INSTITUTO TECNOLÓGICO DE LA PAZ
DIVISION DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
MAESTRÍA EN SISTEMAS COMPUTACIONALES

PROCESAMIENTO DE IMÁGENES DE SATÉLITE EN PARALELO, PARA CUANTIFICAR LA DEGRADACIÓN DE ECOSISTEMAS TERRESTRES

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRA EN SISTEMAS COMPUTACIONALES

PRESENTA

CLAUDIA MARTÍNEZ VÁZQUEZ

DIRECTORES DE TESIS

DR. SAÚL MARTÍNEZ DÍAZ

DR. RAÚL OCTAVIO MARTÍNEZ RINCÓN

LA PAZ, B. C. S., MÉXICO, NOVIEMBRE 2016

Dedicatoria

A mi familia…

Agradecimientos

A todas las personas e Instituciones que de una u otra manera me brindaron apoyo o facilidades para hacer esto posible.

A mi familia, mis directores de tesis, mis profesores, compañeros de generación, y desde luego al Centro de Investigaciones Biológicas del Noroeste (CIBNOR), a todos muchas gracias.

Resumen

Las imágenes de satélite son imágenes de la tierra o del espacio que son capturadas por diversos satélites que orbitan la tierra. Éstas imágenes usualmente contienen grandes cantidades de datos, lo que implica que su lectura o analisis requiera de elevados tiempos de procesamiento computacionalmente hablando. El objetivo principal de este trabajo fue desarrollar un ambiente de trabajo integrado por múltiples tecnologías (CUDA C, OpenCV y R) como una alterntiva al software convencional para el procesamiento de imágenes de satélite (sistemas de información geográfica), que permitiera disminuir el tiempo de procesamiento, mediante el uso de paralelismo en la unidad de procesamiento de gráficos (GPU, por sus siglas en ingés).

Es así, que derivado de la tarea de calcular el promedio histórico y la tendencia de la vegetación en el Estado de Baja California Sur en los últimos 15 años, utilizando imágenes de satélite con datos de índices de vegetación de diferencia normalizada (NDVI, por sus siglas en inglés) colectados por el sensor de imágenes de espectroradiometro de resolución moderada (MODIS, por sus siglas en ingés) de la NASA. Estas imágenes representan promedios de datos recolectados por el sensor durante 16 días. Para el periodo de febrero del 2000 a diciembre del 2014.

Finalmente, el reto fue dar solución al elevado tiempo de procesamiento de analizar 342 imágenes, cada una con dimensiones de 3,455 x 1,758 pixeles, es decir, un total de 6,073,890 de pixeles por imágen. La solución propuesta fue el desarrollo de un algoritmo en CUDA, el cual demostró que mediante el uso de cómputo en paralelo se logró reducir 69 veces el tiempo de procesamiento en el caso del *cálculo del promedio* y 155 veces para el *cálculo de la tendencia*. El tiempo promedio de ejecución utilizando procesamiento en paralelo (GPU) fue de 75 segundos para el cálculo del *promedio* y 114 segundos para el cálculo de la *tendencia*. Y cuando se utilizó el método tradicional (CPU) el proceso tuvo una duración de 5,271 segundos para el cálculo del *promedio* y 19,088 segundos para el cálculo de la *tendencia*.

Absract

Satellite images are images of the earth or space that are captured by different satellites orbiting the earth. These images usually contain large amounts of data, which implies that their reading or analysis requires high processing times computationally speaking. The main objective of this work was to develop a work environment integrated by multiple technologies (CUDA C, OpenCV and R) as an alternative to the conventional software for the processing of satellite images (geographic information systems), that would allow to reduce the time of Processing, by using parallelism in the graphics processing unit (GPU).

Thus, derived from the task of calculating the historical average and trend of vegetation in the State of Baja California Sur in the last 15 years, using satellite images with normalized difference vegetation index data (NDVI, for its acronym in English) collected by the NASA moderate resolution imaging spectroradiometer (MODIS, for its acronym in English) sensor. These images represent averages of data collected by the sensor for 16 days. For the period from February 2000 to December 2014.

Finally, the challenge was to solve the high processing time of analyzing 342 images, each with dimensions of 3,455 x 1,758 pixels, that is, a total of 6,073,890 pixels per image. The proposed solution was the development of an algorithm in CUDA, which showed that using parallel computation, it was possible to reduce the processing time by 69 times in the case of the *average calculation* and 155 times to *calculate the trend*. The average execution time using parallel processing (GPU) was 75 seconds for the calculation of the average and 114 seconds for the calculation of the trend. When using the traditional method (CPU) the process had a duration of 5,271 seconds for the calculation of the average and 19,088 seconds for the calculation of the trend.

Índice

Capítulo 1. Introducción

1.1 Antecedentes

Vivimos un momento histórico en donde las sociedades modernas están avasallando los recursos naturales del planeta, y existe un claro desequilibrio entre los recursos naturales que consumimos y la capacidad de regeneración de los ecosistemas que habitamos. El inminente cambio climático potenciado por las actividades humanas está propiciando todo tipo de catástrofes naturales, desde inundaciones hasta sequías prolongadas. Las actividades primarias como la agricultura y ganadería se ven afectadas, pero es un círculo en el que las personas negligentemente agotan los recursos del suelo y después son los principales afectados al terminar con la fertilidad del mismo, lo que resulta en una escasa producción de alimentos, que a su vez repercute en otros estratos de una sociedad que se encuentra entrelazada (Hori, Stuhlberger & Simonett, 2011).

El rápido crecimiento de las manchas urbanas y la falta de legislación en el uso del suelo también originan una grave deforestación de carácter antropogénico, además de las variables climáticas como el viento y la lluvia que erosionan severamente el suelo, todo esto resulta en pérdida de biodiversidad al forzar a ciertas especies a emigrar. Y no es para menos, pero la proliferación de suelos desnudos en zonas áridas, denominada *desertificación*, en situaciones graves origina problemas de índole social como la pobreza, hambruna y la falta de oportunidades para quienes viven en esas circunstancias, lo que lo vuelve un problema de índole general. Entre otros efectos que se pueden percibir son el aumento en la frecuencia e intensidad de los desastres naturales, o la extinción de especies por mencionar solo algunos (Hori *et al.*, 2011).

La desertificación definida en palabras de la principal organización no gubernamental a nivel internacional que la combate, la Convención de las Naciones Unidas para la Lucha Contra la Desertificación (UNCCD por sus siglas en inglés) que entró en vigor en 1996. Se define como:

> *La desertificación es la degradación de las tierras y la vegetación, la erosión de los suelos y la pérdida de la capa superficial del suelo y de las tierras fértiles en zonas áridas, semiáridas y subhúmedas secas, causada principalmente por las actividades humanas y por las variaciones del clima (*UNCCD, s.f.*).*

Actualmente el uso de tecnologías con fines de monitoreo del medio ambiente va en aumento, y si bien las opciones son variadas, en el caso particular de la vigilancia a la desertificación, las tecnologías de teledetección como satélites son las más utilizadas. En últimos años se han realizado un gran número de investigaciones que utilizan índices de vegetación obtenidos desde sensores abordo de satélites, con la finalidad de conocer la salud de ecosistemas terrestres (en este trabajo se utilizarón índices de vegetación de diferencia normalizada denominados NDVI por sus siglas en inglés). Pero con ello llegan retos tecnológicos, en cuanto a las capacidades para almacenar, analizar y procesar la información disponible. Y no es para menos ya que los datos recolectados durante un solo día por alguno de estos sensores, llegan a tener cantidades exorbitantes. Por lo tanto, el poder de procesamiento que exigen (computacionalmente hablando) es impresionante. En ese sentido el cómputo en paralelo salta a la vista por su enfoque de procesamiento, múltiples instrucciones ejecutándose de manera simultánea, resultando en menores tiempos de procesamiento.

Hablar de tecnologías de cómputo en paralelo involucra una amplia gama, pero abordaremos una en particular, la Arquitectura Unificada de Dispositivos de Computo, CUDA por sus siglas en inglés (en el apartado 3.4 se explicará con mayor detalle, ya que fue la tecnología utilizada en este trabajo de investigación). En un intento por definirla sucintamente, podemos decir que es un lenguaje de programación en paralelo, el cual aprovecha la capacidad de cálculo de las tarjetas de video NVIDIA, debido a su enfoque para el procesamiento de gráficos. Desde luego hay toda una trama de por medio, pero la clave reside en que este lenguaje permite la ejecución de instrucciones en paralelo del lado de la tarjeta de video (conoceremos más adelante como el código del *Device*) y estas instrucciones son invocadas, como una llamada a una función desde lenguaje C, desde el cuerpo de un programa que se está ejecutando en secuencial en la CPU (conoceremos más adelante como el código del *Host*), y brevemente así es CUDA (CUDA NVIDIA, s.f.).

1.2 Descripción del problema

El cuantificar la degradación de ecosistemas terrestres involucra varios factores o indicadores como la fragmentación del hábitat, erosión del terreno, desertificación del suelo, etc. En el caso de Baja California Sur se ha venido utilizando la *desertificación*, que es la única que se abordará desde este trabajo de investigación.

Una forma de evaluar la desertificación de ecosistemas terrestres es mediante el análisis de la tendencia de índices de vegetación como es el caso del NDVI. Para este trabajo se utilizaron imágenes de satélite de los valores del NDVI obtenidos del sensor denominado espectroradiómetro de resolución moderada (MODIS por sus siglas en inglés) a bordo del satélite Terra propiedad de la Administración Nacional de la Aeronáutica y del Espacio (NASA por sus siglas en inglés). Las imágenes que se utilizaron tienen una resolución espacial por pixel de 250 x 250 metros, lo que representa aproximadamente una superficie de 0.06 km^2. Estas imágenes de satélite fueron obtenidas de la base de datos pública de *Land Processes Distributed Active Archive Center* (https://lpdaac.usgs.gov/). Dichas imágenes de satélite se encuentran disponibles para todo el mundo y tienen una resolución temporal del promedio de 16 días.

Para el caso de la superficie del Estado de Baja California Sur (Fig. 1) se requirió la combinación de dos imágenes o cuadrantes (en inglés *tiles*, una traducción aproximada de loseta o azulejo), cada imagen contiene 4800 x 4800 pixeles. La serie de tiempo que se analizó es de febrero 2000 a diciembre 2014 y consta de un total de 342 imágenes. Pero al ser 2 cuadrantes los implicados resultó en un total de 684 imágenes que sumaban un total de 15GB (aproximadamente). El procesamiento de estos datos se se ha venido realizando con un enfoque "tradicional" en secuencial desde la CPU (utilizando software R) lo que ha representado un alto costo computacional debido al excesivo consumo de memoria y recursos del procesador que demandan por el tipo de cálculos (operaciones con matrices gigantescas para cacular promedios pixel por pixel y cálculos de la pendiente de un modelo de regresión lineal) que requieren, resultando en exorbitantes tiempos de procesamiento.

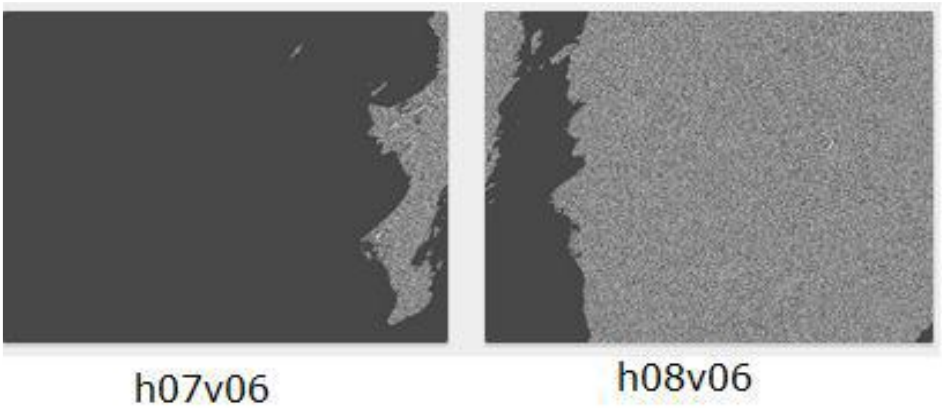

Figura 1. Cuadrículas MODIS entre los que se encuentra el estado Baja California Sur.

1.3 Objetivo General

Desarrollar un algoritmo utilizando la arquitectura de cómputo en paralelo de CUDA, que permita el procesamiento y análisis del NDVI, calculando el promedio histórico y la tendencia (mediante el cálculo de la pendiente de un modelo de regresión lineal) de los índices de vegetación de una serie de tiempo dada (2000-2014), que pueda servir para evaluar la desertificación de los suelos al conocer el comportamiento que han tenido los índices de vegetación en dicho periodo de tiempo. Con la finalidad de reducir los elevados tiempos de procesamiento actuales.

1.3.1 Objetivos Particulares

El algoritmo deberá considerar los siguientes puntos:

- Poder leer las imágenes en el *HOST*.
- Diseñar estrategia para el envío de las imágenes entre el *HOST* y el *DEVICE*.
- Implementar cálculo del promedio histórico del NDVI en el *Device* (GPU).
- Implementar cálculo de la pendiente de un modelo de regresión lineal en el *Device* (GPU).
- Grabar la salida del resultado de los cálculos del promedio y la tendencia del NDVI.

1.4 Limitaciones y alcances

El algoritmo propuesto para cuantificar la desertificación a través del procesamiento de índices de vegetación, solo se podrá ejecutar bajo arquitecturas de tarjetas NVIDIA, siempre y cuando estas soporten CUDA.

El algoritmo propuesto solo funcionará sin ningún tipo de configuración en tarjetas NVIDIA GeForce GTX Titan X (12GB), que si bien ofrece capacidades importantes (tamaño de memoria, cantidad de núcleos CUDA), se infiere que, a mayor capacidad de la tarjeta utilizada, se obtendría una mayor aceleración en el tiempo de procesamiento.

Otro factor a considerar es que los índices de vegetación del satélite MODIS solo están disponibles desde febrero del año 2000, y a un mayor número de datos disponibles arrojaría resultados más precisos.

Por último, este trabajo solo se enfocó a procesar índices de vegetación de un polígono equivalente a la región de Baja California Sur (específicamente pertenecen a los productos de Tierra MODIS), aunque en un futuro podrían integrarse otros productos MODIS (océanos, aerosoles, temperaturas, etc.).

1.5 Justificación

Es importante contar con una alternativa al software convencional para el procesamiento de grandes volúmenes de datos geoespaciales en el menor tiempo y costo posible. Es así que una solución a la medida utilizando la arquitectura de cómputo paralelo CUDA, que aprovecha la potencia de cálculo de la GPU brinda la posibilidad de acelerar el procesamiento de índices de vegetación extraídos de imágenes de satélite.

Además, el co-procesamiento en una GPU de una tarjeta de video de gama media (NVIDIA GeForce GTX Titan X) respecto a estaciones de trabajo de última generación resulta una opción más asequible. Aunado a ello el contar con una alternativa que permita determinar el grado de desertificación de determinada región, o el obtener una perspectiva general del comportamiento que han tenido los índices de vegetación a través del tiempo es de sumo interés para determinar la salud de cierto ecosistema.

Finalmente, el uso de imágenes satelitales para cuantificar la degradación de los ecosistemas terrestres obtenidas gracias a las tecnologías de teledetección actuales, es considerado como una herramienta de bajo costo y sobre todo disponibles de manera gratuita para el público en general.

1.6 Estado del arte

Existen sin fin de investigaciones que están cimentadas en el uso de imágenes de satélite, procesamiento en paralelo y técnicas de procesamiento de imágenes, etc., pero si bien existen coincidencias en cuanto a las tecnologías que utilizan y las implementadas en este trabajo, ninguna de las investigaciones consultadas se encontró que hiciera exactamente lo que se propuso para este trabajo.

Enseguida se mencionarán solo algunas de las investigaciones más significativas en cuanto a la similitud de tecnologías involucradas (las demás consultadas han sido solamente citadas en la bibliografía de este escrito):

Una investigación en la meseta de Ordos en China, para evaluar y cuantificar la desertificación en el año 2009. Utilizó imágenes satelitales LANSAT, para extraer el Índice de Vegetación de Diferencia Normalizada (NDVI), y el Índice de Desviación Estándar Dinámico (MSDI), y el albedo de la tierra. En este trabajo utilizaron una serie de tiempo de 30 años (1980-2000), lo resultados de la investigación los plantearon en intervalos de 10 años y el criterio con el cual los obtuvieron fue el método de clasificación de "árbol de decisión". Finalmente, los datos arrojaron que el cambio climático y la expansión de las actividades humanas son las principales razones de la desertificación en esa zona (Xu, Kang, Qiu, Zhuang & Pan, 2009).

En el Estado de Sinaloa Lopez (2014) desarrolló una metodología que identificaría zonas propensas a desertificación. Para ello utilizó imágenes de satélite obtenidas con el sensor MODIS, variables biofísicas y antropogénicas, así como técnicas de evaluación multicriterio y tecnologías de información geográfica.

En otra investigación Liu, Feld, Xue, Member, Garcke & Soddeman (2015), realizaron dos implementaciones eficientes de un algoritmo de recuperación de Profundidad Óptica de Aerosoles (AOD por sus siglas en inglés), a partir de imágenes de Espectro-radiómetro de Resolución Moderada (MODIS por sus siglas en inglés), se empleó dos arquitecturas de cómputo de alto rendimiento diferentes: procesadores de múltiples núcleos y una unidad de procesamiento de gráficos (GPU).

También Zhang, You & Gruenwald (2011), hicieron una implementación de técnicas de compresión de datos raster geoespaciales y técnicas de compresión populares. Presentaron una aplicación paralela de compresión de datos raster geoespaciales implementada en una GPGPU en comparación con una CPU multi-core con 16 hilos, y consiguen una aceleración de 12x en comparación con la implementación multi-core (con 16 hilos) de la popular librería de compresión Zlib.

En una más Zhang, Wang & Chen (2010), presentaron soluciones basadas en las ventajas de operaciones en paralelo de una GPGPU. La comparativa consistió en dos algoritmos de procesamiento de imágenes representativos, el detector de bordes de Sobel y filtrado homomórfico. Los datos de las imágenes de prueba fueron de diferentes resoluciones y se utilizaron tanto en la plataforma de la CPU y la GPU para poder comparar su eficiencia computacional.

En la última investigación Yang, Zhu & Pu (2008), hicieron un análisis sobre las distintas características de CUDA GPU y resumen el modo de programa general de CUDA. Además, ponen en práctica varios algoritmos de procesamiento de imágenes clásicos de CUDA, como la ecualización del histograma, la eliminación de las nubes, detección de bordes y codificación/decodificación de transformada discreta del coseno (DTC por sus siglas en inglés) etc., sobre todo introduce los dos primeros algoritmos.

Capítulo 2. Marco Teórico

2.1 Área de estudio

El área de estudio de este trabajo es el estado de Baja California Sur, conformado por cinco municipios: Comondú, Mulegé, La Paz, Los Cabos y Loreto, siendo la capital de la entidad la ciudad de La Paz. La superficie total del estado es de aproximadamente 73,922.47 km^2, lo que representa el 3.8% del territorio nacional. Baja California Sur se encuentra rodeada al oeste por el Océano Pacífico y al este por el mar de Cortes, cuenta con 2,230 km de litoral, lo que representa el 22% del litoral del país (Cuéntame INEGI, 2016).

El relieve del estado está constituido iniciando por el norte con la Sierra de la Giganta en donde existen algunas mesetas que alcanzan hasta los 1,740 msnm que se extiende hasta llegar a la bahía de La Paz. En el suroriente se localiza la sierra La Laguna (reserva de la biosfera) con 2,080 msnm. Al oriente, predominan zonas bajas representadas por lomeríos y llanuras interrumpidos por sierras de 800 msnm. Al occidente de Ciudad Constitución, se ha formado una zona de dunas (montañas de arena). Y finalmente en la parte occidental, hay una serie de barras o cordones litorales (barrera de arenas y gravas junto a la costa) que se han formado desde Puerto San Andresito hasta Bahía de Santa Marina (Cuéntame INEGI, s.f.).

En general la vegetación predominante es de matorrales, le siguen en importancia las selvas secas en la región de Los Cabos y los manglares de las costas. En la parte sur se localizan los bosques, principalmente de encinos en la Sierra La Laguna, en la parte litoral existe vegetación de dunas costeras y mezquitales que se presentan en los cauces de arroyos intermitentes (Cuéntame INEGI, s.f.).

El clima predominante en el estado es catalogado como “Muy seco”, en verano sobrepasa los 40°C y durante el invierno desciende hasta 0°C en ciertas zonas, y en conjunto este y todos los demás factores favorecen la desertificación del Estado. Esto es particularmente relevante para Baja California Sur, ya que las predicciones indican una tendencia a la acentuación de la aridez, y con ello un mayor riesgo a la desertificación (Cuéntame INEGI, s.f.).

Existe una cadena montañosa que atraviesa longitudinalmente una parte del Estado, la cual propicia varios microclimas. La vegetación más representativa del Estado está formada por matorrales, manglares en las zonas costeras hasta selva seca y bosques de pino-encino en regiones de la Sierra La Laguna. La vegetación es principalmente matorrales en un 86% y el clima es 92% predominante seco. Esto último propicia prolongadas sequias la mayor parte del año originando escasez de agua por la falta de lluvias. Uno de los principales problemas de la entidad es el garantizar el suministro de agua potable, la cual depende esencialmente de 11 mantos acuíferos y aguas desaladas, además del tratamiento de aguas residuales para uso agrícola principalmente (Cuéntame INEGI, s.f.).

2.2 Tecnología de teledetección

Para comprender las tecnologías de teledetección de lo general a lo particular, se inicia con los rayos de luz que el sol irradia sobre la superficie de la tierra, los cuales viajan en forma de ondas electromagnéticas a una velocidad constante aproximada de 300,000 km/s. Estas ondas electromagnéticas no necesitan un medio material para propagarse, como son los rayos UV, las ondas de radio, los rayos gamma, etc. (Peña & Dreyfus ,2012; Muñoz, 2015).

Dichas ondas son medidas de acuerdo a la distancia real que recorren en un determinado intervalo, denominada longitud de onda y se mide su frecuencia en Hertz (Hz). La ordenación de los diversos tipos de radiación electromagnética por su frecuencia recibe el nombre de espectro electromagnético. Dentro del espectro existe un pequeño rango de ondas que son visibles al ojo humano (lo que conocemos como luz visible). La parte visible de la luz tiene longitudes de onda que van desde los 420 a 650 nanómetros (Fig.2). Mientras mayor es la longitud de onda menor es la frecuencia de la misma. Estas longitudes de ondas pueden ser absorbidas y reflejadas por los cuerpos sobre las que inciden (Peña & Dreyfus ,2012).

El ojo humano ve los colores que son reflejados por algún cuerpo en particular, de tal manera que los objetos que vemos de color verde absorben todas las longitudes de onda a excepción de las correspondientes a este color. Tal es el caso de las plantas verdes, que absorben casi todas las longitudes de onda y reflejan energía en la longitud de onda del color verde.

Algunos satélites que orbitan la tierra cuentan con sensores que permiten observar y cuantificar la energía que reflejan los cuerpos en diferentes longitudes de onda, por ejemplo, el sensor MODIS. En el apartado 2.3 se explicará a detalle el sensor MODIS.

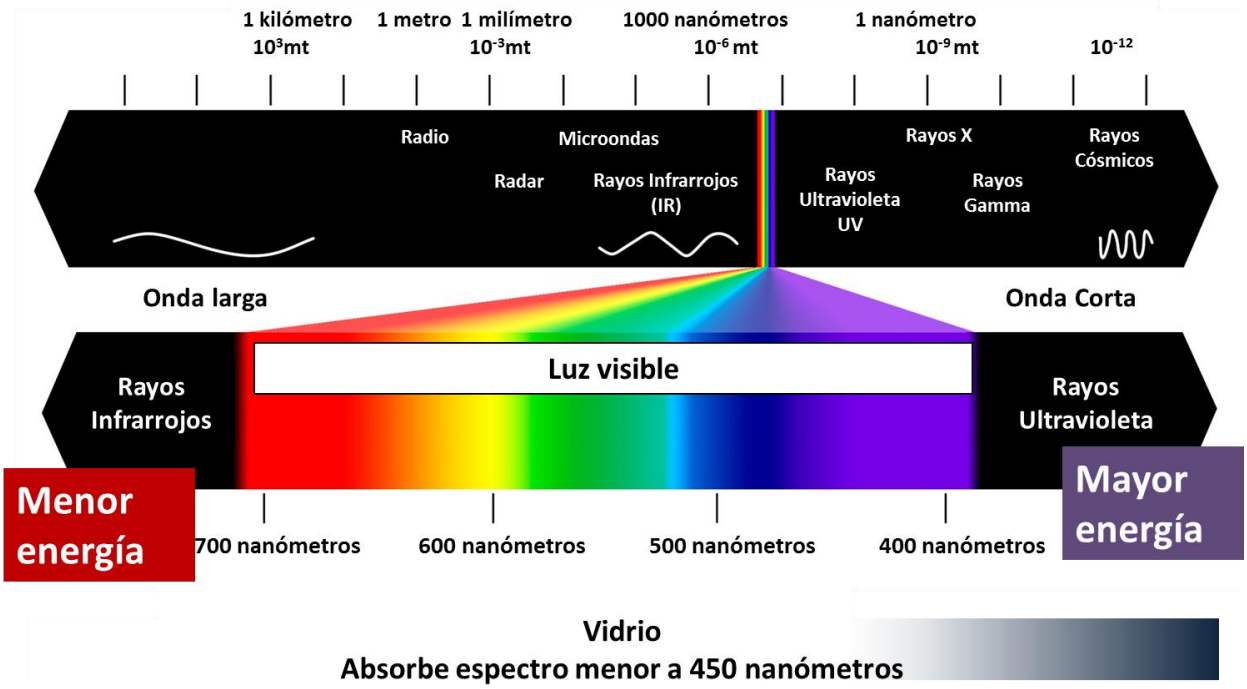

Figura 2. Espectro electromagnético. Fuente: Reimpreso de, Energía y Desarrollo Humano, En Blogger, 13 septiembre 2015, Recuperada 21 octubre 2016 de, http://5cdszillicarola.blogspot.mx/2015_09_01_archive.html. Copyright © 1999-2016 Google.

La percepción remota involucra varios elementos según se describen en la Fig. 3, y a cotinuación veremos con mayor detalle (INEGI, s.f.):

- Fuente de energía o iluminación, es la que emite la energía electromagnética al objeto de interés.
- Atmósfera, ya que la energía interactúa con este elemento al viajar del sensor al objeto y viceversa.
- Objetos, dependiendo de sus propiedades es la interacción o respuesta a la energía recibida y reflejada.

- Sensor remoto, que recoge y graba la radiación electromagnética reflejada o emitida por el objeto y la atmósfera. Este instrumento se monta en una plataforma llamada satélite.
- Transmisión, recepción y procesamiento. La energía grabada por el sensor se transmite a una estación receptora, en donde los datos se procesan y son convertidos a imágenes digitales.
- Interpretación y análisis, consiste en interpretar la imagen para extraer la información de los objetos captados.
- Usuario final que le da una aplicación a la información extraída de las imágenes para un mejor conocimiento de los objetos de interés.

