UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

CENTRO DE GEOINFORMÁTICA Y SEÑALES DIGITALES

Algoritmos paralelos para el análisis del terreno sobre información raster en plataformas de cómputo heterogéneas y no dedicadas

Tesis presentada en opción al título de Máster en Informática Aplicada

Autora: **Ing. Grethell Castillo Reyes**

Tutor: **Dr. C. Liesner Acevedo Martínez**

La Habana, Diciembre 2015

DEDICATORIA

A mi papá Emilio, porque eres mi meta, porque me lo has dado todo...

A mi "esposo" Guillermo, porque eres la mitad de este esfuerzo compartido...

A mi hermano David, porque "no tienes ni idea" sobre que es esto, pero igual dices que soy un "crack". A mi hermano Jorgito, porque siempre te he sentido orgulloso de mi...

AGRADECIMIENTOS

A mi tutor de tesis Liesner Acevedo Martínez, porque aprendí muchísimo desde la primera clase, gracias por ese curso "Programación para multinúcleos", por todas las correcciones a tiempo y sobre todo...gracias por ponerme metas más altas cada día.

Al Dr. Valle, porque en algún momento fue el precursor de este resultado, y quien también contribuyó a mi crecimiento profesional y humano en general.

A mi amigo Armando Batista Piñeda, a quien veo como un tutor más, agradezco infinitamente su ayuda incondicional para lograr el perfeccionamiento de la investigación.

A los miembros del grupo de investigación "Procesamiento de Señales y Geoinformación", al MSc. Romanuel Ramón Antunez, al MSc. Eddy Dangel Quesada y al Dr. Rafael Arturo Trujillo, por sus recomendaciones certeras.

A Reisel y Osvel, por todos los trastazos con T-Arenal.

A los muchachos de GeneSIG, Aplicativos y del Centro Geoinformática y Señales Digitales, gracias por estar pendientes cada día. En especial a Abel, Pacheco, Dianita.

Mi eterno agradecimiento a toda mi familia y amigos por siempre estar pendientes...esto también va para ustedes...

A todos los que contribuyeron al desarrollo de la investigación, mi más sincero agradecimiento.

DECLARACIÓN JURADA DE AUTORÍA

Yo Grethell Castillo Reyes, con carné de identidad 89120523032, declaro que soy la autora principal del resultado que expongo en la presente memoria titulada "Algoritmos paralelos para el análisis del terreno sobre información *raster* en plataformas de cómputo heterogéneas y no dedicadas", para optar por el título de Máster en Informática Aplicada.

El presente trabajo fue desarrollado individualmente en el transcurso de los años 2013-2015.

Finalmente declaro que todo lo anteriormente expuesto se ajusta a la verdad, y asumo la responsabilidad moral y jurídica que se derive de este juramento profesional.

Y para que así conste, firmo la presente declaración jurada de autoría en La Habana a los 11 días del mes de Diciembre del año 2015.

_____

Firma de la Maestrante

Resumen

El modelo de datos *raster* es la base para el cálculo de varios parámetros de caracterización de la topografía del terreno, tales como la pendiente, la rugosidad y el sombreado. El cálculo de estos parámetros resulta de vital importancia para los Sistemas de Información Geográfica, debido a que sus aplicaciones tienen impacto directo en la toma de decisiones a la hora de evaluar las características del terreno en situaciones de emergencia. La mayoría de los algoritmos utilizados para realizar este tipo de cálculos son complejos desde un punto de vista computacional y dependen del tamaño y la resolución de los modelos de entrada. Uno de los principales retos en este sentido consiste en el diseño e implementación de algoritmos paralelos que utilicen todo el potencial de procesamiento de las arquitecturas de cómputo paralelo.

El aporte fundamental de la investigación consiste en la aplicación de un conjunto de técnicas de computación paralela para disminuir el tiempo de respuesta de los algoritmos para el análisis del terreno sobre información geográfica almacenada en formato *raster*, dirigidas a la utilización de varias arquitecturas de cómputo paralelo: las arquitecturas multinúcleos, la potencia de cálculo de las Unidades de Procesamiento Gráfico y los beneficios de los sistemas distribuidos por cómputo voluntario, teniendo en cuenta las limitaciones del entorno computacional de las entidades dedicadas al procesamiento y análisis de la información geospacial y la heterogeneidad entre diferentes plataformas de cómputo. Los experimentos llevados a cabo muestran buenos resultados de manera general.

Palabras claves: análisis del terreno, heterogeneidad, procesamiento paralelo, *raster*.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El vertiginoso desarrollo de las nuevas tecnologías satelitales de alta resolución, la teledetección y el incremento de la fotogrametría terrestre y aérea para la digitalización de la superficie, han posibilitado la obtención de imágenes que contienen información geográfica de cada vez mayor resolución. La información contenida en las imágenes satelitales tiene aplicaciones en varias ramas, entre las que se destacan: la agricultura, para la medición de campos e identificación de suelos y cultivos; la hidrología, para el modelado de la erosión del suelo y la extracción de redes de drenaje; la minería, para el monitoreo de recursos naturales y la exploración minera; la cartografía, mediante los Sistema de Información Geográfica (SIG) para el análisis de la información representada; y de manera general en la vigilancia del medio ambiente [Romero, 2006].

La información obtenida mediante estas tecnologías es almacenada y manipulada a través de un modelo de datos denominado modelo *raster*. Su estructura constituye normalmente una matriz bidimensional A con n filas y m columnas, que conforman n x m celdas, donde $n, m \in \mathbb{N}$. Cada celda $a_{ij} \in A$ donde $0 \leq i \leq n$ y $0 \leq j \leq m$, representa un valor numérico que toma valores en $\mathbb{R}$ y que describe una característica del terreno en un punto, por ejemplo la elevación [Smith y col., 2013]. La dimensión del área del terreno cubierta por una celda, determina la resolución espacial con la que se representan los datos.

El modelo *raster*, entre otras aplicaciones, es utilizado en los SIG para realizar análisis que permitan caracterizar la topología del terreno mediante el cálculo de sus principales atributos, como es el caso de la pendiente, la rugosidad y el sombreado del terreno. Estos parámetros son usados como base para la formulación de varios tipos de modelos ambientales, tal es el caso de los modelos hidrológicos propuestos por [Wallis y col., 2009, Tesfa y col., 2011], el estudio de la inclinación de superficies y determinación de zonas de poca pendiente favorables para la construcción [Rodríguez y Suárez, 2010] o zonas de mucha pendiente que determinan erosión o deslizamientos de tierra [Biesemans y col., 2000]. Permiten además, percibir la profundidad de una superficie en tres dimensiones [Jenny, 2001] y determinar la variabilidad de un relieve en un entorno determinado [Seitavuopio y col., 2005].

La familia de algoritmos utilizada para calcular estos parámetros, llamada algoritmos de análisis focal dentro del álgebra de mapas, tiene una complejidad temporal dependiente de las dimensiones de la matriz *raster*. La disponibilidad de datos de la superficie del terreno almacenados en formato *raster* ha ido en aumento sostenido y cada vez con un mayor nivel de resolución espacial y precisión. Este elemento es directamente proporcional al tamaño que pueden llegar a alcanzar dichos datos, pues a medida que aumenta la resolución y precisión, mayor tamaño

tendrá el modelo utilizado para su almacenamiento. Los proyectos GeoEye[1] y WorldView[2] son un ejemplo de ello, equipados con sensores con la capacidad de tomar imágenes de la superficie terrestre con un nivel de resolución de 50cm, para generar una base digital de datos topográficos de la Tierra en alta resolución [Farr y col., 2007].

En pruebas preliminares realizadas para analizar el comportamiento de los algoritmos en función del aumento de la resolución de los datos, se realizó el cálculo del sombreado del terreno utilizando un *dataset* que constituye un Modelo Digital de Elevación (MDE) generado por el proyecto *Shuttle Radar Topography Mission* (SRTM), que cubre el 80 % de la región Sudamericana con un nivel de resolución de 500m y un tamaño de 86 404 x 28 804 píxeles. Como resultado, se obtuvo un tiempo de procesamiento de 1 122,8 segundos (18 minutos) en una PC HP Pavilion g4-1174la con 4 GB de memoria RAM y un procesador AMD Dual-Core A4-3300M APU a 1.90 GHz.

Para calcular un parámetro como el sombreado, la resolución de 500m se considera muy baja. Una resolución más adecuada, aunque no ideal, pudiera ser a 10m. La misma región a esta resolución tendría dimensiones de $50n$ x $50m$, por lo que se espera un tiempo de procesamiento de $50 * 50 * 18$ minutos aproximadamente (45 000 minutos o 31 días), lo que se considera un tiempo inaceptable.

Teniendo en cuenta esto, varios autores [Zhan y Qin, 2012, Qin y col., 2014a], se enfocan en la utilización de sistemas distribuidos para reducir el tiempo de ejecución de los algoritmos para el análisis del terreno. Las soluciones propuestas son eficientes respecto a las variantes secuenciales, sin embargo, requieren que se cuente con una infraestructura de cálculo dedicada a ese fin (por ejemplo: un clúster o multiprocesador). Este tipo de arquitectura es costosa y en ocasiones no accesible a organizaciones y universidades de bajo presupuesto.

Además, actualmente una gran parte de las organizaciones, entre ellas las entidades dedicadas al análisis y visualización de la información geográfica donde suelen realizarse este tipo de cálculos, se caracterizan por poseer una infraestructura de red compuesta por múltiples estaciones con características heterogéneas y servidores de "bajas prestaciones". Para entornos como este, han surgido tecnologías que permiten utilizar los recursos de hardware disponibles en las organizaciones, para potenciar el procesamiento con el fin de reducir el costo computacional de las aplicaciones, tal es el caso de la computación grid y la computación voluntaria [Anderson y Fedak, 2006].

Por otro lado, con la llamada "Revolución Multinúcleo" [Herlihy, 2007], las computadoras personales cuentan con arquitecturas paralelas que, bien aprovechadas, pueden usarse para disminuir los tiempos de respuesta de las aplicaciones. Por otro lado, las capacidades de cálculo y almacenamiento de las Unidades de Procesamiento Gráfico (GPU) se han incrementado significativa-

[1]Disponible para consulta en: http://www.landinfo.com/geo.htm
[2]Disponible para consulta en: http://www.digitalglobe.com

mente, posibilitado por el alto desempeño de su arquitectura paralela con múltiples procesadores especializados en la realización de cálculos matemáticos a bajo costo [Nvidia, 2011].

Teniendo en cuenta los elementos mencionados anteriormente, se arriba al siguiente **problema de investigación**: ¿Cómo disminuir el tiempo de respuesta de los algoritmos para el análisis del terreno sobre información geográfica almacenada en formato *raster*, en arquitecturas de cómputo heterogéneas y no dedicadas?

El **objeto** sobre el que se enfoca el **estudio** desde el punto de vista teórico-metodológico con vistas a la solución del problema planteado, consiste en el procesamiento paralelo de datos *raster* en SIG, enfocándose en las técnicas de computación paralela aplicadas al procesamiento de datos *raster* en arquitecturas de cómputo heterogéneas y no dedicadas, lo cual constituye el **campo de acción** de la investigación.

Con el propósito de brindarle una solución al problema planteado se define como **objetivo general** de la investigación: Diseñar algoritmos paralelos que hagan uso eficiente de las arquitecturas paralelas heterogéneas y no dedicadas para disminuir el tiempo de respuesta en el proceso de análisis del terreno sobre información geográfica almacenada en formato *raster*.

En el marco de la presente investigación se propone una nueva variante para el procesamiento de datos *raster*, en concreto para el análisis del terreno, partiendo de la siguiente **hipótesis**: El diseño de un conjunto de algoritmos paralelos que hagan uso eficiente de las arquitecturas paralelas heterogéneas y no dedicadas para el análisis del terreno sobre información geográfica almacenada en formato *raster*, incidirá directamente en la disminución del tiempo de respuesta requerido para el procesamiento de dicha información.

Para darle cumplimiento al objetivo antes planteado se han definido las siguientes **tareas de investigación**:

- Caracterización del modelo de datos *raster* y de los algoritmos secuenciales para el análisis del terreno.
- Caracterización de las variantes de solución existentes en el tratamiento del problema planteado mediante la realización de un estudio de los referentes teórico - prácticos que preceden la realización de esta investigación.
- Diseño de un conjunto de algoritmos paralelos para el análisis del terreno sobre datos *raster*, haciendo uso de diferentes arquitecturas de cómputo.
- Desarrollo de una herramienta informática para el análisis del terreno en la Biblioteca de Abstracción de Datos Geoespaciales (GDAL) que haga uso de los algoritmos propuestos.
- Validación experimental a través de la herramienta implementada, para establecer comparaciones entre los resultados obtenidos a partir de la utilización de los algoritmos propuestos

y los resultados arrojados por los algoritmos secuenciales hasta el momento.

Se considera que el **aporte teórico** fundamental de la presente investigación consiste en la aplicación de un conjunto de técnicas de computación paralela para disminuir el tiempo de respuesta de los algoritmos para el análisis del terreno sobre información geográfica almacenada en formato *raster*, dirigidas a la utilización de varias arquitecturas de cómputo no dedicadas: las arquitecturas multinúcleos de los procesadores actuales, la potencia de cálculo de las GPU y los beneficios del cómputo voluntario, teniendo en cuenta las limitaciones del entorno computacional y la heterogeneidad entre diferentes plataformas de cómputo. Como **aporte práctico** se proporciona una herramienta para el análisis del terreno, a incorporarse como una de las utilidades que brinda la GDAL para el procesamiento *raster* en SIG.

Para dar cumplimiento al objetivo propuesto anteriormente se combinaron una serie de métodos de la investigación científica que permitieron la búsqueda y análisis de la información en el proceso de investigación. Los más relevantes son:

Métodos teóricos: El método *analítico - sintético* para consultar la bibliografía especializada en cuanto al tema abordado e identificar los elementos claves que contribuyan a la solución del problema científico planteado, así como para evaluar, analizar y sintetizar conceptos claves que ayuden a comprender la solución del problema; el método *hipotético - deductivo* para elaborar la hipótesis de la investigación y definir criterios que permitan dar cumplimiento al objetivo planteado a partir de los elementos abarcados durante las etapas de la investigación; el método *histórico - lógico* para el estudio crítico de los trabajos anteriores que constituyen referentes teórico - prácticos en el tratamiento del problema planteado, y tomarlos como base de comparación con los resultados alcanzados; el método de *modelación* para visualizar y evaluar las propuestas y alternativas científicas en la propuesta de solución.

Métodos empíricos: El método *análisis documental* en la revisión de la literatura especializada para consultar la información necesaria en el proceso de investigación; el método de *observación* para evaluar el comportamiento de los algoritmos para el procesamiento *raster* y análisis del terreno en relación con el aumento de la resolución espacial; y el método *experimental* para la comprobación de los resultados obtenidos, en condiciones controladas, y establecer comparaciones con los resultados de los algoritmos secuenciales.

Estructura del documento

El presente documento se encuentra dividido en tres capítulos. En el primer capítulo se exponen las principales características del modelo *raster*, las funciones utilizadas para el análisis del terreno sobre este tipo de dato, así como las principales limitantes de realizar su procesamiento de forma secuencial. Se caracterizan los principales elementos de computación paralela de interés para la investigación y se realiza un diagnóstico del estado actual de la temática en cuestión.

En el segundo capítulo se proponen un conjunto de algoritmos dirigidos a la utilización de varias arquitecturas paralelas que garanticen la heterogeneidad entre diferentes plataformas de cómputo para el análisis del terreno sobre información *raster*. En el tercero, se realiza un análisis de los resultados a través de la aplicación de las diferentes métricas para evaluar algoritmos paralelos: la ganancia de velocidad y la eficiencia. Además, se presentan una serie de conclusiones, recomendaciones, referencias y acrónimos.

Capítulo 1

FUNDAMENTOS TEÓRICOS Y ESTUDIO DIAGNÓSTICO

En este capítulo se abordan conceptos asociados al dominio del problema, que sustentan la investigación y contribuyen a la comprensión de su contenido. Se reflejan los elementos fundamentales relacionados con la computación paralela que resultan de interés para la presente investigación. Además, se presenta el resultado de un análisis realizado sobre las contribuciones más relevantes que constituyen referentes teórico - prácticos de la investigación, así como las principales tendencias en el área del procesamiento de datos *raster* en entornos paralelos.

1.1. El modelo de datos *raster*

En el **modelo *raster***, según [Smith y col., 2013], la información geográfica es representada a través de una estructura matricial A, conocida como **malla regular**, en la que cada celda $a_{ij} \in A_{n,m}(\mathbb{R})$ tiene un valor y una localización determinada, como se muestra en la Figura 1.1.

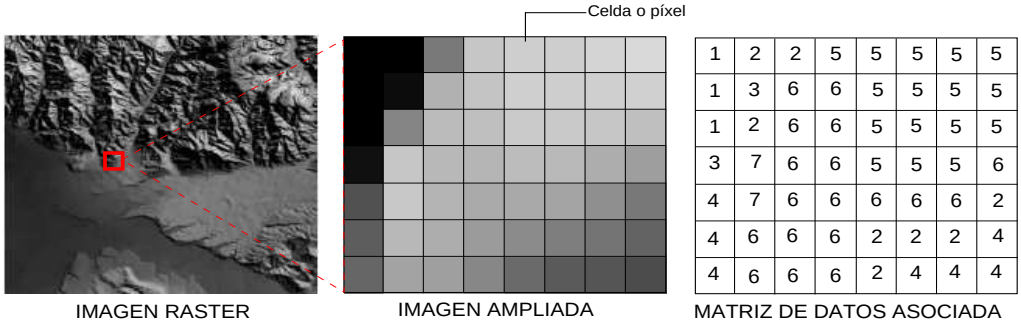

Figura 1.1: Esquema de representación de la información en el modelo *raster*.

La malla regular se conoce como **estructura *raster***. Mediante este modelo la localización de las entidades geográficas se define con la referencia directa a la matriz de datos en la que cada celda está asociada a una porción del terreno [Llopis, 2008].

1.1.1. Tamaño de celda y resolución espacial

La celda es la unidad mínima de información de un *raster*, por lo tanto, su tamaño representa la precisión con la que son definidos los elementos geográficos en el modelo [Llopis, 2008]. A su vez, la resolución espacial de una superficie representada en un *raster*, depende en gran medida del tamaño de las celdas de la matriz de datos, expresado en metros sobre el terreno [ESRI, 2012].

Cuanto menor sea el tamaño de dicha celda y por ende de la zona representada, mayor es el número de celdas que se representarán mediante el *raster* [Llopis, 2008]. La Figura 1.2 muestra una representación de la variación en el tamaño de celda de una superficie determinada.

