen2 = *(inputRed + offset);
    RED[offset] = (paRed[offset] * Imagen2 + pbRed[offset]);
/*Espera a que se terminen de calcular los procesos*/
    barrier(CLK_GLOBAL_MEM_FENCE);
}
__kernel void NDVI(const int ancho,
                   const int alto,
                   __global float *NIR,
                   __global float *entradaNIR,
                   __global float *entradaRED,
                   __global float *RED,
                   __global float *NDVI)
{
      const int columna = get_global_id(0);
      const int fila = get_global_id(1);
      int offset = fila * ancho + columna;
      float diferencias = 0;
      float diferencias2 = 0;
      float trans = 0;
      float trans2 = 0;
      if(entradaNIR[offset] > NIR[offset])
            diferencias = entradaNIR[offset] - NIR[offset];
      else
            diferencias = NIR[offset] - entradaNIR[offset];
      NIR[offset] = NIR[offset] + 3 * diferencias;
      if(entradaRED[offset] > RED[offset])
            diferencias2 = entradaRED[offset] - RED[offset];
      else
            diferencias2 = RED[offset] - entradaRED[offset];
      RED[offset] = RED[offset] + 3 * diferencias2;
      trans = NIR[offset] - RED[offset];
      trans2= NIR[offset] + RED[offset];
      NDVI[offset] = (trans / trans2) ;
}
__kernel void Temperatura(__global const float* restrict inputTIR1,
                          const int ancho,
                          const int alto,
                    __global float *Temperatura,
                    __global const float* restrict inputTIR2)
{
    const int columna = get_global_id(0);
    const int fila = get_global_id(1);
```

```
      int offset = fila * ancho + columna;
      float k1_TIR1 = 774.89;
      float k2_TIR1 = 1321.08;
      float k1_TIR2 = 480.89;
      float k2_TIR2 = 1201.14;
      float Ap = 0.10000;
      float Mp = 0.0003342;
      float TIR1 = convert_float(inputTIR1[offset]);
      float TIR2 = convert_float(inputTIR2[offset]);
      if(convert_float(inputTIR1[offset]) != 0)
      {
        TIR1 = Ap + (Mp * TIR1);
        TIR1 = k2_TIR2 / log((k1_TIR2 / TIR1) + 1);
      }
      if(convert_float(inputTIR2[offset]) != 0)
      {
        TIR2 = Ap + (Mp * TIR2);
        TIR2 = k2_TIR1 / log((k1_TIR1 / TIR2) +1);
      }
      Temperatura[offset] = (TIR1 + TIR2) / 2;
}
__kernel void TDVI(__global float *Temperatura,
                   __global float *NDVI,
                   __global float *TVDI,
                   const int ancho,
                   const float a,
                   const float minim,
                   const float b)
{
       const int columna = get_global_id(0);
       const int fila = get_global_id(1);
       int offset = fila * ancho + columna;
       if(Temperatura[offset] != 0)
       TVDI[offset] = (Temperatura[offset] - minim)/(a + b * NDVI[offset] - minim);
       else
       TVDI[offset] = 0;
}
__kernel void Imagenes (__global float *NDVI,
                        __global float *TDVI,
                        __global uchar *ImageTDVI,
                        __global uchar *ImageNDVI,
                        const int ancho,
                        const float mintdvi,
                        const float maxtdvi,
                        const float minndvi,
                        const float maxndvi,
                        __global float *RED,
                        __global float *NIR,
                        __global uchar *ImageRED,
                        __global uchar *ImageNIR)
{
       const int columna = get_global_id(0);
       const int fila = get_global_id(1);
       int offset = fila * ancho + columna;
       TDVI[offset] = floor ((255 * ((TDVI[offset] - mintdvi)/ (maxtdvi - mintdvi)))
+ 0.5);
       *(ImageTDVI + offset) = convert_uchar(*(TDVI + offset));
       *(ImageNDVI + offset) = convert_uchar(*(NDVI + offset));
```

```
        *(ImageNIR + offset) = convert_uchar(*(NIR + offset));
        *(ImageRED + offset) = convert_uchar(*(RED + offset));
}
```