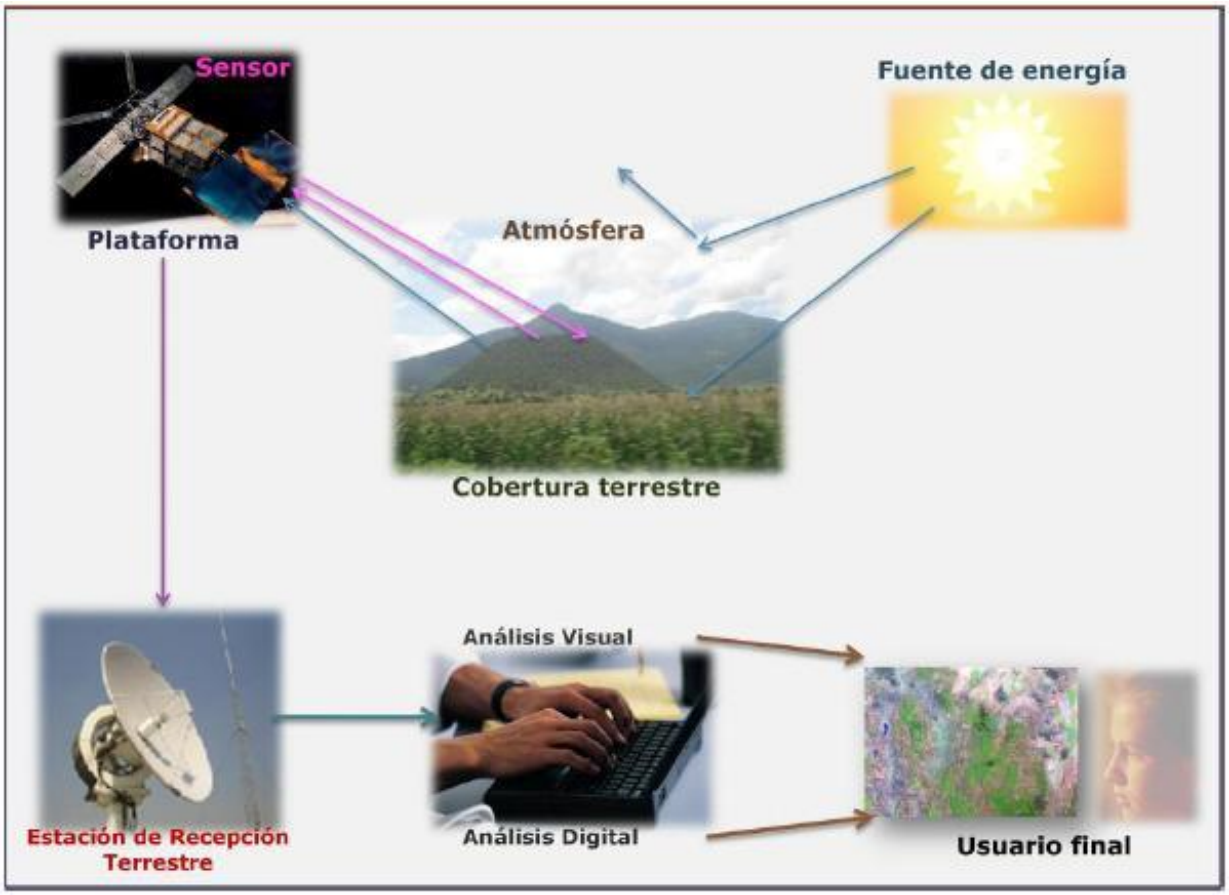

Figura 3. Proceso de percepción remota, Fuente: Reimpresa de, Elementos de percepción remota, En INEGI, s.f., Recuperada 21 octubre 2016 de, http://www.inegi.org.mx/geo/contenidos/imgpercepcion/imgsatelite/elementos.aspx, Derechos Reservados © INEGI.

2.3 Sensor MODIS

El sensor con el que fueron generadas las imágenes utilizadas para este trabajo se le denomina MODIS por sus siglas en inglés, que significa *Imágenes de Espectro Radiómetro de Resolución Moderada* y opera desde el satélite Terra que es un satélite de investigación que pertenece a la *Administración Nacional de la Aeronáutica y del Espacio* (NASA por sus siglas en inglés) que comenzó transmisiones desde febrero del año 2000 (TERRA, s.f.).

MODIS tiene un ancho de franja de visión de 2,330 kilómetros y recorre la superficie del planeta de norte a sur y viceversa cada uno o dos días; el cual puede recolectar información en 36 bandas espectrales que miden distintas longitudes de onda entre 0.405 y 14.385 μm de la energía reflejada desde la superficie del planeta (TERRA, s.f.).

El producto que se utilizó para este trabajo está disponible en tres resoluciones espaciales 250m, 500m y 1,000m. En conjunto los datos obtenidos por el sensor MODIS y otros instrumentos de medición también a bordo, se transfieren a las estaciones terrestres en White Sands, Nuevo México para un primer pre-procesamiento de los datos en bruto previo a su publicación en el sitio FTP de la NASA (LP DAAC, s.f.).

En el año de 1990 se eligió el Formato de Jerarquía de Datos (HDF por sus siglas en inglés) para ser el formato de archivo estándar del Sistema de Observación de la Tierra (EOS por sus siglas en inglés) para la recopilación de datos de los sensores de los satélites. Este es un empaquetado de varias capas que solo es posible ver a través de un visor o biblioteca especializada de archivos HDF; y el tamaño de un solo archivo HDF-MODIS puede oscilar entre 4.9 – 277 megas (HDF Group, s.f.).

2.4 NDVI

Los índices de vegetación de diferencia normalizada (NDVI por sus siglas en inglés) se consideran un indicador o parámetro de la salud de un ecosistema en función de la cobertura vegetal presente. Este método consiste en medir la reflectancia de la vegetación

en las longitudes de onda en el espectro del rojo y cercano al infrarrojo (Fig. 4), las cuales ya no son visibles al ojo humano (Weier & Herring, 2000).

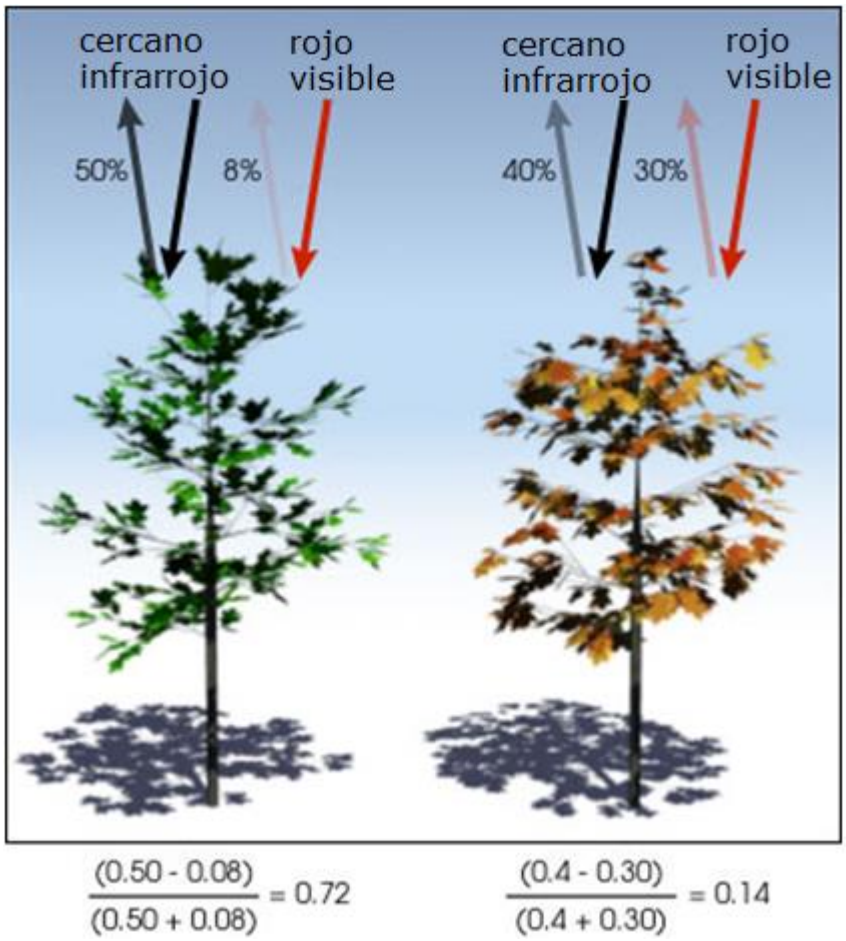

Figura 4. Esquema de la reflectancia en la vegetación origen del NDVI. Fuente: Adaptada de, Measuring Vegetation (NDVI & EVI), En Earth Obsrvatory,30 agosto 2000, Recuperada 21 octubre 2016 de, http://earthobservatory.nasa.gov/Features/MeasuringVegetation/. Copyright © Earth Observatory - NASA.

Estos índices se utilizan para identificar plantas saludables o plantas estresadas según sea el caso. Una planta saludable absorbe una mayor cantidad de energía en la parte visible de la luz, en adición refleja una mayor cantidad de energía en las longitudes de onda del infrarrojo cercano. Por el contrario, una planta menos saludable o estresada absorbe una menor cantidad de luz en la parte visible de la luz, por lo tanto, refleja una mayor cantidad de energía en estas longitudes de onda (Weiner & Herring, 2000).

A partir de estas características de absorbancia/reflectancia de las plantas fue creado el NDVI. Su definición matemática es el cociente de la diferencia de la reflectancia (R) observada en la banda del infrarrojo cercano (NIR por sus siglas en inglés) y la banda del rojo (R_{NIR} - R_{RED}) entre la adición de las mismas (R_{NIR} + R_{RED}) como se muestra a continuación (Weiner & Herring, 2000):

$$NDVI = \frac{R_{NIR} - R_{Red}}{R_{NIR} + R_{Red}}$$

Los valores del NDVI tienen un rango entre -1 y 1 (Fig. 5), siendo los valores negativos los que se observan en zonas con nieve o cuerpos de agua, valores positivos cercanos al cero corresponden a suelos desnudos o rocosos, valores entre 0.1 y 0.3 usualmente están relacionados a zonas de pastizales, valores superiores a 0.7 usualmente se observan en ecosistemas con alta densidad de vegetación como son los bosques y manglares.

Figura 5. Escala de valores NDVI. Fuente: Adaptada de, Measuring Vegetation (NDVI & EVI), En Earth Obsrvatory,30 agosto 2000, Recuperada 21 octubre 2016 de, http://earthobservatory.nasa.gov/Features/MeasuringVegetation/. Copyright © Earth Observatory - NASA.

Los índices de vegetación tienen una gran historia de uso en una amplia gama de disciplinas. Enseguida se presentan solo algunos ejemplos: (UN-SPIDER[1], s.f.)

- Monitoreo de vegetación;
- Estudios de sequía;
- Actividades agrícolas;
- Modelado hidrológico y del clima;

Sin embargo, ninguno de los ejemplos anteriores sería una realidad sin el uso de la percepción remota.

1 UN-SPIDER: Acrónimo de "United Nations Platform for Space-based information for Disaster Management and Emergency Response".

2.5 Tiles Grid MODIS

Las imágenes MODIS de la tierra se producen en tres proyecciones: sinusoidal, acimutal de Lambert y geográfica, todos sus productos están disponibles para su descarga gratuita desde el sitio web filial de la NASA denominado Centro de Archivos Activos de Procesos Distribuidos de la Tierra (MODIS Land, s.f.).

Con la finalidad de mantener las descargas de los archivos de mayores resoluciones espaciales con tamaños razonables, están divididos en una representación de una red/rejilla de losetas/azulejos (seria la traducción aproximada de *Tiled Grid)* que abarca toda la superficie de la tierra, y la mayoría de los productos MODIS de tierra son distribuidos en la rejilla de proyección sinusoidal (Fig. 6), excepto los productos de hielo marino, que son generados en una rejilla con proyección acimutal de Lambert (Fig.7) (MODIS Land, s.f.).

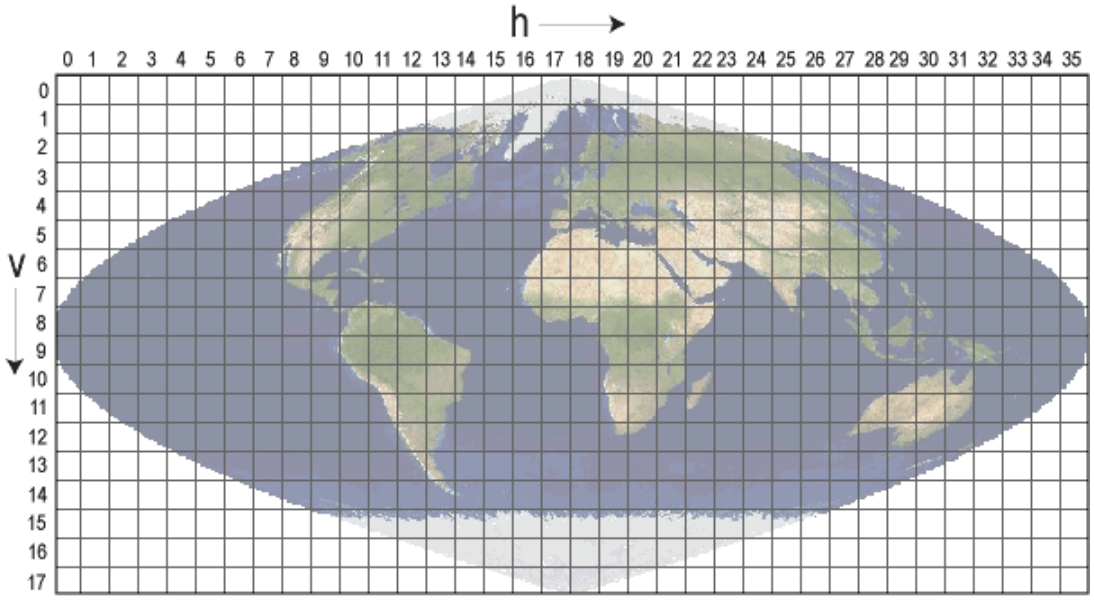

Figura 6. Proyección sinusoidal. Fuente: Reimpresa de, Modis grids, En MODIS Land Team, s.f., Recuperada 21 octubre 2016, https://modis-land.gsfc.nasa.gov/MODLAND_grid.html. Copyright © MODIS Land Team.

Es así que observando la rejilla de proyección sinusoidal se puede apreciar que para la superficie del Estado de Baja California Sur queda en el eje horizontal 07 y 08, pero en el eje vertical 06, dando la combinación de cuadrantes que vimos en la Fig.1.

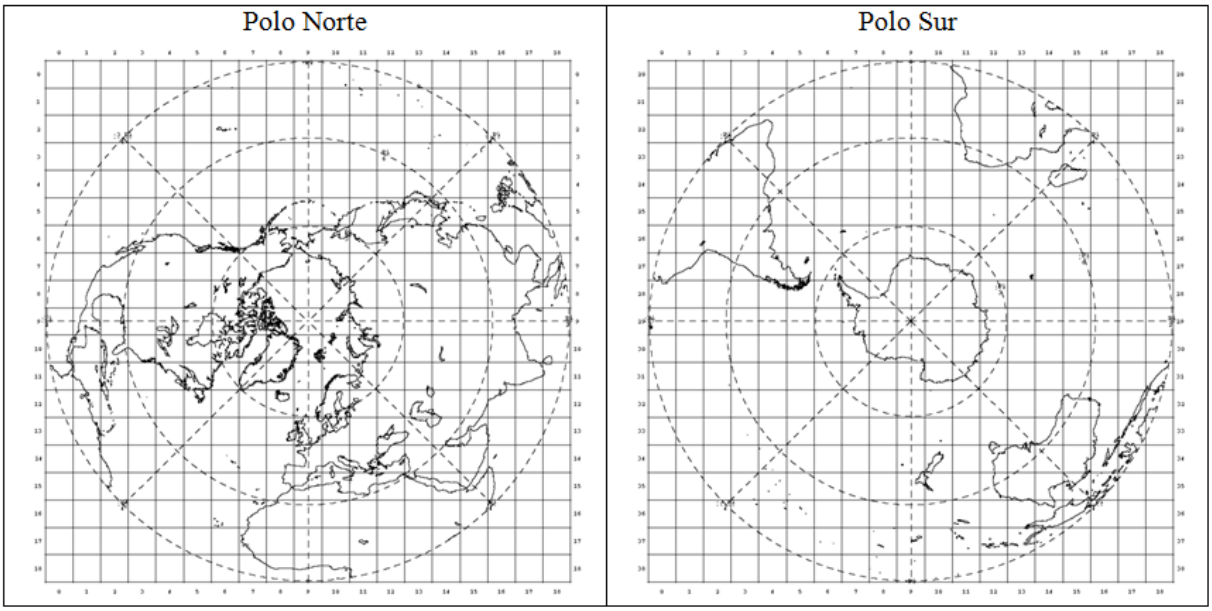

Figura 7. Proyección acimutal de Lambert, Fuente: Adaptada de, Modis grids, En MODIS Land Team, s.f., Recuperada 21 octubre 2016, https://modis-land.gsfc.nasa.gov/MODLAND_grid.html. Copyright © MODIS Land Team.

2.6 MOD13Q1

MOD13 es sinónimo de los productos MODIS VI sobre Índices de Vegetación, en donde "MOD" indica que fue creado con datos del sensor a bordo de la plataforma TERRA y por el contrario cuando es de la plataforma AQUA dice "MYD", después las últimas dos letras "Q1" del final indica la resolución espacial que en este caso sería cuarto de kilometro (quarter) es decir 250 metros. En general se describe el producto en inglés como: "Vegetation Index 16-Day Global 250m (MOD13Q1)", el cual representa las variaciones espacio temporales en la actividad de la vegetación del promedio resultante de 16 días en intervalos mensuales, en promedio anualmente si las condiciones meteorológicas (nubosidad) lo permiten se crean 22 a 24 imágenes aproximadamente (Solano, Didan, Jacobson & Huete, 2010) .

Cabe señalar que el producto MODIS denominado "MOD13Q1" fue el concretamente utilizado para este trabajo, el cual está conformado a su vez (por ser formato HDF) de un total de 12 capas como se ilustra en la Tabla 1, de las cuales específicamente se utilizó el NDVI (Solano *et al*., 2010).

Tabla 1
Capas o sub-archivos del producto MOD13Q1

Data set	*Units*	*Data type*	*Validrange*	*Scale factor*
16 days NDVI	NDVI	Int16	-2000, 10,000	0.00001
16 days EVI	EVI	Int16	-2000, 10,000	0.00001
16 days VI Quality detailed QA	Bits	uint16	0, 65534	NA
16 days red reflectance (Band 1)	Reflectance	int16	0, 10000	0.0001
16 days NIR reflectance (Band 2)	Reflectance	int16	0, 10000	0.0001
16 days blue reflectance (Band 3)	Reflectance	int16	0, 10000	0.0001
16 days MIR reflectance (Band 7)	Reflectance	int16	0, 10000	0.0001
16 days view zenith angle	Degree	int16	-9000, 9000	0.01
16 days sun zenith angle	Degree	int16	-9000, 9000	0.01
16 days relative azimuth angle	Degree	int16	-3600, 3600	0.1
16 days composite day of the year	Day of year	int16	1, 366	NA
16 days pixel reliability summary QA	Rank	int8	0, 3	NA

Fuente*:* Solano, Didan, Jacobson & Huete (2010), *MODIS Vegetation Index User's Guide*. Recuperada 21 octubre 2016 de, https://vip.arizona.edu/documents/MODIS/MODIS_VI_UsersGuide_01_2012.pdf

2.7 TIFF

Tagged Image File Format (TIFF por sus siglas en inglés) es un formato de archivo de imagen utilizado para el almacenamiento e intercambio de datos, fue lanzado originalmente 1986 por Aldus Corporation (TIFF, 1992).

La estructura de organización de este tipo de archivos se divide principalmente en tres secciones (Fig. 8): encabezado de archivo de imagen, IFH por sus siglas en inglés (Image File Header), directorio de archivo de imagen, IFD por sus siglas en inglés (Image File Directory) que puede haber más de uno por archivo, y por último los valores del mapa de bits (TIFF, 1992).

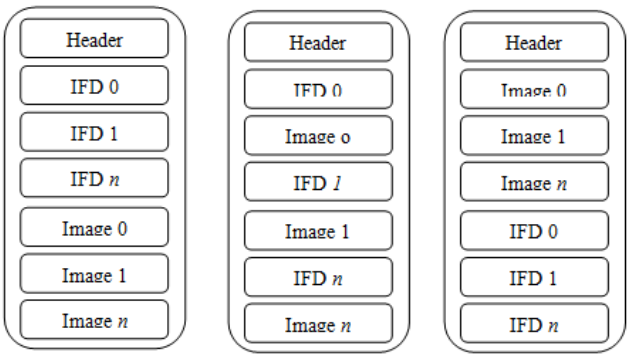

Figura 8. Estructuras posibles de un archivo TIFF.

El IFH es la única sección que tiene una localización fija dentro del archivo, siempre está al inicio en los primeros 8 bytes, y contiene información en los últimos 4 bytes hacia el primer IFD (TIFF, 1992).

EL IFH contiene la siguiente información, que a continuación se detalla (TIFF, 1992):

Bytes 0-1: Tipo de byte de orden utilizado dentro del archivo, los valores pueden ser "II" o "MM".

Bytes 2-3: Un arbitrario pero significativo número (42) cuidadosamente escogido que identifica el archivo como *archivo TIFF*.

Bytes 4-7: El desplazamiento (en bytes) hacia el primer IFD, el cual puede estar en cualquier ubicación del archivo después del IFH. Un IFD debería seguir de los datos de la imagen que describe. Los lectores deben seguir el apuntador dondequiera que este los conduzca.

El término desplazamiento en bytes (bytes offset) es siempre utilizado según lo especificado en el documento de la revisión 6 del formato TIFF, se refiere a una localización con respecto al inicio del archivo TIFF. El primer byte de archivo tiene un desplazamiento de cero.

EL IFD contiene una o más estructuras llamadas etiquetas (Tags) o entradas de directorio (*Directory Entry o IFD entry*), es una especie de arreglo de espacios consecutivos. Los primeros 2 bytes de cada IFD indican el número de entradas de directorio o Tags que se esperan encontrar en ese IFD, seguidos de una secuencia de 12 bytes por cada Directory Entry, las cuales sirven para almacenar información sobre la imagen (número de filas, número de columnas, profundidad de los pixeles, etc.). Los últimos 4 bytes de cada IFD indicaran la ubicación hacia el siguiente IFD (TIFF, 1992; FileFormat.Info, s.f.).

Debe haber al menos un IFD por archivo TIFF, y cada IFD debe tener al menos un Directory Entry, cada elemento dentro del Directory Entry se le denomina campo (*field*) (FileFormat.Info, s.f.).

Cada 12 bytes el Directory Entry o Tag tiene el siguiente formato (TIFF, 1992):

Bytes 0-1: El Id de la etiqueta, de acuerdo a este valor se puede conocer que tipo de informacion esta almacenando, por ejemplo, el numero de filas de la imagen, o columnas de la imagen, profundidad de los pixeles (64 bits), etc.

Bytes 2-3: El tipo de dato(s) del valor almacenado por este Tag, los valores pueden ser del 1 al 5 (1-Byte, 2-Ascii, 3-short, 4-Long, 5-Rational).

Bytes 4-7: El número de valores, cantidad de valores almacenados (y/o apuntados) de acuerdo al tipo indicado en el *field* anterior. Que pueden ser serie de valores consecutivos (por ejemplo, cada uno de los valores de los pixeles) o un solo dato (número de filas de la imagen).

Bytes 8-11: El valor del desplazamiento hacia donde se encuentran los valores (o valor) almacenados (o apuntados) por esta etiqueta. Solo en caso que los valores almacenados esten conformados por una serie de valores consecutivos, este *field* almacenara el desplazamiento al inicio de esos valores, siempre partiendo desde el inicio del archivo. Pero si el *field* anterior indicaba el valor de 1 (uno), el valor de ese único dato estará almacenado en este *field*, de lo contrario el valor almacenado representará el desplazamiento hacia donde inician los datos (TIFF, 1992).

En conclusión, el formato TIFF es capaz de albergar una gran variedad de estilos de imágenes, es decir imágenes de dos niveles (bi-level), en escala de grises (grayscale), paleta de colores (palette-color) e imágenes a todo color (full-color images). Pero sin importar el tipo de imagen que sea, la estructura del formato TIFF parte de la línea base explicada anteriormente (TIFF, 1992).

2.7.1 GeoTIFF

De acuerdo al sitio oficial de la especificación GeoTIFF, lo describen como un archivo formato TIFF utilizado para el intercambio de imágenes raster georreferenciadas. Y es porque el formato GeoTIFF adopta por completo la especificación TIFF 6.0, salvo unos pequeños cambios (GeoTIFF, 2000).

Un ejemplo es la información almacenada dentro de algunos de los principales "Tags" (que son privados y se ilustran en la Tabla 2) en GeoTIFF, a través de los cuales se puede describir toda la información de la especificación que representa proyecciones, sistemas de coordenadas, datums, elipsoides, etc. (GeoTIFF, 2000).

Tabla 2
Tags ID Summary

TIFF Tags	***Id***	***Owner***
ModelPixelScaleTag	33550	SoftDesk
ModelTransformationTag	34264	JPL Carto Group
ModelTiepointTag	33922	Intergraph
GeoKeyDirectoryTag	34735	SPOT
GeoDoubleParamsTag	34736	SPOT
GeoAsciiParamsTag	34737	SPOT

Fuente: GeoTIFF, (2000), Recuperada 21 octubre 2016 de, http://web.archive.org/web/20160306050539/http://remotesensing.org/geotiff/spec/geotiff6.html#6.2

El Tag denominado "GeoKeyDirectoryTag" es capaz de representar cientos o miles de parámetros involucrados en la representación de información geográfica, utilizando "Meta-tag". El "Meta-tag" se refiere a un tipo de ordenación/estructuración que sirve para almacenar *keys* (también llamados *GeoKeys*) que son virtualmente similares a los Tags de TIFF. Los Id de los *Keys* pueden tener valores en el rango de 0 a 65,535 y existe una subclasificación por rango de valores como se ilustra en la Tabla 3 (GeoTIFF, 2000).

Tabla 3
Sub-clasificacion id keys

Rango Id Keys	***Descripción***
[0, 1023]	Reservados
[1024, 2047]	Keys de configuracion GeoTIFF
[2048, 3071]	Keys de parametros del sistema de coordenadas geografico/geocetrico
[3072, 4095]	Keys de parametros del sistema de coordenadas proyectado
[4096, 5119]	Keys de parametros del sistema de coordenadas vertical
[5120, 32767]	Reservados
[32768, 65535]	Uso privado

Fuente: GeoTIFF, (2000), Recuperada 21 octubre 2016 de http://web.archive.org/web/20160305042932/http://remotesensing.org/geotiff/spec/geotiff2.7.html#2.7

El Tag GeoKeyDirectoryTag (34735) almacena un arreglo de 4 valores *unsigned short* que representan un *Header*. Después del header se almacenan inmediatamente N arreglos también de 4 valores *unsigned short* por cada KeyEntry definido en el Header (GeoTIFF, 2000).

La estructura del **`Header`**:

```
[ KeyDirectoryVersion, KeyRevision, MinorRevision, NumberOfKey ]
```

Donde (GeoTIFF, 2000):

`KeyDirectoryVersion` Indica la versión actual de implementación *Key*.

`KeyRevision` Indica que revisión del conjunto *Keys* está utilizando.

`MinorRevision` Indica que revisión del conjunto *Codes* se están utilizando.

`NumberOfKey` Indica cuantos Keys hay definidos en el resto del Tag.

La estructura del **`KeyEntry`**:

```
[ KeyID, TIFFTagLocation, Count, Value_Offset ]
```

Donde (GeoTIFF, 2000):

KeyID Proporciona el valor del Id del Key (idéntico al Id Tag de TIFF, pero completamente independiente de los espacios disponibles de los Tags).

TIFFTagLocation Indica el Id del Tag-TIFF que contiene los valores de ese Key. Si TIFFTagLocation es 0, entonces el valor almacenado será de tipo SHORT, y estará almacenado en el Value_Offset del mismo KeyEntry. De lo contrario, el tipo de valor (formato) implicado estará almacenado en un Tag TIFF definido para contener ese tipo de valores [GeoDoubleParamsTag (34736) ó GeoAsciiParamsTag (34737)].

Count Indica la cantidad de valores en este Key.

Value_offset Indica el índice de desplazamiento hacia el TagArray indicado por TIFFTagLocation, sí es diferente de 0. Sí TIFFTagLocation es 0, entonces el Value_Offset contiene el valor actual (SHORT) del Key, y el Count=1 es implícito. Nota que el offset no es un desplazamiento en bytes, pero más bien un índice basado en el tipo de dato natural del especificado por TagArray.

Ejemplo de una estructura de **GeoKeyDirectoryTag** (GeoTIFF, 2000):

```
    GeoKeyDirectoryTag=(     1,     1, 2,     6,     <-- Header
                          1024,     0, 1,     2,
                          1026, 34737,12,     0,
                          2048,     0, 1, 32767,     Keys entries
                          2049, 34737,14,    12,
                          2050,     0, 1,     6,
                          2051, 3 4736,1,     0)
GeoDoubleParamsTag(34736)=(1.5)
GeoAsciiParamsTag(34737)=("Custom File|My Geographic|")
```

Una vez explicado el papel que desempeñan los *Keys*, se debe abordar también el concepto de *Codes* que son los valores almacenados en los primeros (también pueden tener valores del rango de 0 a 65,535). Sin embargo, los *Codes* del 32,768 hacia arriba son para implementaciones de uso privado, quien las utilice las deberá utilizar *solo de manera personal* y bajo su propio riesgo. Por lo tanto, los *Codes* de 0 a 32,767 son públicos y reservados para la especificación GeoTIFF (2000).

Algunos valores de *Codes* comunes y definidos para mantener consistencia entre los diversos sistemas de códigos numéricos (tipo SHORT), son los siguientes (GeoTIFF, 2000):

0 = undefined

32767 = user-defined

Donde:

undefined Indica que este Key este intencionalmente omitido por cualquier razón (GeoTIFF, 2000).

User-defined Indica que esta característica no está en el listado estándar, y ha sido explícitamente definido por el usuario para esta característica (*key*) (GeoTIFF, 2000).