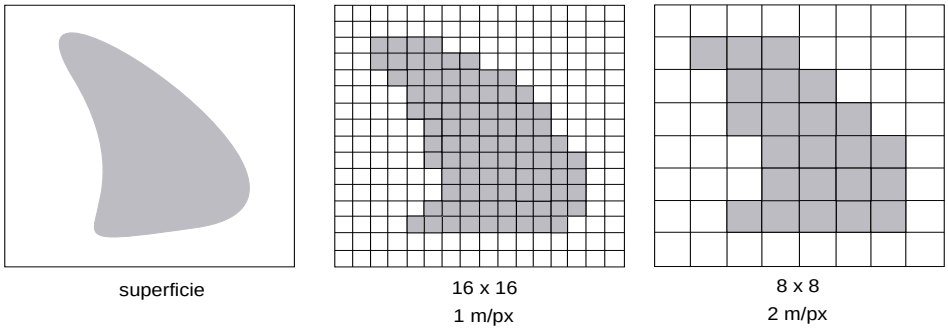

Figura 1.2: Representación del tamaño de celda en el modelo *raster*.

En el manual publicado por el Centro de Recursos de ArcGIS[1] [ESRI, 2012], se afirma que los tamaños de celda más pequeños en modelos *raster* grandes representan una superficie completa, por lo tanto, se necesita un espacio de almacenamiento mayor, elemento que implica también mayor tiempo de procesamiento de los datos.

1.1.2. Tendencias en el aumento de la resolución de los datos

Con el perfeccionamiento de las técnicas de adquisición de datos de la superficie terrestre [Max, 2005], la información almacenada en los modelos *raster* es cada vez más detallada. Este hecho está dado como consecuencia de que el tamaño de las celdas en la matriz de datos se hace más pequeño, adquiriendo de esta forma mayores niveles de resolución espacial. La Tabla 1.1[2], fue elaborada a partir de un estudio realizado sobre las características de algunas de las imágenes *raster* obtenidas como resultado de la puesta en órbita de satélites con sensores de alta resolución y muestra la tendencia en el aumento de la misma en este tipo de dato.

Según la información representada, la evolución comienza en el año 1972, con el lanzamiento a la órbita de la primera versión de la serie de satélites LANDSAT, que logra obtener una imagen de la superficie terrestre con un nivel de resolución de 80m, resultado que fue mejorado posteriormente por la tercera y séptima versión de esta serie con una resolución de 40m y 30m respectivamente. En septiembre de 1999 se lanza Ikonos, el primer satélite con un sensor capaz de tomar imágenes de la Tierra en blanco y negro con 1m de resolución, hecho que marcó un hito significativo en relación con la precisión y resolución espacial de imágenes de la superficie terrestre.

[1] Familia de productos de software para la captura, edición y análisis de información geográfica.

[2] La información mostrada constituye una recopilación de datos extraídos de los sitios mostrados en el campo "Fuente de información" de la propia tabla y otros publicados por *Satellite Imaging Corporation* disponible en http://satimagingcorp.com.

Tabla 1.1: Tabla resumen de imágenes satelitales de la superficie terrestre en alta resolución.

Satélite	**Resolución (m/px)**	**Año de lanzamiento**	**Fuente de información**
LandSAT-1	80	1972	http://landsat.gsfc.nasa.gov/
LandSAT-3	40	1978	http://landsat.gsfc.nasa.gov/
LandSAT-7	30	1999	http://landsat.gsfc.nasa.gov/
ASTER	15	1999	http://asterweb.jpl.nasa.gov
Ikonos	1	1999	http://www.geoeye.com
EROS A	1,8	2000	http://www.imagesatintl.com
QuickBird-2	0,6	2001	http://www.eurimage.com
OrbView	1	2003	http://glcf.umd.edu/data/orbview
EROS B	0,7	2006	http://www.imagesatintl.com
GeoEye	0,5	2006	http://www.landinfo.com/geo.htm
WorldView-2	0,46	2009	http://www.digitalglobe.com

A partir de este momento los satélites de resolución submétrica comenzaron a revolucionar el mundo de la alta resolución con la puesta en órbita en 2001 del satélite QuickBird. Estos resultados son superados por GeoEye y WorldView-2, al obtener imágenes con 0,5m de resolución en los años 2006 y 2009. Según la revisión bibliográfica realizada hasta el momento, los resultados más recientes del año 2014 fueron arrojados por el satélite TAnDEM-X, puesto en órbita desde el año 2010 con el objetivo de obtener un MDE del 100 % de las tierras emergidas del planeta, lo que se traduce en 149 millones de kilómetros. La resolución alcanzada por este sensor tiene un nivel de detalle de 2m sobre el terreno. Pero sin lugar a dudas los resultados esperados por la compañía *Digital Globe* para el año 2016 son superiores con el lanzamiento del satélite WorldView-4, a través del cual se obtendrán imágenes con 0,30m de resolución por píxel.

Como conclusión del estudio realizado, es evidente que la disponibilidad de información *raster* de la superficie terrestre es amplia y su perfeccionamiento no se detiene, elemento que conlleva a innumerables consecuencias benéficas en la rama de la Geomática, sobre todo para su integración con SIG, pero que también tiene su contraparte, pues exige al unísono el perfeccionamiento de las técnicas utilizadas para su procesamiento y análisis.

1.2. Análisis del terreno sobre datos *raster*

1.2.1. Métodos para el cálculo y extracción de parámetros del terreno

Dentro del álgebra de mapas, las funciones para el análisis del terreno a partir de la extraccción de sus parámetros, son clasificadas como funciones de análisis focal o de vecindad, en relación con la distribución de las celdas que se utilizan para obtener un resultado. Esto se debe a que

el valor de una celda se obtiene a partir de su propio valor y teniendo en cuenta el valor de sus vecinos más próximos, definiendo una matriz de recorrido de tamaño n x n alrededor de la celda analizada, como muestra la Figura 1.3. Las funciones más habituales emplean matrices con $n = 3$ [ESRI, 2012]. Posteriormente, esta matriz se emplea para obtener el valor de ese punto en el terreno derivado de la aplicación de un algoritmo de análisis espacial.

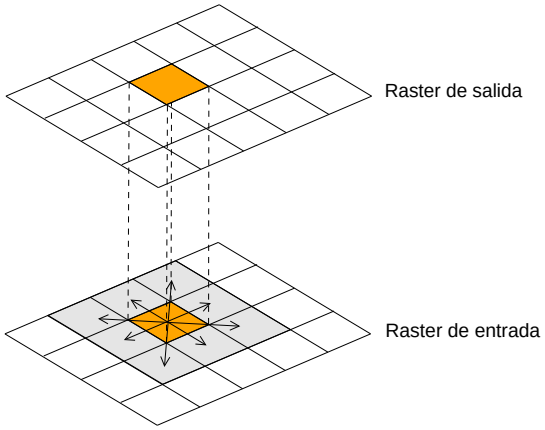

Figura 1.3: Ventana de análisis focal con $n = 3$.

Pendiente y orientación

La pendiente y la orientación se definen en la literatura como parámetros primarios del terreno, debido a que constituyen el punto de partida para la realización de otros análisis basados en la obtención de parámetros más complejos derivados de estos y que permiten aportar mayor información sobre las características del relieve en un entorno determinado [Serrano y col., 1998]. Existen varios enfoques matemáticos para su cálculo [Fleming y Hoffer, 1979, Horn, 1981, Zevenbergen y Thorne, 1987], sin embargo, estudios realizados [Zhou y Liu, 2004, Rodríguez y Suárez, 2010] arrojan como conclusión que los más apropiados son el método de Diferencia Finita de segundo grado [Zevenbergen y Thorne, 1987] y el método de Diferencia Finita de tercer grado [Horn, 1981].

Dado un punto a_{ij} del terreno, la pendiente $P(a_{ij})$ se calcula como sigue:

$$P(a_{ij}) = \arctan\left(\sqrt{(g_x)^2 + (g_y)^2}\right) \tag{1.1}$$

$$O(a_{ij}) = 270^\circ + \arctan\left(\frac{g_y}{g_x}\right) - 90^\circ \frac{g_x}{|g_x|} \tag{1.2}$$

donde g_x y g_y corresponden a las funciones del gradiente Este - Oeste y Norte - Sur respectivamente, partiendo de la matriz de recorrido $M_{3,3} \in A_{n,m}(\mathbb{R})$ correspondiente al punto a_{ij}, expresada en la siguiente ecuación:

$$M_{3,3} = \begin{pmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{pmatrix} \tag{1.3}$$

Según el método de Diferencia Finita de segundo grado definido por [Zevenbergen y Thorne, 1987], g_x y g_y se expresan en las siguientes ecuaciones:

$$g_x = \frac{a_{i-1,j} - a_{i+1,j}}{8\delta} \tag{1.4}$$

$$g_y = \frac{a_{i,j-1} - a_{i,j+1}}{8\delta} \tag{1.5}$$

Según el método de Diferencia Finita de tercer grado definido por [Horn, 1981], g_x y g_y se calculan como sigue:

$$g_x = \frac{((a_{i-1,j+1} + 2a_{i-1,j} + a_{i-1,j-1}) - (a_{i+1,j+1} + 2a_{i+1,j} + a_{i+1,j-1}))}{8\delta} \tag{1.6}$$

$$g_y = \frac{((a_{i-1,j-1} + 2a_{i,j-1} + a_{i+1,j-1}) - (a_{i-1,j+1} + 2a_{i,j+1} + a_{i+1,j+1}))}{8\delta} \tag{1.7}$$

donde δ corresponde al tamaño de celda.

Sombreado

La generación de mapas de sombra a partir del análisis del sombreado del terreno, es una técnica empleada para destacar visualmente los elementos del terreno simulando los efectos de la iluminación del sol. Dada una posición del sol definida por su azimut ϕ y su inclinación con respecto al horizonte ω, la insolación $H(a_{ij})$ de un punto a_{ij} con pendiente p y orientación o se define en la siguiente ecuación [Shary y col., 2005]:

$$H(a_{ij}) = \frac{100 \tan(p)}{\sqrt{1 + \tan^2(p)}} \left[\frac{\sin(\phi)}{\tan(p)} - \cos(\omega) \sin(\phi - o) \right] \tag{1.8}$$

donde los valores obtenidos en $H(a_{ij})$ se sitúan en el rango de $0 - 100$.

Rugosidad

La rugosidad permite caracterizar la complejidad del terreno en función de su variabilidad o irregularidad. Siguiendo las definiciones expuestas en [Dartnell, 2000, Wilson y col., 2007], y dado un punto a_{ij} del terreno, la rugosidad $R(a_{ij})$ se expresa como la diferencia entre el máximo valor $V_{max}(M)$ y el mínimo valor $V_{min}(M)$ de los n vecinos definidos en la ecuación 1.3 que representa la matriz de recorrido $M_{3,3} \in A_{n,m}(\mathbb{R})$:

$$R_{ij} = V_{max}(M) - V_{min}(M) \tag{1.9}$$

1.2.2. Análisis del procesamiento secuencial

Teniendo en cuenta la descripción anterior, siendo f el MDE a procesar, $A_{n,m}(\mathbb{R})$ la matriz asociada, donde $\forall a_{ij} \in A \exists a'_{ij} \in A'$, tal que a'_{ij} es el parámetro del terreno en a_{ij}, el flujo de las operaciones realizadas por los algoritmos para la extracción de los parámetros del terreno se describe de forma genérica en el Algoritmo 1.

Algoritmo 1 Algoritmo secuencial para el cálculo de los parámetros del terreno

```
1:  procedure GenericMatrix3x3(f)
2:      n ← GetNSize(f)
3:      m ← GetMSize(f)
4:      outputDEMFile ← CreateOutputDataset() {Se crea el archivo MDE de salida}
5:      A ← [] {Crear vector fila de tamaño m para realizar la lectura}
6:      for i ← 1, 2, ..., n − 1 do
7:          A ← ReadRow(i,f) { Leer la fila i de f}
8:          ComputeAtEdges(n,m,A) {Procesar los límites del raster}
9:          V ← [] {Crear vector fila de tamaño m para guardar los resultados de procesar la fila i}
10:         W ← [] {∀a_ij ∈ A Crear matriz de vecindad}
11:         for j ← 1, 2, ..., m − 1 do
12:             W[0] ← A[i-1][j-1]
13:             W[1] ← A[i-1][j]
14:             W[2] ← A[i-1][j+1]
15:             W[3] ← A[i][j-1]
16:             W[4] ← A[i][j]
17:             W[5] ← A[i][j+1]
18:             W[6] ← A[i+1][j-1]
19:             W[7] ← A[i+1][j]
20:             W[8] ← A[i+1][j+1]
21:             V[j] ← ExtractParameter(W) {Almacenar el cálculo del elemento j-ésimo en V}
22:         end for
23:         WriteOutputDataset(outputDEMFile, V) {Escribir el vector V en el archivo de salida}
24:     end for
          return outputDEMFile
25: end procedure
```

La primera operación que se ejecuta es la lectura de los metadatos del *dataset raster* a procesar.

A partir de estos metadatos se crea el *dataset* de salida. Una vez realizadas las operaciones anteriores, para cada fila $i \in A$, $i = 1, 2, ..., n-1$, se realiza la lectura de los datos y para cada celda $a_{ij} \in A$ se ejecutan dos operaciones esenciales: se crea la matriz de vecindad $M_{3,3}$ y en cada iteración i, M se emplea para calcular el parámetro del terreno del punto a_{ij} mediante los gradientes correspondientes partiendo de las celdas vecinas, utilizando los métodos matemáticos descritos en el epígrafe anterior. El resultado de procesar cada fila se almacena en un vector fila V_{1m} y una vez finalizada la iteración, este vector se escribe en el fichero MDE de salida.

A los efectos de esta investigación, estas secuencias de operaciones son las más importantes. Precisamente esto se debe a que, independientemente del tamaño de la estructura a analizar, el procesamiento de los datos se realiza fila por fila, como se ilustra en el esquema de la Figura 1.4.

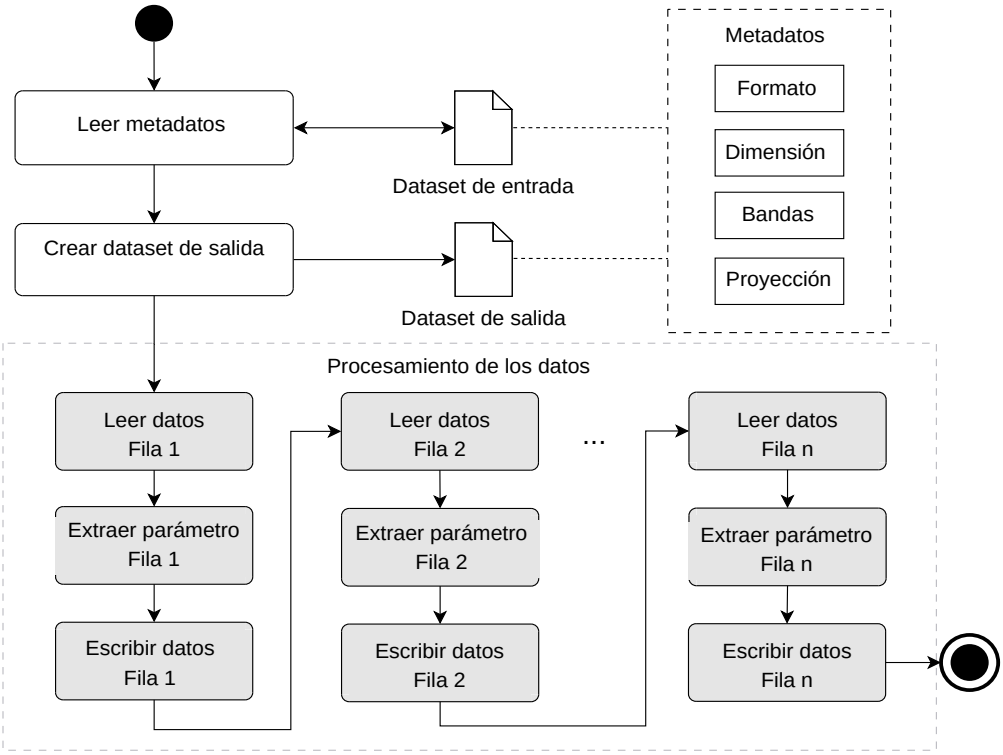

Figura 1.4: Esquema del procesamiento secuencial mediante los algoritmos de análisis *raster*.

Desde el punto de vista algorítmico, las operaciones externas al *Procesamiento de los datos*, Figura 1.4, son consideradas operaciones elementales de orden $\Theta(1)$, por lo que no influyen significativamente en el tiempo de procesamiento de los algoritmos. Sin embargo, teniendo en cuenta que la matriz de entrada A tiene n filas y m columnas y que se itera por cada fila i y columna j desplazando la matriz de vecindad M $\forall a_{ij} \in A$, entonces el tiempo para realizar los

cálculos mediante los algoritmos secuenciales está dado por la ecuación:

$$T(n,m) = \sum_{i=1}^{n-1} \left(\sum_{j=1}^{m-1} \Upsilon \right) \tag{1.10}$$

siendo Υ un valor constante que representa la cantidad de operaciones elementales ejecutadas en cada iteración. Aplicando la definición de cota superior, se deduce que $T(n,m) = n * m$, por lo que la complejidad temporal de los algoritmos es de orden $\Theta(n * m)$.

1.2.3. Evaluación del proceso secuencial en GDAL

La Tabla 1.2 muestra un resumen de los resultados obtenidos en pruebas preliminares realizadas para analizar el comportamiento de los algoritmos en relación con el tiempo de procesamiento secuencial a medida que aumenta la resolución y el tamaño de los datos que son analizados. Las pruebas fueron realizadas haciendo uso de las implementaciones disponibles en la GDAL, biblioteca creada en el año 1998 [Warmerdam, 2008], a partir de la necesidad de lograr el acceso genérico - ya sea para la lectura, escritura o traducción - del amplio rango de formatos de datos *raster* existentes. Una de sus características fundamentales es que presenta un único modelo de datos abstracto que permite manipular la mayoría de formatos *raster*, pero que además se basa en el enfoque de la extensibilidad permitiendo la implementación de nuevos formatos en caso de que así se requiera.

Por esta razón, los principales desarrollos de software para el manejo de información geoespacial, tanto libres como propietarios, utilizan GDAL como motor de manipulación de datos *raster*, tal es el caso de QuantumGIS, GRASS, ArcGIS v9.2+, gvSIG, Google Earth y MapServer [Tyler, 2005, Hall y Leahy, 2008].

Como parte de sus utilidades, la GDAL cuenta con un conjunto de herramientas de línea de comandos para el procesamiento *raster* con diferentes propósitos [GDAL Development Team, 2014], entre los que destacan los algoritmos de análisis del terreno descritos en el epígrafe 1.2.1.

Las pruebas fueron realizadas en dos entornos de hardware:

- Entorno 1 (E1): HP Pavilion g4-1174la con 4 GB de memoria RAM, un procesador AMD Dual-Core A4-3300M APU a 1.90 GHz y una tarjeta gráfica integrada Radeon HD 6480G de 2033 MB.
- Entorno 2 (E2): ASUS x550L con 4 GB de memoria RAM, un procesador Intel(R) Core(TM) i3-2120 a 3.30 GHz.

Se procesaron cuatro MDE utilizando la herramienta `GDALDEM`[3], con el objetivo de generar los mapas de sombra y rugosidad correspondientes a cada uno de los modelos proporcionados. Los tiempos de respuesta obtenidos, son el resultado del promedio de los tiempos arrojados en la ejecución de varias iteraciones por cada uno de los modelos utilizados.