En cuanto a la especificación GeoTIFF hay sin fin de parámetros que puede utilizar, pero se resume en que gracias a las particularidades de la especificación TIFF y el uso "Meta-Tag" se puede almacenar prácticamente cualquier tipo de imagen geoespacial (GeoTIFF, 2000).

2.8 Regresión Lineal

Un modelo de regresión lineal es utilizado, como su nombre lo indica, para evaluar que tan fuerte o no es la relación lineal que existe entre dos variables numéricas. De acuerdo con Crawley (2013), la esencia del análisis de regresión lineal es utilizar una muestra de datos para estimar los coeficientes, así como sus errores estándar. La representación matemática de un modelo de regresión lineal simple, es decir, una variable predictiva es la siguiente:

Ecuación de la línea recta:

$$y = a + bx$$

Donde *y* es la variable dependiente, *a* es el intercepto, *b* es la pendiente y *x* es la variable independiente.

De manera sencilla podemos decir que un modelo de regresión lineal evalúa el efecto de *x* sobre *y*, es decir, si *b* tiene un valor positivo significa que *y* aumenta de manera proporcional cuando *x* aumenta, por el contrario, si *b* tiene un valor negativo, implica que *y* disminuye proporcionalmente cuando *x* aumenta. Si el valor de *b* es cero, significa que no existe una relación lineal entre ambas variables. En la Fig. 9 se muestra un esquema de la ecuación de un modelo de regresión lineal simple.

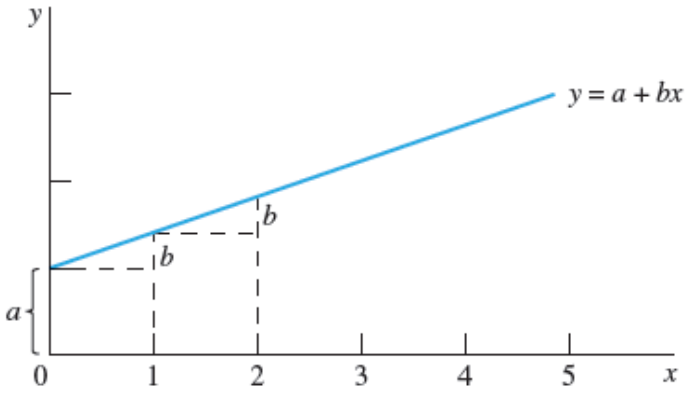

Figura 9 Gráfica de dispersión de x contra y. Fuente: Reimpresa de *Introduccion a la probabilidad y estadística* (pp. 105-109), por W. Mendenhall, R.J. Beaber, B. M. Beaver, México: CENGAGE, Copyright 2008.

El método empleado para estimar los mejores valores de *a* y *b* es el conocido como método de mínimos cuadrados (Mendenhall, Beaber & Beaver, 2010). Este método toma su nombre de las ecuaciones matemáticas involucradas en su cálculo, en este caso el método utiliza la sumatoria de las diferencias cuadráticas de los valores observados y los valores ajustados (la línea recta). El método busca reducir a cero esta sumatoria, sin embargo, esto casi nunca ocurre. En la Fig. 9 se muestra con una línea azul la línea recta que resulta de los coeficientes estimados con el método de mínimos cuadrados, los círculos son las observaciones de *y*.

En este trabajo se calcularon los valores de la pendiente (tendencia) de cada uno de los pixeles de las imágenes que conforman la serie de tiempo que representa los índices de vegetación del estado de B.C.S. Esto fue posible utilizando las siguientes ecuaciones tomadas de Crawley (2013):

$$SSY = \sum (y - \bar{y})^2,$$
$$SSX = \sum (x - \bar{x})^2,$$
$$SSXY = \sum (y - \bar{y})(x - \bar{x})$$
$$b = \frac{SSXY}{SSX}$$

Donde *y* es el valor observado (cada uno de los 6,073,890 de pixeles de cada una de las 342 imágenes), $\bar{y}$ representa el promedio de *y* (hemos venido llamándolo el promedio histórico), *x* es la serie de tiempo (1 a 342), $\bar{x}$ representa el promedio de *x*, finalmente *b* es el valor de la pendiente.

2.9 Taxonomia de Flynn's

La taxonomía de Flynn's según Cook (2013):

> Es una clasificación para diferenciar arquitecturas de computación. Los diversos tipos son los siguientes:
>
> SIMD: Single instruction, multiple data (una traducción aproximada seria una sola instrucción, múltiples datos).
> MIMD: Multiple instructions, multiple data (una traducción aproximada seria multiples instrucciones, multiples datos).
> SISD: Single instruction, single data (una traducción aproximada sería una sola instrucción y un solo dato).
> MISD: Multiple instruction, single data (una traducción aproximada sería multiples instrucciones y un solo dato).

La mayoría de las personas estará familiarizada con el estándar de programación serial, seguido del modelo SISD. Esto es, que en el flujo de trabajo hay una sola instrucción con un único elemento de un dato en cualquier punto en el tiempo. Esto equivale, a un core de CPU habilitado para una sola tarea a la vez.

Los sistemas MIMD son los que nosotros vemos hoy en día en computadoras de escritorio dual-core o quad-core. Estos tienen un pool de trabajo de hilos/procesos que el sistema operativo gestionara para cada uno de los N cores del CPU.

Los sistemas SIMD tratan de simplificar el enfoque anterior, en particular con el modelo de paralelismo de datos. Ellos siguen un solo flujo de instrucción en cualquier punto del tiempo. De esta manera, requieren un conjunto único de la lógica al interior del dispositivo para decodificar y ejecutar el flujo de instrucciones, en lugar de decodificar las rutas de múltiples instrucciones.

Los sistemas MISD son aquellos donde muchas unidades de procesamiento desempeñan diferentes cálculos con el mismo dato de entrada. Hay varias arquitecturas que pueden ser clasificadas en este tipo. Entre las más populares están las arquitecturas Pipeline y las computadoras a prueba de fallos. Una arquitectura Pipeline es un sistema que toma un conjunto de datos y ejecuta una serie de operaciones en los datos. El procesador individual de una unidad de procesamiento de gráficos, cae dentro de esta categoría, pero la unidad de procesamiento de gráficos en si es un MIMD (Hamilton, 2013).

En caso de las GPU toman una ligera diferencia en el enfoque del SIMD. Es un modelo implementado por NDVIA llamado SIMT (single instruction, multiple thread) que revisaremos con mayor detalle en capítulos posteriores.

2.9.1 Granularidad de paralelismo

La granularidad del paralelismo según Bhujade (2009):

> El cómputo en paralelo enfatiza el uso de varios elementos de procesamiento (procesadores) con el principal objetivo de obtener velocidad en la realización de tareas con mucho costo computacional. El cómputo en paralelo enfatiza la explotación de la concurrencia disponible en un problema. La concurrencia de procesos de cómputo podría ser vista dentro de sistemas en paralelo en varios niveles (granularidad del paralelismo). La granularidad de paralelismo puede ser a través de la cantidad de trabajo (cálculos) que constituye a los elementos de procesamiento. Un sistema computacional puede tener varias granularidades coexistiendo. Las siguientes granularidades de paralelismo pueden ser fácilmente identificadas en los sistemas actuales.
>
> 1. Paralelismo a nivel de programa.
> 2. Paralelismo a nivel de tareas o procesos.
> 3. Paralelismo a nivel d grupo de sentencias.
> 4. Paralelismo a nivel de sentencias.
> 5. Paralelismo dentro de una sentencia.
> 6. Paralelismo a nivel de instrucción.
> 7. Paralelismo dentro de una instrucción.
> 8. Paralelismo a nivel de lógica y circuitos.
>
> Las granularidades arriba listadas son enumeradas en sentido de menor a mayor grado de finura. El nivel 1 es el nivel más grueso (paralelismo de grano grueso), mientras que el nivel 8 es el nivel más fino (paralelismo de grano fino).

Capítulo 3. Tecnologías involucradas

3.1 Linux Mint

Para realización de este trabajo se utilizó la versión 17 del sistema operativo Linux Mint, que tiene como propósito ser moderno, elegante y confortable, a la vez que potente y fácil de usar. Es considerado el tercer sistema operativo de distribución de escritorio más utilizado en el hogar, solo por detrás Microsoft Windows y Apple Mac OS (Linux Mint, s.f.).

Algunas de las razones del éxito de este sistema operativo son (Linux Mint, s.f.):

- Funciona desde el primer momento, con soporte multimedia completo y extremadamente fácil de usar.
- Es gratis y a la vez *open source.*
- Es comunitario. Los usuarios son alentados para enviar retroalimentaciones al proyecto para que sus ideas puedan ser utilizadas para mejorar Linux Mint.
- Basado en las distribuciones de Debian y Ubuntu, provee cerca de 30,000 paquetes y uno de los mejores Administradores de software.
- Es seguro y confiable. Gracias a un enfoque conservador de actualizaciones de software, un Administrador de actualizaciones único y la solidez de su arquitectura Linux, Linux Mint requiere muy poco mantenimiento (no hay regresiones, no hay ningún antivirus, anti-spyware, etc.).

3.1.1 Linux OOM Killer

Es una función nativa del sistema operativo que se dedica a seleccionar procesos para terminarlos (killer) cuando la memoria es consumida excesivamente, es conocido comunmente como “The Out-Of-memory (OOM) killer”, que en español seria algo aproximado a *Asesino de sin memoria*. El *OOM Killer* intenta escoger el mejor proceso para terminarlo con el fin de mejorar el agotamiento de memoria, donde el “mejor”

candidato es determinado por un rango de factores. Por ejemplo, cuanto mas memoria un proceso este consumiendo será más probable que sea candidato. Para asesinar un proceso seleccionado, el OOM killer proporciona una señal a SIGKILL. Pero es posible especificar una calificación (oom_adj file) a los procesos desde la versión del kernel 2.6.11 que puede influenciar la calificación (oom_score) de un proceso. Este archivo puede ser definido en cualquier valor en el rango de -16 a +15, donde valores negativos decrementan la posibilidad de ser asesinados y valores positivos la incrementan. Existe el valor especial -17 que exceptua al proceso por completo como candidato para ser seleccionado por el OOM killer (Kerrisk, 2010).

3.2 The R Project

Para este trabajo fue la utilizada la versión 3.2.1 de *The R Project*; el cual es comúnmente conocido solo como R, un lenguaje y entorno de computación y gráficos estadísticos muy utilizado por la comunidad científica con fines de investigación, fue lanzado en 1993 por Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Auckland (R Core team, 2015).

Entre las ventajas de R se puede contar la gran comunidad activa que lo utiliza, es muy fácil de instalar y usarse. Además, gracias a la extensa comunidad que lo emplea es altamente probable re-utilizar scripts que alguien más desarrolló sobre la misma necesidad que este requiriendo. Por ser *open source* permite codificar sus propias funciones o hacer contribuciones o mejoras a alguna de las más de 8000 librerías disponibles que existen al día que se escribió este documento. Otro de los puntos fuertes de R es la facilidad y lo bien diseñados que pueden ser producidos gráficos con calidad de publicación, incluidos símbolos y fórmulas matemáticas cuando sea necesario (R Core team, 2015).

R está disponible como software libre bajo los términos de la Licencia Pública General de *Free Software Foundation GNU General* en forma de código fuente. Se compila y se ejecuta en una amplia variedad de plataformas UNIX y sistemas similares (incluyendo FreeBSD y Linux), Windows y MacOS (R Core team, 2015).

3.2.1 Librería raster

La librería "raster" tiene por título: *Modelado y Análisis de Datos Geográficos* según su propio manual de referencia disponible en el repositorio de librerías de R (CRAN –The Comprehensive R Archive Network) y se describe como: "La lectura, la escritura, la manipulación, el análisis y modelado de datos espaciales cuadriculadas. El paquete implementa las funciones básicas y de alto nivel. El procesamiento de archivos de gran tamaño es compatible". Tiene compatibilidad con sistemas operativos Windows, Macintoch y Linux. La primera versión disponible fue la 1.0.0-1 el 20 en marzo del 2010 y la más reciente es 2.5-2 desde el 19 diciembre de 2015. Actualmente contiene más de 100 funciones disponibles (Hijmans, 2016).

3.2.2 Librería rgdal

La librería "rgdal" tiene por título: Enlaces para la Librería de Abstracción de Datos Geoespaciales (GDAL por sus siglas en inglés) según el manual de referencia y se describe: "Provee enlaces a la Librería de Abstracción de datos Geoespaciales de Frank Warmerdam (GDAL >= 1.6.3) y acceso a operaciones de transformación y proyección de la librería PROJ.4.". Tiene compatibilidad con Sistemas Operativos Windows, Macintoch y Linux. La primera versión disponible fue la 0.2-5 el 26 de noviembre 2003 y la más reciente es 1.1-10 desde el 12 mayo de 2016. Actualmente contiene más de 25 funciones o clases que la constituyen (Bivand & Rowlingson, 2016).

3.2.3 Librería gdalUtils

La librería "gdalUtils" tiene por título: "Wrappers" (envoltorio es la traducción aproximada) para utilerías de la Librería de Abstracción de datos Geoespaciales (GDAL), según su manual de referencia la versión más reciente es la 2.0.1.7 del 08 de octubre del 2015 y la primera versión fue la 0.2.0 el 08 de enero 2014, actualmente contiene más de 30 funciones o clases miembro. Tiene compatibilidad con sistemas operativos Windows, Macintoch y Linux (Greenberg & Mattiuzzi, 2014).

3.3 OpenCV

Es una librería de visión por computadora y software de aprendizaje por computadora lanzada en 1999. Para este trabajo fue la utilizada la versión 2.4.13., que cuenta con más de 2500 algoritmos optimizados. Se distribuye bajo licencia Berkeley Software Distribution (BSD por sus siglas en inglés) por lo tanto es considerada libre tanto para uso académico y comercial. Soporta interfaces para C ++, C, Python, Java, MATLAB y es compatible con los sistemas operativos: Windows, Linux, Mac OS, iOS y Android (OpenCV, s.f.).

OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. Fue escrito en C / C ++ optimizado, es muy utilizado y cuenta con una comunidad activa de más de 47 mil personas y tiene un número estimado de descargas superior a los 9 millones (OpenCV, s.f.).

Entre algunos de los usos y aplicaciones que se pueden encontrar con OpenCV (s.f.) son:

- Detección y reconocimiento de rostros.
- Identificación de objetos.
- Clasificar acciones humanas en videos.
- Registrar los movimientos de la cámara.
- Registrar el movimiento de objetos.
- Extraer modelos 3D de objetos.
- Encontrar imágenes similares de una base de datos de imágenes.
- Remover los ojos rojos de una imagen tomada usando flash.
- Seguir el movimiento de los ojos.

3.4 CUDA

Arquitectura Unificada de Dispositivos de Cómputo (CUDA por sus siglas en inglés), es una plataforma de cómputo paralelo y un modelo de programación propuesto por el fabricante de tarjetas de gráficos NVIDIA, fue desarrollado originalmente como una extensión del lenguaje C que permite que código para la GPU sea escrito en C regular (la

versión utilizada para este trabajo fue Toolkit 7.0). La plataforma de CUDA permite aumentos impresionantes en el rendimiento de los cálculos al aprovechar la potencia de la unidad de procesamiento de gráficos (GPU por sus siglas en inglés) (CUDA NVIDIA, s.f.).

NVIDIA sabía que un hardware impresionantemente rápido, tenía que combinarse con herramientas de hardware y software intuitivas, e invitó a Ian Buck a unirse a la compañía y empezar a hacer evolucionar una solución que ejecutara C a la perfección en la GPU. Al reunir el software y el hardware, NVIDIA lanzó CUDA en 2006, la primera solución del mundo para computación general en las GPU (CUDA NVIDIA, s.f.).

Con millones de dispositivos GPU habilitados con soporte para CUDA vendidos a la fecha, desarrolladores de software, cientificos e investigadores encontraron un amplio rango de usos para la computación de GPU con CUDA, enseguida algunos ejemplos (CUDA NVIDIA, s.f.):

- Identificar la capa oculta de las arterias (medicina): los ataques al corazón son la principal causa de muerte alrededor del mundo. Ingenieros de Harvard, la escuela de medicina de Harvard, y el Brigham & Women`s Hospital, formaron una agrupación de trabajo que utiliza GPUs para simular el flujo de la sangre e identificar la capa oculta de las arterias sin técnicas invasivas de imagenología o cirugías invasivas (CUDA NVIDIA, s.f.).

- Análisis del flujo de tráfico aéreo (aviación): El Sistema del Espacio Aéreo Nacional (NAS por sus siglas en inglés) de E.U.A., administra la coordinación del flujo de tráfico aéreo a nivel nacional. Modelos por computadora ayudan a identificar nuevas maneras para aliviar el congestionamiento y mantener el tráfico de los aviones en movimiento eficientemente. Utilizando el poder computacional de las GPUs, un equipo en la NASA obtuvo una gran ganancia de desempeño, reduciendo el tiempo de análisis de 10 minutos a 3 segundos (CUDA NVIDIA, s.f.).

- Visualización de moléculas (bioquímica): una simulación molecular llamada NAMD (nanoscale molecular dynimics en inglés) obtuvo un gran desempeño impulsada por GPUs. La aceleración es resultado de la arquitectura en paralelo de GPUs, que permitió a los desarrolladores de NAMD utilizar CUDA en porciones de la aplicación de cálculos intensivos (CUDA NVIDIA, s.f.).

La arquitectura propia de la GPU está constituida por un mayor número de Unidades Aritmético Lógicas (ALU por sus siglas en inglés) respecto a las Unidades de Procesamiento Central (CPU por sus siglas en inglés) esto se ilustra en la Fig. 10, por lo anterior es lógico pensar que los tiempos de procesamiento de tareas que involucren cálculos intensivos se reducen significativamente. Esto es posible siempre y cuando las tareas a ejecutar soporten paralelización y la complejidad del cálculo a efectuarse sea importante (CUDA Toolkit, s.f. a).

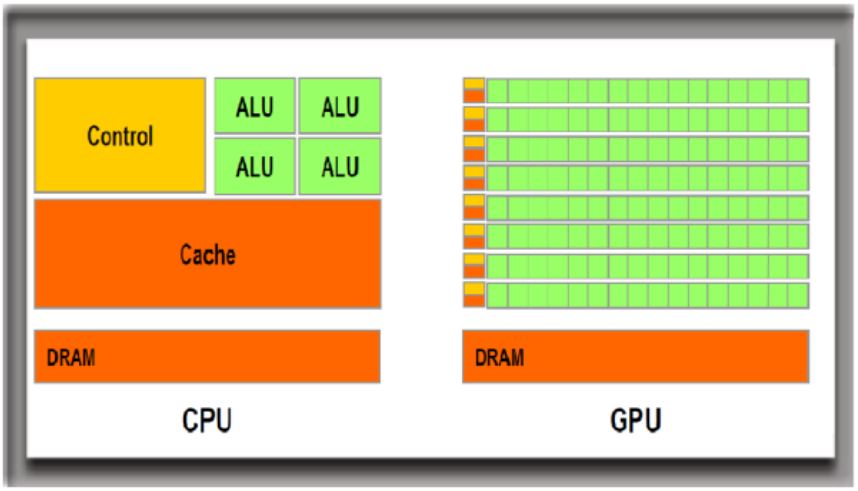

Figura 10. Esquema comparativo de la arquitectura de una CPU y GPU, Fuente: Reimpresa de, 1. Introduction, En CUDA Toolkit, s.f. a, Recuperado 21 oct 2016 de, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction, Copyright © 2007-2016 NVIDIA Corporation.

3.4.1 Modelo de programación CUDA

CUDA es ideal para problemas embarazosamente paralelos, porque bajo el principio "divide y vencerás" consigue que problemas sumamente complejos, llevarlos a su mínima expresión y ahí tratarlos con los beneficios del paralelismo. La división de los problemas es

posible empatarla a la conceptualización o jerarquización que soporta en su modelo de programación, básicamente hay que considerar los siguientes conceptos importantes: *Host*, *Device*, *grids of blocks*, *block of threads*, *threads y kernel* (una traducción aproximada seria: anfitrión, el dispositivo, cuadricula de bloques, bloques de hilos, hilos y semillas). En la Fig. 11 es posible apreciar con más claridad la interrelación de dichos conceptos (en apartados siguientes se explicarán a detalle cada unos de ellos) (Cook, 2013).

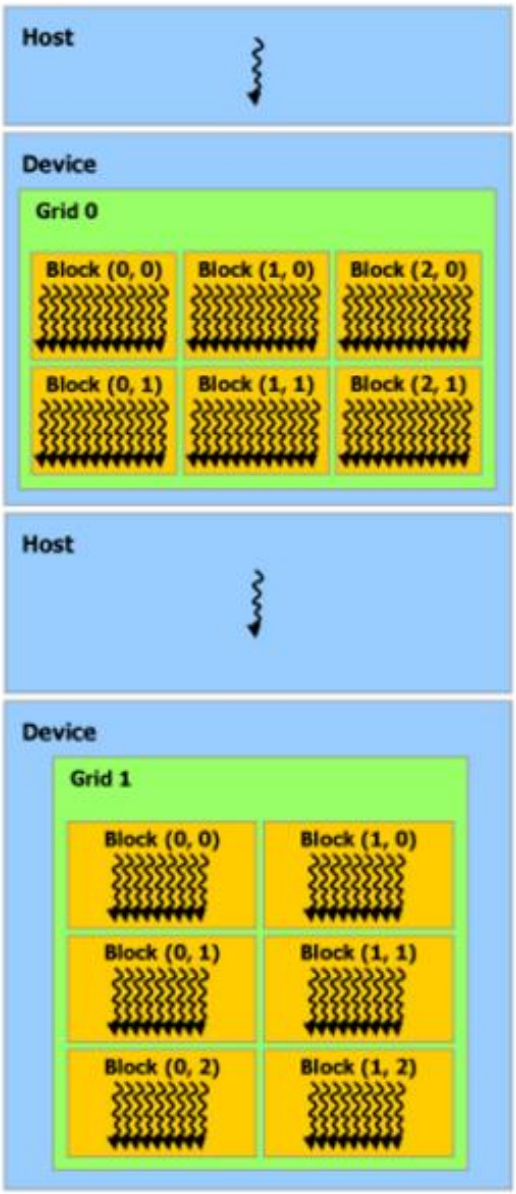

Figura 11. Esquema de programa de CUDA, Fuente: Adaptada de 2.4. Heterogeneous Programming, En CUDA Toolkit, s.f. a, Recuperado 21 octubre 2016 de, *http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#heterogeneous-programming*, Copyright © 2007-2016 NVIDIA Corporation.

3.4.1.1 Host & Device

El modelo de programación de CUDA permite ejecutar aplicaciones en sistemas heterogéneos, gracias a una sencilla anotación de código con un pequeño conjunto de extensiones para el lenguaje de programación C. Un ambiente heterogéneo consiste de CPUs complementadas por GPUs, cada cual con su propio apartado de memoria conectadas PCI Express bus. Por lo tanto, es importante recordar la siguiente distinción (Cheng, Grossman & Mckercher, 2014):

- *Host*: la CPU y su memoria (*Host* memory).
- *Device*: la GPU y su memoria (*Device* memory).

El código del *Host* se ejecuta en el CPU y el código del *Device* se ejecuta en la GPU. Una aplicación ejecutándose en la plataforma heterogénea es típicamente inicializada por la CPU. El código del CPU es responsable de administrar el ambiente, el código, los datos para el *Device* antes de cargar tareas intensivas de cálculo en el *Device* (*kernels* que veremos en el apartado 3.4.1.5) (Cheng *et al*., 2014).

3.4.1.2 Grid of blocks

Un grid es el máximo nivel de abstracción en la organización de los hilos en el *Device*. El grid representa un conjunto de bloques que contienen hilos. También pueden ser pensados como una fila (o grid) de procesos (blocks), pero sin comunicación entre sus procesos (Cook, 2013).

Los grids pueden declararse de máximo tres dimensiones (x, y, z). Dependiendo de la versión de la arquitectura de la tarjeta utilizada, es posible realizar ejecuciones concurrentes de distintos kernels (valorando la cantidad de recursos que consumirán y la disponibilidad en el *Device*). La ejecución de un kernel es asociada a los recursos gestionados de un solo grid, es decir un grid por kernel (Fig. 12) (CUDA Toolkit, s.f. a).

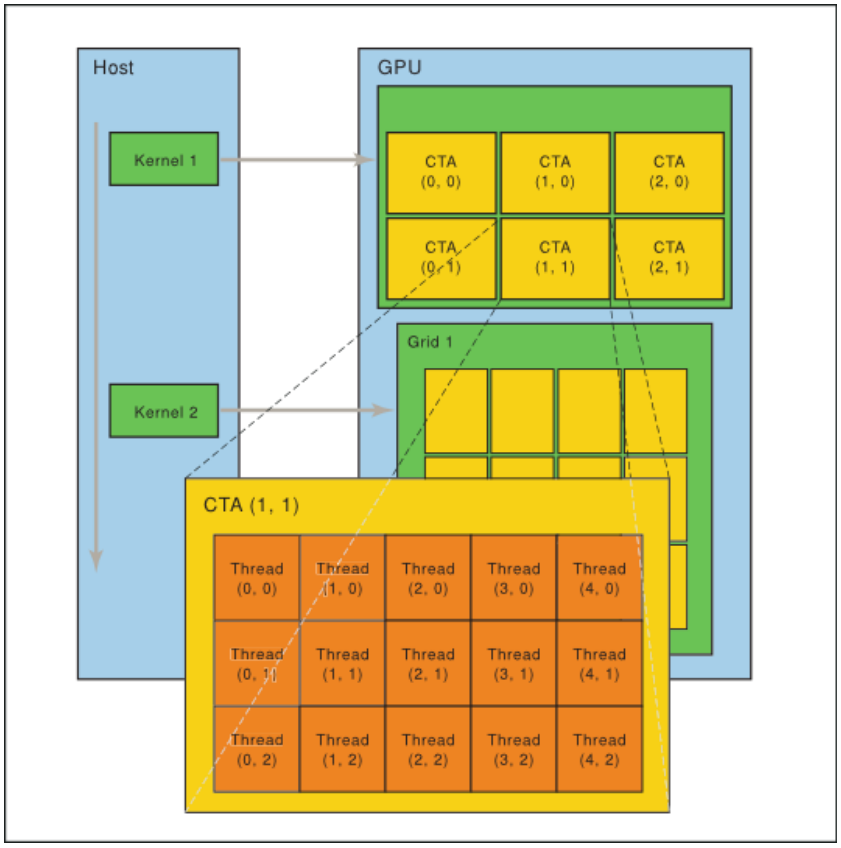

Figura 12. Esquema de ejecución de múltiples kernels, Fuente: Adaptada de 2.4. Heterogeneous Programming, En CUDA Toolkit, s.f. a, Recuperado 21 octubre 2016 de, http://docs.nvidia.com/cuda/parallel-thread-execution/#grid-of-cooperative-thread-arrays , Copyright © 2007-2016 NVIDIA Corporation.

3.4.1.3 Block of threads

Un bloque es un conjunto de hilos que están ejecutando concurrentemente el mismo programa (kernel) y pueden cooperar entre ellos para calcular un resultado en conjunto (Patterson & Hennessy, 2014).

Dentro de cada bloque los hilos son operados cooperativamente en lotes llamados *warps* (abordaremos en el siguiente apartado en qué consisten). Los bloques pueden definir como arreglos de máximo tres dimensiones (x, y, z), y tienen un máximo número de hilos que soportan; estas características van en función de la versión de la arquitectura de la tarjeta que se esté utilizando. Para finalizar es importante mencionar que los bloques de hilos utilizan la memoria compartida, y es por eso que todos los hilos dentro de un mismo bloque pueden colaborar para cálculos en conjunto. Los bloques son asignados para su

procesamiento a nivel de hardaware en la GPU a los Steaming multiprocessors (SMs) de acuerdo a la disponibilidad que exista, pero solo bloques completos (CUDA Toolkit, s.f. a).

3.4.1.4 Threads

Los hilos son los últimos en la jerarquía de organización para ejecutar kernels, estos son los que finalmente generan las múltiples instancias de un kernel que será lanzado, tantas veces como hilos hayan sido gestionados. Los hilos tienen una memoria local y registros para cargar/almacenar datos de los cálculos que deben efectuar. Además, también tienen acceso a la memoria global, texturizada y constante (Fig. 13) (CUDA Toolkit, s.f. a).

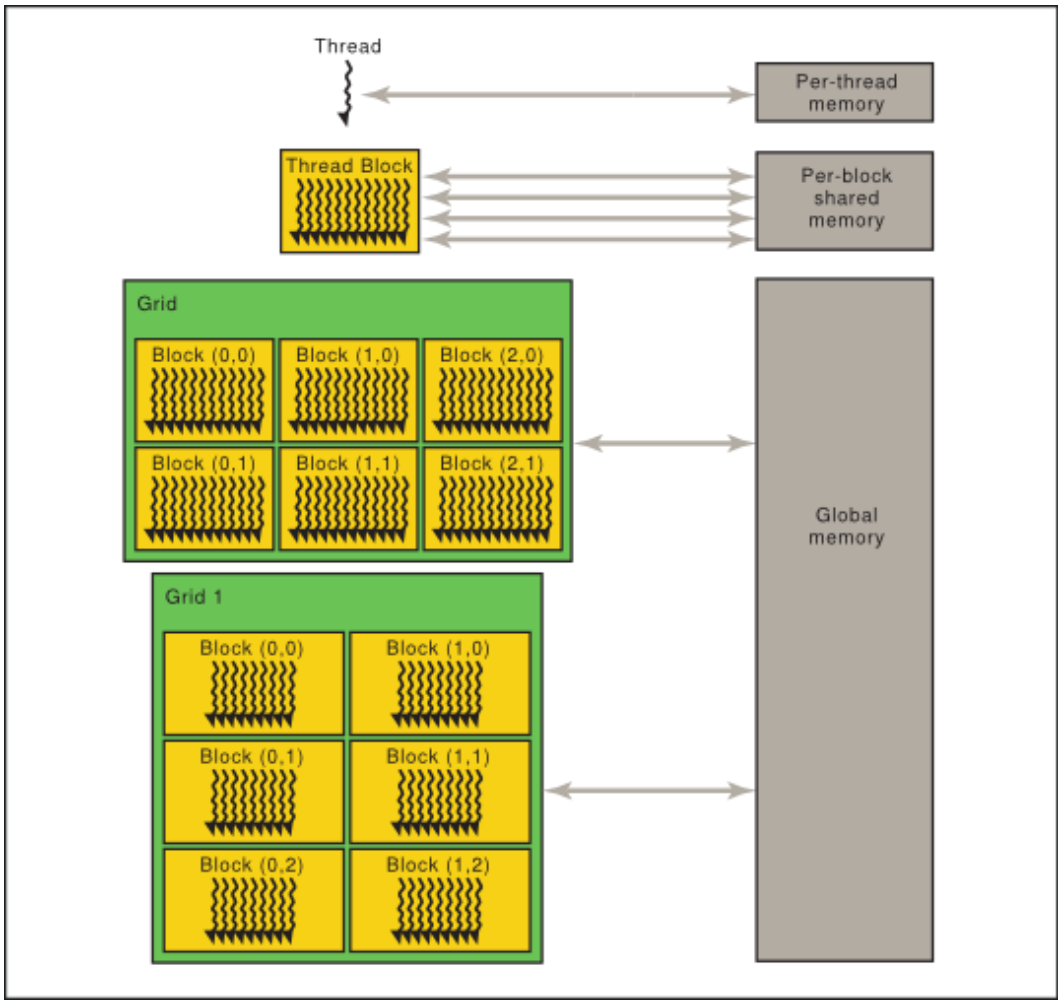

Figura 13. Esquema de la jerarquía de la memoria en el *Device*, Fuente: Adaptada de 2.3 Memory Hierachi, En CUDA Toolkit, s.f. a, Recuperado 21 octubre 2016 de, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#memory-hierarchy, Copyright © 2007-2016 NVIDIA Corporation

CTA: Cooperative thread array.

Por último, los warps que explicaremos en el apartado 3.4.2, son grupos o lotes de 32 hilos (lo mismo para todas las arquitecturas de CUDA existentes). El warp es la unidad de medida utilizada al momento de procesar los hilos dentro de la GPU a nivel de hardware (CUDA Toolkit, s.f. a).

3.4.1.5 kernel

CUDA es una extensión del lenguaje C, por lo que le permite al programador definir funciones de C, pero llamadas *kernel*, que cuando las llamamos, son ejecutadas N veces en paralelos por N diferentes CUDA threads que sería lo opuesto para funciones regulares de C. En resumen, un *kernel* viene a ser solo el nombre de una función que se ejecuta en la GPU (*Device*) (CUDA Toolkit, s.f. a).

Un kernel es definido usando la declaración especifica __global__ (es doble guion bajo antes y despues) y el número de CUDA threads que ejecutarán el kernel. A cada hilo que ejecute el kernel le corresponde un ID thread (`threadIdx.x`) único que solo es accesible desde dentro del kernel (CUDA Toolkit, s.f. a).

Enseguida dos ejemplos de la declaración de un kernel y su invocación, uno más sencillo que el otro. El primero es del clásico "Hello World", y el otro solo ligeramente más elaborado, realiza la suma de 2 vectores A y B, guardando el resultado en un tercer vector C (CUDA Toolkit, s.f. a).

Ejemplo 1 (Sanders & Kandrot, 2010):

```
__global__ void kernel (void) {
}

int  main(void){
kernel<<1,1>>();
printf("Hello, World! \n");
return 0;
}
```

Ejemplo 2 (CUDA Toolkit, s.f. a):

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

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

3.4.2 SIMT

De acuerdo a la documentación disponible desde el sitio web oficial de CUDA Toolkit (s.f.), se describe a la implementación del hardware (de la GPU) y la arquitectura SIMT de la siguiente manera:

> La arquitectura NVIDIA GPU es construida alrededor de un arreglo escalable de multihilos Streaming Multiprocessors (SMs). Cuando un programa de CUDA en el *Host* invoca un grid de kernels, los bloques del grid son enumerados y distribuidos entre los SMs de acuerdo a la dispobilidad de la capacidad de ejecución. Los hilos de un bloque se ejecutan concurrentemente en un SM, y multiples bloques de hilos pueden ser ejecutados concurrentemente en un SM. Cuando un bloque de hilos termina, nuevos bloques son lanzados en los SMs vacantes (CUDA Toolkit, s.f. a).
>
> Un SM es diseñado para ejecutar miles de hilos concurrentemente. Para administrar tal cantidad de hilos, emplea una arquitectura única llamada SIMT (Single Instruction Multiple Thread). Las instrucciones son canalizadas para potenciar el paralelismo a nivel de instrucción dentro de un solo hilo, así como el paralelismo a nivel de hilo simultáneamente a través de hardware multithreading. A diferencia de los núcleos de la CPU son publicados en orden, sin embargo, no hay predicción de bifurcaciones ni especulación de ejecuciones (CUDA Toolkit, s.f. a).

Dentro de la arquitectura SIMT, los SMs crean, administran y programan, y ejecutan hilos en grupos de 32 hilos paralelos llamados warps. Hilos individuales que componen un warp inician juntos en la misma dirección del programa, pero tienen sus propias instrucciones de contador de direcciones y registros de estado, por lo tanto, son libres de diversificarse y ejecutarse independientemente. El termino warp es originario de *weaving* (tejido en inglés), la primera tecnología de hilos en paralelo. El termino half-warp (mitad de warp) puede ser ya sea la primera o la segunda mitad del warp. Y el termino quarter-warp (un cuarto de warp) puede ser el primer, segundo, tercero o la cuarta parte del warp (CUDA Toolkit, s.f. a).

Cuando a un SM se le proporciona uno o más bloques de hilos para ejecutar, estos son particionados dentro de warps y cada warp es programado por el *warp sheduler* (programador de warps) para su ejecución. La manera en que un warp es particionado en warps es siempre la misma; cada warp contiene hilos consecutivos, incrementándose los Id (identificador) de los hilos con el primer warp conteniendo el hilo 0. La jerarquía de hilos es descrita como Id de los hilos relacionado con los índices en el bloque (CUDA Toolkit, s.f. a).

Un warp ejecuta una instrucción común a la vez, asi la eficiencia es completada cuando los 32 hilos de un warp acuerdan su ruta de ejecución. Si los hilos de un warp divergen por una bifurcación condicional de datos-dependientes, el warp serializa las ejecuciónes de las distintas rutas tomadas, deshabilitando hilos que no estén en esa ruta, y cuando todas las rutas se completan, los hilos convergen de nuevo en la misma ruta de ejecución. Las bifurcaciones ocurren solo dentro del warp; diferentes warps se ejecutan independientemente con independencia de que se esten ejecutando rutas de código comunes o disjuntas (CUDA Toolkit, s.f. a).

Para finalizar, la arquitectura SIMT es un tipo de organización de vectores SIMD (single instruction multiple data, arquitectura que vimos en capítulos anteriores) en que una sola instrucción controla multiples elementos de procesamiento (CUDA Toolkit, s.f. a).

3.4.3 CUDA cores

En junio 2008, NDVIA introdujo una importante revisión a la arquitectura G80. La segunda generación de arquitectura unificada (GT200) incrementando el número de *streaming processor cores*, los cuales posteriormente fueron llamamos CUDA cores (Architecture Fermi, 2009).

Los cuda cores son el equivalente a las unidades aritmético lógicas (ALU por sus siglas en inglés) en la CPU. Los cuda cores cuentan con una unidad para operaciones de punto flotante (FP unit por sus siglas en inglés) y una unidad de operaciones enteras (INIT unit por sus siglas en inglés) como se aprecia en la Fig. 14. De ahí podemos vislumbrar la gran capacidad de cálculo que brindan las GPU, debido a la especialización de sus componentes, y además a la gran cantidad de cuda cores presentes (Architecture Fermi, 2009).

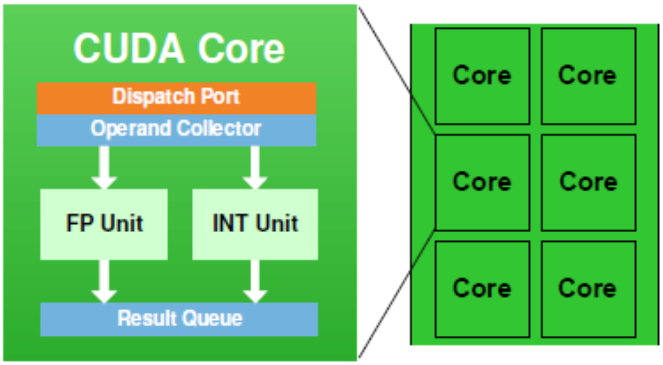

Figura 14. Esquema del contenido dentro de un CUDA core, Fuente: Adaptada de, Arquitecture Fermi, En NVIDIA, Recuperado 21 octubre 2016 de, http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf , Copyright © 2009 NVIDIA Corporation

Los cuda cores se localizan dentro de los SMs (streaming multiprocessors), y dependiendo de la versión (compute capabilitie en inglés) de la arquitectura de la GPU, se han organizado en grupos de 16 o 32 cuda cores, al menos en las últimas arquitecturas así fue. En la Tabla 4 se pueden ver las distintas generaciones de las arquitecturas GPU NVIDIA que hay hasta el momento. Para este trabajo la tarjeta utilizada pertenece a la arquitectura 5.2 (Architecture Fermi, 2009; Harris, 2014).

Tabla 4
Compute capabilities de la arquitectura GPU NVIDIA

Versiones					
2.X	3.0	3.2	3.5, 3.7, 5.0, 5.2	5.3	6.X
Fermi	Kepler		Maxwell		Pascal

3.4.4 NVCC

Es el compilador de NVIDIA para generar los ejecutables de los programas en CUDA. Su propósito es ocultar los intrincados detalles de la compilación a los desarrolladores. En las fases de compilación el código del *Host* es atendido por compiladores como gcc ó g++, para ambientes en Linux o Windows respectivamente. Y el código del *Device* en el compilador de CUDA (CUDA Toolkit, s.f. a).

.

Las instrucciones que permiten compilar el código de CUDA/OpenCV (pkg-config, s.f.):

`nvcc `pkg-config --cflags opencv` `pkg-config --libs opencv`** *nombre_programa.cu* **`-o` *nombre_ejecutable*

Donde:

`nvcc` Hace la llamada al compilador de NVIDIA para CUDA.

`pkg-config` Es una utilería de software libre que ayuda a recuperar información de librerías instaladas y requeridas al momento de compilar un programa, las cuales se utilizan durante la fase de enlace en la compilación.

`--cflags` Imprime las banderas del pre-procesador y compilador necesarios para compilar en línea de comandos, incluyendo las banderas de todas sus dependencias.

`--libs` Esta opción es idéntica a la anterior, solo que imprime las banderas de la fase de enlace.

`-o` Es la opción abreviada de **`–ouput-file`** *nombre_archivo*, sirve para especificar el nombre y la ubicación del archivo de salida.

3.4.5 Tarjeta NVIDIA GeForce GTX TITAN X

La tarjeta utilizada para este trabajo fue una NVIDIA GeForce GTX TITAN X de la arquitectura Maxwell 2da generación (Fig.15). En la Tabla 5 podemos ver sus características principales. La información técnica o especificaciones de cualquier tarjeta NDVIA está disponible desde la utilería `Device Query` (GeForce, s.f.).

Tabla 5
Tarjeta NVIDIA GeForce GTX TITAN X

Especificaciones técnicas	
Multiprocesadores	24
Núcleos CUDA (128 nucleos por Multiprocesador)	3072
Memoria global	12 GB
Máximo número de hilos residentes por multiprocesador	2048
Máximo número de hilos por bloques	1024
Máximo número de *warps* residentes por Multiprocesador	64

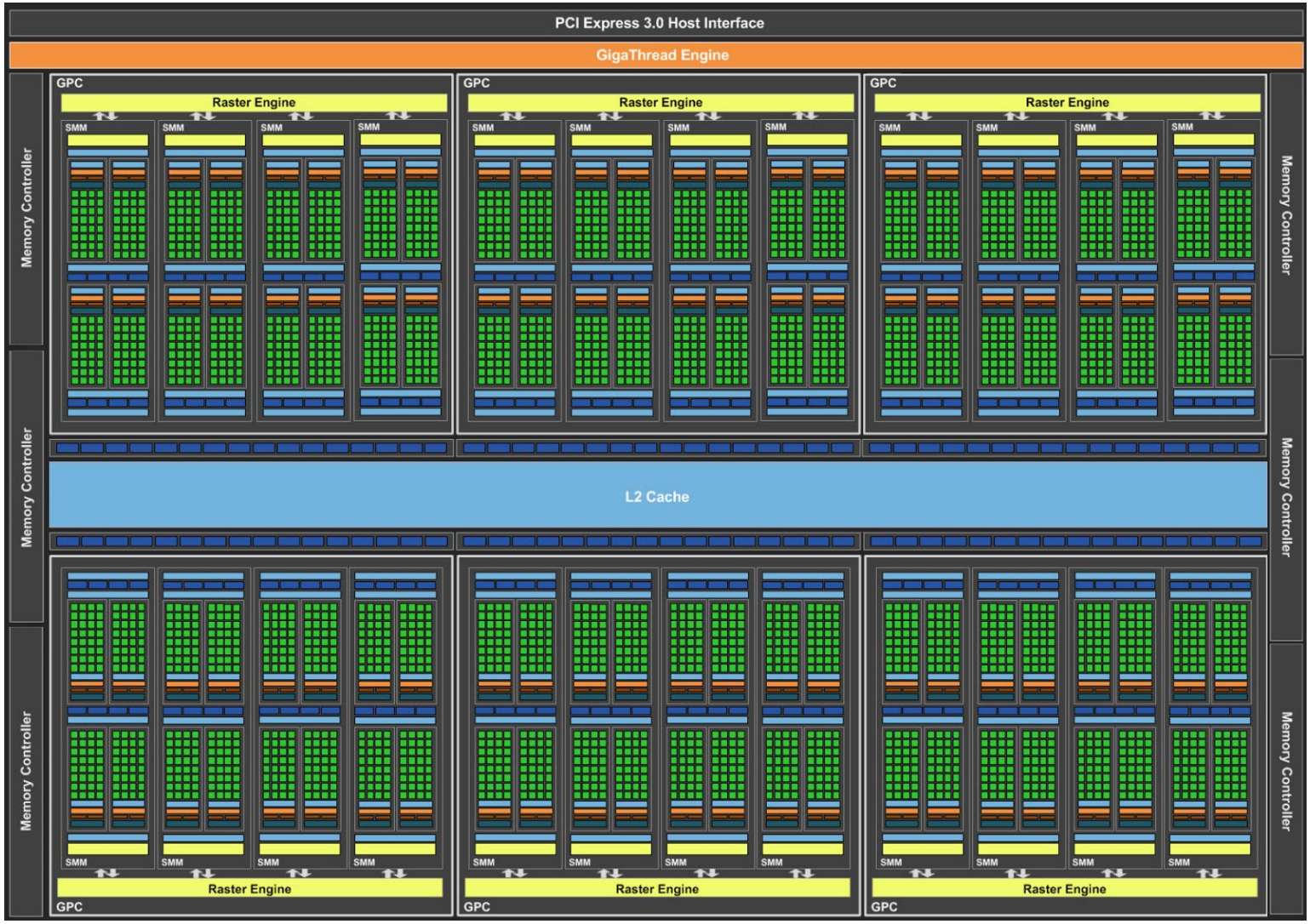

Figura 15. Esquema de la arquitectura Maxwell 2da generación (SMM, Streaming Multiprocessors Maswell), Fuente: Reimpresa de Blog Bigdata by Kenji, En Blogger, s.f., Recuperada 21 octubre 2016 de http://databigdata.blogspot.mx/2016/02/cuda-devicequery.html

Capítulo 4. Metodología

4.1 Fases

La metodología y/o actividades realizadas durante el desarrollo de este trabajo fueron extensas como en todo nuevo proyecto que se inicia, sin embargo, se intentará sintetizar de manera clara y puntual las actividades y/o fases que permitieron llegar a la solución final de esta investigación (el algoritmo en CUDA). En la Fig. 16 se muestra una abstracción que describe las actividades que se realizaron a lo largo de todo el proceso de trabajo.

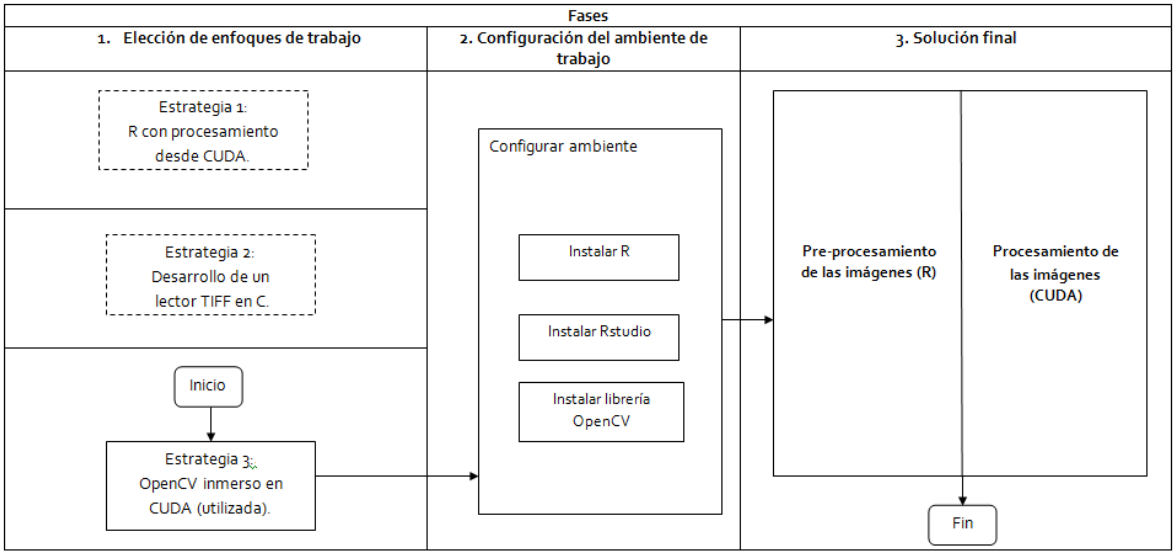

Figura 16. Fases del proceso de desarrollo del presente trabajo de investigación.

4.2 Elección del enfoque de trabajo

En el camino hacia el objetivo de este trabajo (el algoritmo para el análisis y procesamiento de los índices de vegetación MODIS) se abordaron y desarrollaron diversas estrategias o enfoques antes de llegar a la solución final. Los caminos previos condujeron a: utilizar R con procesamiento desde CUDA, desarrollar un lector de TIF en C, y concluyeron con la integración de OpenCV en el ambiente de trabajo con CUDA, que finalmente fue la estrategia elegida como punto de partida.

En los apartados siguientes conoceremos las estrategias estudiadas durante esta fase experimental de la elección del enfoque de trabajo, que si bien en su momento fueron implementadas y probadas, finalmente fueron descartadas (en los apartados correspondientes se expondrán las causas circunstanciales de su eliminación). Pero es importante mencionarlas desde la perspectiva del abanico de opciones tecnológicas que existen hoy en día, y pudieran servir como alternativas para otras investigaciones. Por lo anterior, únicamente se plantean con el objetivo de conocerlas. Los ejemplos que se presentarán son funcionales, pero para mayores dudas o ejemplos más complejos deberán referirse a la bibliografía consultada.

Enseguida, las características de la computadora utilizada en la implementación de las estrategias antes mencionadas, incluida la estrategia definitiva por la que se optó (Tabla 6).

Tabla 6
Especificaciones del PC utilizado

	Especificaciones
Procesador	Intel i7 (3.60 GHz)
Memoria RAM	GTX Titan X (12 GB)
Disco duro	Kingston (240GB) estado sólido
Tarjeta madre	Gigabyte Technology Co.

4.2.1 Estrategia 1: R con procesamiemto desde CUDA

En la búsqueda de opciones para procesar las imágenes del NDVI en CUDA, se planteó la idea de aprovechar que las imágenes ya estaban en memoria durante su pre-procesamiento en R (que se explicará a detalle en el apartado 4.4.1), y aprovechar esta situación para hacer el envío de los datos a CUDA. Investigando al respecto se encontró que R soporta interfaces (comunicación) con CUDA, prueba de ello son librerías de R que cuentan con funciones mejoradas (aceleradas) a través CUDA. Una vez confirmado que era posible la comunicación entre las dos plataformas, es decir la invocación de cáculos de funciones en R, pero realmente ejecutándose en CUDA (de manera invisible al usuario final).

La fuente consultada, señala que la comunicación de R con CUDA es posible a través de funciones denominadas *wrapper* (envoltorio), básicamente lo que hacen estas funciones es encapsular el procedimiento o llamada del código de CUDA. El proceso para lograr la comunicación de R y CUDA se expresa en la Fig. 17 (Peng & Leeuw, 2002; Zhao, 2014).

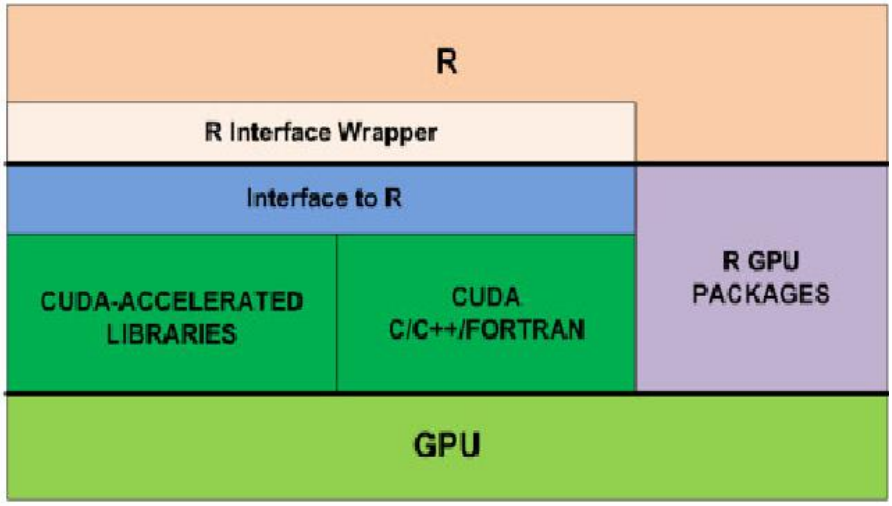

Figura 17. Esquema de comunicación de R con CUDA, Fuente: Adaptada de Accelarate R Applications with CUDA, En Parallel Forall, 2014, Recuperado 21 octubre 2016 de https://devblogs.nvidia.com/parallelforall/accelerate-r-applications-cuda/, Copyright © 2016 NVIDIA Corporation.

Para lograr la comunicación de R y CUDA, se hicieron pruebas a través de una sencilla función *wrapper* que enviaba un vector desde R hacia CUDA, y se le incrementaba o sumaba una unidad (1) al valor de cada elemento del vector en CUDA, y se devolvía resultado para desplegarse en consola de R (Fig. 36).

Enseguida se presentan la serie de pasos que seguimos para realizar el ejemplo antes descrito (suma de vectores). Todas las instrucciones fueron ejecutadas desde la consola de Linux:

Paso 1,
Compilar y crear el empaquetamiento del código de CUDA en conjunto con librerías de R (archivo extensión `.so` – *Shared Object*):

```
nvcc -g -G -I/usr/local/cuda/include -Xcompiler "-
I/usr/share/R/include -fpic" -c sumaVectorGPU.cu -o
sumaVectorGPU.o
nvcc -shared -lm sumaVectorGPU.o -o sumaVectorGPU.so
```

Paso 2,
Copiar el empaquetado (`sumaVectorGPU.so`) al directorio trabajo en R:

```
cp sumaVectorGPU.so /home/posgrado/R/
```

Paso 3,
Crear la función *wrapper* en R (las instrucciones de la creación de la función *wrapper* que invocará el código de CUDA se puede consultar en el Apéndice.

Paso 4,
Crear el vector de prueba que será enviado (desde la consola de R):

```
V<-c(0.2303, 0.2367, 0.2029,0.2040,0.2188,0.2209,0.2350,0.2304)
```

Paso 5,
Invocar la función wrapper (desde la consola de R), en nuestro caso la llamamos *ndviGPU*.
Ejemplo de invocación desde la consola de R:

```
ndviGPU(V)
```

En la Fig. 18 se puede ver un ejemplo del resultado en consola de la llamada a la función *wrapper* (**ndviGPU**) de este ejemplo.

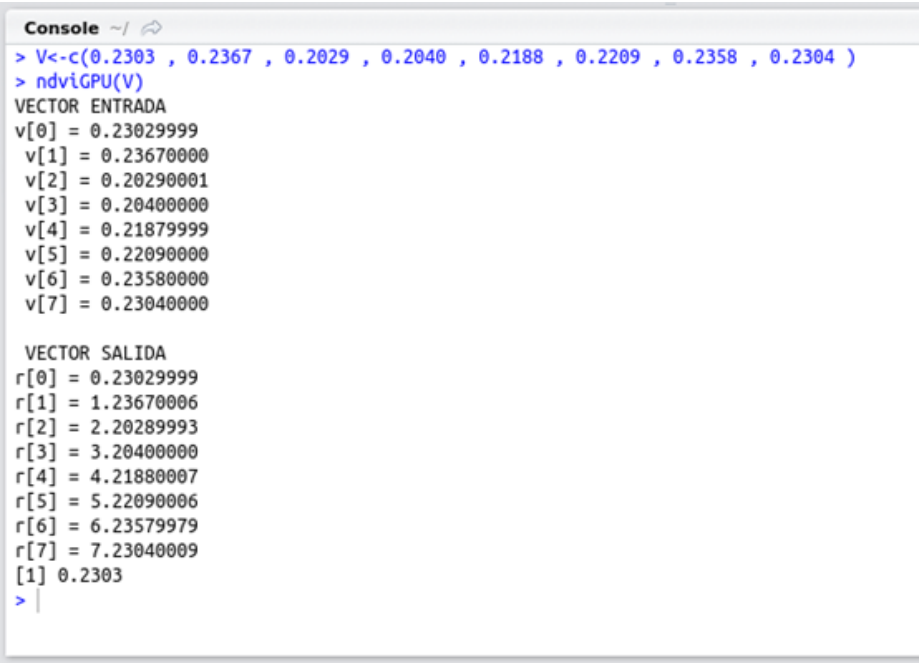

Figura 18. Consola de R de la función *wrapper* invocando CUDA.

Sin embargo, a pesar de haber tenido éxito en las pruebas de comunicación de R con CUDA (a través de una sencilla operación con vectores), este enfoque de trabajo fue dejado de lado, debido a que durante las pruebas del envío de los datos reales (las imágenes del NDVI) hacia el código en CUDA, generaba errores que en su momento fueron analizados, y después de un tiempo prudente se replanteó el enfoque de trabajo, lo que llevó a idear la propuesta que vera en el apartado 4.2.2.

4.2.2 Estrategia 2: Desarrollo de un lector TIFF en C

El desarrollo del lector TIF surgió de la necesidad de leer las imágenes del NDVI directo en el código de CUDA. Y debido a la similitud de C y CUDA, se optó por codificarlo en C y una vez terminado migrarlo a CUDA. Pero como vimos en capítulos pasados (dentro del Marco Teórico, apartado 2.7.1 Geotiff), las características del formato (GeoTIF) de las imágenes del NDVI es complejo, por lo cual era importante comprenderlo y tener la facilidad de acceder a sus distintas propiedades (Tags), que era crucial para el diseño de la estrategia del envío de los datos entre el *Host* y el *Device*.