Tabla 1.2: Pruebas experimentales del procesamiento secuencial con la GDAL.

Modelo	Dimensiones	Tiempo promedio (s)			
		Mapa de sombra		Mapa de rugosidad	
		E1	E2	E1	E2
Modelo 1	2049 x 2049	8	5	6	5
Modelo 2	4097 x 4097	36	17	30	16
Modelo 3	19685 x 8972	274	241	236	199
Modelo 4	86404 x 28804	1 416	1 122	1 086	923

A partir de los resultados obtenidos se deduce que la deficiencia de este mecanismo, radica en que independientemente de las características de hardware de los entornos de procesamiento, mientras mayor sea la resolución y el tamaño del *raster*, entonces la cantidad de filas y columnas de la matriz a procesar aumenta, por lo que se incrementa el costo de cómputo y el tiempo de procesamiento, disminuyendo el rendimiento del proceso en cuestión.

Uno de los enfoques de solución más utilizados cuando se trata el problema de procesar grandes volúmenes de datos en el menor tiempo posible, se basa en explotar al máximo las posibilidades de ejecución en paralelo que brindan las arquitecturas multinúcleos actuales, siempre y cuando las instrucciones a ejecutar y los datos a procesar así lo permitan. Como plantean [Guan y Clarke, 2010], desde la perspectiva de computación, el procesamiento de datos *raster* es paralelizable usando una técnica de descomposición de dominio, dado que la estructura matricial utilizada para su almacenamiento puede ser particionada en varias submatrices, constituyendo el procesamiento de cada una de ellas una tarea independiente de las otras.

1.3. Elementos de computación paralela

El paralelismo, como plantea [Gove, 2011], consiste en utilizar múltiples recursos computacionales simultáneamente para resolver un problema dado, con el fin de lograr menor tiempo de procesamiento.

Según la forma en la que se intercambian los datos, existen dos tipos de arquitecturas paralelas: en memoria compartida y en memoria distribuida [Petersen y Arbenz, 2004, Gebali, 2011], ver

[3]Herramienta para analizar y visualizar MDE que provee la biblioteca GDAL.

figura 1.5[4]. Para cada uno de estos tipos de arquitecturas se pueden aplicar distintas técnicas y herramientas de la computación paralela.

Mediante la arquitectura de memoria compartida, según [Petersen y Arbenz, 2004], todos los elementos de proceso acceden al mismo espacio de memoria común, Figura 1.5(a). El acceso a la memoria debe ser controlado por los propios algoritmos a través de diferentes métodos de sincronización. La comunicación entre los procesadores se realiza utilizando la memoria, en el momento en que un proceso determinado escribe en un espacio de memoria y éste es leído por otro proceso. Bajo esta arquitectura, se han realizado diferentes implementaciones como es el caso de *Pthreads* [Lewis y Berg, 1997], *Threading Building Blocks* [Contreras y Martonosi, 2008], *Parallel Pattern Library* [Karypis y Amin, 1994] y el estándar *Open Multi - Processing* (OpenMP) [Dagum y Menon, 1998, Chapman y Huang, 2007].

Por su parte, mediante la arquitectura de memoria distribuida, cada procesador está asociado a un espacio de memoria físico propio o local [Gebali, 2011] y existe una red de interconexión que une los procesos, Figura 1.5(b). Ante los usuarios, los sistemas implementados bajo esta arquitectura, aparecen como una única máquina para resolver un determinado problema. En este caso, la comunicación entre los procesadores se realiza mediante el mecanismo de paso de mensajes. Este mecanismo se encuentra implementado a través de diversas bibliotecas como es el caso de la Máquina Virtual Paralela (PVM) [Geist y col., 1994], PARMACS [Calkin y col., 1994] y la Interfaz de Paso de Mensajes (MPI) [Karniadakis y Kirby, 2002].

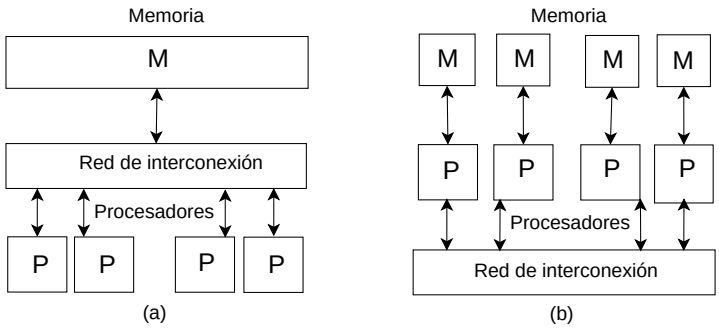

Figura 1.5: Variantes principales para el intercambio de datos en entornos paralelos. (a) Arquitectura de memoria compartida. (b) Arquitectura de memoria distribuida.

1.3.1. Sistemas distribuidos por computación voluntaria

Una de las variantes de implementación de sistemas distribuidos, es la computación voluntaria, a través de la cual el procesamiento se realiza en terminales con características heterogéneas identificadas en la red y no en máquinas dedicadas, bajo el control y la gestión de un proyecto determinado [Beberg y col., 2009], como es el caso de los clúster de alto rendimiento. Esta

[4]Fuente: Elaboración propia, basada en las Figuras 3.1 y 3.2 de [Gebali, 2011]

tecnología, toma ventaja del tiempo de inactividad de una Computadora Personal (PC) ofrecida como voluntaria, para utilizar el poder de su Unidad Central de Procesamiento (CPU) [Kwesi y Amoako-Yirenkyi, 2008].

El esquema básico de funcionamiento de la computación voluntaria está compuesto por un servidor encargado de administrar la distribución del procesamiento y que mantiene comunicación periódica con los voluntarios para informar nuevas tareas y obtener los resultados de las ya completadas.

Varios autores [Anderson y Fedak, 2006, Kwesi y Amoako-Yirenkyi, 2008, Beberg y col., 2009], plantean que mediante este enfoque se proporciona una potencia de cálculo y procesamiento elevada, lo que permite reducir el tiempo requerido por las aplicaciones en condiciones normales de ejecución, obteniendo resultados con alto rendimiento. En los estudios realizados por [Anderson y Fedak, 2006] se demuestra que el potencial de este tipo de sistemas, se extiende mucho más allá de las tareas intensivas de la CPU y abarca aplicaciones que requieren elevado uso de la memoria y el disco.

Uno de los factores que a los efectos de esta investigación y a consideración de la autora, constituye una de las características más relevantes de la computación voluntaria, es que dicha tecnología emerge como una vía económica para la implementación de sistemas distribuidos para el procesamiento paralelo en entornos de bajas prestaciones con características heterogéneas, si se tiene en cuenta que el poder de cómputo no recae sobre un grupo de computadoras de altas prestaciones que conforman un clúster dedicado, sino que utiliza el poder de cálculo y procesamiento de los voluntarios distribuidos a través de una red.

1.3.2. OpenMP para sistemas de memoria compartida

La biblioteca OpenMP [Dagum y Menon, 1998, Chapman y col., 2008] es una Interfaz de Programación de Aplicaciones (API) para los lenguajes de programación Fortran y C/C++ usada para implementar paralelismo en sistemas de memoria compartida, aunque diversas teorías [Karniadakis y Kirby, 2002] abogan por su utilización en esquemas paralelos híbridos[5]. El modelo de programación paralela que aplica OpenMP es el *Fork - Join*, mediante el cual el hilo maestro genera P hilos que se ejecutan en paralelo. OpenMP está compuesto por un conjunto de directivas que describen el paralelismo en el código fuente y denotan su portabilidad, pues en entornos que no usan OpenMP, las directivas son tratadas como simples comentarios e ignoradas. Además proporciona capacidad para paralelizar de forma incremental un programa secuencial, independientemente del hardware.

[5]Se refiere a un algoritmo paralelo concebido para una arquitectura de memoria distribuida que bajo ciertas condiciones, pueda combinarse con un sistema que posea una arquitectura de memoria compartida.

1.3.3. MPI para sistemas de memoria distribuida

Por su parte, el método de paso de mensajes de manera general, se concibe como una tecnología que proporciona las herramientas necesarias para comunicar procesos situados en diferentes nodos de cómputo [Karniadakis y Kirby, 2002]. En relación con este concepto, [Gove, 2011] define a MPI como un modelo de paralelización que permite que una aplicación pueda ser ejecutada en múltiples núcleos ubicados tanto en redes locales LAN o en redes de área amplia WAN. La comunicación entre los nodos se realiza, como su nombre indica, haciendo pasar mensajes entre ellos. Los mensajes pueden ser transmitidos a todos los nodos o dirigidos a uno en particular. MPI provee una API estándar para la programación mediante los lenguajes Fortran y C/C++ en sistemas de memoria distribuida. [Cameron Hughes, 2003] destaca sus capacidades para ofrecer un rendimiento portátil dada su disponiblidad de distribuciones prácticamente para cualquier arquitectura.

1.3.4. GPU: Plataformas de cómputo de propósito general

Impulsado por la demanda del mercado de los gráficos en tres dimensiones, el procesamiento de imágenes en tiempo real y la creciente industria de los video juegos, las GPU han evolucionado hasta convertirse en elementos significativos en comparación con las CPU, tomando gran ventaja sobre éstas en términos de capacidades de cómputo [Nvidia, 2011].

Dado que las GPU poseen una arquitectura que toma como base la naturaleza altamente paralela de las operaciones de cómputo asociadas a la producción de gráficos por computadora, dedicando la gran mayoría de sus componentes a la realización de grandes volúmenes de cálculos matemáticos [Martínez, 2011], son utilizadas como plataformas de cómputo de propósito general, o *General Purpose GPU* (GPGPU). Este paradigma ha generado un creciente interés por parte de la comunidad científica dedicada a investigaciones [Lv y col., 2012, Steinbach y Hemmerling, 2012, Gao y col., 2013] relacionadas con el procesamiento de datos geográficos en paralelo.

A partir de la evolución de las GPU han surgido bibliotecas y plataformas que permiten el desarrollo de aplicaciones GPGPU. Este es el caso de CUDA, arquitectura de cómputo paralelo de propósito general, que incluye un modelo de programación y un conjunto de instrucciones que aprovechan el motor de cálculo paralelo de las tarjetas gráficas NVIDIA para resolver problemas computacionales complejos de una manera más eficiente [Nvidia, 2011].

Luego del surgimiento de CUDA y a partir de la colaboración de los miembros del Grupo Khronos[6], surge el *Open Computing Language* (OpenCL), estándar abier-

[6]Consorcio industrial enfocado en la implementación de estándares abiertos para la creación y aceleración de la computación paralela y los gráficos. Está integrado, entre otros, por AMD, Intel y Nvidia.

to para la programación de colecciones heterogéneas de CPU, GPU y otros procesadores [Munshi y col., 2012]. Ambos, han sido ampliamente utilizados por varios autores [Xia y col., 2011, Steinbach y Hemmerling, 2012, Qin y Zhan, 2012] en diversas investigaciones relacionadas con el procesamiento de información geográfica en entornos paralelos, incluyendo el procesamiento de datos almacenados en formatos *raster*.

Varios autores [Komatsu y col., 2010, Daga y col., 2011, Fraire y col., 2013], coinciden en que una de las diferencias más notables entre OpenCL y CUDA, es que esta última, al ser dependiente de la plataforma y del fabricante, hace difícil que el desarrollador aproveche completamente la potencia disponible en los sistemas modernos de computación. Sin embargo, OpenCL fue diseñado para explotar al máximo el paralelismo dentro de un mismo dispositivo, permitiendo la ejecución de tareas en múltiples núcleos de múltiples dispositivos, independientemente de la plataforma y su fabricante. Su objetivo, es lograr códigos eficientes y portables al mismo tiempo.

1.3.5. Estrategias de descomposición de dominio

Para obtener beneficios en un entorno de procesamiento paralelo, [Armstrong y Densham, 1992] plantean que se deben concebir procedimientos, a partir de los modelos secuenciales, en los que el paralelismo pueda ser explotado al máximo. Este proceso es conocido como descomposición de dominios, y se considera un elemento fundamental en el procesamiento de datos *raster* en paralelo.

En la descomposición de dominio, los datos se dividen en subdominios y cada una de las particiones obtenidas es asignada a un procesador, donde se ejecuta el algoritmo secuencial correspondiente [Wagner y Scott, 1995] utilizando los datos del subdominio proporcionado. En los algoritmos paralelos para procesar información *raster*, comúmmente los datos se descomponen en subdominios rectangulares utilizando tres estrategias comunes: descomposición por filas, descomposición por columnas o descomposición en bloques [Qin y col., 2014b], Figura 1.6.

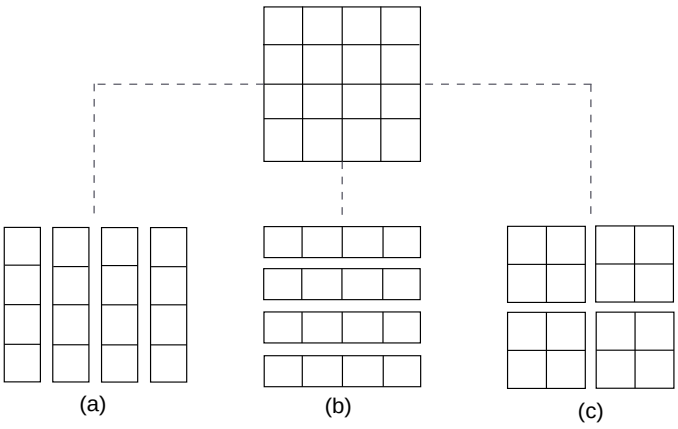

Figura 1.6: Estrategias de descomposición de dominio (a)En columnas. (b)En filas. (c)En bloques.

En su investigación, [Wagner y Scott, 1995] plantean que el rendimiento de un algoritmo paralelo con respecto a la descomposición de los datos, depende en gran medida de la correspondencia que exista entre el tamaño de los subdominios obtenidos y el balance en la asignación de estos subdominios a cada uno de los elementos de procesamiento. En este sentido, el equilibrio entre los procesadores se logra si se particionan los datos en subdominios del mismo tamaño, de manera que se logre uniformidad en la distribución y en consecuencia que algunos procesadores no operen con mayor cantidad de datos que otros.

Por su parte, los resultados obtenidos en experimentos realizados por [Guan y Clarke, 2010], demostraron que el enfoque de descomposición por filas es más adecuado que la descomposición por columnas y bloques en cuanto a tiempo de ejecución e integridad de los resultados.

1.4. Antecedentes del uso de la programación paralela en el procesamiento *raster*

A partir del notable avance en las Ciencias de la Información Geográfica (*GISciences*), el incremento en el volumen y resolución de los datos geoespaciales almacenados en formato *raster*, y la complejidad de los algoritmos y métodos utilizados para su procesamiento, han sido fundamentadas y aplicadas diversas teorías dentro del campo de la computación paralela con el fin de lograr mejores resultados si de rendimiento se trata. En esta sección se muestra el resultado de un análisis realizado sobre las principales investigaciones en el área del procesamiento de modelos *raster* en entornos paralelos.

Aunque en la presente investigación son de mayor interés las teorías relacionadas con los algoritmos paralelos para el análisis del terreno a partir de datos *raster*, sí resulta importante destacar los trabajos que han marcado hitos en el área del procesamiento de modelos *raster* en paralelo de manera general.

Los autores [Guan, 2009, Guan y Clarke, 2010, Guan y col., 2013] desarrollaron una biblioteca de programación denominada *parallel Raster Processing Library* (pRPL) aplicada a la aceleración de un modelo de un autómata celular. Por su parte [Ouyang y col., 2013] propone la utilización de una arquitectura de acceso paralelo a la información *raster* basada en la utilización de MPI. Bajo el mismo concepto de utilizar MPI, [Wang y col., 2012] desarrollaron una arquitectura para el procesamiento en paralelo de imágenes de teledetección, haciendo énfasis en las imágenes multiespectrales por el alto costo computacional que requieren.

[Xia y col., 2011] proponen una metodología genérica basada en la utilización de la GPU y el modelo de programación CUDA en el análisis geoespacial a través de algoritmos de interpolación. Por su parte [Steinbach y Hemmerling, 2012] presentan una estrategia eficiente para el almacenamiento en caché de los datos *raster* y la aceleración de su procesamiento utilizando la

GPU. Sus resultados fueron comprobados a través de la implementación de un componente de software para el SIG GRASS.

1.4.1. Principales aproximaciones al diseño de algoritmos paralelos para la extracción de parámetros del terreno

El aporte de [Lv y col., 2012] se enfoca en la utilización de la GPU empleando CUDA, para acelerar el análisis geoespacial de la pendiente o inclinación de una superficie terrestre. [Jiang y col., 2013a] realizan una propuesta de algoritmos paralelos para el pre-procesado de MDE en la extracción de redes de drenaje usando un clúster de computadoras con el modelo de programación MPI.

Se considera que entre los principales exponentes del procesamiento *raster* en paralelo, figuran las investigaciones realizadas por [Zhan y Qin, 2012, Qin y col., 2014a, Qin y col., 2014b]. La principal contribución de [Zhan y Qin, 2012] consiste en el diseño de dos variantes aplicadas en la biblioteca GDAL, teniendo en cuenta las tres estrategias de descomposición de dominio de los datos *raster* mostradas en la Figura 1.6. En ambos casos se utilizó en la implementación el modelo de programación MPI.

En una primera variante de solución, [Zhan y Qin, 2012] plantean la utilización de un proceso maestro encargado de almacenar todos los datos en la memoria del sistema, con el objetivo de que otros hilos de procesos accedan a los datos a través de la comunicación con el proceso maestro, como muestra la Figura 1.7. La principal deficiencia de esta variante se presenta cuando el *raster* a procesar excede en tamaño la capacidad de la memoria disponible en alguno de los nodos de cómputo.

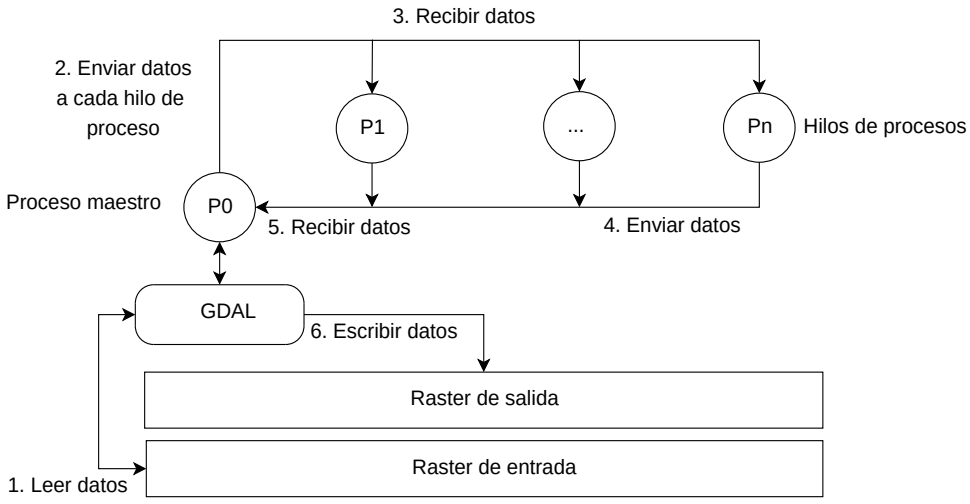

Figura 1.7: Esquema de E/S en serie para el procesamiento de datos *raster* en paralelo utilizando GDAL. Fuente: Elaboración propia basada en la Figura 1 de [Zhan y Qin, 2012]

A diferencia de esta variante, el segundo modelo que propone [Zhan y Qin, 2012] contempla la posibilidad de que cada proceso pueda leer y escribir datos en los archivos de entrada y salida respectivamente, como muestra la Figura 1.8. En este caso, el proceso maestro utiliza la GDAL para obtener los metadatos útiles en el procesamiento, por ejemplo la extensión espacial y la proyección. A partir de estos datos crea el archivo *raster* de salida correspondiente. Luego, en concordancia con alguna de las estrategias de descomposición de dominio envía la información extraída a los procesos de trabajo. Basándose en la información del subdominio recibida cada uno de los procesos de trabajo lee el subdominio correspondiente, realiza los cálculos y una vez obtenidos los resultados, utiliza la GDAL para abrir y escribir dichos resultados en el archivo *raster* de salida.

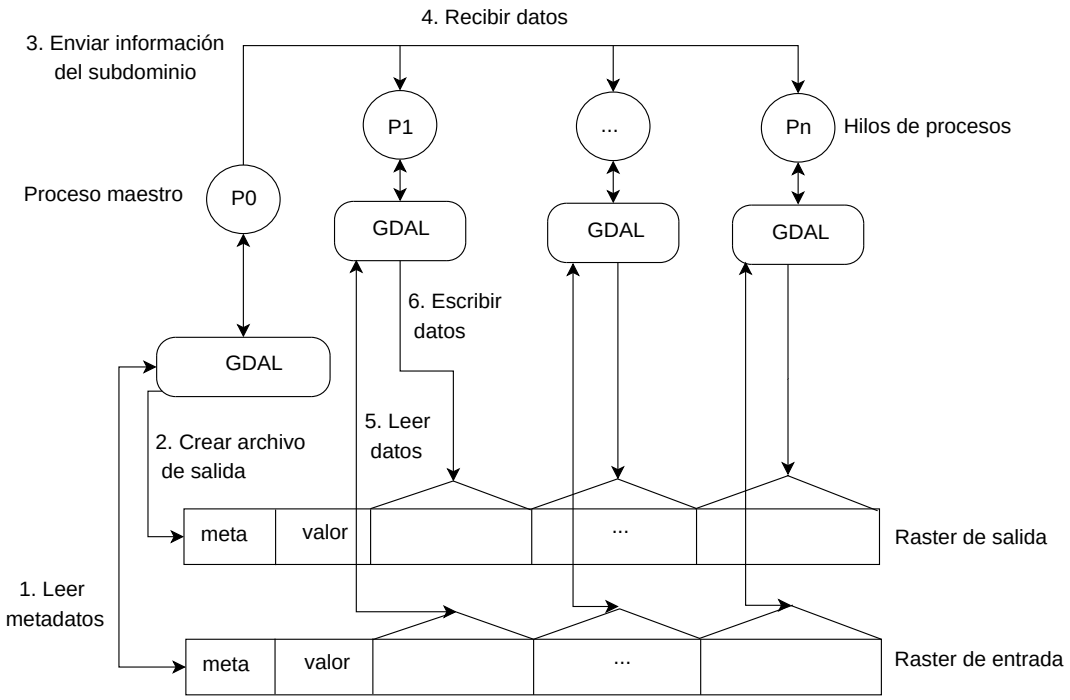

Figura 1.8: Esquema de E/S en paralelo para el procesamiento de datos *raster* en paralelo utilizando GDAL. Fuente: Elaboración propia basada en la Figura 2 de [Zhan y Qin, 2012]

Para este caso la integridad de los resultados se ve afectada cuando se utiliza la descompocisión por columnas o por bloques, o cuando los datos *raster* están almacenados en formato de imagen. Los experimentos aplicados por [Qin y col., 2014a], permitieron corroborar que las deficiencias presentadas por la propuesta de [Zhan y Qin, 2012] son atribuibles al mecanismo de almacenamiento en caché que implementa la GDAL. La dimensión del bloque de caché en GDAL es a menudo igual a la dimensión de la trama a procesar, lo que provoca que se obtengan resultados incorrectos cuando diferentes procesos intentan escribir en la misma región de un archivo *raster* compartido. Esta característica afecta el procesamiento en paralelo en el caso de que los datos se dividan en columnas o en bloques, pues al realizar la lectura de los datos *raster* estos son

reorganizados como un flujo lineal contiguo de valores, que se inicia desde la celda de la esquina superior izquierda en la matriz *raster* y termina con la celda de la esquina inferior derecha, Figura 1.9. Como resultado, cada proceso debe acceder a varias secciones no contiguas del archivo *raster*.

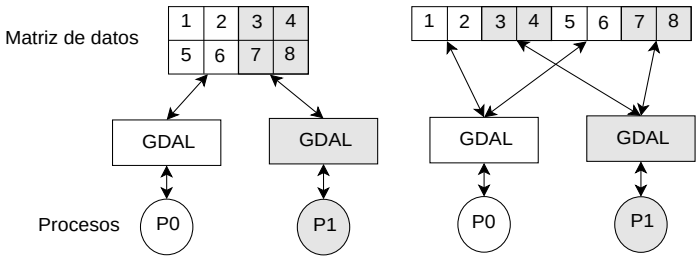

Figura 1.9: Reorganización de la matriz *raster* en disco.

En este sentido, el principal aporte de [Qin y col., 2014a], consiste en lograr erradicar las deficiencias de la variante propuesta por [Zhan y Qin, 2012], Figura 1.8, referidas a la carencia de flexibilidad en cuanto a la utilización de las estrategias de descomposición por columnas y bloques.

A partir de la determinación de las causas que provocan estos resultados incorrectos, [Qin y col., 2014a] proponen el diseño de un módulo de redistribución de los datos. Este enfoque utiliza una estrategia de comunicación entre procesos para redistribuir los datos, de manera que cada proceso compute un trozo contiguo de datos.

1.4.2. Combinación de arquitecturas paralelas

Las investigaciones analizadas hasta el momento se enfocan en la utilización de MPI. Aunque la aplicación de esta teoría proporciona una solución eficiente, requiere que se cuente con una infraestructura de cálculo dedicada a ese fin. Además el proceso pudiera acelerarse aún más si se combina con la utilización de las arquitecturas multinúcleos y la arquitectura paralela que por naturaleza caracteriza a las GPU.

En este sentido, las redes de recursos heterogéneos, en el ámbito de la computación paralela, se han convertido en una alternativa viable en la implementación de sistemas paralelos con respecto a las tecnologías de alto rendimiento. Formalmente, [Lastovetsky, 2008] define una plataforma heterogénea como una infraestructura computacional donde están disponibles un conjunto P de procesadores ($P = \{P_0, P_1, P_2, ...P_{p-1}\}$) de diferentes características. Los elementos que constituyen fuente de heterogeneidad en un sistema paralelo son: los procesadores, la memoria y la forma de acceso a la misma (memoria compartida o memoria distribuida).

Sin lugar a dudas, una de las soluciones más abarcadoras en el sentido de la heterogeneidad a través de la utilización de varias plataformas de cómputo, es la propuesta realizada recientemente

por [Qin y col., 2014b], Figura 1.10. En su investigación, se enfocan en el diseño de una biblioteca para el procesamiento *raster* que hace uso de sistemas paralelos a través de la combinación MPI/OpenMP en un clúster SMP[7], así como sistemas distribuidos mediante clústeres Beowulf[8] con MPI. Su principal contribución radica en utilizar además las características de las GPU para acelerar el procesamiento de los datos, tarea que realizan empleando la tecnología de cómputo de propósito general CUDA.

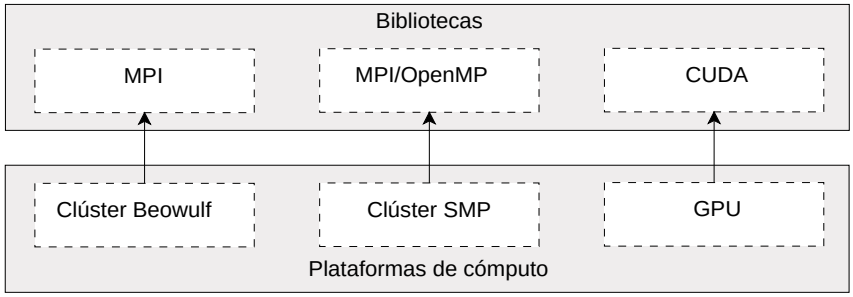

Figura 1.10: Principales elementos de la estrategia propuesta por [Qin y col., 2014b].

Aunque el empleo de CUDA se adentra ya en la utilización de la computación GPGPU para lograr mejores resultados en cuanto a la aceleración de los cálculos, presenta una limitación relevante. CUDA no está implementado para tener en cuenta la computación heterogénea entre diferentes plataformas de cómputo, en otras palabras, con su utilización no se aprovecha la potencia de los procesadores gráficos unida a la de las CPU y otros dispositivos de cómputo. Además, CUDA ha sido implementado y optimizado para obtener alto desempeño en tarjetas gráficas NVIDIA solamente. Como consecuencia de esto, los resultados obtenidos no pueden ser aprovechados en cualquier plataforma o arquitectura (por ejemplo en las tarjetas gráficas de Intel y AMD), al contrario de esto, son dependientes del hardware fabricado por NVIDIA.

Adicionalmente, dichos resultados fueron evaluados en un clúster IBM SMP con 134 nodos de cómputo. Cada nodo consta de dos CPUs Intel Xeon E5650 a 2.0 GHz con seis núcleos y 24 GB de memoria RAM DDR3, generalmente empleados como servidores por sus altas prestaciones[9]. La GPU utilizada es una NVIDIA Corporation Tesla M2075 con 448 núcleos de procesamiento y 6 GB de memoria gráfica[10] [Qin y col., 2014b]. Sin embargo, a pesar del incremento en las velocidades de cómputo y la disponibilidad de memoria, terminales de estas características, en cuanto a sus altas prestaciones de hardware, no son las que predominan en el entorno de las entidades y proyectos dedicados al procesamiento y análisis de información geoespacial.

[7] Arquitectura de computadores de alto rendimiento en la que varias unidades de procesamiento comparten una única memoria central [Jost y col., 2003].

[8] Conglomerado de elementos de cómputo, gestionado por un sistema Unix, donde cada nodo es una computadora personal sin teclado, mouse, tarjeta de video o monitor [González, 2010].

[9] Consultar: http://procesadores.findthebest.es/l/230/Intel-X5650

[10] Consultar: http://www.nvidia.com/docs/IO/105880/DS-Tesla-M-Class-Aug11.pdf

1.4.3. Resultados del análisis realizado

En la Tabla 1.3, se ilustra un análisis comparativo entre las estrategias más relevantes de la bibliografía que centran su atención en la implementación de alternativas para el procesamiento paralelo de información *raster*. En la tabla se representan características evaluadas con un nivel (alto, medio, bajo) asociado a cada una de las contribuciones citadas.

Tabla 1.3: Análisis comparativo de las principales aproximaciones estudiadas.

Estudios	**Plataforma de cómputo**	**Heterogeneidad entre plataformas**	**Confiabilidad de los resultados**	**Aplicación en entornos no dedicados**
[Zhan y Qin, 2012] (E/S en serie)	CPU	Baja	Baja	Baja
[Zhan y Qin, 2012] (E/S en paralelo)	CPU	Baja	Baja	Baja
[Qin y col., 2014a]	CPU	Baja	Alta	Baja
[Qin y col., 2014b]	CPU (Clúster Beowulf y Clúster SMP), GPU	Media	Alta	Baja

A modo de resumen del estudio realizado, en la Tabla 1.4 se muestran las principales contribuciones de los últimos diez años en relación con el procesamiento de datos *raster* en entornos paralelos, que a consideración de la autora, fundamentan propuestas de significación para la presente investigación.

Tabla 1.4: Estudios sobre procesamiento de datos *raster* en entornos paralelos.

Estudios	**Aportes**
[Guan, 2008, Guan, 2009, Guan y Clarke, 2010, Wang y col., 2012, Guan y col., 2013, Ouyang y col., 2013]	Uso de MPI en el procesamiento *raster*.
[Zhao y col., 2010, Xia y col., 2011, Lv y col., 2012, Qin y Zhan, 2012, Osterman, 2012, Gao y col., 2013]	Uso de la GPU con el modelo de programación CUDA.
[Zhan y Qin, 2012, Qin y Zhan, 2012, Qin y col., 2014a]	Uso de MPI y OpenMP en el procesamiento paralelo de datos *raster* aplicado en GDAL.
[Qin y col., 2014b]	Incorpora a la estrategia propuesta por [Qin y col., 2014a] el uso de la GPU con el modelo de programación CUDA para acelerar el procesamiento de los datos.

1.5. Conclusiones parciales

El estudio realizado demostró que el perfeccionamiento de las tecnologías y mecanismos para la obtención de datos de la superficie terrestre ha propiciado la obtención de información *raster* cada vez más detallada y con mayor nivel de resolución y precisión espacial, lo que ha generado un crecimiento gradual del volumen de los modelos *raster*. Por lo tanto, se requieren altas prestaciones computacionales para apoyar su análisis, por lo que el reto fundamental es la implementación de técnicas que contribuyan a reducir el tiempo para su procesamiento. Como consideraciones finales de este capítulo se relacionan las siguientes:

- A pesar de que se aprecia un avance científico en el área del procesamiento paralelo de *raster*, en su mayoría, las propuestas estudiadas, modelan el procesamiento paralelo en torno a la CPU a través de la utilización de estrategias paralelas mediante OpenMP y MPI o a la GPU mediante CUDA, sin explotar al máximo la coexistencia de múltiples arquitecturas de cómputo.
- La aproximación propuesta por [Qin y col., 2014b] satisface este factor, sin embargo el empleo de CUDA limita su campo a las GPU del fabricante NVIDIA. Adicionalmente, el entorno computacional empleado para el despliegue de la solución propuesta no es el más asequible, en términos de costo, a las posibilidades de las organizaciones especializadas en el desarrollo de SIG y dedicadas al procesamiento de información geoespacial.

Capítulo 2

MODELACIÓN DE LA SOLUCIÓN

Teniendo en cuenta los elementos abordados en el capítulo anterior, la investigación se centra en el diseño de un conjunto de algoritmos dirigidos a la utilización de varios entornos paralelos: en memoria compartida y memoria distribuida; tomando como base la arquitectura paralela de las GPU y los procesadores multinúcleos, además de las ventajas de procesamiento de los sistemas distribuidos por cómputo voluntario en redes de procesadores heterogéneos y no dedicados.

En el presente capítulo se explican de forma detallada todos los elementos que intervienen en el procesamiento. En primer lugar, se diseña un esquema genérico que muestra el flujo general que siguen los algoritmos paralelos propuestos para realizar el procesamiento de los datos, teniendo en cuenta las diferentes arquitecturas de cómputo empleadas. Posteriormente, se describen cada uno de los algoritmos propuestos, en correspondencia con las plataformas de cómputo incluidas en el esquema diseñado.

2.1. Estrategia de paralelización

La Figura 2.1 muestra el flujo de operaciones seguidas en la paralelización de los cálculos. De manera general el mismo consta de tres fases: Inicialización, Ejecución y Finalización.

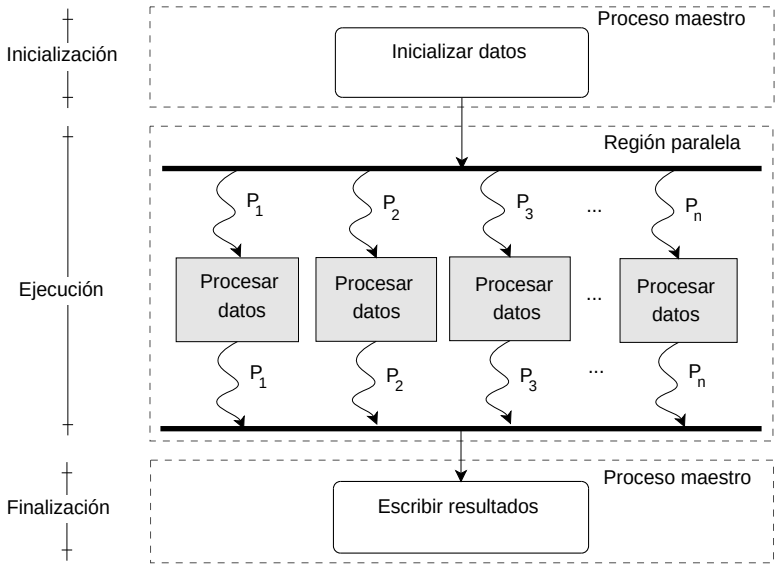

Figura 2.1: Esquema general de procesamiento paralelo.

Las fases de Inicialización y Finalización constituyen la parte secuencial del algoritmo y son ejecutadas por el procesador o hilo maestro. En la primera se inicializan las estructuras de datos a utilizar, así como otras variables necesarias, mientras que en la última se escriben en disco los resultados del cálculo. La fase de Ejecución es la parte paralelizable del algoritmo, donde P procesadores denotados por $p = 0, 1, 2, ..., p - 1$ intervienen en el cálculo simultáneamente.

La Figura 2.2 muestra las tres variantes de cómputo paralelo propuestas. La heterogeneidad radica en hacer uso de sistemas multinúcleos (CPU o GPU) en memoria compartida (a), así como sistemas distribuidos por cómputo voluntario en caso de que los *dataset* a procesar sean lo suficientemente grandes como para no poder ser almacenados en la memoria de un sistema de memoria compartida (b). Se propone además como tercera variante un sistema híbrido (c) como resultado de la combinación de los anteriores, mediante la distribución de subtareas a nivel grueso en una red de procesadores multinúcleos en los que sea posible a su vez ejecutar subtareas en paralelo a un nivel más fino (ya sea en CPU o GPU) en correspondencia de las características de hardware de las terminales que conformen el sistema.