Una vez terminada la codificación del lector TIF, se procedió a hacer pruebas lectura con las imágenes reales del NDVI. Aunque el lector fue capaz de reconocer el contenido de las imágenes en su totalidad, a raíz de esto, nos percatamos que los datos (valores de los pixeles del NDVI) de las imágenes venían codificados. Cuando se estudió la especificación del formato TIFF aprendimos que los datos pueden o no venir codificados (opciones: sin comprensión, RLE, LZW, CCITT Group 3 y Group 4, JPEG). En el caso de las imágenes del NDVI utilizaban el algoritmo de compresión sin pérdida LZW. En este punto, nuevamente tuvimos que valorar las implicaciones de continuar adelante con esta estrategia (desarrollo del lector), que imponían una importante curva de aprendizaje del funcionamiento del algoritmo de encriptado.

Por lo anterior se optó por hacer a un lado esta propuesta, y retomar la búsqueda de una nueva estrategia para poder leer las imágenes en CUDA. Fue así como, en esta ocasión consultando con expertos del campo de procesamiento de imágenes, sugirieron la inclusión de OpenCV en el ambiente de trabajo.

Finalmente, cabe precisar que el lector desarrollado quedó funcional únicamente para el reconocimiento en su totalidad de la estructura (Tags) de imágenes TIF.

4.2.3 Estrategia 3: OpenCV.

La integración de la librería de OpenCV se planteó como recomendación de especialistas en el área de procesamiento de imágenes, durante el proceso de búsqueda de nuevas alternativas (que nos dejó la estrategia 2) para lograr leer las imágenes del NDVI.

La elección de OpenCV como estrategia quedó confirmada con una sencilla prueba de leer una única imagen (realizada desde QT – es un framework para desarrollos). Una vez demostrado el soporte del formato (GeoTIFF) de las imágenes a través de OpenCV, y consultado la compatibilidad de interfaces con CUDA. Por fin era claro el camino a seguir.

En los apartados siguientes, se presentará toda la investigación realizada para lograr la integración de OpenCV en el ambiente de trabajo (instalación de OpenCV, configurar la compatibilidad con CUDA, compilar CUDA con OpenCV, etc.).

4.3 Configuración del ambiente de trabajo

La configuración del ambiente trabajo definitivo (ya que hubo otras propuestas como veremos en el Capítulo de Resultados) consistió desde la implementación de la librería OpenCV 2.4.13, sin pasar por alto R 3.2.1 y el IDE (Integrated Development Enviroment) de Rstudio 0.98.110.

Las guías o asistentes utilizados en su mayoría fueron los disponibles desde los sitios de descarga oficiales. Más adelante en capítulos posteriores se describen la serie de pasos que se siguieron, específicamente para el ambiente de trabajo de esta investigación.

Los pasos se describen divididos por cada una de las tecnologías antes mencionadas, así como configuraciones particulares que se detectaron sobre la marcha al realizar las implementaciones.

4.3.1 Instalación de R Project 3.2.1

Enseguida se describe una serie de pasos para la instalación de R, en función del sistema operativo con que se trabajó, Linux Mint.

Paso 1,

En la página oficial de R (https://www.r-project.org/), se debe ubicar el link para descargas (Fig. 19), el cual a su vez redirecciona al *The Comprehensive R Archive Network* (CRAN por sus siglas en inglés).

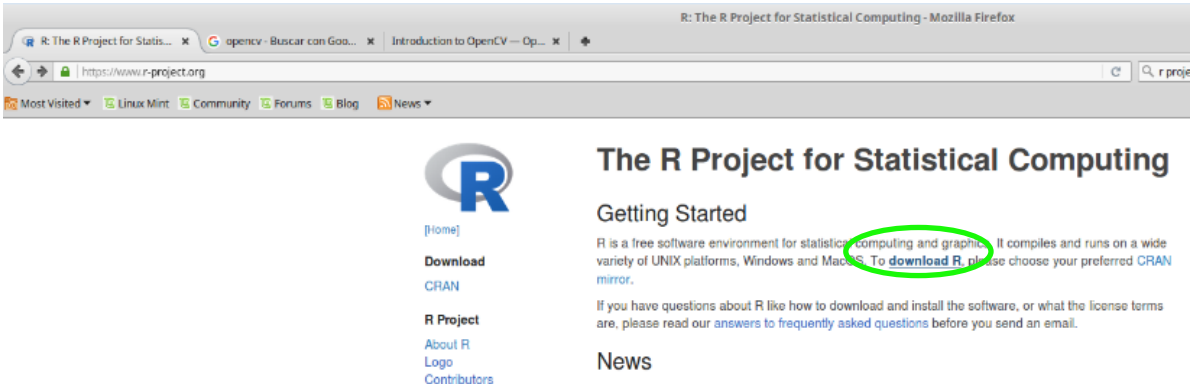

Figura 19. Página web de bienvenida de R,

Paso 2,

Ya en la página del CRAN (https://cran.r-project.org/mirrors.html) que una traducción aproximada sería "Red Integral de Archivos de R", viene a ser como el repositorio de las versiones disponibles por países como se aprecia en la Fig. 20, los cuales a su vez forman parte de la comunidad del proyecto de R.

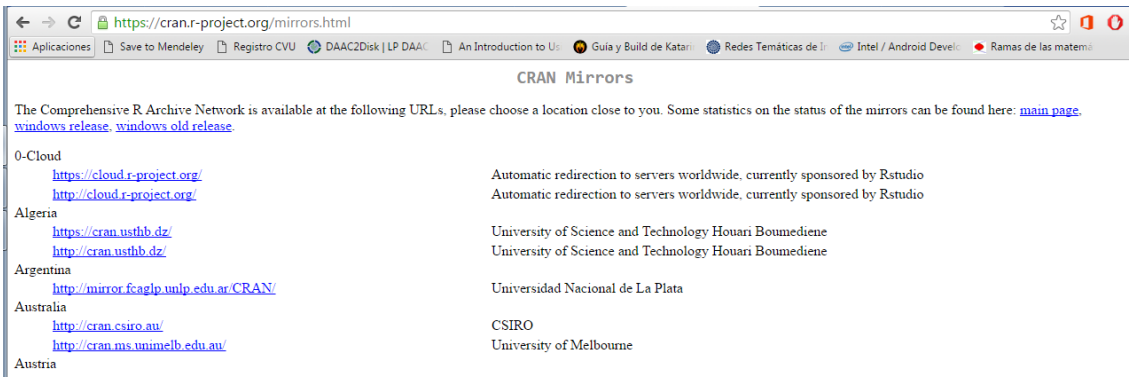

Figura 20. Página web del CRAN,

Paso 3,

Se elige servidor (por país) para iniciar la descarga, en nuestro caso utilizamos de México como se ilustra en la Fig. 21 el cual se encuentra en el Instituto Tecnológico Autónomo de México (ITAM).

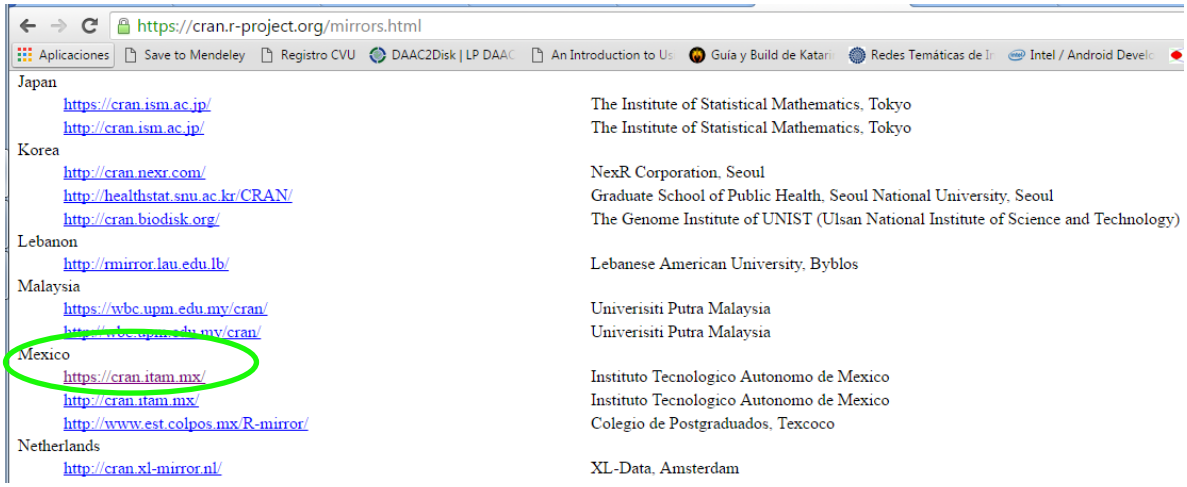

Figura 21. Servidor del ITAM,

Paso 4,

Ya dentro del sitio de descarga (https://cran.itam.mx/) elegido, se hace clic en la liga de descarga de acuerdo al sistema operativo en donde será instalado (nosotros fue Linux) como se ilustra en la Fig. 22.

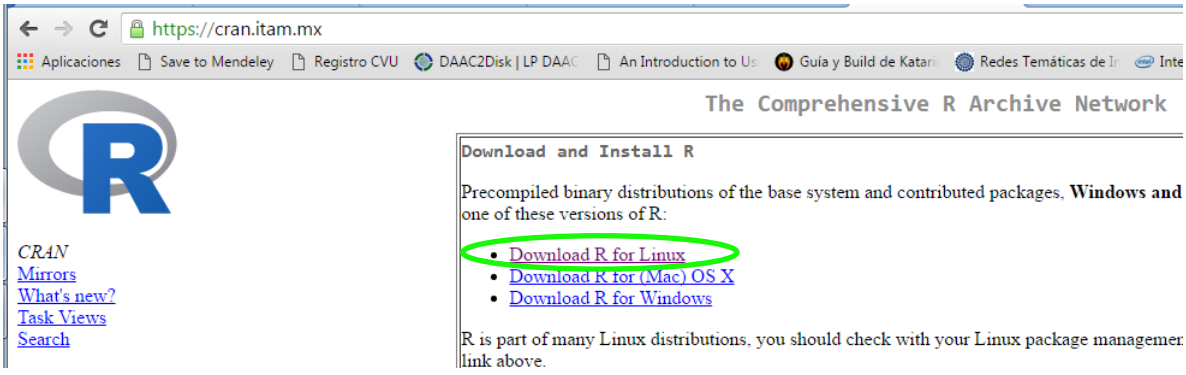

Figura 22. Sistemas operativos compatibles con R,

Paso 5,

De acuerdo a la categoría en la que se clasifica la versión de Linux Mint que se utilizó, se eligió Ubuntu como se aprecia en la Fig. 23.

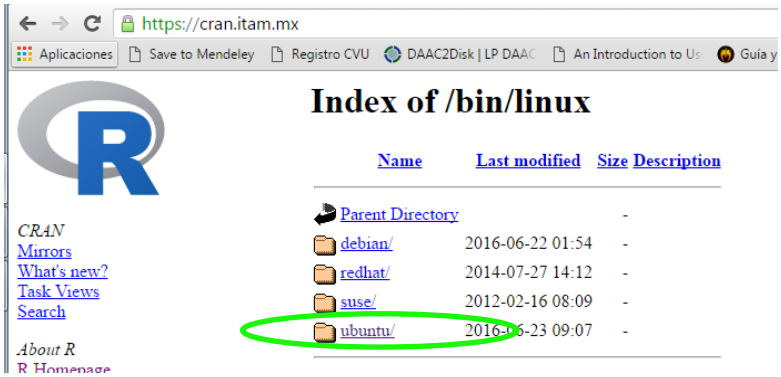

Figura 23. Opciones de descarga para Linux**,**

Paso 6,

Una vez dentro de la guía de instalación, se elige la liga Installation como se ilustra en la Fig. 24 para terminar la instalación de R, siguiendo las instrucciones que enseguida se listan desde una terminal de Linux.

Instrucciones a ejecutar (R Core Team, 2015):

```
sudo apt-get update
sudo apt-get install r-base
sudo apt-get install r-base-dev
```

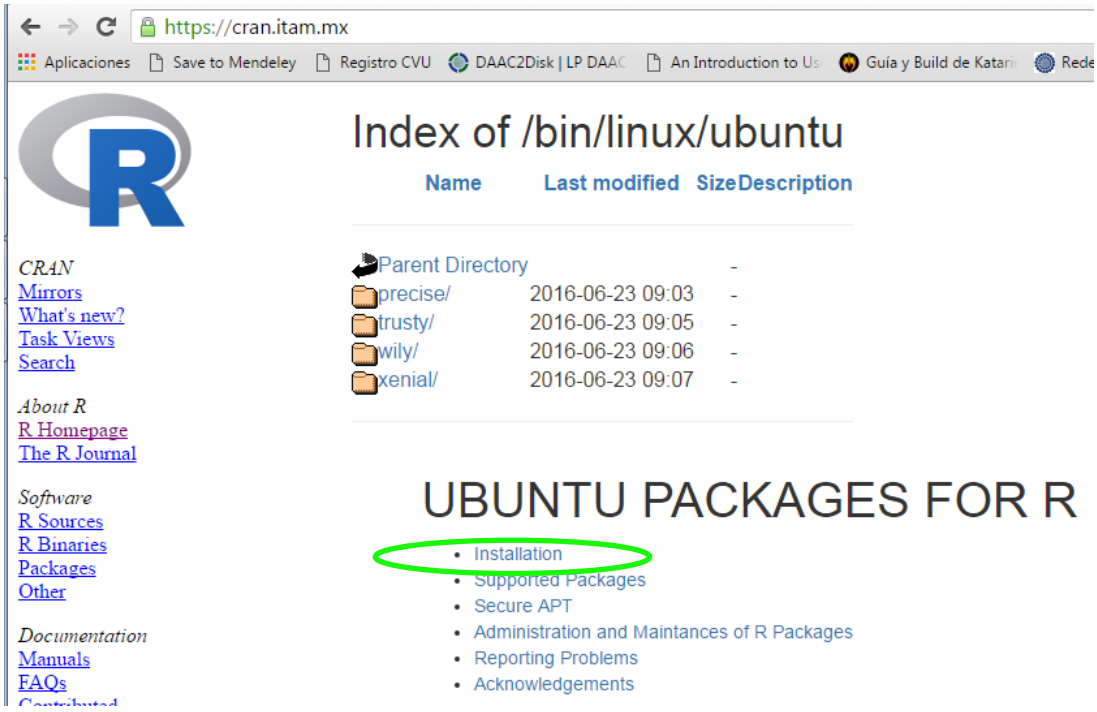

Figura 24. Instalación de R con Ubuntu,

Paso 7,

Una vez ejecutadas las instrucciones del paso anterior, se puede constatar que R ya quedó instalado, desde el *Menú* principal en la categoría de *Gráficos* como se aprecia en la Fig. 25.

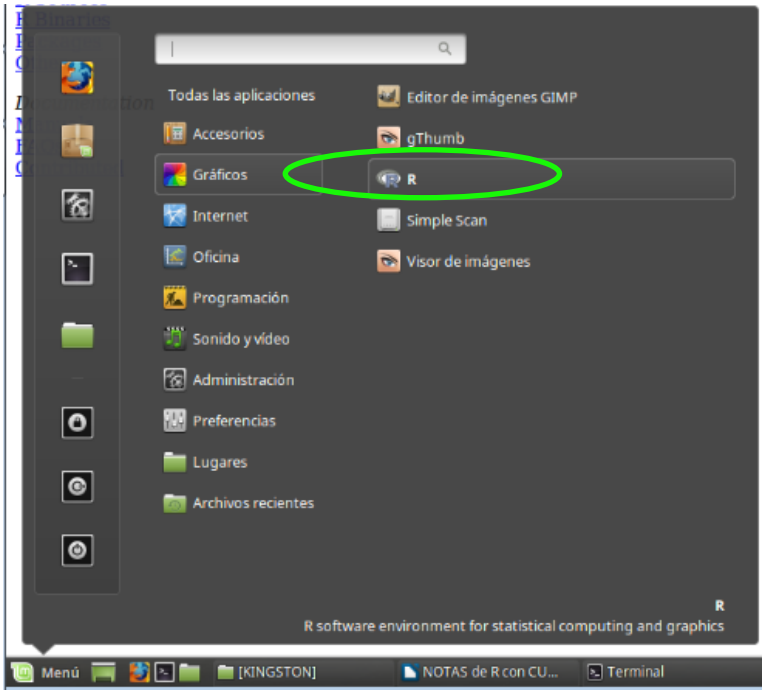

Figura 25. Menú de acceso a R,

4.3.1.1 Instalación de RStudio 0.98.110

Paso 1,

La instalación del IDE complementario RStudio, cuenta con un asistente de instalación (https://www.rstudio.com/), solo se debe descargar desde el enlace señalado en la Fig. 26

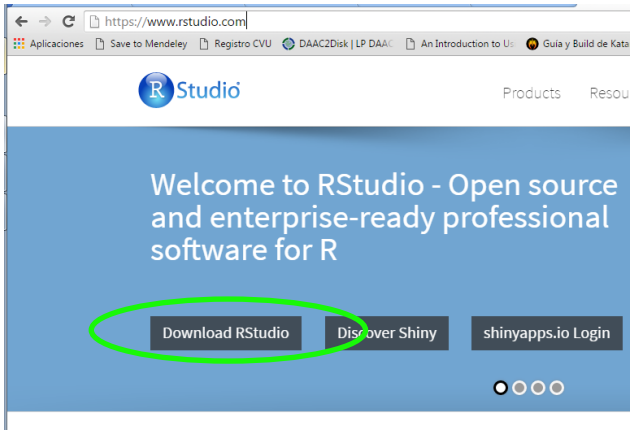

Figura 26. Página web de bienvenida de RStudio,

Paso 2,

Una vez dentro de la opción de *Download RStudio*, se debe elegir la edición *Desktop* como se aprecia en la Fig. 27.

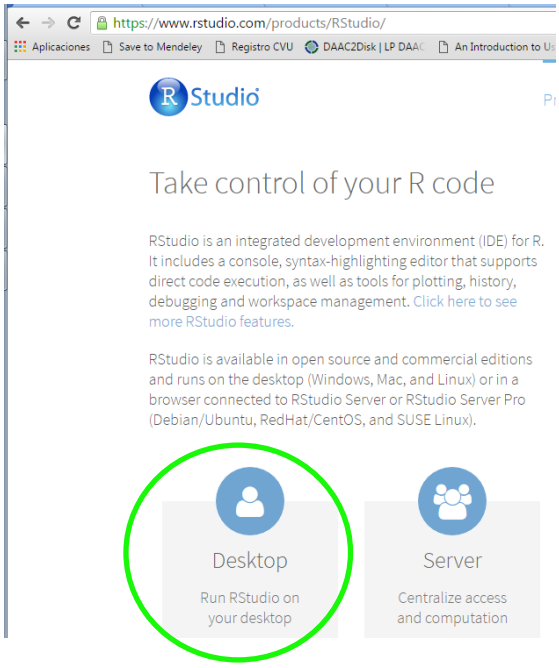

Figura 27. Ediciones disponibles de RStudio,

Paso 3,

Se debe seleccionar la versión *Open Source* por ser gratuita como se ilustra en la Fig. 28.

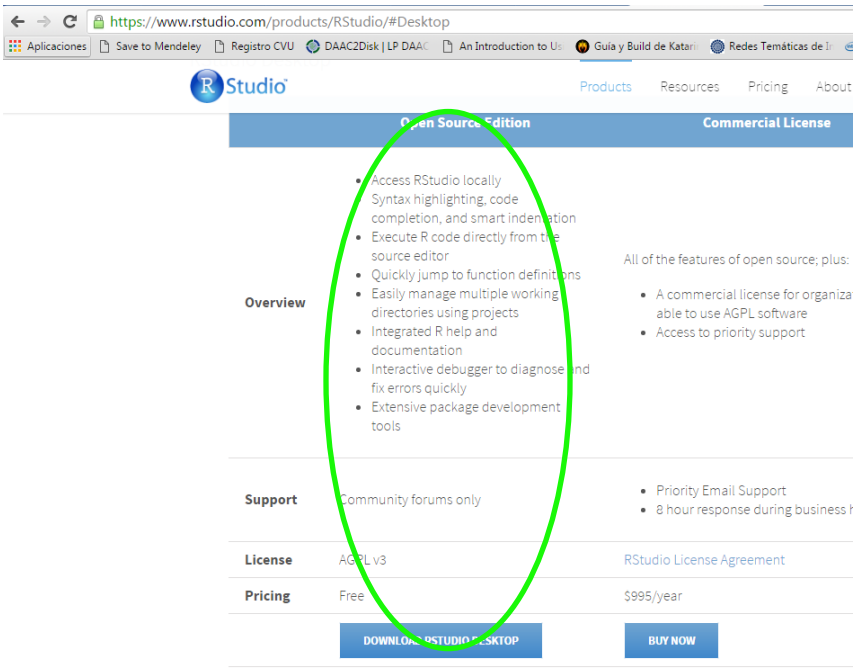

Figura 28. Versiones disponibles de RStudio,

Paso 4,

Una vez dentro de las opciones de descarga gratuitas (*Open Source*), se elige una acorde al sistema operativo en el cual será instalada como se ilustra en la Fig. 29, en nuestro caso la opción afín es Ubuntu 64 bits (por utilizar Linux Mint).

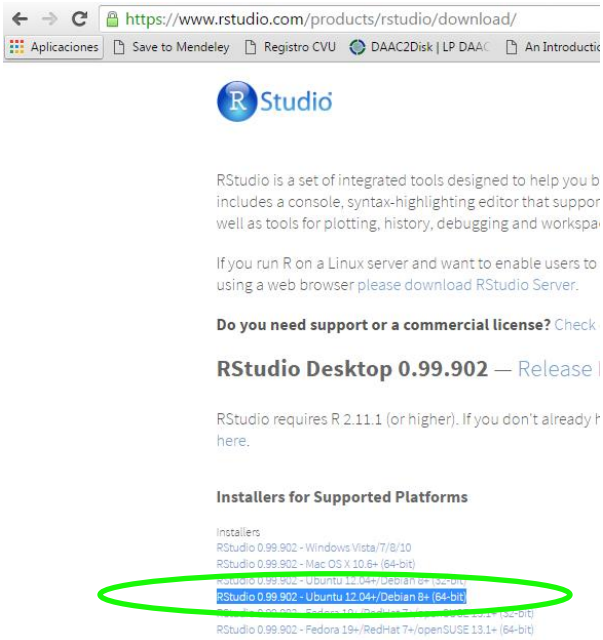

Figura 29. Sistemas operativos compatibles con RStudio,

Paso 5,

Una vez iniciada la descarga del asistente de instalación como se ilustra en la Fig. 30, solo se deberá ejecutar el asistente, y con ello concluye la instalación de RStudio.

Figura 30. Descarga del asistente de instalación,

Paso 6,
Para constatar que RStudio quedó instalado se puede acceder a él desde el *Menú* principal en la categoría *Programación* como se aprecia en la Fig. 31.

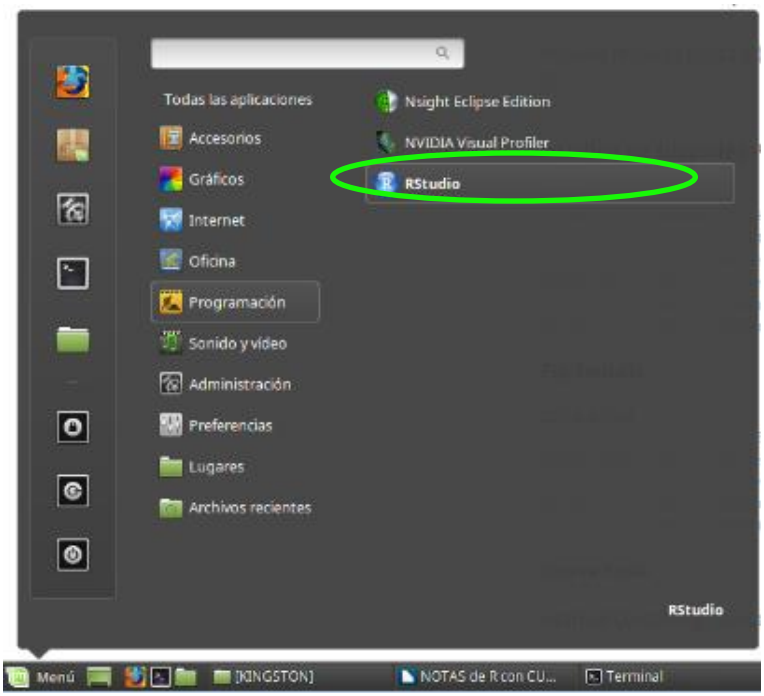

Figura 31. Menú de acceso a RStudio,

4.3.2 Instalación de OpenCV 2.4.13

Enseguida se describe la instalación de OpenCV para sistemas operativos de la familia de Linux (Mint). En esta ocasión no se utilizó la guía de instalación sugerida desde el sitio oficial de descargas, sino la de un blog (https://geekytheory.com/opencv-en-linux/), por ser más clara (Pérez, s.f.).

Los pasos a seguir fueron los siguientes:

Paso 1,
La descarga del paquete de instalación se efectuó desde el sitio oficial de OpenCV (http://opencv.org/downloads.html) eligiendo la versión para Linux como se ilustra en la Fig. 32.

Figura 32. Sistemas operativos compatibles con OpenCV,

Paso 2,

Una vez descargado el paquete de instalación, se descomprime y dentro de la carpeta descomprimida se crea una nueva carpeta llamada "build" como se ilustra en la Fig. 33.

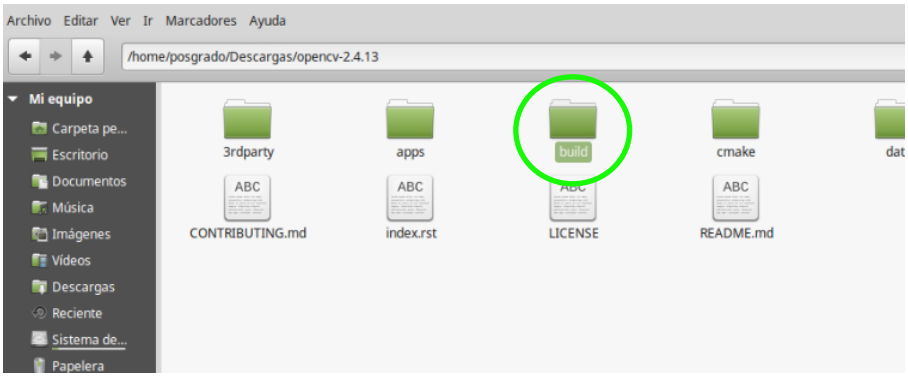

Figura 33. Creación de la carpeta "build",

Paso 3,

Una vez terminado el paso anterior, desde una terminal de Linux se ejecutan las siguientes instrucciones una a la vez.

Instrucciones a ejecutar:

```
sudo apt-get update

sudo apt-get upgrade

sudo apt-get install libopencv-dev

sudo apt-get install opencv-doc

sudo apt-get install build-essential
```

```
sudo apt-get install cmake

sudo apt-get install pkg-config


sudo apt-get install libjpeg-dev libtiff4-dev libjasper-dev
libopenexr-dev libtbb-dev libeigen2-dev yasm libfaac-dev
libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev
libvorbis-dev libxvidcore-dev libx264-dev libqt4-dev libqt4-
opengl-dev sphinx-common texlive-latex-extra libv4l-dev
libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-
dev
```

Paso 4,

Una vez terminado el paso anterior, se obtiene la ruta hacia la carpeta "build" del paso 2, porque dentro de ella se generará el Makefile que define que partes de OpenCV necesitan ser compiladas, en nuestro caso las instrucciones exactas que requerimos son las siguientes y se deberán ejecutar una a la vez.

Instrucciones a ejecutar:

```
cd /home/posgrado/Descargas/opencv-2.4.13/build

cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D -DFORCE_VTK=ON
WITH_GDAL=ON -D WITH_TBB=OFF -D BUILD_NEW_PYTHON_SUPPORT=ON -
D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D
INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D
WITH_QT=OFF -D WITH_OPENGL=OFF -DWITH_XINE=ON WITH_GTK=ON
OPENCV_BUILD_3RDPARTY_LIBS=ON WITH_CUDA=ON ..
```

Paso 5,

Una vez terminado el paso anterior, lo siguiente será compilar y esto demorará varios minutos.

Instruccion a ejecutar:

```
make
```

Paso 6,
Una vez terminado el paso anterior, para instalar la biblioteca ejecutamos la siguiente instrucción:

```
sudo make install
```

Paso 7,
Se debe definir la ruta para acceder a las librerías de OpenCV, para ello se crea un archivo llamado "**opencv.conf**" y se le agregará la siguiente línea "`/usr/local/lib`" (sin las comillas dobles).

Ejecutar la siguiente instrucción:

```
sudo gedit /etc/ld.so.conf.d/opencv.conf
```

Una vez escrito "`/usr/local/lib`" dentro del archivo, guardar y cerrar el archivo.

Paso 8,
Por último, se crean los enlaces hacia las librerías de OpenCV que acabamos de definir en el paso anterior, con la siguiente instrucción:

```
sudo ldconfig -v
```

Paso 9,
En este momento OpenCV ya está instalado, para comprobar que todo está en orden, se probará uno de los ejemplos que vienen disponibles desde la carpeta de instalación que se descomprimió en el paso 2 (si se encuentra actualmente dentro de la carpeta **build**, deberá salir de ella y llegar a la carpeta **samples**), pero para ello se compilarán todos los ejemplos.

Ejecutar las siguientes instrucciones:

```
cd /home/posgrado/Descargas/opencv-2.4.13/samples/c
chmod +x build_all.sh
./build_all.sh
```

Y para finalizar probamos un ejemplo cualquiera, en este caso la siguiente instrucción deberá mostrar la fotografía de la Fig. 34, lo cual nos confirmaría que la instalación fue correcta.

Ejecutar la siguiente instrucción:

```
./facedetect lena.jpg
```

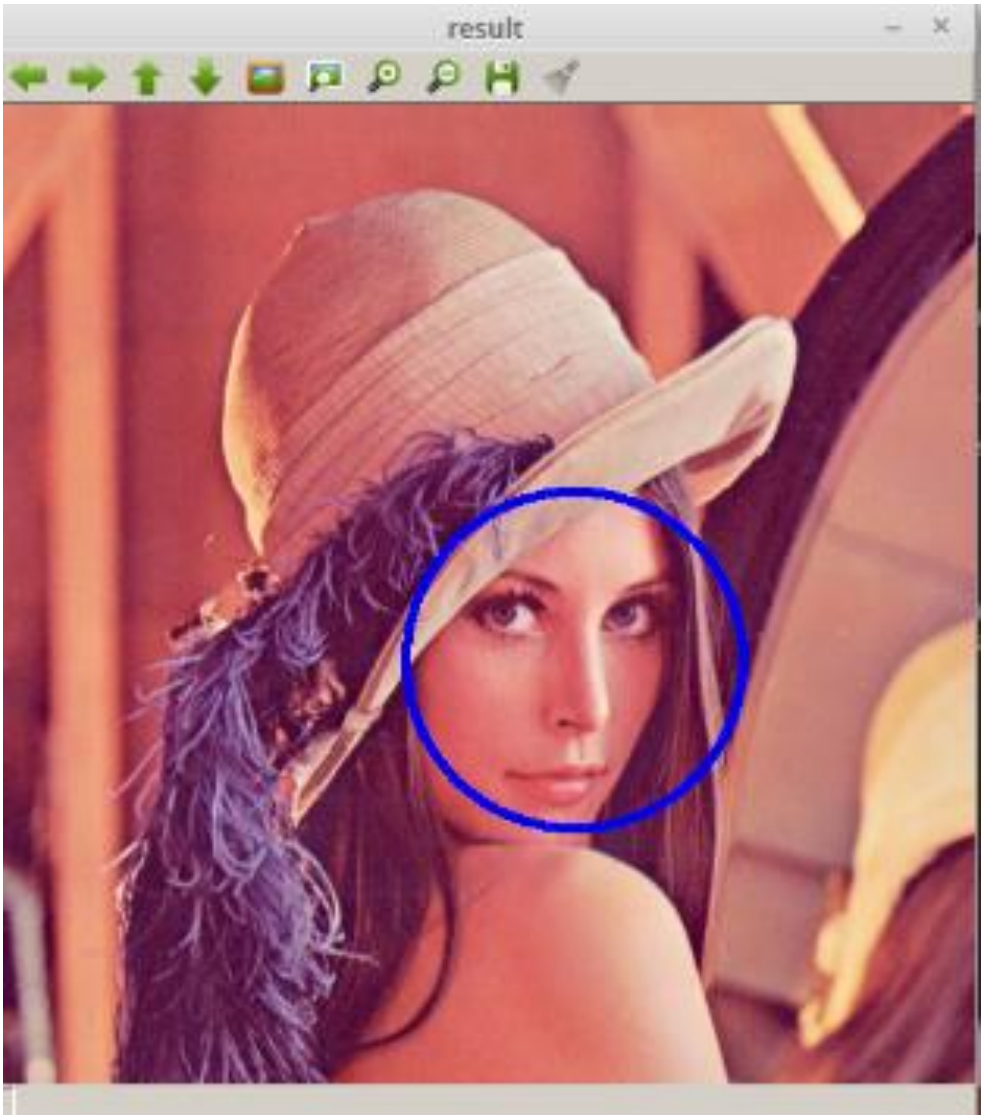

Figura 34. Imagen de ejemplo “Lena”, Fuente: Internet.

4.4 Solución final

Alcanzar la solución final a la problemática planteada para este trabajo, involucró dos fases (Fig. 35), una primera en donde se debían preparar los datos que se procesarían, y una segunda fase donde se haría la codificación del algoritmo como tal. Cada una de estas acciones, serán expuestas en los apartados siguientes.

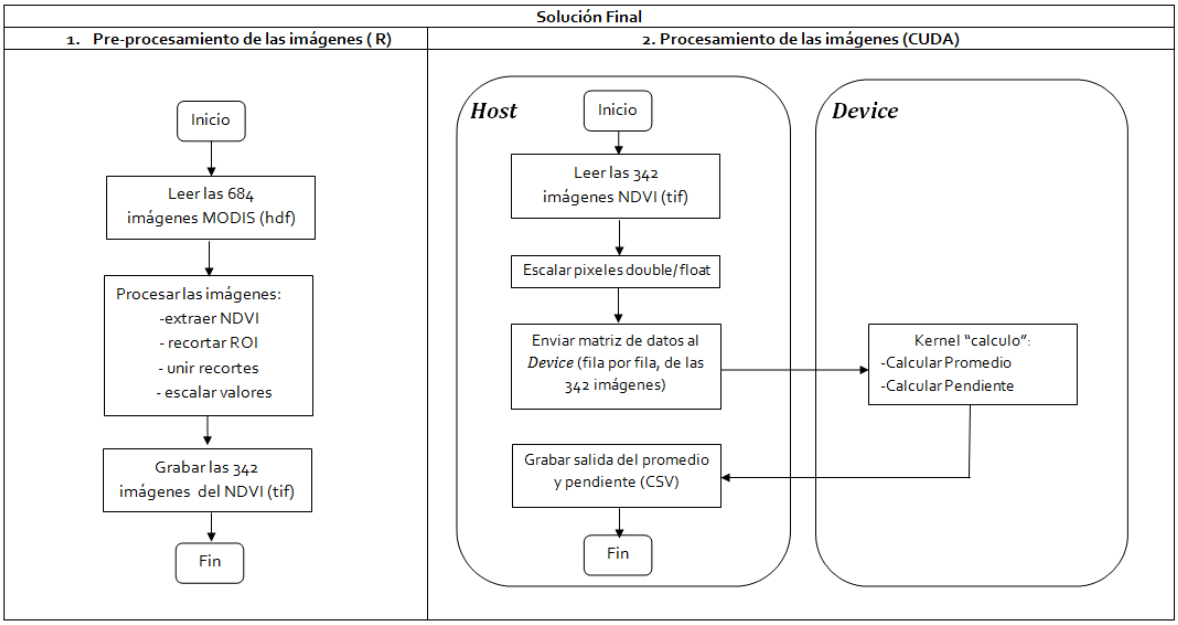

Figura 35. Diagrama de fases de desarrollo de la Solución Final.

4.4.1 Pre-procesamiento

La fase del pre-procesamiento se le denominó al procesamiento previo que se hizo a las imágenes desde R, utilizando un script (en el *Apéndice A* se anexa el código original). El procesamiento a la imágenes involucró desde el desempaquetado del archivo HDF para la extracción de la capa de NDVI, posteriormente el recorte a la región de interés de cada cuadrante como los que se ilustran en la Fig.1, quitando primero la zona del Océano Pacífico y después el Estado de Sonora respectivamente, una vez recortados se procedió a unir los cuadrantes, y para finalizar se cambió la proyección, debido a que las imágenes originalmente vienen con proyección Sinusoidal se cambió a WGS84.

Una vez terminado el pre-procesamiento de los 684 recortes originales que representan la superficie del Estado de Baja California Sur, concluyó en 342 imágenes finales (después de unir los recortes), estas fueron grabadas en disco en formato GeoTiff (extensión TIF) para

posteriormente leerlas desde CUDA en la siguiente fase. Para este procedimiento en R se uilizaron las librerías raster, rgdal y gdalUtils.

Enseguida un listado de las funciones que se utilizaron en el script, y una breve explicación de la finalidad con la que se utilizaron.

`list.files` Genera una lista con los nombres de los archivos que se encuentran en cierto directorio previamente definido, además soporta indicar un patrón de búsqueda, es decir solo leer archivos con cierta extensión o que comiencen con cierto caracter en el nombre del archivo (R Core Team, s.f.).

`gdal_translate` Extrae las capas de archivos HDF, en nuestro caso la de NDVI, el formato de la capa extraída la graba en disco con extensión TIF (R Core Team, s.f.).

`raster` Crea un raster, en nuestro caso a partir de la imagen TIF de la capa del NDVI que fue extraída del archivo HDF (R Core Team, s.f.).

`stack` Sirve para apilar un raster encima de otro, en nuestro caso creamos un raster base y después apilamos el resto de las 341 capas restantes por cada cuadrante como se ilustra en Fig. 1. (R Core Team, s.f.).

`crop` Funciona para recortar regiones de interés de una imagen, se deben indicar las coordenadas donde los puntos cruzan (R Core Team, s.f.).

`merge` Se utilizó para unir los dos recortes que conforman la superficie completa de Baja California Sur, es importante que se unieran antes de cambiar la proyección, de lo contrario genera inconsistencias (R Core Team, s.f.).

projectRaster Permite hacer el cambio en la proyección, en conjunto con la función `projectExtent` que es la que almacena el tipo de proyección a la que se desea cambiar (R Core Team, s.f.).

writeRaster Permite grabar en disco la(s) capa(s) raster en diversos formatos, en nuestro caso se utilizó el formato TIF una vez que se concluyó el pre-procesamiento de las imágenes (R Core Team, s.f.).

Pseudocodigo del Script en R:

```
1   Cargar_Librerias (raster, rgdal ,gdalUtils);
2   lista_HDF ← Leer("ruta_directorio_archivos_HDF"); /* 684 archivos */
3   lista_TIF ← Extraer_NDVI(lista_HDF); /* extrae 684 capas de NDVI */
4   raster_h07v06 ← Apilar_Raster(cuadrante_h07v06); /* apila 684  h07v06 */
5   raster_h08v06 ← Apilar_Raster (cuadrante_h08v06); /* apila 684  h08v06*/
6   raster_h07v06 ← Recortar_ROI(raster_h07v06); /* quitar zona océano */
7   raster_h08v06 ← Recortar_ROI(raster_h08v06); /* quitar zona Sonora */
8   bcs ← Unir(raster_h07v06, raster_h08v06); /* resultan 342 imágenes */
9   bcs ←bcs/100000000; /* escala los valores entre 0 y 1 */
10  bcs ← Cambiar_proyeccion(Sinusoidal, WGS84);
11  Grabar_TIF(bcs, "ruta_directorio"); /* graba en disco las 342 imágenes */
```

4.4.2 Procesamiento

La fase del procesamiento (es una subetapa dentro de la fase del desarrollo de la solución final, ver Fig.16) se refiere a la codificación en sí del algoritmo en CUDA. En los apartados siguientes se presentarán los procesos más importantes dentro del algoritmo.

Pero, primero un listado de las principales funciones que se utilizaron en el Algoritmo, y una breve explicación de la finalidad con que se usaron.

Mat.imread Permitió leer las 342 imágenes TIF, para posteriormente cambiar el tipo de dato de los pixeles de *double* a *float*.

cudaMallocPitch Se gestionó el arreglo bidimensional (o matriz) del lado del *Device*, en donde se copiaban los datos de la fila en turno a la que se calculaba el promedio.

cudaMemcpy Permitió copiar en la matriz de salida (del promedio histórico), el promedio calculado en el *Device* de la fila en turno.

4.4.2.1 Lectura de las imágenes del lado del Host

Una vez generadas las 342 imágenes TIF (3455 x 1758 pixeles) en la fase del *pre-procesamiento*, se procedió a leerlas desde CUDA, esto se logró utilizando la librería `highgui.h` de OpenCV (específicamente con el objeto `Mat`).

Para leer las (342) imágenes del lado del *Host*, fue necesario convertir los valores de los pixeles originalmente *double* (8 bytes) a *float* (4 bytes). Para poder las imágenes en una sola pasada, debido a que la capacidad de la memoria no era suficiente. Sin el escalamiento que sufirieron los datos representaban aproximadamente 14GB.

Si bien la memoria disponible del lado del *Host* se presume eran 12GB (capacidad de la tarjeta utilizada), el sistema operativo tiene que cubrir sus requerimientos de operatividad, por lo que realmente solo quedaban disponibles 8 GB, los cuales no eran suficientes. De modo que una vez convertidos los pixeles a *float* sumanaban 7.73 GB de datos a procesar. Cabe señalar que el escalamiento de los pixeles fue posible porque no significaba perdida en la precisión de los valores almacenados en los pixeles.

4.4.2.1.1 Librería highgui.h

Es la librería de OpenCV que permitió leer las imágenes MODIS (TIF) que se procesarían en la GPU. Específicamente a través de su clase `Mat`, se realizó la lectura de las 342 imágenes con pixeles de profundidad de 64 bits (double = 8 bytes). Las imágenes fueron escaladas de tipo *double* a *float* utilizando la misma clase Mat, que si bien soporta otros tipos de operaciones, enseguida se presenta un listado solo de las que se utilizaron para este trabajo:

cv::imread Es la función que permitió leer las imágenes MODIS en formato TIF, ya que emula tareas como los tradicionales apuntadores de

archivos de C (* FILE), como fopen y fread, pero ambas fusionadas en una sola (Mat-OpenCV, s.f.).

Mat.clone Genera una copia del objeto Mat (imagen) con dirección de memoria distinta de la imagen de origen, es decir los cambios realizados en la copia no afectan a la imagen base (Mat-OpenCV, s.f.).

Mat.convertTo Permite cambiar el tipo de dato de los pixeles de la imagen, por ejemplo, nosotros cambiamos de double a float (Mat-OpenCV, s.f.).

Mat.push_back Sirve para agregar elementos a un vector de objetos de Mat, es decir otro objeto Mat (Mat-OpenCV, s.f.).

Mat.ptr Permite acceder a las filas de la imagen y a sus columnas (Mat-OpenCV, s.f.).

4.4.2.2 Envío de las imágenes entre el *Host* y el *Device*

La estrategia utilizada para el envío de los datos entre el *Host* y el *Device*, consistió en realizar una serie de iteraciones para procesar las imágenes por filas. Es decir, en una primera iteración se enviaba solamente la *fila 1*, pero de todas las imágenes y así sucesivamente hasta llegar a la *fila n* (3455). En cada iteración se estaba enviando una matriz de 342 x 1758 (arreglo bidimensional gestionado con `cudamallocpith`). La estrategia surgió debido a las dimensiones de las imágenes, ya que al enviarse todos los datos al *Device* provocaban violaciones de segmento en la memoria.

Del lado del *Device* dentro de un solo *kernel* se calculó el promedio y la pendiente a la fila en turno. Los cálculos se efectuaron con un total de 1758 hilos (se gestionaron 2 bloques con 879 hilos) como se aprecia en la Fig. 36 donde a cada hilo le correspondía una columna de la matriz de datos recibida (342 x 1758). Los resultados del promedio y la pendiente (de la fila

en turno) se enviaban de regreso al *Host* y se almacenaban en las respectivas matrices de salida, una para el promedio y otra para la pendiente.

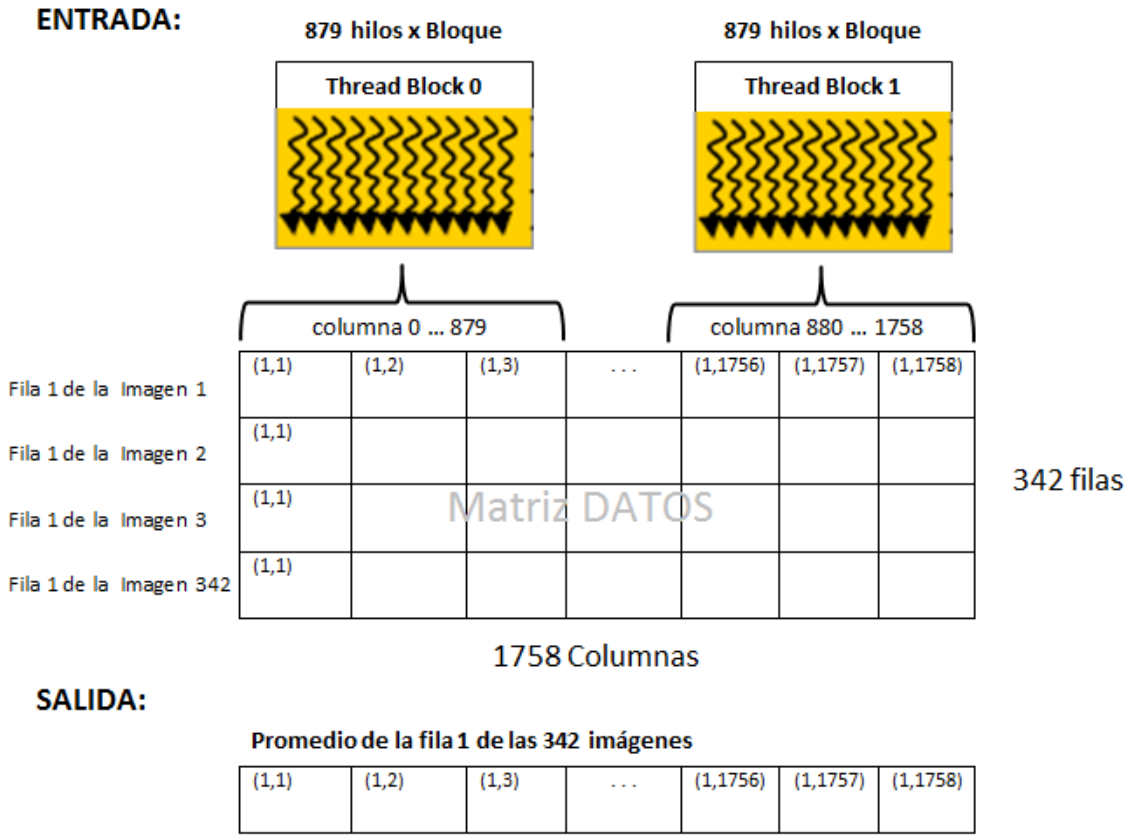

Figura 36. Asignación de los hilos para procesar la matriz recibida.

4.4.2.2.1 cudamallocpitch

`cudamallocpitch` es una función nativa de CUDA (palabra reservada que forma parte del lenguaje de CUDA C), y sirve para gestionar arreglos bidimensionales (ancho*alto en bytes de memoria lineal) del lado del *Device* y devuelve un apuntador a la memoria gestionada; `Cudamallocpitch` fue fundamental para desarrollar la estrategia de paralelización con la que se enviaron los datos de las imágenes a procesar dentro del *Kernel* (CUDA Toolkit, s.f. b).

La sintaxis de *cudamallocpitch*:

```
cudaError_t cudaMallocPitch ( void**    devPtr,
                              size_t*   pitch,
                              size_t    width,
```

```
    size_t    height
)
```

Donde (CUDA Toolkit, s.f. b):

devPtr Apuntador a la memoria gestionada en el *Device*.

pitch Apuntador que almacena el tamaño del espacio en bytes del **width** gestionado.

width Requerimiento del ancho en bytes a gestionar (*sizeof* según un tipo de dato).

height Requerimiento de altura a gestionar (en bytes según el tipo de dato del *width*).

4.4.3 Grabando los resultados

Las matrices resultantes del cálculo del promedio y la pendiente se grabaron a disco utilizando el formato CSV (*comma separated values* en inglés). Dos archivos separados para almacenar los respectivos cálculos (promedio.csv y pendiente.csv).

Pseudocogido del Algoritmo en CUDA:

```
1   Cargar_Librerias ("librerías de CUDA");
2   Cargar_Librerias ("librerías de OpenCV");
3   Definir_Constantes (REN= 3455, COL=1758, THR=879,BLK=2,TOT_IMG=342);
4   OOM_Killer("No terminar el proceso actual"); /* -17 exceptúa el proceso*/
5   listado_imagenes ←Leer("ruta_directorio_imagenes_TIF"); /* 342 imagenes */
6   listado_imagenes ←ConvertTo(listado_imagenes,float)/*orginalmente double*/
7   matriz_filasD ← GestionarMemDevice(row[342],col[1758]); /*datos al DEVICE*/
8   promedioFila ← GestionarMemoriaDevice(row[1],col[1758]);
9   promedioHistorico ← GestionarMemoriaHost(row[3455],col[1758]);
10  pendienteFila ← GestionarMemoriaDevice(row[1],col[1758]);
11  pendienteHistorico ← GestionarMemoriaHost(row[3455],col[1758]);
12  Para (r=0 hasta >=REN)/* 3455 renglones por imagen */
13    Para (i = 0 hasta listado_imagenes.size) /*342 imágenes en total*/
14        matriz_filasD[i]← CopiarFila(listado_imagenes[i].row[r]);
15    Fin Para
16    (promedioFila,pendienteFila)← Kernel_prom_pend<<BLK,THR>>(matriz_filasD);
17     promedioHistorico.row[r] ← CopiarFila (promedioFila);
18     pendienteHistorico.row[r] ← CopiarFila (pendienteFila);
19  Fin Para
20  Liberar_Memoria_Gestionada();
21  GrabarArchivo("promedio.csv",promedioHistorico);
22  GrabarArchivo("pendiente.csv",pendienteHistorico);
```

Capítulo 5. Resultados

5.1 Validación de resultados

Después de calcular los valores del promedio y la pendiente del NDVI para Baja California Sur durante el periodo febrero 2000 – diciembre 2014 mediante el algoritmo implementado en CUDA, se compararon con los resultados obtenidos desde R, pixel por pixel. Para esto se emplearon dos métodos: 1) Inspección visual y 2) Inspección numérica (cálculo del porcentaje de error).

Debido a que el algoritmo de CUDA no generó la salida de los resultados con representación gráfica, en formato imagen (la librería `highgui.h` utilizada para la lectura de las imágenes, no soporta grabar imágenes con pixeles mayores a 16 bits de profundidad, solo puede leerlas). Por lo que los resultados del promedio y la pendiente se exportaron en archivos de texto separados por coma (CSV). Estos archivos tienen un formato matricial, de tal manera, que cada valor representa un punto geográfico (Longitud y Latitud).
Posteriormente los archivos CSV fueron importados en R en formato de matrices y después se convirtieron en objetos ráster (formato para sistemas de información geográfica que soporta imágenes espaciales). Una vez generados los objetos ráster con los promedios y pendientes calculados desde CUDA y R, se compararon sus valores mínimos y máximos (Fig. 37 y Fig.38). Durante el análisis se detectaron diferencias mínimas en los valores obtenidos. Lo que corrobora que ambos resultados son consistentes entre si.

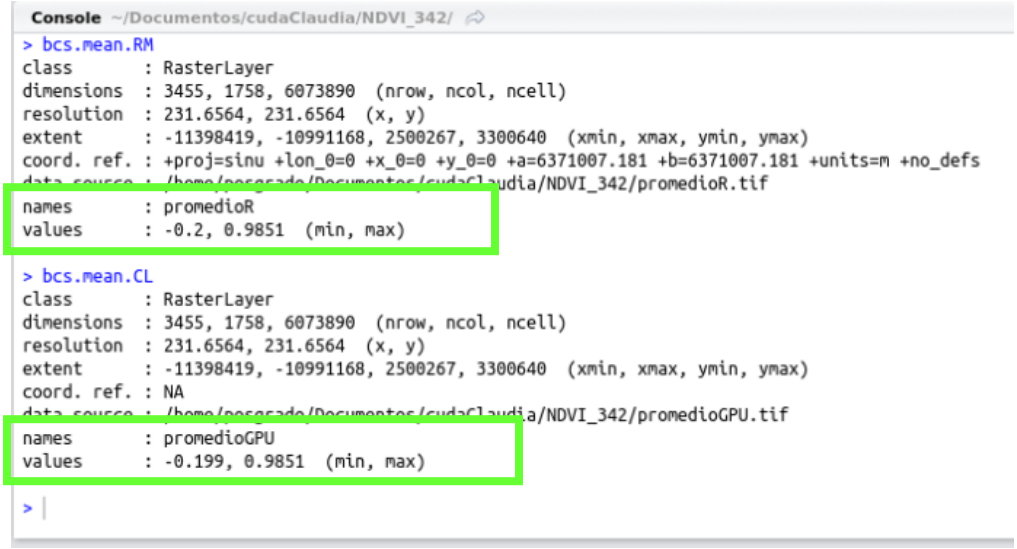

Figura 37. Valores del promedio del NDVI calculado desde R (bcs.mean.RM) y CUDA (bcs.mean.CL).

```
Console ~/Documentos/cudaClaudia/NDVI_342/
> slope.RM
class       : RasterLayer
dimensions  : 3455, 1758, 6073890  (nrow, ncol, ncell)
resolution  : 231.6564, 231.6564  (x, y)
extent      : -11398419, -10991168, 2500267, 3300640  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=sinu +lon_0=0 +x_0=0 +y_0=0 +a=6371007.181 +b=6371007.181 +units=m +no_defs
data source : in memory
names       : pendienteR
values      : -0.001308451, 0.001922987  (min, max)

> slope.CL
class       : RasterLayer
dimensions  : 3455, 1758, 6073890  (nrow, ncol, ncell)
resolution  : 231.6564, 231.6564  (x, y)
extent      : -11398419, -10991168, 2500267, 3300640  (xmin, xmax, ymin, ymax)
coord. ref. : NA
data source : in memory
names       : pendienteGPU
values      : -0.001308, 0.001923  (min, max)

>
```

Figura 38. Valores de la pendiente del NDVI calculada desde R (slope.RM) y CUDA (slope.CL).

Para finalizar con la inspección visual de los resultados del promedio y la pendiente, calculados desde CUDA (GPU) y R (CPU). Se procedió a calcular la diferencia entre las respectivas matrices, pixeles por pixel, y se almaceno en una tercera matriz (Fig. 39 y Fig. 40). Las diferencias resultantes para ambos cálculos (promedio y pendiente) fueron muy cercanas a cero, ya que algunos pixeles presentaron diferencias mínimas. Lo que indica una alta concordancia espacial entre ambos resultados (CUDA y R).

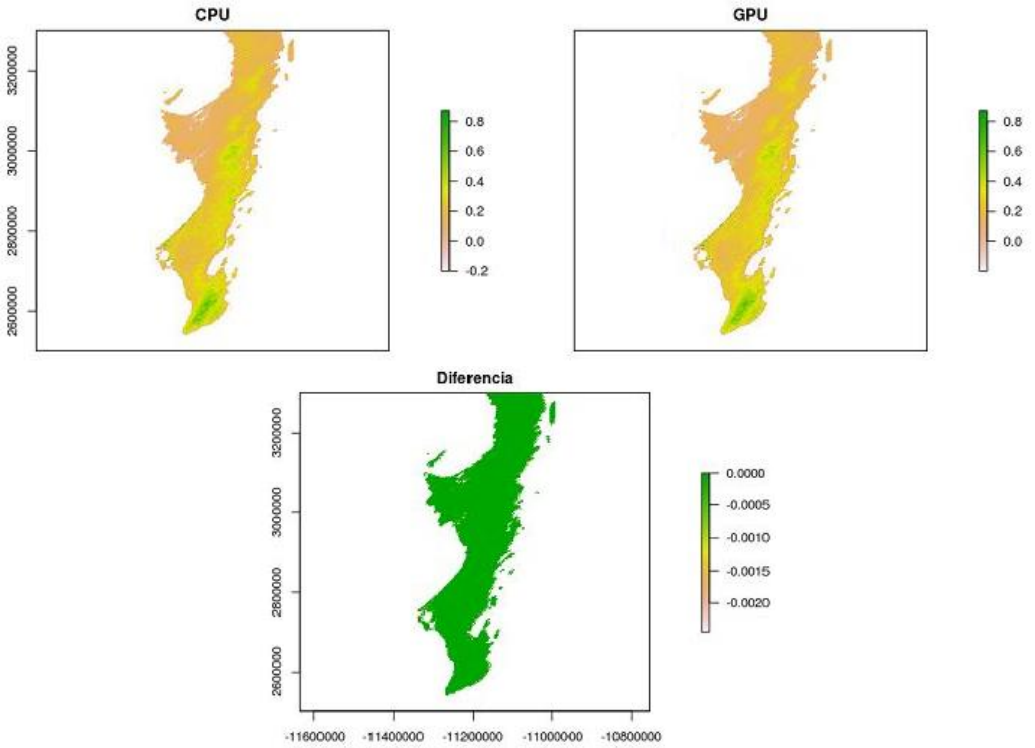

Figura 39. Comparación de los resultados del cálculo del promedio de R y CUDA.