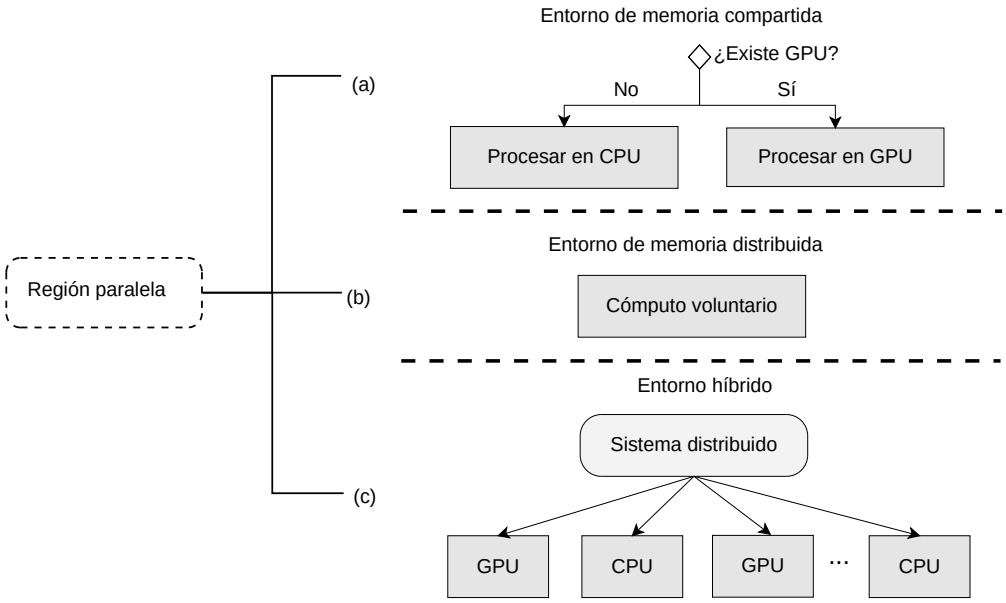

Figura 2.2: Entornos de cómputo paralelo propuestos.

2.2. Paralelización en un entorno de memoria compartida

Para un entorno paralelo de memoria compartida, se propone tanto la utilización de la CPU como de la GPU. Mediante esta variante, no es necesario dividir la matriz de datos entre los procesos, los procesadores comparten toda la matriz, por lo que los datos deben ser cargados totalmente en la memoria RAM del sistema. Esto garantiza que no existan problemas de dependencia entre datos a la hora de conformar la matriz de vecindad, teniendo en cuenta que para cada celda a_{ij},

es necesario acceder a la filas $i-1$ e $i+1$ y a las columnas $j-1$ y $j+1$. Además, dado que todos los cálculos son totalmente independientes, no se requiere sincronización entre procesos.

2.2.1. Algoritmo paralelo en CPU

Teniendo en cuenta los resultados obtenidos en las pruebas experimentales expuestas en [Guan y Clarke, 2010, Zhan y Qin, 2012, Qin y col., 2014a], la investigación asume el esquema de descomposición de los datos en subdominios horizontales, dividiendo la matriz de entrada en bloques de filas. En el algoritmo propuesto se inician tantos hilos como procesadores existan. Siendo p el número de procesadores y n la cantidad de filas de la matriz, entonces a cada procesador P_i con $i = 0, 1, 2, ..., p-1$, se le asigna un bloque de tamaño $\frac{n-2}{p}$, Figura 2.3.

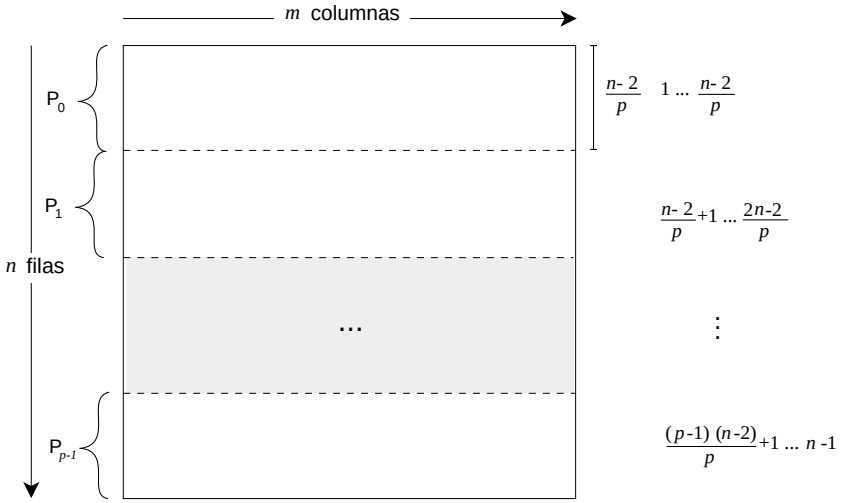

Figura 2.3: Esquema de descomposición de los datos para arquitecturas multinúcleos.

En el caso de que n no sea múltiplo de p, el último bloque tendría un tamaño de $n-2 \ mod \ p$. De esta forma, un procesador P_i cualquiera recibe como parámetro a d_k^s donde s y k se denotan como sigue:

$$s = 1, \frac{n-2}{p} + 1, ..., \frac{(p-1)(n-2)}{p} + 1 \tag{2.1}$$

$$k = \frac{n-2}{p} + 1, \frac{2n-2}{p} + 1, ..., n-1 \tag{2.2}$$

Inicialmente en el Algoritmo 2 se realiza la lectura del fichero *raster* de entrada. En concordancia con las metadatos leídos se crea el fichero de salida. Las matrices de entrada/salida se disponen en la memoria de la CPU en forma de *buffer* lineal, donde los elementos de la primera fila de la matriz se sitúan al inicio del *buffer* seguidos por los elementos de la segunda fila y

así sucesivamente. Siendo m la cantidad de columnas, el acceso a un espacio de índice ij de la matriz de dos dimensiones $A_{n,m}$ en el *buffer* B se realiza como sigue:

$$a_{ij} = B_{i*m+j} \tag{2.3}$$

Algoritmo 2 Algoritmo paralelo para memoria compartida

```
1:  procedure CalculateParameterOnCPU(inputDEMFile)
2:      A ← ReadInputDataset(inputDEMFile) {Se realiza la lectura del archivo MDE de entrada y se
3:      almacena en la memoria en forma de buffer lineal}
4:      outputDEMFile ← CreateOutputDataset() {Se crea el archivo MDE de salida}
5:      n ← GetNSize(inputDEM)
6:      m ← GetMSize(inputDEM)
7:      ComputeAtEdges(n,m,A) {Procesar los límites del raster}
8:      A' ← [] {Crear buffer lineal de tamaño n x m para almacenar el resultado del cálculo}
9:      En paralelo con p hilos de ejecución: para cada pr = {0, 1, 2..., p − 1}
10:     Variables privadas: j, W
11:     Variables compartidas: i, n, m, A
12:     {Distribución de los elementos en subdominios de tamaño (n−2)/p}
13:     for i ← pr(n-2)/p + 1, (pr+1)(n-2)/p + 1 do
14:         W ← []
15:         {Cada hilo "pr" realiza el cálculo del parámetro en el subdominio correspondiente.
16:         ∀a_ij ∈ A Crear matriz de vecindad}
17:         for j ← 1,m-1 do
18:             W[0] ← A[(i-1) * m + j-1]
19:             W[1] ← A[(i-1) * m + j]
20:             W[2] ← A[(i-1) * m + j+1]
21:             W[3] ← A[i * m + j-1]
22:             W[4] ← A[i * m + j]
23:             W[5] ← A[i * m + j+1]
24:             W[6] ← A[(i+1) * m + j-1]
25:             W[7] ← A[(i+1) * m + j]
26:             W[8] ← A[(i+1) * m + j+1]
27:             A'[i * m + j] ← ExtractParameter(W) {Realizar el cálculo del parámetro y
28:             almacenarlo en el buffer A'}
29:         end for
30:     end for
31:     WriteOutputDataset(outputDEMFile, A') {Escribir el resultado en el archivo de salida}
32: end procedure
```

La función ComputeEdges es empleada para procesar las celdas correspondientes a los bordes del *raster* (filas 0 y n, columnas 0 y m) debido a la insuficiencia de datos para conformar la matriz de vecindad en estos casos. Luego, cada procesador P_i se desplaza por el bloque de filas que le corresponde y para cada celda genera la matriz de vecindad correspondiente, almacenando el resultado del cálculo en el *buffer* de salida. Luego el hilo maestro se encarga de escribir

estos resultados en el fichero de salida. Esta secuencia de pasos se encuentra formalizada en el Algoritmo 2 que recibe como parámetro al fichero MDE para analizar ($inputDEMFile$).

2.2.2. Algoritmo paralelo en GPU

Para lograr la coexistencia entre CPU y GPU en el procesamiento, el trabajo se inclina por el uso de OpenCL. La arquitectura de OpenCL está descrita a través de varios modelos: (1) el modelo de la plataforma, (2) el modelo de ejecución, (3) el modelo de memoria y (4) el modelo de programación [Fraire y col., 2013]. El modelo de la plataforma consta de los siguientes elementos: un *host* (generalmente la CPU), conectado a uno o varios dispositivos OpenCL integrados por unidades de cómputo, donde cada unidad a su vez se compone de varios elementos de procesamiento (definidos como E_i, $i = 0, 1, 2, ..., n$ en la Figura 2.4). En el *host* se ejecuta un programa encargado de iniciar la aplicación central, realizar todas las configuraciones, gestionar los dispositivos e inicializar el contexto OpenCL.

En el modelo de memoria se gestiona la memoria utilizada por el *host* y el dispositivo OpenCL. La memoria *host* define la región a la que tiene acceso el programa *host*, y la memoria global es el espacio reservado para la lectura y/o escritura de los elementos de procesamiento del dispositivo. Por su parte la región de memoria global que se mantiene constante durante la ejecución es denominada memoria constante. En correspondencia con la capacidad de memoria del dispositivo, los datos resultantes de las operaciones de lectura y escritura en la memoria global pueden ser almacenados temporalmente en caché.

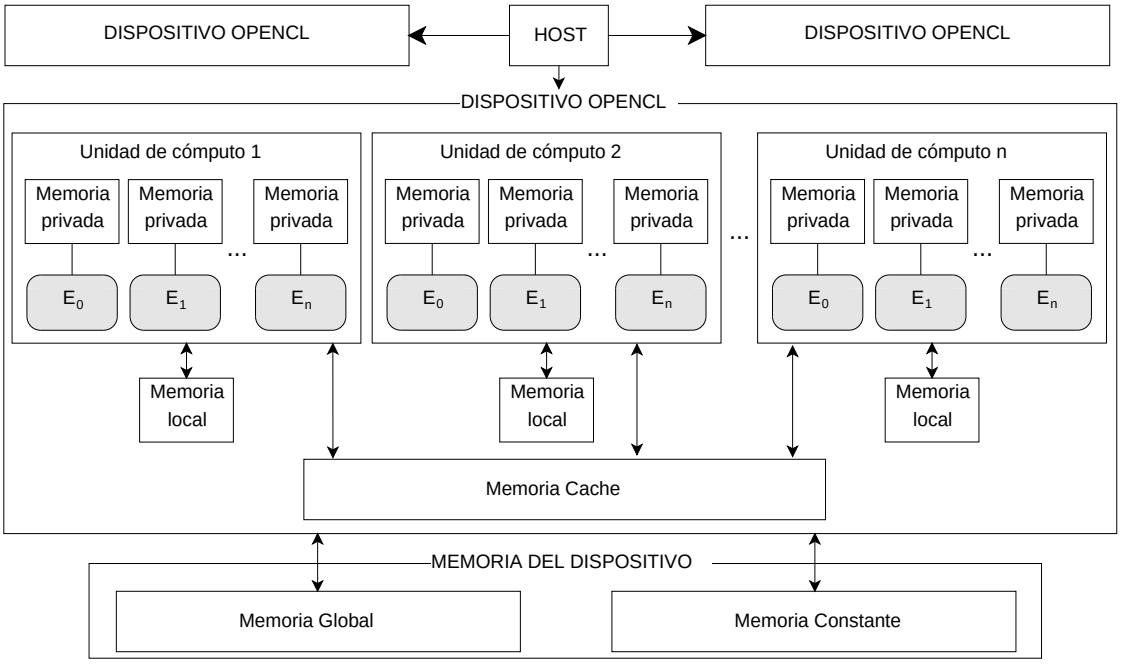

Figura 2.4: Modelo arquitectónico de una tarjeta gráfica en OpenCL.

Por su parte el modelo de ejecución está compuesto por el programa *host* y por los *kernels*. Los *kernels* son porciones de código que se ejecutan en el dispositivo OpenCL en paralelo y que están escritos en un lenguaje especificado por OpenCL similar a C.

Los elementos de procesamiento en OpenCL son los encargados de ejecutar los *kernels* y cada instancia de un *kernel* recibe el nombre de "*work - item*", identificado a través de un único valor conocido como "Global ID", utilizado como índice de referencia que apunta a una región específica del *buffer* en memoria.

El flujo general de operaciones del programa OpenCL diseñado consta de tres fases fundamentales que se describen en el esquema de la Figura 2.5. Las fases de Inicialización y Finalización son ejecutadas por el *host*, mientras que en la fase de Ejecución cada uno de los elementos de procesamiento E_i del dispositivo OpenCL ejecuta la función *kernel* operando en paralelo sobre una zona específica de la matriz de datos de entrada/salida.

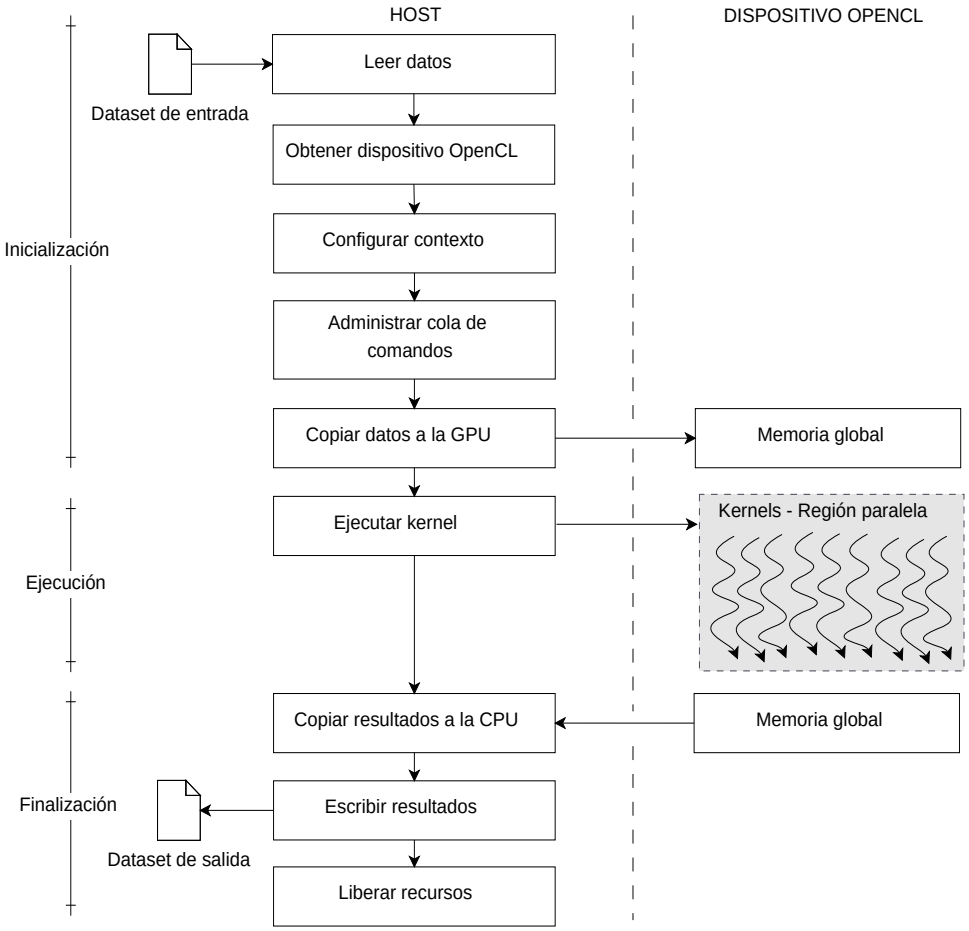

Figura 2.5: Flujo general de operaciones del programa OpenCL.

Basado en este esquema, se plantea la implementación de un algoritmo que consta de dos partes: la parte *host* ejecutada en la CPU y el *kernel* ejecutado en el dispositivo OpenCL. La parte *host*

se muestra en el Algoritmo 3 y recibe como parámetros el fichero con los datos del MDE de entrada (*inputDEM*) y el código del programa *kernel* a ejecutar en el dispositivo.

En las líneas 6 y 7 de la parte *host* se crean los *buffers* de datos de entrada/salida de manera que luego sean accesibles desde el *kernel*. Seguidamente en la línea 8, mediante la función ReadInputData se realiza la lectura de los datos del archivo MDE de entrada.

Algoritmo 3 Programa *host* ejecutado en la CPU

```
 1: procedure CalculateParameterOnGPU(inputDEM, sourceKernel)
 2:     n ← GetNSize(inputDEM)
 3:     m ← GetMSize(inputDEM)
 4:     outputDEM ← CreateOutputFile() {Se crea el fichero MDE de salida}
 5:     {Se crean los buffer en memoria para transferir los datos a la memoria de la GPU}
 6:     inputBuffer ← [n * m * sizeof(float)]
 7:     outputBuffer ← [n * m * sizeof(float)]
 8:     inputBuffer ← ReadInputData(inputDEM) {Leer el fichero MDE de entrada}
 9:     ComputeEdges(inputBuffer,outputBuffer,n,m) {Procesar los bordes del raster}
10:     numPlatforms ← GetOpenCLDevices() {Obtener los dispositivos OpenCL disponibles}
11:     if numPlatforms >0 then
12:         platform ← GetPlatformIds(CL_DEVICE_TYPE_GPU)
13:         if platform == 0 then {Consultar si existe GPU disponible, en otro caso se procesa
14:             en la CPU}
15:             CalculateParameterOnCPU()
16:             GoTo line 36 {En este caso se liberan los recursos y finaliza el procesamiento}
17:         else
18:             device ← GetGPUDevice(platform, CL_DEVICE_TYPE_GPU)
19:         end if
20:         context ← CreateContext(device) {Crear el contexto OpenCL}
21:         queue ← CreateCommandQueue(context, device) {Crear la cola de comandos a
22:         ejecutar}
23:         program ← CreateProgram(context, sourceKernel) {Crear el código de programa
24:         a ejecutar en el kernel}
25:         kernel ← CreateKernel(program, "Process3x3Matrix") {Crear el objeto kernel}
26:         SetKernelArguments(inputBuffer,outputBuffer,m) {Setear los argumentos del
27:         kernel}
28:          global_work_size = n - 2 {Creando n - 2 hilos de procesamiento en la GPU}
29:          status ← ExecuteKernel(queue,kernel,global_work_size) {Ejecutar el kernel}
30:         if status == 0 then
31:             ReadBuffer(queue,ouputBuffer) {Copiar los resultados a la memoria del  host }
32:             WriteResult(outputDEM,ouputBuffer) {Escribir los resultados en el fichero MDE
33:             de salida}
34:         end if
35:     end if
36:     ReleaseResources() {Liberar los recursos}
37: end procedure
```

En las líneas 10 - 19 se realiza el proceso para detectar las plataformas OpenCL disponibles en la

arquitectura y seleccionar la más apropiada. Seguidamente se crea el contexto de los dispositivos detectados (línea 20) y la cola de comandos a ejecutar (línea 21). En las líneas 23 - 25 se crea el programa *kernel* y luego en la línea 26 todos los datos requeridos para su ejecución, incluyendo los *buffers* de entrada/salida creados, son transferidos desde la memoria del *host* a la memoria global del dispositivo.