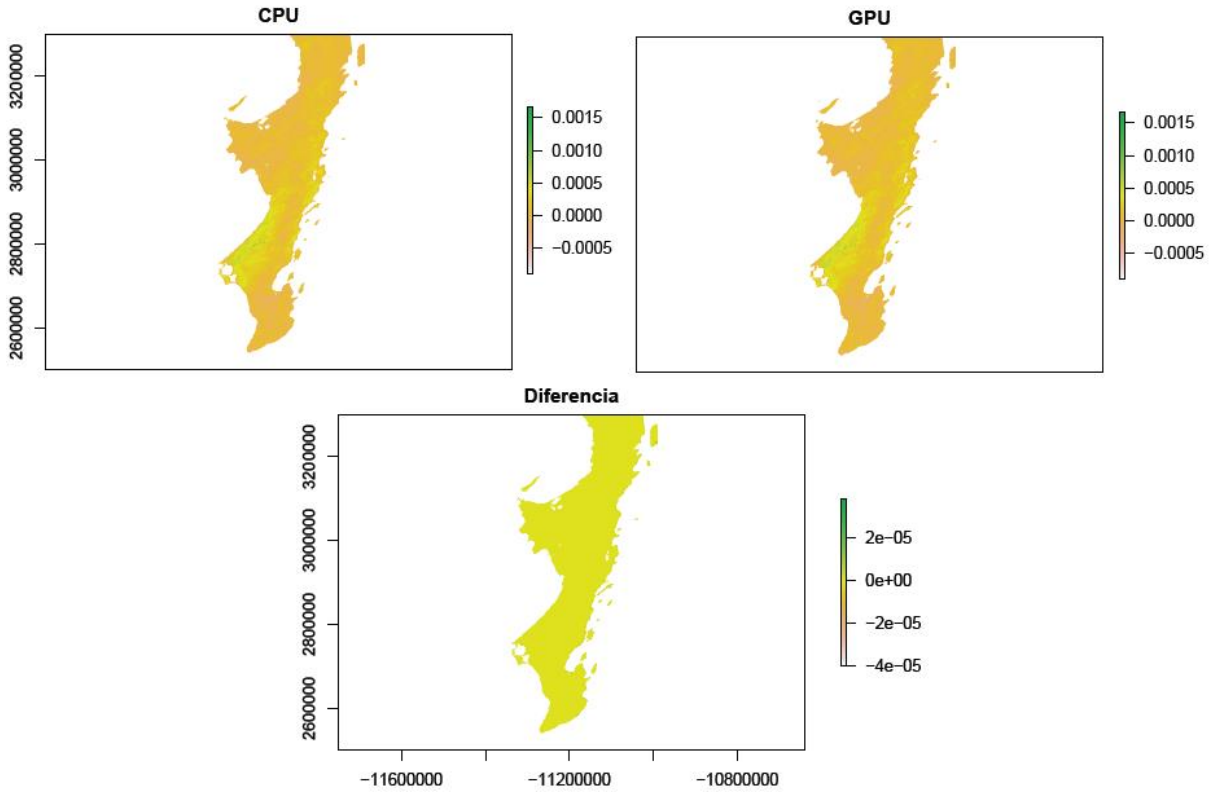

Figura 40. Comparación de resultados del cálculo de la pendiente de R y CUDA**.**

Concluida la inspección visual, se continuó con la inspección numérica de los datos (Fig. 37 y Fig. 38), que consistió en calcular el porcentaje de error de los valores minimos y máximos de los resultados obtenidos en CUDA respecto R (Tabla 7), utilizando la siguiente ecuación (Vargas *et al*., 2008):

$$\%E = \left|\frac{M - m}{M}\right| \times 100$$

Donde *m* es el valor experimental y *M* es el valor teórico, el porcentaje de error relaciona el valor teórico con el valor experimental (Vargas et al., 2008). En la ecuación del cálculo de porcentaje de error (%E) la *m* representa los valores obtenidos en CUDA y la *M* los valores de R.

Tabla 7

Porcentaje de error en los resultados del promedio y la pendiente de CUDA vs R

	Promedio		***Pendiente***	
Plataforma	***valor minimo***	***valor maximo***	***valor minimo***	***valor maximo***
CUDA	-0.199	0.9851	-0.001308	0.001923
R	-0.2	0.9851	-0.001308451	0.001922987
% E	0.50	0.00	0.03	0.00

Del porcentaje de error calculado, se presentó solo en los valores minimos de ambos cálculos (Tabla 7). El del 0.5% del valor minmo del promedio, fue originado por el redondeo del lado de R (valores minimos: R = - 0.2, CUDA= - 0.199). Y el error del 0.003% del valor minimo de la pendiente, fue originado por el redondeo del lado de CUDA (valores minimos: CUDA= - 001308, R= -0.001308451). De este último tenemos la certeza que fue una consecuencia del escalamieno que sufrieron las imágenes (originalmente los pixeles eran double y se cambiaron a float) durante el proceso de lectura desde disco (ver apartado 4.4.2.1), que aunque se analizó el no perder la precisión de los valores almacenados en los pixeles, no se previó la magnitud de los resultados de sus cálculos.

5.2 Promedio y Pendiente del NDVI (2000 – 2014)

A partir de los valores calculados del promedio y la pendiente del NDVI para el estado de Baja California Sur, usuarios de Sistemas de Información Geográfica (SIG) crean mapas que sirven para describir la distribución espacial de la vegetación a lo largo del estado (Fig. 41 y Fig. 42), lo que les facilita la toma desiciones sobre el análisis de fenómenos naturales para ciertas areas de estudio, por ejemplo: la cuantificación de la degradación algún ecosistema terrestre, basado en el indicador de la "desertificación".

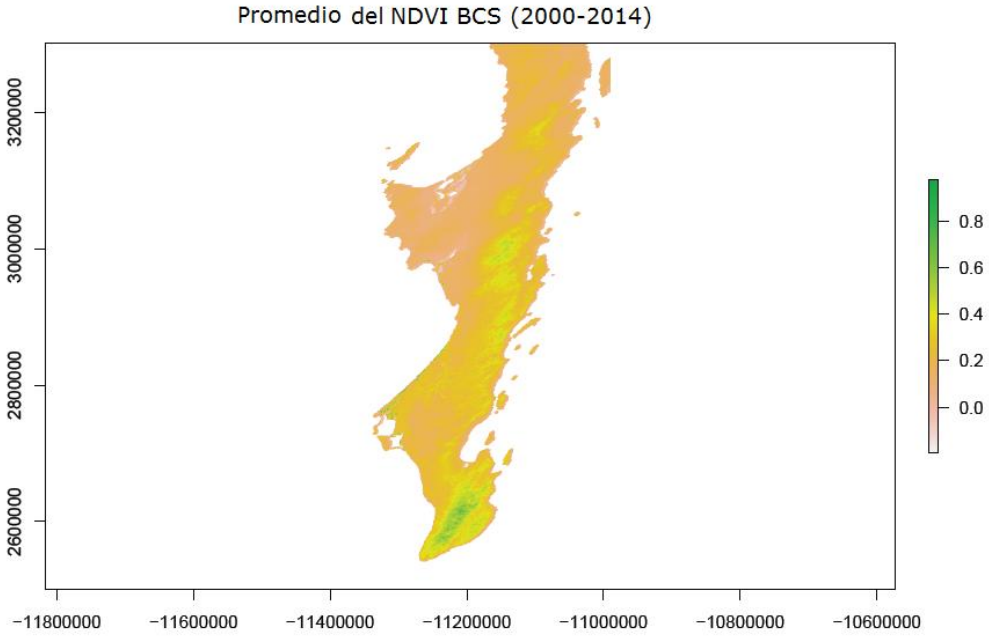

Figura 41. Mapa del promedio del NDVI.

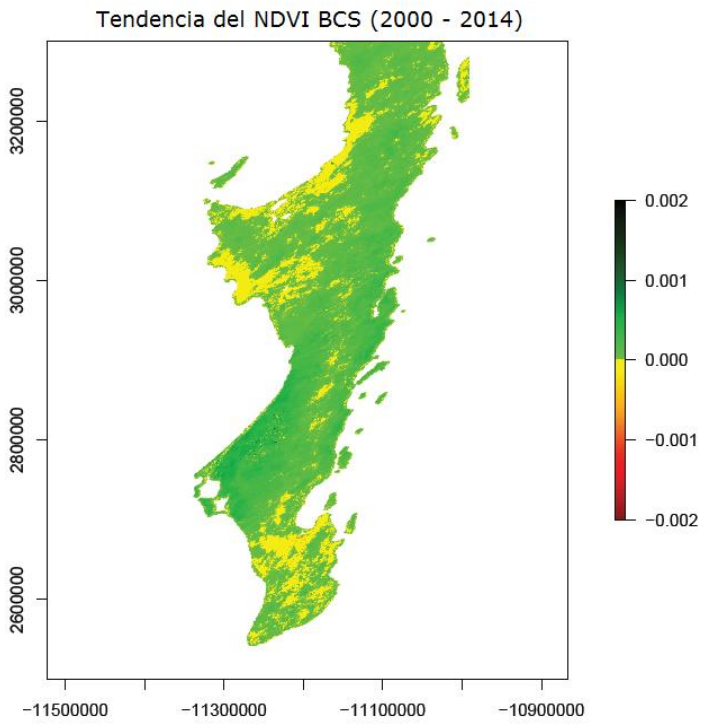

Figura 42. Tendencia del NDVI registrada durante el periodo 2000-2014.

En el caso del cálculo de promedio del NDVI, los valores más altos fueron observados en la zona sur del Estado en le región conocida como "Reserva de la biosfera Sierra La Laguna". Se caracteriza por contar con los siguientes tipos de vegetación (Arriaga y Ortega, 1988):

Bosque de encino Bosques en donde predomina el encino. Suelen estar en climas templados y en altitudes mayores a los 800 m.

Selva baja caducifolia Comunidad vegetal de 4 a 15 m de altura en donde más del 75 % de las especies pierden las hojas durante la época de secas.

Bosque de pino-encino Asociación de pinos y encinos en donde predominan los pinos. Se distribuyen en zonas templadas y zonas frías.

Bosque de encino-pino Asociación de encinos y pinos en donde predominan los encinos. Suelen estar en climas templados y en altitudes mayores a los 800 m.

Bosque de encino-pino y pastizal	Asociación de encino, pino y pastizal que ha sido introducido por cambio en el uso del suelo. Predominan los encinos.
Selva baja caducifolia y bosque de encino	Asociación de selva baja caducifolia y bosque de encino en donde predomina la selva baja caducifolia.
Matorral inducido y bosque de encino	Asociación de matorral que ha sido introducido por cambio en el uso del suelo y bosques de encino. Predominan los matorrales.
Bosque de encino y selva baja caducifolia	Asociación de encino y selva baja caducifolia en donde predominan los encinos.
Selva baja caducifolia y matorral xerófilo	Asociación de selva baja caducifolia y matorral xerófilo en donde predomina la selva baja caducifolia.
Matorral xerófilo y selva baja caducifolia	Asociación de matorral xerófilo y selva baja caducifolia en donde predominan los matorrales xerófilos.

Y por el contrario los valores más bajos del promedio del NDVI se localizaron al norte del estado en el municipio de Mulegé, en la región conocida como "Reserva de la biosfera El Vizcaino", que se caracteriza por vegetación del tipo (Programa Biosfera Vizcaino, 2000):

Matorral Sarcocaule	Esta agrupación se caracteriza por la dominancia fisonómica de árboles y arbustos de tallo grueso, de crecimiento tortuoso, semisuculentos, de madera blanda y con algunas especies que poseen corteza papiracea y exfoliante
Matorral Sarco-Crasicaule	Bajo esta denominación es posible agrupar en una misma comunidad tanto a las especies sarcocaules como a las crasicaules que se encuentran dentro del área de la Reserva. Este matorral se caracteriza por la dominancia de cactus, muchos de ellos de crecimiento candelabriforme y talla elevada, regularmente Pachycereus pringlei es el dominante fisonómico.

Matorral Halófilo Esta asociación agrupa especies con un alto nivel de tolerancia a la salinidad y alcalinidad del suelo. Se localiza en zonas que estuvieron bajo la superficie del mar o reciben su influencia directa. Esta agrupación vegetal comprende desde unos pocos metros sobre el nivel del mar, fuera de la influencia directa de las mareas, hasta unos 60 m., de elevación.

Matorral de Dunas La superficie ocupada por las dunas es relativamente inestable ya que la acción del viento y las precipitaciones que ocurren en la región, mueven progresivamente importantes volúmenes del sustrato. La flora de esta asociación parece tratar de fijarse al suelo inmediato y estas plantas sirven de refugio a la fauna, permitiendo la convivencia intra e interespecífica y el desarrollo de relaciones tróficas.

Matorral Desértico Microfilo Inerme Comprende una estrecha franja entre el matorral de dunas y el matorral halófilo pero con una mayor densidad vegetal y cobertura que la de ambas. Se encuentra en suelos arenosos, más afín con las dunas, y su pedregosidad es menor que en el caso del matorral halófilo. Se caracteriza por la dominancia de especies herbáceas y semiarbustivas y sobre todo por la ausencia casi total de elementos espinosos.

Vegetación de dunas costeras Esta asociación presenta grandes afinidades con el matorral halófilo. Se localizan sobre montículos arenosos en la proximidad de la franja litoral cuyo sustrato no es inundable. Su composición florística suele variar de un sitio a otro.

Eriales Se localizan en amplias superficies llanas cercanas a los cuerpos lagunares del Pacífico (Ojo de Liebre y San Ignacio), en donde por influencia de las mareas altas, el agua de mar alcanza varios

> kilómetros tierra adentro. Los vegetales que aquí llegan a encontrarse son muy escasos en número y en especie, proceden de las agrupaciones adyacentes sobre todo de las áreas cercanas a la costa. Los factores ambientales a los que se tienen que enfrentar dichas especies son: Alto nivel de salinidad y alcalinidad del suelo, viento y radiación solar elevada.

Por otro lado, en cuanto al cálculo de la pendiente, la escala de valores de sus resultados osciló en un rango de -0.0010 a 0.0015. Cabe señalar que en la Fig. 42 se modificó la escala de colores para poder ver los valores positivos y negativos con mayor facilidad. Los valores de las tendencias negativas del NDVI sirven para identificar las regiones que han sufrido un deterioro o pérdida del vigor de su vegetación y valores positivos recuperación de la cobertura vegetal o nuevos surgimientos. Es asi, que podemos observar que durante los últimos 15 años (2000-2014) para el estado de Baja California Sur, el municipio de Comomdú es el único que muestra una pronunciada tendencia positiva en su vegetación (ver Fig. 42), por el contrario, las regiones del norte y sur del estado se observan pronunciadas tendencias negativas del comportamiento de su vegetación.

5.3 Tiempos de ejecución

Durante la ejecución de los calculos desde CUDA y R, debido a que la GPU y CPU utilizadas no eran exclusivas, por lo que no estaban exentas de realizar otras tareas al momento de efectuar las ejecuciones de los algoritmos. Para poder comparar los tiempos de ejecución de ambas plataformas, se calcularon tiempos promedio a partir del registro de 30 ejecuciones.

Los tiempos de las 30 repeticiones de las dos plataformas (para los dos cálculos promedio y pendiente) se pueden observar en las tablas 7 y 8, respectivamente. Cabe mencionar, que en los registros del cálculo de la pendiente desde R intencionalmente solo se realizaron 4 repeticiones, debido a lo costoso que era completar la serie de 30, ya que cada ejecución duraba aproximadamente 5 horas (Tabla 9).

Los tiempos promedio obtenidos de las 30 repeticiones, para el cálculo del promedio del NDVI, en el caso de CUDA fueron 75.778 segundos, mientras que desde R fueron 5,271.553 segundos. Representando una marcada diferencia entre ambas plataformas, en

otras palabras, el algoritmo de CUDA fue 69 veces más rápido o más eficiente que R. Y para el cálculo de la pendiente del NDVI, nuevamente el algoritmo de CUDA resultó más impresionante, con un tiempo promedio de 114.205 segundos, en comparación con los 19,088.827 segundos del algoritmo de R. En este caso el algoritmo de CUDA fue 155 veces más rápido que R.

Tabla 8
Tiempos de ejecución del cálculo del promedio en segundos.

No.	***Paralelo****	***Secuencial****
1	73.567	5,276.323
2	79.322	5,274.362
3	72.665	5,294.125
4	79.190	5,291.351
5	73.862	5,302.275
6	76.006	5,279.982
7	79.348	5,275.968
8	77.348	5,294.117
9	75.202	5,305.052
10	75.295	5,288.155
11	75.149	5,287.130
12	74.733	5,296.652
13	75.122	5,277.610
14	75.881	5,300.499
15	75.083	5,278.333
16	75.083	5,279.009
17	75.523	5,247.277
18	76.006	5,116.870
19	79.348	5,118.594
20	77.348	5,294.380
21	75.202	5,265.268
22	75.295	5,290.345
23	75.149	5,297.409
24	74.733	5,301.934
25	75.122	5,291.659
26	75.881	5,226.794
27	75.083	5,285.056
28	75.523	5,272.011
29	75.854	5,239.234
30	74.428	5,298.825
Tiempo promedio	75.778	5,271.553

Nota: **Paralelo = Algoritmo CUDA, *Secuencial = Algoritmo R.* Los tiempos son exclusivos del cálculo del promedio, es decir sin la implementación del cálculo de la pendiente.

Tabla 9
Tiempos de ejecución del cálculo de la pendiente en segundos.

No.	***Paralelo****	***Secuencial ****
1	152.997	18,980.123
2	118.227	19,239.420
3	110.750	19,072.866
4	144.833	19,062.900
5	117.464	- - -
6	110.922	- - -
7	119.408	- - -
8	112.213	- - -
9	116.431	- - -
10	123.883	- - -
11	116.588	- - -
12	105.754	- - -
13	104.737	- - -
14	107.094	- - -
15	106.100	- - -
16	105.527	- - -
17	106.259	- - -
18	107.574	- - -
19	186.938	- - -
20	106.475	- - -
21	108.533	- - -
22	107.605	- - -
23	105.589	- - -
24	101.515	- - -
25	105.845	- - -
26	101.417	- - -
27	103.677	- - -
28	103.997	- - -
29	103.380	- - -
30	104.419	- - -
Tiempos promedio	114.205	19,088.827

Nota: **Paralelo = Algoritmo CUDA, *Secuencial = Algoritmo R.*

5.4 Estimación de la capacidad de cálculo utilizada

El tiempo de la lectura de las imágenes (7.7GB) desde disco a la memoria de la CPU (de laldo del *Host*) tiene una duración promedio de 48.375 segundos, representando el 42% del tiempo total de la ejecución del algoritmo.

El proceso de copiar las imágenes en la memoria global (del lado del *Device*) tiene una duración promedio de 23.492 segundos, representando el 21% del tiempo total de la ejecución del algoritmo.

Los dos procesos anteriormente mencionados representan el 63% de la duración total del algoritmo (Tabla 10). Y fueron estimados a partir de la ejecución de 10 repeticiones de cada uno de los procesos mencionados (el registro de las 10 repeticiones se puede consultar en el Apendice).

Tabla 10

Tiempos promedio de los principales procesos dentro del algoritmo (segundos)

Proceso	***Tiempo***	***Porcentaje***
Lectura de imágenes desde disco.	48.375	42%
Copiar imágenes a la memoria global.	23.492	21%
Ejecucion del kernel*	42.338	37%
Duración total de ejecución del algoritmo en CUDA	114.205	100%

Nota: *Representa solo el cálculo del promedio y la pendiente.

Por lo anterior, se puede inferir que a una mayor segmentación de las imágenes al momento de leerlas desde disco (en caso de que la memoria disponible en la CPU fuese menor y no sea posible leerlas todas en una sola pasada), el tiempo de lectura incrementaría y por ende el tiempo de duración total del algoritmo también.

Otro hecho sumamente interesante, es que investigando el funcionamiento de la arquitectura de la GPU (de la tarjeta utilizada), para poder dimensionar cuánta capacidad de cálculo (3072 núcleos CUDA) realmente se estaba utilizando durante la ejecución de los *kernels*. Lo que se encontró es que solo un (aproximado) 4% de la capacidad de la tarjeta es involucrada. De los 24 multiprocesadores que conforman la capacidad total de la tarjeta (128 nucleos por multiprocesador), solo se está utilizando un multiprocesador (1 de 24); Por lo anterior, podemos concluir que una tarjeta con mayor número de núcleos, no representaría ninguna mejora en los tiempos ejecución del algoritmo aquí planteado (lo anterior tiene sustento en el funcionamiento de la arquitectura SIMT y las características propias de la tarjeta utilizada, visto en los apartados 3.4.2 y 3.4.5).

Cabe señalar, que la conjetura anterior deriva de la estrategia para al procesamiento de los datos en la GPU, planteada en este trabajo (visto en el capítulo *4.3 Procesamiento y generación de la salida en CUDA*). La cual solo gestiona 1758 hilos, que son los

estrictamente necesarios para efectuar los cálculos que se requieren (arreglo bidimensional de 342 x 1758, donde a cada hilo le es asignada una sola columna del arreglo). Pero, si en un futuro se modificara esta estrategia, se deberá valorar en su momento el número de hilos a gestionar, y quizá represente un incremento en la utilización de la capacidad de cálculo de la GPU.

Para finalizar, solo diremos que el aprovechamiento de la totalidad de la potencia de calculo de la GPU, solo se conseguiría bajo esquemas de cálculo que gestionarán el máximo número de hilos que soporta la tarjeta (24 multiprocesadores con capacidad de 2,048 hilos cada uno de ellos = 49,152 hilos), y la estrategia actual no requiere.

5.5 Conclusiones

5.5.1 Cálculo del promedio histórico del NDVI (2000-2014)

El cálculo del promedio histórico del NDVI en CUDA, resultó ser 69 veces más rápido que el procedimiento en R. Lo anterior se estimó con base en los tiempos promedio de la ejecución del algoritmo en CUDA y R (Tabla 10), en función de las 30 repeticiones presentadas en la Tabla 7. Con esto se confirma la tendencia del uso de tecnología GPU para cálculos intensivos o de grandes volúmenes de datos, por la impresionante reducción de tiempos de procesamiento que brindan (cabe señalar que estos tiempos fueron estimados sin la implementación del cálculo de la pendiente dentro del *kernel*).

Tabla 10
Tiempos Promedio del cálculo del promedio (segundos)

GPU	***CPU***
75.778	5, 271.553

El *speedup* es una métrica que representa el índice de rendimiento conocido como aceleración, y se calcula:

$$\textbf{\textit{Speedup}} = \frac{\text{Tiempo secuencial}}{\text{Tiempo paralelo}} = \frac{5{,}271.553}{75.778} = \mathbf{69.565}$$

5.5.2 Cálculo de la tendencia del NDVI (2000-2014)

El cálculo de la pendiente en CUDA, resultó ser 155 veces más rápido que el procedimiento en R. Lo anterior con base en los tiempos de ejecución promedio del cálculo desde CUDA y R (Tabla 11), estimados en función de las 30 y 4 repeticiones respectivamente (Tabla 8).

Tabla 11
Tiempos Promedio del cálculo de tendencia (segundos)

GPU	***CPU***
114.205	19,088.827

$$\textbf{\textit{Speedup}} = \frac{\text{Tiempo secuencial}}{\text{Tiempo paralelo}} = \frac{19{,}088.827}{114.205} = \mathbf{155.21}$$

5.6 Trabajo Futuro

- El Algoritmo tiene factibilidad para escalarse a otro tipo de operaciones con matriciales utilizadas con imágenes satelitales;
- Ampliar la gama de productos MODIS soportados hacia: oceanográficos, atmosféricos, etc., actualmente solo trabaja con productos de la Tierra (específicamente NDVI);
- Adecuar el algoritmo para que pueda ejecutarse en tarjetas NVIDIA de menor capacidad.

Referencias

Arquitectura Fermi. (2009). NVIDIA's Next Generation CUDATM Compute architecture: FermiTM. *NVIDIA*. Recuperado 21 octubre 2016 de: http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf

Arriaga, L. y Ortega, A (editores). 1988. La Sierra de la Laguna de Baja California Sur. Publicación no. 1. CIB, BCS.México. Recuperado 16 noviembre 2016: http://www.conabio.gob.mx/conocimiento/regionalizacion/doctos/rtp_001.pdf

Bivand, R., Keitt, T. & Rowlingson, B. (2015). *rgdal: Bindings for the Geospatial Data Abstraction Library. R package version 0.9-2*. Recuperada 21 octubre 2016 de: http://CRAN.R-project.org/package=rgdal

Bhujade, M.R. (2009). *Parallel Computing* (2da ed.). UK: New Age Science Limited.

Cheng, J., Grossman, M. & Mckercher, T. (2014). *Professional CUDA® C Programming.* Indianapolis, IN: John Wiley & Sons, Inc.

Cook, S. (2013). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Waltham, MA: ELSEVIER.

Crawley, M. J. (2013). *The R Book* (2da ed.). Recuperado 21 octubre 2016 de: http://www.bio.ic.ac.uk/research/mjcraw/therbook/index.htm

CUDA NVIDIA. (s.f.). *Plataforma de computación paralela CUDA*. Recuperado 21 octubre 2016 de: http://www.nvidia.com.mx/object/cuda_home_new_la.html

CUDA ToolKit. (s.f.)a. *CUDA ToolKit V8.0 - CUDA C Programming Guide*. Recuperado 21 octubre 2016 de: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz4OWa8mZC1

CUDA ToolKit. (s.f.)b. *CUDA Runtime API - Memory Management*. Recuperado 21 octubre 2016 de: http://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY_1g32bd7a39135594788a542ae72217775c

Cuéntame INEGI. (2016). *Bienvenidos a Cuéntame de México*. Recuperado 21 octubre 2016 de: http://cuentame.inegi.org.mx/monografias/informacion/bcs/default.aspx?tema=me&e=03

FileFormat.Info (s.f.) *TIFF File Format Summary*. Recuperado 21 octubre 2016 de: http://www.fileformat.info/format/tiff/egff.htm

GeForce. (s.f.). *GeForce GTX TITAN X*. Recuperado 21 octubre 2016 de: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x

GeoTIFF. (2000). *GeoTIFF Format Specification: GeoTIFF Revision 1.0*. Recuperado 21 octubre 2016 de: http://web.archive.org/web/20160403164508/http://www.remotesensing.org/geotiff/spec/geotiffhome.html

Greenberg, J. A. & Mattiuzzi, M. (2014). *gdalUtils: Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities. R package version 0.3.1*. Recuperado 21 octubre 2016 de: http://CRAN.R-project.org/package=gdalUtils

Hamilton, S. L. (2013). Introduction to Parallel Programing. USA: Autor.

Harris, M. (2014). *Maxwell: The Most Advanced CUDA GPU Ever Made*. Recuperado 21 octubre 2016 de: https://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/

Hijmans, R. J. (2015). *raster: Geographic Data Analysis and Modeling. R package version 2.3-40.* Recuperado 21 octubre 2016 de: http://CRAN.R-project.org/package=raster

HDF Group. (s.f.).Why *HDF?*. Recuperado 21 octubre 2016 de: https://www.hdfgroup.org/why-hdf/

Hori, Y., Stuhlberger, C. & Simonett, O. (2011). Desertification: a visual synthesis. *UNCCD - Zoi Environment Network*. Recuperado 21 octubre 2016 de: http://www.unccd.int/Lists/SiteDocumentLibrary/Publications/Desertification-EN.pdf

INEGI. (s.f.). *Imágenes del Territorio*. Recuperado 21 octubre 2016 de: http://www.inegi.org.mx/geo/contenidos/imgpercepcion/imgsatelite/elementos.aspx

Kerrisk, M. (2010). *THE LINUX PROGRAMMING INTERFACE: A Linux and UNIX® System Programming Handbook*. San Francisco, CA: No Starch Press, Inc.

Linux Mint (s.f.). *About Us*. Recuperado 21 octubre 2016 de: https://www.linuxmint.com/about.php

Liu, J., Feld, D., Xue, Y., Member, S., Garcke, J. & Soddemann, T. (2015). Multicoreprocessors and graphicsprocessingunitacceleratorsforparallelretrieval of aerosol opticaldepthfromsatellite data: implementation, performance, and energy efficiency. *IEEE*, 8(5), 2306–2317.

López Beltrán, M.A. (2014). *Integración de imágenes del sensor MODIS y cartografía temática para la simulación de modelos geoespaciales para obtener zonas propensas a desertificación en el Estado de Sinaloa, México.* (Tesis de Maestría inédita). Universidad Autónoma de Sinaloa. Sinaloa, México.