Antes de iniciar el *kernel* (línea 29), se especifica en la variable `global_work_size` la cantidad de elementos de procesamiento que deben ser iniciados, teniendo en cuenta la cantidad de filas de la matriz de datos y que además la primera y la última fila no deben ser procesadas debido a que corresponden a los bordes del *raster* (línea 28). Una vez culminada la ejecución del *kernel* los resultados del procesamiento son transferidos desde la memoria del dispositivo a la memoria de la CPU (línea 31) para luego ser escritos en el archivo MDE de salida mediante la función `WriteResult` en la línea 32. Finalmente son liberadas todas las estructuras de datos y los recursos utilizados (línea 36).

Teniendo en cuenta que la matriz de entrada se almacena en la memoria global del dispositivo en forma de *buffer* lineal, la paralelización del problema se enfoca en un espacio de una dimensión ($N = 1$). El programa *host* inicia $n-2$ instancias del *kernel* `Process3x3Matrix`, (Algoritmo 4), siendo n la cantidad de filas de la matriz de entrada, Figura 2.6. Esto propicia la eliminación del ciclo externo del algoritmo secuencial. El ciclo interno donde se conforma la matriz de vecindad sigue formando parte del *kernel*, por lo que cada elemento de procesamiento ejecuta $m-2$ iteraciones, siendo m la cantidad de columnas de la matriz. Cada elemento de procesamiento recorre una fila actualizándola en el *buffer* de salida con el resultado de los cálculos correspondientes.

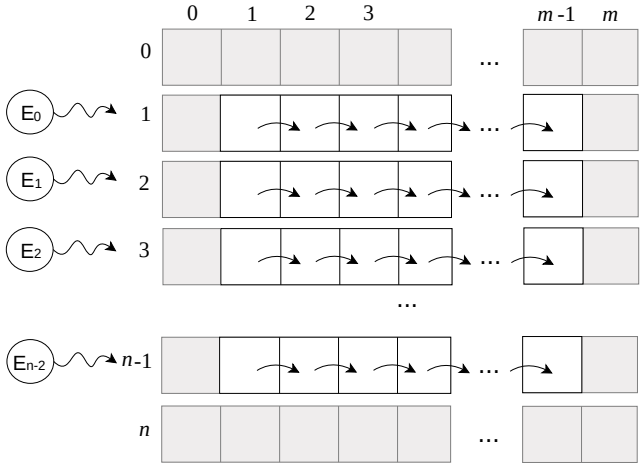

Figura 2.6: Esquema de la ejecución paralela del *kernel* en la GPU.

El *kernel* recibe los siguientes parámetros:

- $inputBuffer = \{v_1, v_2, v_3, ..., v_n\}$: *buffer* de datos de entrada, donde cada v_i contiene el valor que representa la elevación de la superficie del terreno en un punto dado.

- $outputBuffer = \{v'_1, v'_2, v'_3, ..., v'_n\}$: *buffer* de salida, donde cada v'_i contiene el valor obtenido como resultado de la aplicación de un algoritmo de análisis espacial determinado para extraer los parámetros del terreno empleando los datos de elevación en $inputBuffer$.
- $m \in \mathbb{N}$: número de columnas de la matriz *raster*.

Este algoritmo utiliza la memoria global (`_global`) del dispositivo para almacenar tanto los datos de entrada como los de salida. En la línea 2 se obtiene el identificador global del *kernel* en ejecución (único para los $m - 2$ *kernels*). Mediante el ciclo de las líneas 7 - 21 se realiza un desplazamiento por el *buffer* de datos de entrada ($inputBuffer$) para conformar la matriz de vecindad 3x3. El desplazamiento en la memoria lineal del dispositivo se calcula mediante la ecuación 2.3 empleando el índice global i como identificador que permite establecer una correspondencia con el índice de las filas de la matriz inicial. Una vez calculado el parámetro indicado mediante la función `ExtractParameter`, el resultado se almacena en el *buffer* de salida ($outputBuffer$) (línea 20) y se transfiere al espacio de memoria del *host*.

Algoritmo 4 *Kernel* ejecutado en el dispositivo OpenCL

```
1:  function Process3x3Matrix(_global inputBuffer, _global outputBuffer, m)
2:      num ← get_global_id(0)
3:      i ← num + 1 {Se obtiene el ID de cada work-item como índice de referencia a las filas
4:      de la matriz}
5:      {Cada work-item recorre una fila formando la matriz de vecindad correspondiente a cada
6:      celda para luego realizar los cálculos}
7:      for j ← 1, m - 1 do
8:          _local afWin ← []
9:          afWin[0] ← inputBuffer[(i-1) * m + j-1]
10:         afWin[1] ← inputBuffer[(i-1) * m + j]
11:         afWin[2] ← inputBuffer[(i-1) * m + j+1]
12:         afWin[3] ← inputBuffer[i * m + j-1]
13:         afWin[4] ← inputBuffer[i * m + j]
14:         afWin[5] ← inputBuffer[i * m + j+1]
15:         afWin[6] ← inputBuffer[(i+1) * m + j-1]
16:         afWin[7] ← inputBuffer[(i+1) * m + j]
17:         afWin[8] ← inputBuffer[(i+1) * m + j+1]
18:         {Almacenar el resultado del cálculo para cada celda en el buffer de salida según su
19:         disposición en la memoria lineal con respecto a la matriz original }
20:         outputBuffer[i * m + j] ← ExtractParameter(afWin)
21:     end for
22: end function
```

2.3. Paralelización en un entorno de memoria distribuida

A pesar de que la variante de que todos los procesadores compartan la estructura de datos a través del acceso a un espacio de memoria compartido proporciona ventajas en cuanto a la dependencia entre los datos, también es cierto que implica que cuando el tamaño de los modelos es mayor que el tamaño de la memoria disponible en el sistema, estos no puedan ser leídos en su totalidad y deban ser procesados por bloques, o en otro caso ocurre un "colapso" en la memoria. En un entorno de memoria distribuida este problema se atenua, pues los datos se dividen en subdominios más pequeños que son cargados en la memoria local de cada procesador para luego ser procesados simultáneamente.

2.3.1. Descomposición y granularidad de los datos

Como se aprecia en el esquema de la Figura 2.7, cada uno de los procesadores recibe un subdominio de datos de igual tamaño para garantizar el balance de carga durante el procesamiento. Al generar las particiones de datos, cada procesador P_i recibe como parte de su subdominio correspondiente las dos últimas filas de la partición generada con anterioridad. Esto garantiza que al procesar cada fila de la partición se cuente con la fila $i+1$ e $i-1$ necesarias para generar la ventana de análisis para cada celda de la partición.

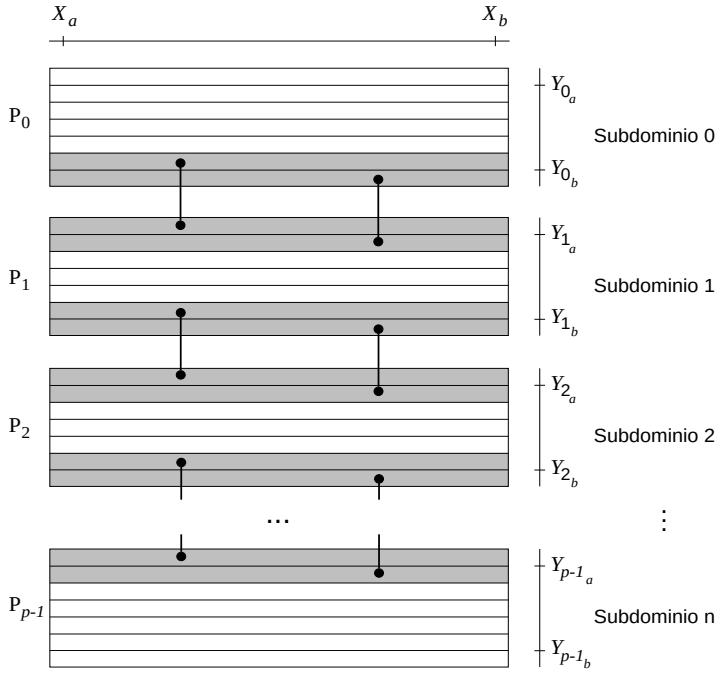

Figura 2.7: Esquema de descomposición de los datos en subdominios horizontales en sistemas de memoria distribuida.

Un elemento importante a tener en cuenta al particionar los datos es la granularidad del pa-

ralelismo. Se entiende como granularidad el grado de comunicación entre los procesadores que intervienen en el cómputo. De acuerdo con lo expuesto en [Jiang y col., 2013b] y en concordancia con el esquema de descomposición de dominio definido, la granularidad es el tamaño de los datos asignados a cada procesador, en este caso el tamaño de los subdominios, y está dada por la siguiente ecuación:

$$G = nRows * mColumns * sifeof(float) \tag{2.4}$$

Donde G es el tamaño de la granularidad, $nRows$ y $mColumns$ la cantidad de filas y columnas de cada subdominio respectivamente y la función $sifeof()$ calcula el tamaño en *bytes* de los datos de tipo flotante. Para el caso de la descomposición por filas, el valor de $mColumns$ y de $sizeof()$ es constante, por tanto, la ecuación 2.4 se puede resumir como:

$$G = nRows \tag{2.5}$$

Por tanto la cantidad de particiones generadas en total es:

$$Pt = \frac{n}{G} \tag{2.6}$$

Donde n es la cantidad de filas de la matriz de datos global.

De acuerdo con lo expuesto anteriormente, los pares (x_a, y_{i_a}) y (x_b, y_{i_b}) que definen cada subdominio D_p generado, Figura 2.7, pueden ser calculados como sigue:

$$D_p = \begin{cases} x_a, x_b \implies \begin{cases} x_a = 1 \\ x_b = m - 1 \end{cases} \\ \\ y_{i_a}, y_{i_b} \implies \begin{cases} y_{i_a} = p * (G - 2) \\ y_{i_b} = y_{i_a} + G \end{cases} \end{cases} \tag{2.7}$$

Donde m es la cantidad de columnas de la matriz global, p es el número de particiones generadas hasta el momento y G es la granularidad seleccionada.

2.3.2. Algoritmo paralelo para memoria distribuida

El pseudocódigo del Algoritmo 5 muestra la secuencia de operaciones lógicas seguidas en la paralelización del cálculo de los parámetros del terreno en un entorno de memoria distribuida.

En el algoritmo paralelo inicialmente se particionan los datos (línea 4 Algoritmo 5), mediante la invocación de la función `PartitionData(n,m,G)` que ejecuta los pasos descritos en el Algoritmo 6 teniendo en cuenta el esquema de descomposición en subdominios horizontales expuesto en el epígrafe 2.3.1 y la granularidad seleccionada. El procesador *root* procede a la creación del archivo

de salida donde posteriormente se escribirán los resultados del cálculo (líneas 7-9 Algoritmo 5). En cada uno de los nodos subscritos, existe una réplica del *dataset* de entrada, por lo que una vez que conozcan el subdominio D_p que les corresponde, cada uno procede a su lectura (línea 10 Algoritmo 5), cálculo y finalmente al envío de los resultados al *root* (líneas 16-18 Algoritmo 5).

El cálculo se realiza en los ciclos de las líneas 11-15. En el primer ciclo se recorren las filas del subdominio y en el segundo se recorren las columnas. En cada iteración se genera la matriz 3 x 3 correspondiente a cada celda a_{ij} y se realiza el cálculo del parámetro empleando el algoritmo correspondiente.

Cuando el *root* recibe los datos de un procesador P_i, estos son escritos en el *dataset* resultante (líneas 19-22 Algoritmo 5) mediante una estrategia de fusión que toma en cuenta los pares (x_a, y_{i_a}) y (x_b, y_{i_b}) correspondientes a cada subdominio con respecto al *dataset* global. La escritura de un bloque resultante comienza en la esquina superior izquierda correspondiente al par (x_a, y_{i_a}) y culmina en la esquina inferior derecha en correspondencia con el par (x_b, y_{i_b}).

Algoritmo 5 Algoritmo paralelo en memoria distribuida

```
1:  procedure CalculateParameterOnDistributedSystems(inputDEM, G)
2:      n ← GetNSize(inputDEM)
3:      m ← GetMSize(inputDEM)
4:      Dp ← PartitionData(n,m,G) {Particionar los datos en subdominios horizontales}
5:      En paralelo: para cada pr = {0,1,2...,p-1} do {Cada procesador "pr" realiza las
6:      operaciones de lectura y cálculo del parámetro en el subdominio D_p correspondiente}
7:      if pr == root then
8:          outputDEM ← CreateOutputFile() {Se crea el fichero MDE de salida}
9:      end if
10:     A ← ReadInputData(inputDEM, Dp) {Leer el subdominio D_p correspondiente}
11:     for i ← y_{i_a}, y_{i_b} - 1 do
12:         for j ← x_a, x_b - 1 do
13:             A' ← ExtractParameter(A,i,j) {Realizar el cálculo del parámetro ∀a_ij ∈ A}
14:         end for
15:     end for
16:     if pr ≠ root then
17:         Send(A', root) {Cada procesador envía el resultado del cálculo al procesador root}
18:     end if
19:     if pr == root then
20:         Receive(A') {El procesador root recibe los resultados del cálculo}
21:         WriteResult(outputDEM, A') {Escribe los resultados recibidos}
22:     end if
23: end procedure
```

Algoritmo 6 Algoritmo para generar las particiones de datos

```
1:  procedure PartitionData(n,m, blocksize)
2:      x_a ← 1
3:      x_b ← m - 1
4:      y_{i_a} ← 1
5:      y_{i_b} ← blocksize
6:      pendingUnits ← n / blocksize {Calcular la cantidad de bloques según la granularidad}
7:      if n MOD blocksize ≠ 0 then
8:          pendingUnits ← pendingUnits + 1 {Si la división n / blocksize no es entera entonces
9:          se genera un bloque más}
10:     end if
11:     {Mientras existan bloques pendientes se continúa generando particiones}
12:     while pendingUnits ≠ 0 do
13:         D_p ← partition(x_a, x_b, y_{i_a}, y_{i_b})
14:         pendingUnits ← pendingUnits - 1
15:         y_{i_a} ← y_{i_a} + blocksize - 2 {Incrementar y_{i_a} para definir el índice donde comienza el
16:         próximo bloque}
17:         {Se comprueba si solo queda un bloque pendiente para definir su tamaño en caso de que
18:         n MOD blocksize ≠ 0}
19:         if pendingUnits == 1 then
20:             y_{i_b} ← n - y_{i_a}
21:         else
22:             y_{i_b} ← blocksize + 2
23:         end if
24:     end while
25:     return D_p
26: end procedure
```

2.3.3. Plataforma de Tareas Distribuidas T-Arenal

En este trabajo se propone la distribución del procesamiento empleando la Plataforma de Tareas Distribuidas T-Arenal [Jacas, 2011]. T-Arenal es un sistema de cómputo voluntario que permite distribuir tareas entre un conjunto de elementos de cómputo heterogéneos y no dedicados. Consta de dos partes fundamentales: el *front-end* y el *back-end*. Mediante el *front-end* es posible acceder a una interfaz gráfica con las funcionalidades propias del sistema y enviar las peticiones de procesamiento al *back-end*. El *back-end* está dividido en una jerarquía de tres niveles compuesta por tres componentes fundamentales [Jacas y col., 2014]: el servidor central, un conjunto de servidores de peticiones y los clientes, Figura 2.8.

El servidor central es el encargado de recibir las solicitudes desde el *front-end* y enviarlas a los servidores de peticiones disponibles para ejecutar las tareas. Además es quien determina el servidor de peticiones correspondiente a cada cliente disponible en la red.

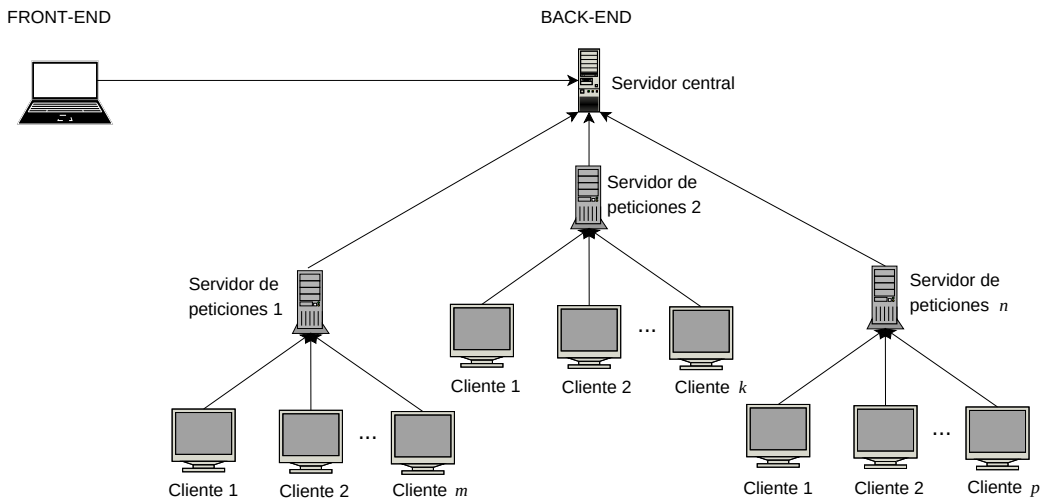

Figura 2.8: Arquitectura general de la Plataforma de Tareas Distribuidas T-arenal. Fuente: Elaboración propia basada en la Figura 1 de [Jacas y col., 2014]

Por su parte el servidor de peticiones es el responsable de particionar las tareas asignadas en subtareas más pequeñas, generando "unidades de trabajo" en correspondencia con el algoritmo de descomposición definido [Jacas y col., 2014] (Algoritmo 6). Cada unidad de trabajo generada es enviada a un cliente y una vez se haya realizado el procesamiento, el servidor de peticiones es quien reúne los resultados de cada cliente con el objetivo de construir la solución final.

Un cliente es cualquier estación de trabajo que desee incorporarse al sistema instalando el módulo cliente de la plataforma. Los mismos son los encargados de procesar las unidades de trabajo recibidas desde el servidor de peticiones correspondiente y luego de retornar los resultados del procesamiento a dicho servidor. Cuando un cliente culmina el procesamiento de una unidad de trabajo, solicita una nueva unidad al servidor de peticiones. En caso de no recibir alguna, el cliente establece un período de pausa para luego comunicarse con el servidor de peticiones nuevamente [Jacas y col., 2014]

Implementación en T-Arenal

La definición de un proyecto de cómputo distribuido sobre T-Arenal requiere heredar de dos clases: *DataManager* y *Task* y redefinir determinadas funciones con el fin de indicar cómo serán generadas las unidades de trabajo y cuál será el algoritmo a ejecutarse en cada uno de los clientes. El diagrama de la Figura 2.9 muestra la definición de las clases responsables de establecer la aplicación distribuida en T-Arenal para realizar el análisis del terreno.