LP DAAC. (s.f.). *MODIS Overview*. Recuperado 21 octubre 2016 de: https://lpdaac.usgs.gov/dataset_discovery/modis

Mat-Opencv. (s.f.). *Basic Structures: Class Mat*. Recuperado 21 octubre 2016 de: http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#mat

Mendenhall, W., Beaber, R.J. & Beaver, B. M. (2010). *Introducción a la probabilidad y estadística*. México: CENGAGE Learning.

Muñoz Puelles V. (2015). *A LA VELOCIDAD DE LA LUZ (El joven Einstein*).México: Anaya.

MODIS Land. (s.f.). *MODIS Grids*. Recuperado 21 octubre 2016 de: https://modis-land.gsfc.nasa.gov/MODLAND_grid.html

OpenCV. (s.f.). *About*. Recuperado 21 octubre 2016 de: http://opencv.org/about.html

Patterson, D. A. & Hennessy, J. L. (2014) *Computer Organization and Design: THE HARDWARE /SOFWARE INTERFACE* (1era ed.). Waltham, MA: ELSEVIER.

Peng, R.D. & Leeuw, J. (2002). An Introduction to the .C Interface to R. *UCLA Department of Statistics.* Recuperado 21 octubre 2016 de: http://www.biostat.jhsph.edu/~rpeng/docs/interface.pdf

Peña, A. & Dreyfus, G. (2012). *La energía y la vida: Bioenergética* (3era ed.). México: Fondo de cultura económica. Recuperado 21 octubre 2016 de: http://www.fondodeculturaeconomica.com/subdirectorios_site/libros_electronicos/desde_la_imprenta/046092R/files/publication.pdf

Pérez, E. (s.f.). *OPENCV EN LINUX. Tutorial de OpenCV. LINUX*. Recuperado 21 octubre 2016 de: https://geekytheory.com/opencv-en-linux/

pkg-config. (s.f.). *Tutorials points: pkg-config - Unix, Linux Command*. Recuperado 21 octubre 2016 de: http://www.tutorialspoint.com/unix_commands/pkg-config.htm

Programa Biosfera Vizcaino. (2000). Programa de Manejo de la Reserva de la Biosfera El Vizcaíno (1era ed.). *Instituto Nacional de Ecología*. Recuperado 16 novimbre 2016 de: http://www.conanp.gob.mx/que_hacemos/pdf/programas_manejo/vizcaino.pdf

R Core Team. (2015). *R: A language and environment for statistical computing. R Foundation for Statistical Computing*. Recuperado 21 octubre 2016 de: http://www.R-project.org/

Sanders, J. & Kandrot, E. (2010). *CUDA by example: an introduction to general-purpose GPU programming.* Boston, MA: Addison-Wesley.

Solano, R., Didan, k., Jacobson, A. & Huete, A. (2010). MODIS Vegetation Index User's Guide: (MOD 13 Series). *The University of Arizona*. Recuperado 21 octubre 2016 de: https://vip.arizona.edu/documents/MODIS/MODIS_VI_UsersGuide_01_2012.pdf

TERRA. (s.f.). MODIS. Recuperado de: http://terra.nasa.gov/about/terra-instruments/modis

TIFF. (1992). *Revision 6.0.* Recuperado 21 octubre 2016 de: https://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf

UNCCD. (s.f.) *History UNCCD*, Recuperado 21 octubre 2016 de: http://www.unccd.int/en/about-the-convention/history/Pages/default.aspx

UN-SPIDER. (s.f.). *Data Application of the month: Vegetation indices*. Recuperado 21 octubre 2016 de: http://www.un-spider.org/links-and-resources/data-sources/daotm/daotm-vegetation

Vargas, J., Ramírez, I., Pérez, S. & Madrigal, J. (2008). *Física mecánica: Coceptos básicos y problemas* (1era ed.). Medellin, Colombia: ITM.

Weier, J. & Herring, D. (2000). *Measuring Vegetation (NDVI & EVI).* Recuperado 21 octubre 2016 de: http://earthobservatory.nasa.gov/Features/MeasuringVegetation/

Xu, D. , Kang, X., Qiu, D., Zhuang, D. & Pan, J. (2009). Quantitative Assessment of Desertification Using Landsat Data on a Regional Scale – A Case Study in the Ordos Plateau, China. *Sensors - Open Access*, 9, 1738-1753. doi: 10.3390/s90301738

Yang, Z., Zhu, Y. & Pu, Y. (Diciembre 2008). Parallel Image Processing Based on CUDA. *CSSE 2008*. Conferencia llevada a cabo en Wuhan, China.

Zhang, J., You, S. & Gruenwald, L. (noviembre 2011). Parallel quadtree coding of large-scale raster geospatial data on GPGPUs. En C.S. Jensen (Presidente), *ACM SIGSPATIAL GIS 201*1. Conferencia llevada a cabo en Chicago, IL.

Zhang, N., Wang, J. & Chen, Y. (Marzo 2010). Image parallel processing based on GPU. *ICACC 2010*. Conferencia llevada a cabo en Shenyang, Liaoning, China.

Zao, P. (2014). *Accelerate R Applications with CUDA*. Recuperado 21 octubre 2016 de: https://devblogs.nvidia.com/parallelforall/accelerate-r-applications-cuda/

Apéndices

A. Pre-procesamiento imagens MODIS (script en R)

```
# P R E - P R O C E S A M I E N T O   I M A G E N E S   M O D I S
library(raster) library(rgdal) library(gdalUtils)

#Leer los nombres de los arhivos HDF
setwd("/home/cuda01/R/analisis NDVI/hdf")
listaHDF<-list.files(pattern=glob2rx("*.hdf"))

#creo una segunda lista con los mismos nombres, pero modifico la extension a TIF
listaTIF<-gsub(".hdf",".tif",listaHDF)

#concateno la palabra NDVI_ al inicio de los nombres y extraigo el archivo de NDVI en formarto TIF
q<-seq(1,342,by=2)
for(i in q){
  gdal_translate(listaHDF[i],paste("NDVI_",listaTIF[i],sep=""),sd_index=1)
  gdal_translate(listaHDF[i+1],paste("NDVI_",listaTIF[i+1],sep=""),sd_index=1)
}

#creo una nueva lista con los nombres de los archivos TIF de NDVI que se extrajeron
listaTIF<-list.files(pattern=glob2rx("*.tif"))

#Creo los raster base o iniciales, y apilo los 342 raster de cada cuadrante
h07v06<- raster(listaTIF[1])
h08v06<-raster(listaTIF[2])
p<-seq(1,342,by=2)
for(i in p){
  h07v06<-stack(h07v06,raster(listaTIF[i]))
  h08v06<-stack(h08v06,raster(listaTIF[i+1]))
}

```

```
#recorto  las regiones de interes quitando zonas del oceano pacifico o estado de Sonora
h07v06<-crop(h07v06,extent(-11398371, -11119505, 2500172, 3300754))
h08v06<-crop(h08v06,extent(-11119505, -10991172, 2800000, 3300754))

#unir los 2 cuadrantes, finalmente resultan en 342
bcs<-merge(h07v06,h08v06)
# escalo los valores entre -2 y 1
bcs<-bcs/100000000

#cambiamos la proyeccion de Sinusoidal a WGS84
proyLongLat<-"+proj=longlat +datum=WGS84 +no_defs"
refGeo<-projectExtent(bcs,crs=proyLongLat)
bcs<-projectRaster(bcs,refGeo)

#Grabarlos 342 archivos NDVI en disco en formato GeoTiff
s<-seq(1,342,by=1)
for (i in s){
  writeRaster(bcs[[i]], filename = paste("NDVI_",paste(i,".tif",sep="") ,sep=""), format = "GTiff", overwrite
= TRUE)
}
```

B. Algoritmo de CUDA (version final)

```
/*****************************************************************************************
 * Programa: Lee imagenes MODIS (formato GeoTIFF) de NDVI, a partir de un directorio        *
 *           especificado, para calcular el   promedio y la pendiente de c/u de los pixeles *
 *           de las imagenes que conforman la serie dada. El resultado lo graba en disco en *
 *           dos archivos (CSV).                                                            *
 * Autor: Claudia Martinez Vazquez  (klauditha@gmail.com)                                   *
 * Fecha:26 de febrero 2016                                                                 *
 * Ultima actualizacion: 29/octubre/2016                                                        *
 *****************************************************************************************/


/* LIBRERIAS CUDA */
#include <cuda_runtime.h>
#include <curand_kernel.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h> // fecha  y hra
#include <stdlib.h>

/* LIBRERIAS DE OPENCV */
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include "opencv2/imgproc/imgproc.hpp"
#include <dirent.h>
#include <stdexcept>


using namespace cv;
using namespace std;


#define THR 879
#define BLK 2
#define REN 3455
#define COL 1758
#define TOT_IMG 342

/* DATOS DE LAS IMAGENES PROCESAR
```

```
	879 X 2 = 1758 hilos
	1758 columnas
	3455 renglones
	6,073,890 pixeles totales
	342 imagenes (15 años 2000 - 2014) <--serie de tiempo*/


/**************** kernel que calcula el promedio y la pendiente de cada pixel ***********/
__global__ void calculo(float * matriz_filas, float* promedio, float * pendiente, float * SSXY)
{
	int tid = threadIdx.x + blockIdx.x * blockDim.x;//direccion fisica del hilo

	/************  PROMEDIO **************/
	/* regresa con el promedio de 1 renglon a la vez, por ejemplo: la fila 1 pero de todas las imagenes (342),
		sera lanzado 3455 veces (tantos renglones existen en una imagen)*/
    promedio[tid] = 0;
	float x_promedio= 0;
	int v =0;
	float pixel= 0;

	for (int r = 0;r < TOT_IMG ;r++)//342 filas, una por cada imagen
	{
			pixel = matriz_filas[r* (THR*BLK) + tid]; /* almacena el valor del pixel en turno */

			if(pixel < -0.2 ) //  contabilizo los NA, para despues quitarlos al momento de calcular el promedio
			{
				v++;
			}
			else
			{
				promedio[tid] = promedio[tid] + pixel;

			}

			x_promedio = x_promedio + (r+1);
	}

	x_promedio = x_promedio / TOT_IMG;
	promedio[tid] =promedio[tid] / (TOT_IMG-v);//la "v" es para restar los NA que haya encontrado

	/************ MODELO DE REGRESION ***********/
	pendiente[tid] = 0;
	float SSX = 0;
```

```
	SSXY[tid] = 0;
	float s=0;

	for (int r = 0;r < TOT_IMG ;r++)//342 filas, una por cada imagen
	{
			pixel = matriz_filas[r* (THR*BLK) + tid];
			s = ( (r+1) - x_promedio )*( (r+1) - x_promedio );//representa elevar al cuadrado
			SSX = SSX + s;

			if( pixel > -0.2 )
			{

				SSXY[tid] = SSXY[tid] +( (pixel - promedio[tid]) * ((r+1) -  x_promedio) );
			}

	}
	pendiente[tid] = SSXY[tid] / SSX;
}

int main(void)
{

	cudaEvent_t tiempo_ini,tiempo_fin;
	cudaEventCreate(&tiempo_ini);
	cudaEventCreate(&tiempo_fin);
	cudaEventRecord(tiempo_ini,0);

	/**********************  DECLARACIONES **********************/
	float tiempo_ejec;
	float* promedioD=NULL;
    float* pendienteD=NULL;
    float* ssxyD=NULL;
    size_t pitch;
	float* matriz_filasD=NULL;
	int miPID;
	char lineaShell[256];
	int x=1;
	FILE * arch;

	vector<cv::String> listado_archivos;
	vector<cv::Mat> listado_imagenes;
	vector<int> parametros;
	Mat matrizPromedio;
```

```
	Mat matrizPendiente;
	cv::String archivo_prom="promedioNDVI.csv";
	cv::String archivo_pend="pendienteNDVI.csv";


	/********************  EVIAR QUE LINUX (OOM) TERMINE MI PROCESO POR EL CONSUMO EXCESIVO DE RECURSOS
*********************************/
	miPID = getpid();
	sprintf(lineaShell,"echo -17 > /proc/%d/oom_adj",miPID);
	system(lineaShell);



	/******************** LEER IMAGENES  ************************/
	cv::String rutaDirectorio("NDVI_*.tif"); //patron del nombre de las imagenes a leer
	cv::glob(rutaDirectorio,listado_archivos,true); // guarda el listado de nombres que coinciden con el patron dentro de
"listado_archivos"

	for (size_t k=0; k < listado_archivos.size(); ++k)
	{
			cv::Mat imagen = cv::imread(listado_archivos[k], CV_LOAD_IMAGE_ANYDEPTH);
			cv::Mat imagenFloat;
			imagen.convertTo(imagenFloat,CV_32F);

		if (imagenFloat.empty())
		{
			printf("NO pudo leer imagen %s \n",listado_archivos[k].c_str () );
			continue;
		}
		else
		{
			try
			{
					if(x == 1 )
					{
						matrizPromedio = imagenFloat.clone();/* creo la matriz de salida donde se guadaran
los RESULTADOS del promedio*/
						matrizPendiente = imagenFloat.clone();/* creo la matriz de salida donde se guadaran
los RESULTADOS del promedio*/
					}

					listado_imagenes.push_back(imagenFloat);
```

```
                                if(! listado_imagenes[k].data )
                                {
                                        printf("No se pudo abrir o encontrar la imagen %s  \n",listado_archivos[k].c_str ()
);
                                        return -1;
                                 }

                        //      printf("\n %s", listado_archivos[k].c_str ());

                }
                catch (std::runtime_error & ex)
                {
                        fprintf(stderr, "Exception converting image to PNG format: %s\n", ex.what());
                        return 1;
                }
                x++;
        }
    }

/************************** INVOCANDO EL KERNEL  *******************************/
    for(int r = 0; r < REN; r ++ )//renglones a recorrer 3455 X Imagen
    {
        /*gestionamos memoria a los apuntadoreas del DEVICE (mem global) */
        cudaMallocPitch((void**) &matriz_filasD,&pitch,COL*sizeof(float),TOT_IMG);
        cudaMalloc((void**)&promedioD, sizeof(float)*COL);
        cudaMalloc((void**)&pendienteD, sizeof(float)*COL);
        cudaMalloc((void**)&ssxyD, sizeof(float)*COL);


        /* recorrido para llenar matriz_filasD del device con los valores de la fila "n" de c/u de las 342 imagenes */

        for(size_t i = 0; i < listado_archivos.size(); i++)
        {
                cudaMemcpy((matriz_filasD+(i*COL)),listado_imagenes[i].ptr<float>(r), sizeof(float)*COL,
cudaMemcpyHostToDevice);

        }

        cudaThreadSynchronize();
        /*invocacando el KERNEl */
        calculo<<<BLK,THR>>>(matriz_filasD,promedioD,pendienteD,ssxyD);
        cudaThreadSynchronize();
```

```
          /*copiar desde el DEVICE  a la imagen de salida, la fila en turno que se calculo el promedio */
            cudaMemcpy(matrizPromedio.ptr<float>(r),promedioD, sizeof(float)*COL, cudaMemcpyDeviceToHost);
            /*copiar desde el DEVICE  a la imagen de salida, la fila en turno que se calculo la Pendiente */
            cudaMemcpy(matrizPendiente.ptr<float>(r),pendienteD, sizeof(float)*COL, cudaMemcpyDeviceToHost);


            //libera memoria
       cudaFree(promedioD);
            cudaFree(pendienteD);
            cudaFree(ssxyD);
            cudaFree(matriz_filasD);
      }


      //detener el timer e imprimir la duracion del proceso
      cudaEventRecord(tiempo_fin,0);
      cudaEventSynchronize(tiempo_fin);
      cudaEventElapsedTime(&tiempo_ejec,tiempo_ini,tiempo_fin);

   printf("\n\n Tiempo %4.6f en miliseg \n",tiempo_ejec);
      cudaEventDestroy(tiempo_ini);
      cudaEventDestroy(tiempo_fin);

      /*
    // despliega la imagen de salida del promedio
      namedWindow( archivo_prom,CV_WINDOW_NORMAL);//la imagen es muy grande y no cabe en la pantalla en su tamaño original

      imshow(archivo_prom, matrizPromedio);
      waitKey(0);

      // despliega la imagen de salida del calculo pendiente del modelo de regresion
      namedWindow( archivo_pend,CV_WINDOW_NORMAL);//la imagen es muy grande y no cabe en la pantalla en su tamaño original

      imshow(archivo_pend, matrizPendiente);
      waitKey(0);
      */

/************************** GRABA EL ARCHIVO DE SALIDA DEL PROMEDIO (CSV) *******************************/
      arch = fopen("promedio.csv", "w");
      for (int r = 0; r < REN; r++)
      {
            for (int c = 0; c < COL;c++)
            {
```

```
                fprintf(arch, "%f,", matrizPromedio.at<float>(r,c));
            }

            fprintf(arch, "\n");
        }

        fclose(arch);


/************************** GRABA EL ARCHIVO DE SALIDA DE LA PENDIENTE (CSV) ******************************/
        arch = fopen("pendiente.csv", "w");
        for (int r = 0; r < REN; r++)
        {
            for (int c = 0; c < COL;c++)
            {
                fprintf(arch, "%f,", matrizPendiente.at<float>(r,c));
            }

            fprintf(arch, "\n");
        }

        fclose(arch);

        return 0;

}
```

C. Funcion *wrapper* (script de R invocando codigo de CUDA)

```
ndviGPU <- function(v)
{

  if(!is.loaded("ndviGPU "))
     dyn.load("sumaVectorGPU.so")#carga el archivo que contiene la funcion de CUDA

        L <- as.integer(length(v))#calculo longitud del vector entrada
  mode(v) <- "single" #vector entrada
        s <- single(1) #declaro vector salida de precision simple

  ListaResultados <- .C("kernel_suma",v,L,salida=s) #ejecuta la funcion de CUDA

  s = ListaResultados$salida

  attr(s,"Csingle") <- NULL #Me aseguro que el formato del resultado sea NUMERIC = float de lado de R

  return (s)
}
```

D. Codigo de CUDA (invocado desde funcion *wrapper* de R)

```
#include <cuda_runtime.h>
#include <curand_kernel.h>

#include <stdio.h>
#include <stdlib.h>

#include <R.h>
#include <Rinternals.h>
#include <R_ext/Rdynload.h>

#define THR 1
#define BLK 1

//KERNEL de cuda
__global__ void suma(float *v, float *r, int L){

        for(int h=0; h< L; h++)
       {
         r[h] = v[h] + h;

       }

}


extern "C" void ndviGPU(float *v, int *l, float *r)
{

    float size = (*l) * sizeof (float);//calculando el tamaño del vector Entrada
       float *dv, *dr; //variables del lado DEVICE

       Rprintf("VECTOR ENTRADA \n");//imprimiendo en consola de R los valores del vector de Entrada
       for (int i =0;i< *l;i++)
       {
              Rprintf("v[%d] = %.8f \n ", i,*(v+i));
              //printf("v[%d] = %.8f \n ", i,*(v+i));
       }

       cudaMalloc( (void**)&dv, size ); //gestionamos memoria del lado del DEVICE
```

```
	cudaMalloc( (void**)&dr, size );
	cudaMemcpy( dv, v, size, cudaMemcpyHostToDevice); //Copiamos el vector recibido hacia el DEVICE

	suma<<<BLK,THR>>>(dv, dr,*l); //ejecutamos el KERNEL

	cudaMemcpy(r, dr, size, cudaMemcpyDeviceToHost); //copiamos el resultado del DEVICE al HOST
	cudaFree(dv);
	cudaFree(dr);
	cudaThreadExit ();// salir del contexto de CUDA, necesario para evitar violaciones de segmento con salidas de R

	Rprintf("\n VECTOR SALIDA \n");//imprimiendo en consola de R los valores del vector de Salida

	for (int i =0;i< *l;i++)
	{
		Rprintf("r[%d] = %.8f  \n", i,*(r+i));
		//printf("r[%d] = %.8f  \n", i,*(r+i));
	}

}

  // DLL scaffolding for R
  R_CallMethodDef callMethods[] = {
    {"ndviGPU", (DL_FUNC)&ndviGPU, 3},
    {NULL, NULL, 0}
  };

  void R_init_myLib (DllInfo *info) {
    R_registerRoutines (info, NULL, callMethods,NULL, NULL);
  }
```

E. Lector de archivos TIF (lenguaje C)

```
#include <stdio.h>
#include <stdlib.h>


int main()
{


            double *imagen;// Host
            FILE *entrada;

            int i=0,j=0;
            long pos =0;
            long limite = 0;
            double pixel = 0;
            long valor=0;
            int n=1;

            /// ============================================
            /// =========== Cabecera del entrada ============
            /// ============================================
            char byteOrden1 = 0;
            char byteOrden2 = 0;
            int identificadorTIF = 0;
            int despl_IFD = 0;
            int despl_IFD_n = 0;
            int NoEtiquetas = 0;
            int idTag = 0;
            int typeData = 0;
            int countData = 0;
            int despl_datos_tag = 0;
            /// ============================================
            /// ============== Datos de la imagen TIF ==============
            /// ============================================

            int imgWidth = 0;
            int imgHeight = 0;
```

```
        int countData273 = 0;
        int offset273 = 0;
        long valor273 =0;
        int typeData273 = 0;

        int countData279 = 0;
        int offset279 = 0;
        long valor279 =0;
        int typeData279 = 0;

        int countData278 = 0;
        int offset278 = 0;
        long valor278 =0;
        int typeData278 = 0;

        /// ======================== C A B E C E R A ============================

//      entrada = fopen ("NDVI.tif","rb");
        entrada = fopen ("NDVI2000_1.tif","rb");


        printf("*********** LECTOR TIFF ***************\n\n ");
        printf("Cabecera TIFF \n\n");

        //Bytes 0-1: The byte order used within the file. Legal values are:
        // "II" (4949.H) "MM" (4D4D.H)
        byteOrden1 = fgetc (entrada);
        byteOrden2 = fgetc (entrada);
        printf ("\tByte de orden: %c",byteOrden1);
        printf ("%c\n",byteOrden2);

        //Bytes 2-3 An arbitrary but carefully chosen number (42) that further
        //identifies the file as a TIFF file.
        fread(&identificadorTIF, 2, 1, entrada);
        printf("\tArchivo TIF:%d\n",identificadorTIF);

        //Bytes 4-7 The offset (in bytes) of the first IFD(Image File Directory)
        fread(&despl_IFD, 4, 1, entrada);
        printf("\tOffset of the first IFD:%d bytes\n\n ",despl_IFD);

        //==================First IFD  (Image File Directory )==================

        printf("I F D  (Image File Directory )\n\n");
```

```
fseek(entrada,despl_IFD,SEEK_SET);//me posiciono al inicio del primer IFD

//IFD consists of a 2-byte count of the number of directory entries
fread(&NoEtiquetas, 2, 1, entrada);
printf("\tNumber of directory entries: %d\n",NoEtiquetas);

//followed by a sequence of 12-byte field entries,
fseek(entrada,(NoEtiquetas*12),SEEK_CUR);

//followed by a 4-byte offset of the next IFD (or 0 if none)
fread(&despl_IFD_n, 4, 1, entrada);
printf("\tSiguiente IFD:%d bytes\n\n",despl_IFD_n);

//Regreso el apuntador al inicio de Directory entries (Tag's)
fseek(entrada,10,SEEK_SET);

//======================== Directory Entries (Tag's) ========================

while(1==1)
{
	printf("Directory entries\n\n");

	for(i =0;i<NoEtiquetas; i++)
	{

		//Bytes 0-1 The Tag that identifies the field.
		fread(&idTag, 2, 1, entrada);
		printf("\tDirectory entrie-------->-%d\n",i+1);
		printf("\tId Tag: %d: \n",idTag);

		//Bytes 2-3 The field Type.
		fread(&typeData, 2, 1, entrada);  // leo el tipo de dato de la etiqueta
		printf("\tTipo de dato(1-Byte, 2-Ascii, 3-short, 4-Long, 5-Rational): %d \n",typeData);

		//Bytes 4-7 The number of values, Count of the indicated Type
		fread(&countData, 4, 1, entrada);
		printf("\tCantidad de datos: %d \n",countData);

		//Bytes 8-11 The Value Offset, the file offset (in bytes) of the Value for the field
		fread(&despl_datos_tag, 4, 1, entrada);
		printf("\tOffset of the Value for the field: %d  \n\n",despl_datos_tag);

		switch(idTag )
```

```
                        {
                                case 256://ImageWidth
                                        imgWidth = despl_datos_tag;
                                        break;
                                case 257: //ImageLength
                                        imgHeight = despl_datos_tag;
                                        break;
                                case 273:    //StripOffsets

                                       countData273 = countData;
                                       offset273 = despl_datos_tag;
                                       typeData273 = typeData;
                                       break;
                                case 278:    //RowsPerStrip

                                       countData278 = countData;
                                       offset278 = despl_datos_tag;
                                       typeData278 = typeData;
                                       break;
                                case 279://StripByteCounts
                                       countData279 = countData;
                                       offset279 = despl_datos_tag;
                                       typeData279 = typeData;
                                       break;
                        }

                }//cierra for, termino de leer las etiquetas (tag) del 1er IFD

                getchar();
                pos = ftell (entrada);/* guardo posicion actual, antes de leer datos de la imagen, porque aun nose si
hay otros IFD*/


                //******************* LEER DATOS IMAGEN ******************
                imagen = (void*) malloc(imgWidth*imgHeight);

                /*  StripOffsets > 0 && StripByteCounts > 0 */
                if (countData273 >0 && countData279>0)
                {
                //     FILE *fileCompress=fopen("tifCompressed.bin","wb");

                        for(i=0;i < countData273;i++) /* StripOffsets contiene el inicio de cada strip, contando desde el
inicio del archivo*/
```

```
		{
			//Leer cada elemento del arreglo de StripOffsets
			//cada elemento tiene un tipo especificado en typeData (1-Byte, 2-Ascii, 3-short, 4-Long,
5-Rational)
			// todas las posiciones son desde el inicio del archivo
			fseek(entrada,(offset273 +(i*4)),SEEK_SET); // aqui usamos 4 bytes porque fue typeData =
Long = 4 bytes
			fread(&valor273, 4, 1, entrada);

			//Leer cada elemento del arreglo StripByteCounts, numero de bytes que contiene cada strip

			fseek(entrada,(offset279 +(i*4)),SEEK_SET);
			fread(&valor279, 4, 1, entrada);

			printf("Strip[%d]-->offset = %d  --->bytes = %d \t",i+1,valor273,valor279);
			getchar();
			char *bytes[countData279];

			//Comienzo a leer los pixeles de la imagen, c/u de tamaño double (8 bytes = 64 bits)
			//avanzo los bytes indicados por el primer elemento de StripOffsets,
			// desde el inicio del archivo.
			fseek(entrada,valor273,SEEK_SET);
			fread(&bytes, valor279, 1, entrada); //Leer pixeles comprimidos (StripByteCounts)

			/*Writing data to file*/
		  // fwrite(bytes,sizeof(bytes), 1, fileCompress);
		}
		/*Closing File*/
//		fclose(fileCompress);

}

//**********************************************************************

//devuelo el apuntador donde estaba antes de leer los datos de las etiquetas
fseek(entrada,pos,SEEK_SET);

getchar();

if(despl_IFD_n>0)//en caso de que hubiese mas de 1 IFD
{
	printf("I F D  (Image File Directory )\n\n");
	//me posiciono en el siguiente IFD, respecto al inicio del archivo
```

```
                        fseek(entrada,despl_IFD_n,SEEK_SET);

                        //2-byte count of the number of directory entries
                        despl_IFD=despl_IFD_n+2;//calculo el inicio de directory entries (Tag's)
                        fread(&NoEtiquetas, 2, 1, entrada);
                        printf("\tNumber of directory entries: %d\n",NoEtiquetas);

                        //followed by a sequence of 12-byte field entries,
                        fseek(entrada,(NoEtiquetas*12),SEEK_CUR);
                        //followed by a 4-byte offset of the next IFD (or 0 if none)
                        fread(&despl_IFD_n, 4, 1, entrada);
                        printf("\tSiguiente IFD:%d bytes\n\n",despl_IFD_n);

                        //Regreso el apuntador justo al inicio de Directory entries (Tag's)
                        fseek(entrada,despl_IFD,SEEK_SET);
                }
                else
                {
                        break;
                }
        }




//fwrite(imagen,biWidth*biHeight,1,salida);
        //cierra archivos
        fclose(entrada);

getchar();

        return 0;

}
```

F. Tiempos principales procesos dentro del Algoritmo

Tiempos de ejecución de los principales procesos dentro del algoritmo (segundos)

No.	Leer imágenes desde disco (memoria CPU)	Copiar imagenes al Device
1	49.749	24.104
2	48.691	23.013
3	48.262	22.697
4	48.097	23.164
5	48.145	22.420
6	48.245	23.061
7	48.007	23.100
8	48.136	22.614
9	48.170	25.931
10	48.247	24.812