La clase *RasterDataManager* se ejecuta en el servidor de peticiones. Mediante la redefinición del método *generateWorkUnit* se particiona el *dataset* de entrada en subdominios más pequeños en función del esquema de descomposición de los datos expuesto en el epígrafe 2.3.1 y la granularidad

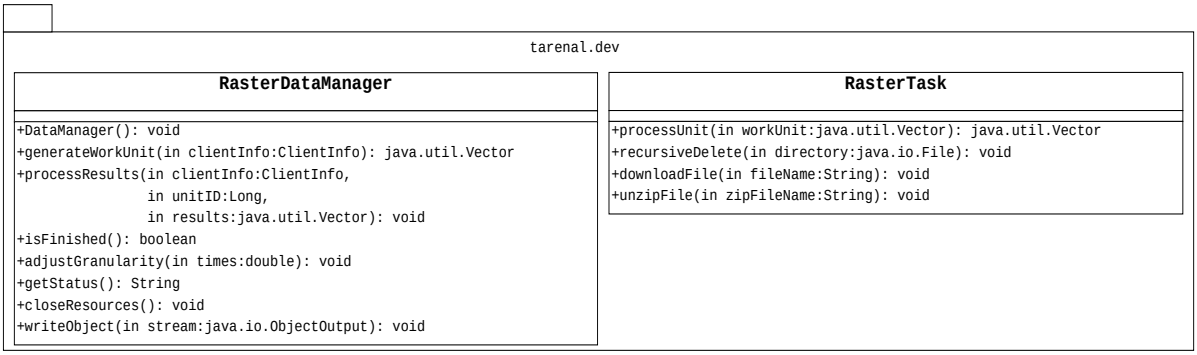

Figura 2.9: Clases responsables de establecer la aplicación distribuida en T-Arenal.

seleccionada. Cada subdominio es enviado como una "unidad de trabajo" a los clientes para que sean procesados.

La clase *RasterTask* se instancia en cada uno de los clientes y mediante el método *processUnit* se ejecuta el algoritmo secuencial sobre la "unidad de trabajo" recibida. Cuando finaliza la ejecución, los resultados se envían al *RasterDataManager* y son escritos en el *dataset* de salida mediante el método *processResult*. Teniendo en cuenta que los resultados son escritos en base al desplazamiento x, y del subdominio con respecto al *dataset* global, estos no deben ser retornados desde los clientes en un orden determinado. El diagrama de la Figura 2.10 muestra la secuencia de operaciones ejecutadas por cada una de las instancias teniendo en cuenta el Algoritmo 5.

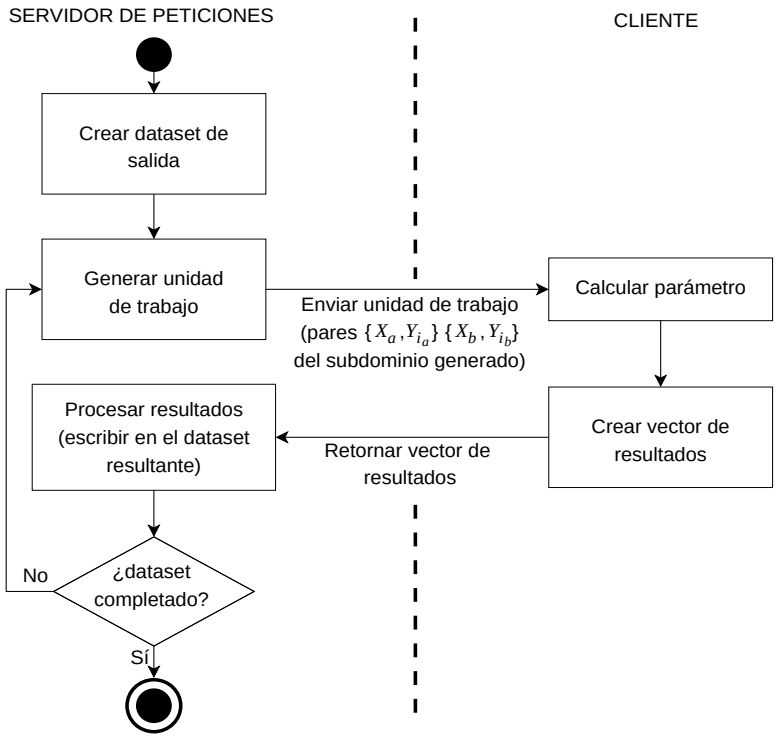

Figura 2.10: Flujo de operaciones para el cálculo de los parámetros del terreno en T-Arenal.

2.4. Paralelización en un entorno híbrido

Luego de haber presentado las variantes paralelas en memoria compartida y distribuida, se propone un algoritmo híbrido para entornos paralelos jerarquizados, en los que como resultado de la combinación de los anteriores, se realice la distribución de subtareas a nivel grueso en una red de procesadores multinúcleos y GPU. En los que sea posible a su vez ejecutar subtareas en paralelo a un nivel más fino en correspondencia con las características de hardware de las terminales que conformen el sistema, Figura 2.11. En el Algoritmo 7 se muestra la secuencia de pasos a seguir en un entorno híbrido.

En un primer nivel de paralelismo el algoritmo sigue el esquema distribuido descrito en el epígrafe 2.3. Mientras que a un segundo nivel, en cada plataforma visible al sistema distribuido (multinúcleo o GPU), el procesamiento de las particiones de datos se realiza mediante algoritmos para memoria compartida muy similares a los descritos en el epígrafe 2.2. En ambos casos, la diferencia con respecto a estos, radica en que los algoritmos para memoria compartida tienen como entrada además del fichero a procesar, el subdominio D_p con los pares (x_a, y_{i_a}) y (x_b, y_{i_b}) que indican el comienzo y el fin de la partición de datos a leer y procesar. El mecanismo para la escritura en el *dataset* de salida se concibe igual al del esquema distribuido.

En la línea 14 se consulta si la plataforma disponible es una GPU. En caso positivo, en la línea 16 se ejecuta el Algoritmo 3, pero con la diferencia de que esta vez se crean los *buffers* para realizar la transferencia de datos al dispositivo GPU y luego se comenzaría a partir de la línea 20 hasta la 31, donde se transfiere el *buffer* de salida a la memoria del *host*. En caso negativo, en la línea 19 se ejecuta el Algoritmo 2 para multinúcleos partiendo de la línea 8 hasta la 30.

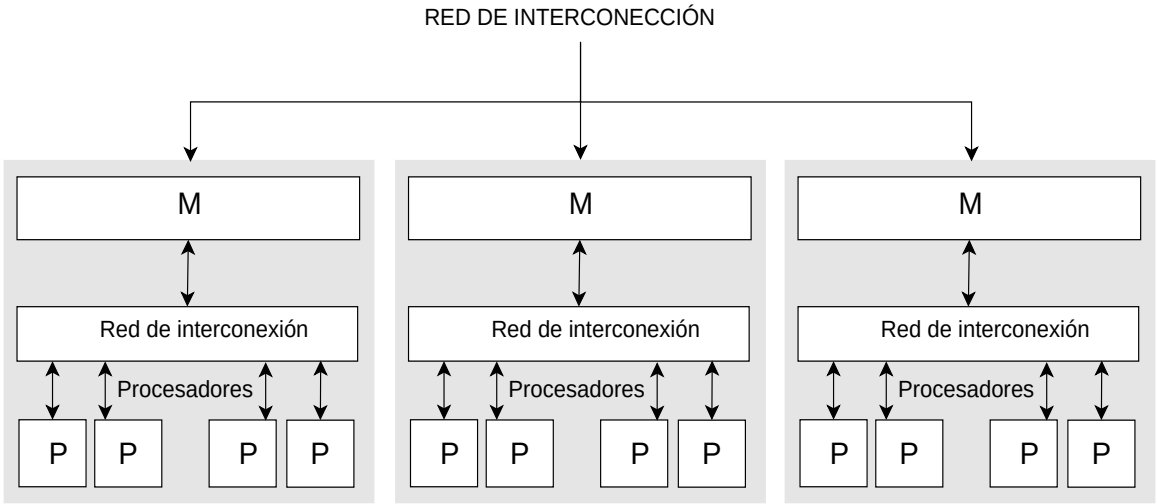

Figura 2.11: Representación de un entorno paralelo híbrido.

Algoritmo 7 Algoritmo paralelo para entornos híbridos

```
1: procedure CalculateParameterOnHibridSystems(inputDEM, G)
2:     n ← GetNSize(inputDEM)
3:     m ← GetMSize(inputDEM)
4:     D_p ← PartitionData(n,m,G) {Particionar los datos en subdominios horizontales}
5:     En paralelo: para cada pr = {0,1,2...,p-1} do {Cada procesador "pr" realiza las
6:     operaciones de lectura y cálculo del parámetro en el subdominio D_p correspondiente}
7:     if pr == root then
8:         outputDEM ← CreateOutputFile() {Se crea el fichero MDE de salida}
9:     end if
10:    A ← ReadInputData(inputDEM, D_p) {Leer el subdominio D_p correspondiente}
11:    numPlatforms ← GetOpenCLDevices() {Obtener los dispositivos OpenCL disponibles}
12:    if numPlatforms >0 then
13:        platform ← GetPlatformIds(CL_DEVICE_TYPE_GPU)
14:        if platform == 0 then {Consultar si existe GPU disponible, en otro caso se procesa
15:            en la CPU}
16:            CalculateParameterOnCPU() {Se procesa en multinúcleos mediante el Algoritmo 2}
17:        else
18:            device ← GetGPUDevice(platform, CL_DEVICE_TYPE_GPU)
19:            CalculateParameterOnGPU() {Se procesa en GPU mediante el Algoritmo 3}
20:        end if
21:    end if
22:    if pr ≠ root then
23:        Send(A', root) {Cada procesador envía el resultado del cálculo al procesador root}
24:    end if
25:    if pr == root then
26:        Receive(A') {El procesador root recibe los resultados del cálculo}
27:        WriteResult(outputDEM, A') {Escribe los resultados recibidos}
28:    end if
29: end procedure
```

2.5. Conclusiones parciales

Como consideraciones del presente capítulo se relacionan las siguientes:

- La utilización de varias arquitecturas paralelas: memoria compartida (tanto en CPU como en GPU), memoria distribuida y esquemas híbridos, permite la portabilidad y heterogeneidad de los algoritmos que se proponen.
- Aunque se propone el uso de la GPU para acelerar el procesamiento, en entornos computacionales que no cuenten con tarjetas gráficas en su configuración de hardware, el proce-

samiento se realiza en la CPU multinúcleo.

- El uso del cómputo voluntario mediante la Plataforma T-Arenal permite realizar el procesamiento en un sistema distribuido adecuado para instituciones con muchas estaciones de trabajo interconectadas a través de una red local y sin recursos especialmente destinados a la realización de tareas de cálculo.
- La implementación de los algoritmos en un esquema híbrido permite explotar la naturaleza heterogénea del sistema, en el que la memoria global es distribuida y los nodos subscritos puede que contengan tecnologías multinúcleos (ya sea CPU o GPU) en las que la memoria local es compartida.

Capítulo 3

RESULTADOS Y DISCUSIÓN

En el presente capítulo se exponen los resultados obtenidos a partir de un conjunto de pruebas realizadas a los algoritmos paralelos propuestos. Primeramente se describe la herramienta para el análisis del terreno implementada y se exponen un conjunto de métricas definidas para la evaluación de algoritmos paralelos. Finalmente se realiza una valoración de los resultados obtenidos, mediante el cálculo de la ganancia de velocidad y de la eficiencia, tomando como entrada los tiempos arrojados en los experimentos prácticos para cada uno de los entornos paralelos propuestos.

3.1. Herramienta para el análisis del terreno

Con el objetivo de realizar una evaluación práctica del desempeño de los algoritmos paralelos propuestos se implementó una herramienta para la biblioteca GDAL que permite realizar el cálculo de tres parámetros del terreno: la pendiente, el sombreado y la rugosidad, teniendo en cuenta que estos constituyen parte de los ya implementados en `GDALDEM`.

La Figura 3.1, muestra a grandes rasgos los principales componentes de la estructura interna de GDAL: utilidades de línea de comando (GDAL Apps), algoritmos utilizados por la biblioteca para el análisis *raster* (GDAL Alg), un paquete de *scripts* escritos en otros lenguajes de programación para la realización de determinadas acciones mediante la API de GDAL (GDAL Swig), así como su núcleo (GDAL Core, GDAL Port, GDAL Driver/Formats).

La herramienta implementada (`GDALDEM_PARALLEL`) se integra al conjunto de utilidades que provee GDAL para el manejo y análisis de los datos *raster*. En su implementación se utilizaron las funciones del núcleo de GDAL para la lectura/escritura de este tipo de dato geográfico, así como otras clases que permiten manejar la variedad de formatos que soporta.

Los algoritmos propuestos fueron implementados en el lenguaje de programación C/C++. La versión para la arquitectura multinúcleo fue implementada utilizando OpenCL y OpenMP. En este último caso el tipo de planificación definida para determinar la cantidad de iteraciones asignadas a cada hilo es estática (`static`), a través de la cual se dividen las iteraciones en bloques de igual tamaño. En el caso de la propuesta para la arquitectura distribuida se utilizó además el lenguaje Java en la redefinición de las clases *DataManager* y *Task*, propias de la Plataforma T-Arenal. La plataforma empleada para el procesamiento en GPU es OpenCL 1.1 AMD-APP-

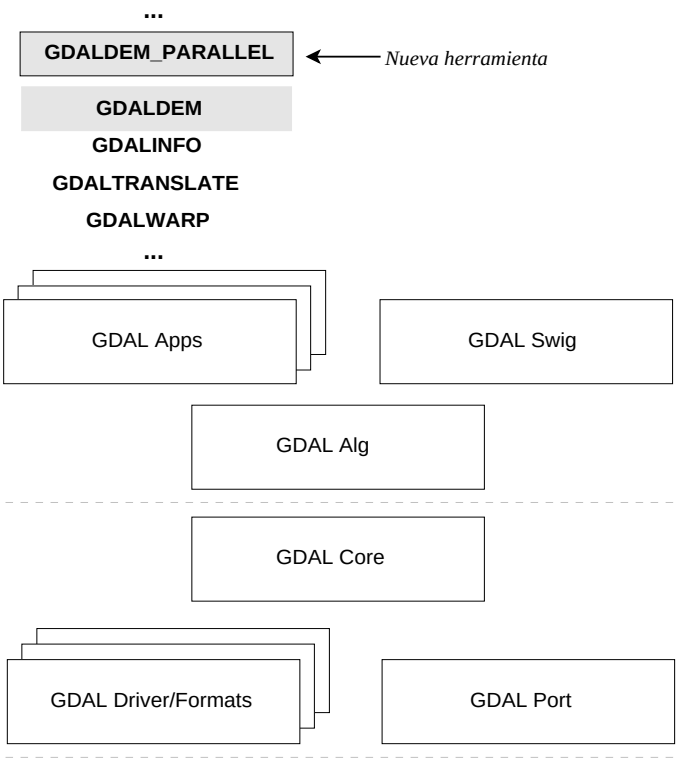

Figura 3.1: Integración de la herramienta implementada en la Biblioteca GDAL.

SDK-v2.4-rc1 (595.9) FULL_PROFILE[1].

Aunque en la herramienta desarrollada se implementaron tres funciones de análisis del terreno, la propuesta realizada puede ser aplicable a otros algoritmos de este tipo que describen funciones de análisis focal utilizando ventanas con tamaño n, como es el caso del pre-procesamiento de los MDE para el análisis de modelos hidrológicos en la extracción de redes de drenaje, la extracción del índice de escabrosidad del terreno, así como su índice de posición topográfica.

3.2. Métricas para la evaluación de algoritmos paralelos

En el análisis y diseño de algoritmos paralelos se tienen en cuenta un conjunto de métricas que permiten definir la calidad de las prestaciones de un algoritmo paralelo, tomando como base el tamaño del problema y la cantidad de procesadores empleados. A continuación se definen las métricas más generalizadas [Grama y col., 2003]:

Tiempo de ejecución

El tiempo de ejecución T_P de un algoritmo paralelo con P procesadores es el tiempo que trascurre

[1]Disponible en: http://www.geeks3d.com/20110407/amd-app-sdk-opencl-v2-4-available/

desde el momento en el que el primer procesador comienza la ejecución del programa hasta que el último procesador culmina. El mismo está dado por la siguiente función:

$$T_P \approx T_A + T_C \tag{3.1}$$

donde T_A es el tiempo empleado por el algoritmo para realizar las operaciones aritméticas y T_C es el tiempo de comunicación entre los procesadores. Además se debe tener en cuenta el tiempo de solapamiento T_O, que es el tiempo en el que las operaciones aritméticas y la comunicación entre los procesadores ocurre de forma simultánea. Sin embargo, la complejidad de su cálculo propicia que se establezca la ecuación anterior como una aproximación.

Ganancia de velocidad

La ganancia de velocidad[2] S_P para p procesadores, se refiere al cociente entre el tiempo del mejor algoritmo secuencial T_S y el tiempo del algoritmo paralelo T_P. El *speed-up* expresa cuánto se ha reducido el tiempo de procesamiento al ejecutar el algoritmo en p procesadores y está dado por la siguiente ecuación:

$$S_P = \frac{T_S}{T_P} \tag{3.2}$$

Se considera que, tomando como base que el tiempo de un algoritmo paralelo T_P ejecutado en p procesadores es, como máximo, p veces inferior al tiempo secuencial T_S del algoritmo, entonces el límite del *speed-up* S_P está definido como $\lim_{n\to\infty} S_P = p$.

Eficiencia

La eficiencia se refiere al porcentaje de aprovechamiento de los procesadores empleados en la ejecución del algoritmo. Su valor está dado en el rango de 0 a 1 y se expresa mediante el cociente entre la ganancia de velocidad S_P y la cantidad p de procesadores.

$$E_P = \frac{S_P}{p} \tag{3.3}$$

3.3. Resultados experimentales

3.3.1. Experimentación en memoria compartida

En los experimentos para memoria compartida se utilizaron cuatro configuraciones de hardware, dos para medir los tiempos de CPU y otras dos para medir los tiempos de GPU. Las Tablas 3.1

[2]Conocida también como *Speed-Up* por su traducción al inglés.

y 3.2 muestran las principales características de cada configuración.

Tabla 3.1: Características de las configuraciones de hardware utilizadas en los experimentos en CPU.

Característica	HARDWARE I (H-I)	HARDWARE II (H-II)
Procesador	Intel(R) Core(TM) i3-2120	Intel(R) Core(TM) i5-3570K
RAM	4096 MB	4096 MB
Velocidad	3.30 GHz	3.40 GHz
Núcleos CPU	4 (2 físicos y 2 lógicos)	4 (físicos)

Tabla 3.2: Características de las configuraciones de hardware utilizadas en los experimentos en GPU.

Característica	HARDWARE III (H-III)	HARDWARE IV (H-IV)
GPU	AMD Radeon HD 6480G	AMD Radeon HD 8510G
Stream Cores	240	384
Memoria gráfica	2033 MB	4096 MB

Como fuente de datos para los experimentos se emplearon ficheros de tipo GeoTIFF[3]. Los datos de las muestras de MDE empleadas en los experimentos se muestran en la Tabla 3.3.

Tabla 3.3: Características de los MDE empleados en los experimentos.

Modelo	Tamaño (nxm)	Cantidad de pixeles	Tamaño (en MB)
M1	3937 x 1794	7 127 562	20,2
M2	7874 x 3589	28 259 786	80,8
M3	11811 x 5383	63 578 613	181
M4	15748 x 7178	113 297 552	323

La Tabla 3.4 muestra los tiempos de CPU obtenidos en la ejecución de la versión secuencial y las versiones paralelas propuestas para memoria compartida en las configuraciones de hardware H-I y H-II, empleando las cuatro muestras de MDE descritas en la Tabla 3.3. Los tiempos de respuesta obtenidos, son el resultado del promedio de los tiempos arrojados en la ejecución de tres iteraciones por cada uno de los modelos utilizados. Durante los experimentos se varió el número de hilos a utilizar (desde 2 hasta 4), para determinar la cantidad que ofrece mejor rendimiento. Los resultados reflejan que con la aplicación de la variante de procesamiento paralelo propuesta, los tiempos de procesamiento pueden ser mejorados de 30,253 a 14,205 segundos en el caso del cálculo de la pendiente para el modelo M4 en la configuración de hardware H-I; y de 22,476 a 7,937 segundos en el caso del cálculo de la rugosidad para el modelo M4 en la configuración

[3]Consultar: http://www.gdal.org/frmt_gtiff.html

de hardware H-II, representando estos valores una reducción aproximada del 53 % y el 65 % del tiempo de ejecución en correspondencia con los algoritmos secuenciales respectivamente.

Tabla 3.4: Tiempos de CPU de los algoritmos propuestos para diferentes muestras de MDE en las configuraciones de hardware H-I y H-II.

Configuración	Algoritmo	Modelo	Cantidad de hilos/Tiempo (s)			
			1	2	3	4
H-I	Pendiente	M1	1,156	0,658	0,568	0,468
		M2	4,422	2,560	2,001	1,745
		M3	9,735	5,435	4,365	3,894
		M4	30,253	19,053	17,011	14,205
	Rugosidad	M1	1,416	0,788	0,656	0,547
		M2	4,871	2,719	2,375	1,975
		M3	10,282	5,969	4,513	4,013
		M4	31,433	23,502	18,090	15,658
	Sombreado	M1	2,949	1,507	1,410	1,119
		M2	6,561	3,919	3,075	2,280
		M3	11,605	6,969	6,013	5,113
		M4	34,220	24,013	19,994	16,231
H-II	Pendiente	M1	0,533	0,365	0,299	0,212
		M2	0,533	0,365	0,299	0,212
		M3	7,187	4,447	3,926	3,101
		M4	17,380	10,971	9,712	7,183
	Rugosidad	M1	0,839	0,519	0,412	0,280
		M2	3,520	2,115	1,561	1,075
		M3	9,001	6,127	4,117	3,259
		M4	19,221	12,481	10,190	7,192
	Sombreado	M1	0,997	0,527	0,410	0,332
		M2	4,530	2,762	1,969	1,480
		M3	9,512	5,563	4,221	3,310
		M4	22,476	13,363	11,361	7,937

Las gráficas de las Figuras 3.2 y 3.3 muestran los resultados reflejados en la Tabla 3.4. Además, con el objetivo de esclarecer la razón por la cual se empleó OpenMP en la paralelización para multinúcleos en lugar de OpenCL, en la Figura 3.2 se realiza una comparación con los resultados

arrojados por OpenCL en la CPU. En este caso, mediante OpenMP se obtienen mejores tiempos que con OpenCL. El trabajo con OpenCL implica la ejecución de llamadas que pueden ocasionar sobrecarga de tiempo debido a su nivel adicional, por ejemplo: la consulta de información sobre los dispositivos existentes, la asignación de memoria, y la configuración y carga de los *kernels*.

Además, un factor que disminuye la eficiencia y el rendimiento de OpenCL en CPU está estrechamente relacionado con la granularidad del paralelismo y la planificación. Mientras que OpenMP implementa paralelismo de grano grueso mediante la planificación estática, al dividir las iteraciones en bloques de igual tamaño para asignarlas a un hilo, OpenCL por su parte ofrece paralelismo de grano fino mediante la asignación de una tarea a cada elemento de procesamiento. Cuando se trata de procesar en la GPU esta característica es provechosa, pues la GPU tiene cientos de núcleos, pero en el caso de la CPU, que cuenta con menos núcleos, esto puede ocasionar una sobrecarga en la planificación. En este sentido, existen variantes de optimización para mejorar el rendimiento de OpenCL en CPU, pero complejizan la implementación, por lo que se decidió no aplicarlas.

No obstante, es válido aclarar que el uso de OpenCL, tanto para las CPU multinúcleo como para las GPU, garantiza mayor simplicidad en la implementación y además mayor portabilidad, debido a que se hace uso de una sola tecnología para la paralelización. Sin embargo, el rendimiento y la eficiencia se ven afectados, por tanto se decide optar por el uso de OpenMP, que además de brindar buenos resultados, igualmente es un estándar portable e incluso más maduro que OpenCL.

Los resultados de OpenCL en CPU en la configuración de hardware H-II, muestran similar comportamiento a los obtenidos en la configuración H-I, en comparación con las variantes OpenMP.

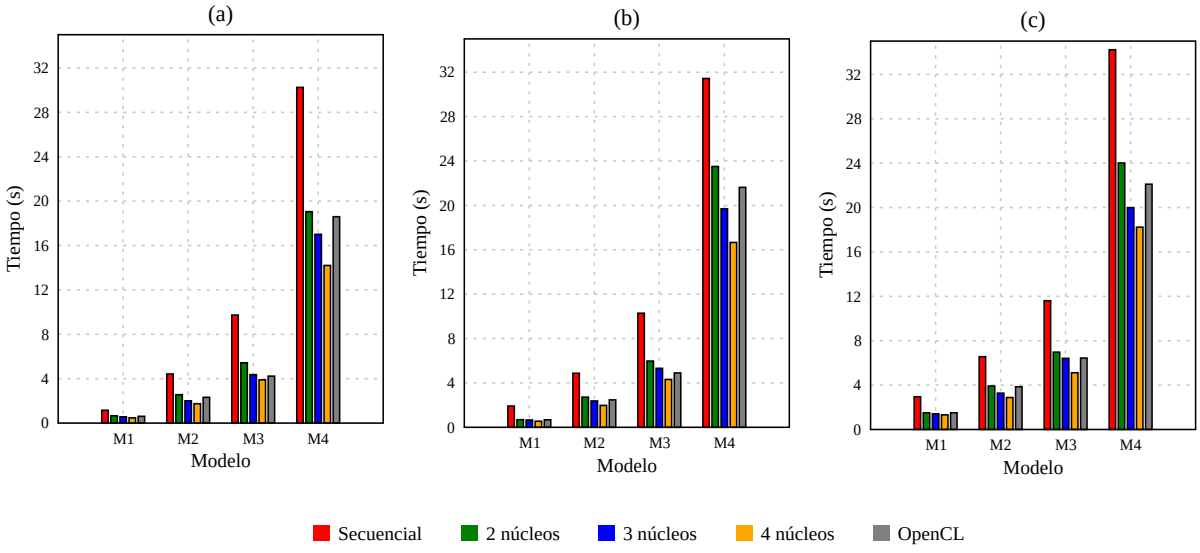

Figura 3.2: Comparación de los tiempos de CPU de la versión secuencial y de los algoritmos paralelos con OpenMP en la configuración de hardware H-I (a) Pendiente (b) Rugosidad (c) Sombreado.

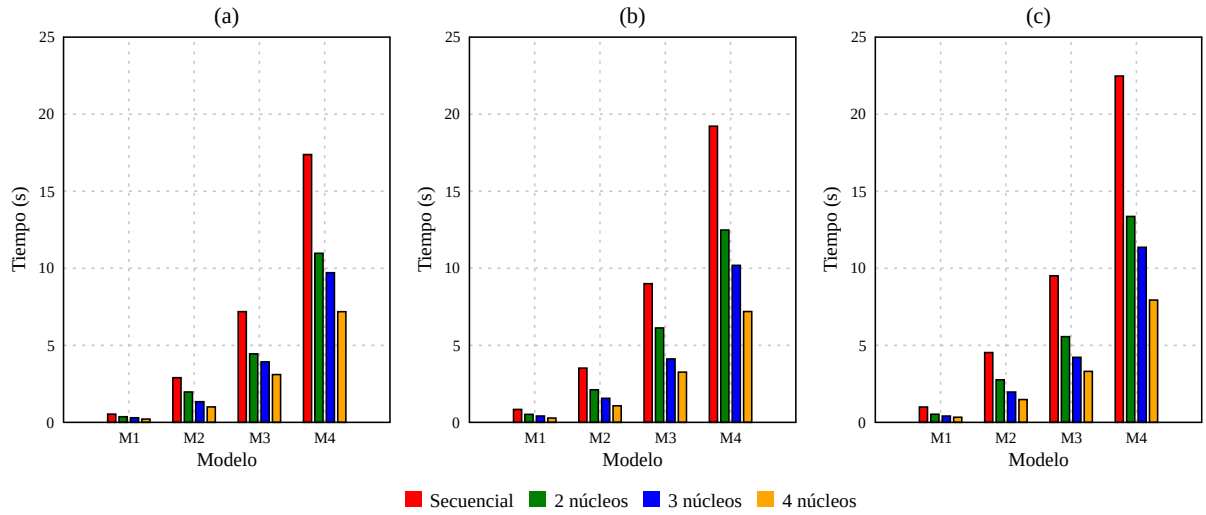

Figura 3.3: Comparación de los tiempos de CPU de la versión secuencial y de los algoritmos paralelos con OpenMP en la configuración de hardware H-II (a) Pendiente (b) Rugosidad (c) Sombreado.

En las gráficas de las Figuras 3.4 y 3.5 se muestra el comportamiento de la ganancia de velocidad para los tres algoritmos a medida que aumenta la cantidad de hilos utilizados. Se puede observar que de manera general todas las muestras presentan similar comportamiento (aunque en la configuración de hardware H-II se obtuvieron mayores valores para esta métrica), indicando que el número óptimo de hilos a utilizar debe ser cuatro, pues este es el caso en que se alcanza mayor ganancia de velocidad, manteniéndose relativamente cerca del óptimo teórico.

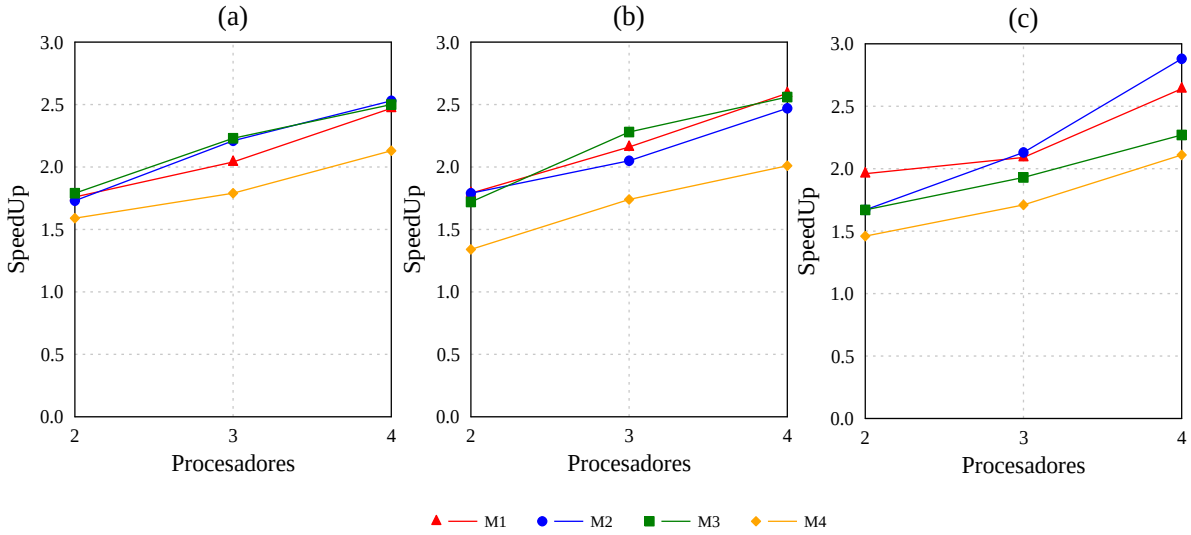

Figura 3.4: Comportamiento de la ganancia de velocidad de la versión paralela OpenMP en la configuración de hardware H-I (a) Pendiente (b) Rugosidad (c) Sombreado.

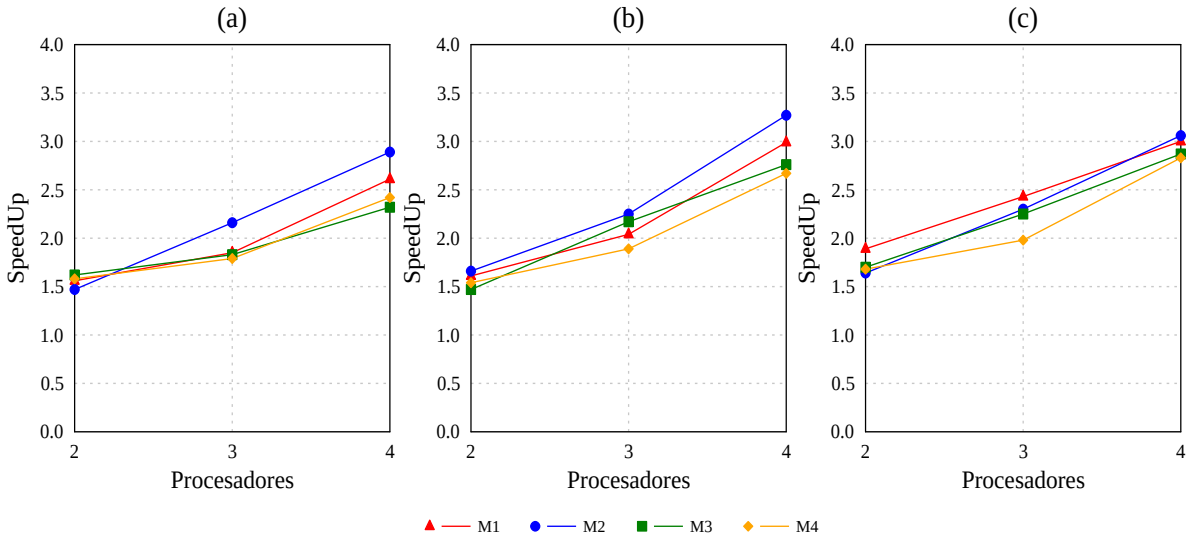

Figura 3.5: Comportamiento de la ganancia de velocidad de la versión paralela OpenMP en la configuración de hardware H-II (a) Pendiente (b) Rugosidad (c) Sombreado.

Sin embargo, en el caso de la configuración de hardware H-I, se puede apreciar que aún cuando la ganancia de velocidad tiende a crecer relativamente cuando se utilizan cuatro procesadores, se está más cerca del óptimo teórico cuando se usan dos. Además, como muestran las gráficas de la Figura 3.6, la mayor eficiencia se alcanza cuando se utilizan dos procesadores. Este comportamiento es totalmente esperado, pues en este caso se cuenta realmente con dos procesadores físicos, y debido a la tecnología *Intel Hyper-Threading* cada uno de estos procesadores físicos simula dos procesadores lógicos, conformando los cuatro procesadores visibles al Sistema Operativo y a OpenMP. Por tanto, teniendo en cuenta que se incrementa el uso de las unidades de ejecución de los procesadores y que además los procesadores virtuales no brindan el mismo desempeño que los físicos, es de esperar que en un momento determinado si existe sobrecarga de procesamiento, los algoritmos tiendan a perder en eficiencia.

Para la configuración de hardware H-II (donde se cuenta con cuatro procesadores físicos sin tecnología *Hyper-Threading*) se obtuvieron mayores valores de eficiencia, en la mayoría de los casos por encima de 0,6, Figura 3.7. Sin embargo, para algunos casos, igualmente se observa que esta métrica tiende a disminuir a medida que aumentan los hilos. Este comportamiento está dado principalmente porque se trata de muestras de MDE de tamaño pequeño. En este sentido, un elemento importante a tener en cuenta cuando se trata de procesar MDE, es que el tiempo de procesamiento está dado por el tiempo de las operaciones de lectura y escritura (parte secuencial del algoritmo), sumado al tiempo de las operaciones aritméticas para cada celda de la matriz (parte paralelizable). Por tanto es de esperar que en un momento dado, manteniendo el tamaño del MDE constante, aunque aumenten los procesadores a utilizar, se deje de ganar velocidad ya que se disminuye el tiempo de la parte paralelizable pero el tiempo de la parte

secuencial se mantiene constante, en este caso la lectura y escritura de los ficheros.

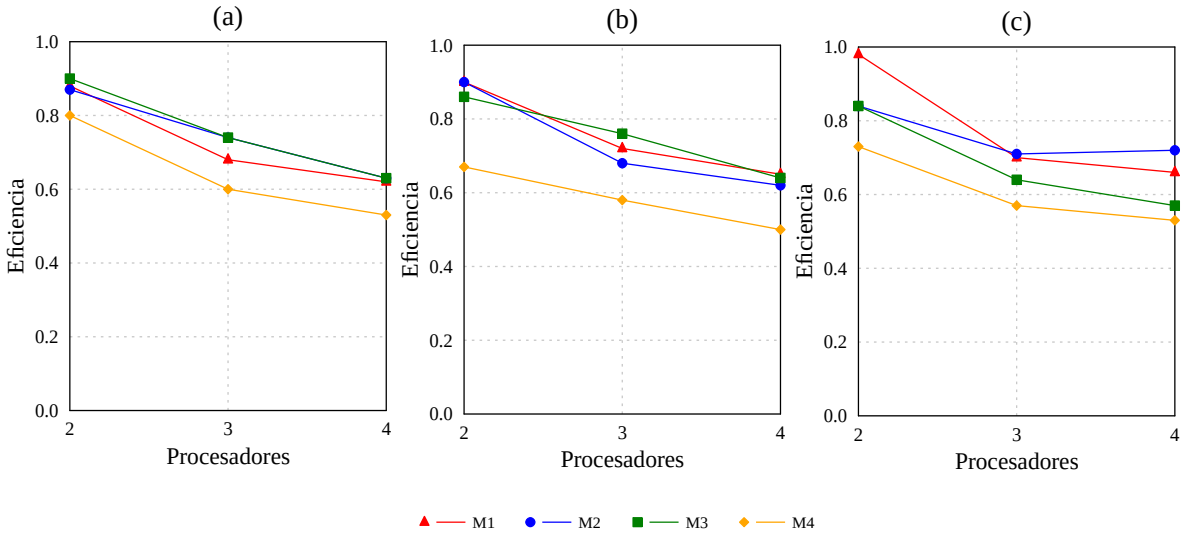

Figura 3.6: Comportamiento de la eficiencia de la versión paralela OpenMP en la configuración de hardware H-I (a) Pendiente (b) Rugosidad (c) Sombreado.

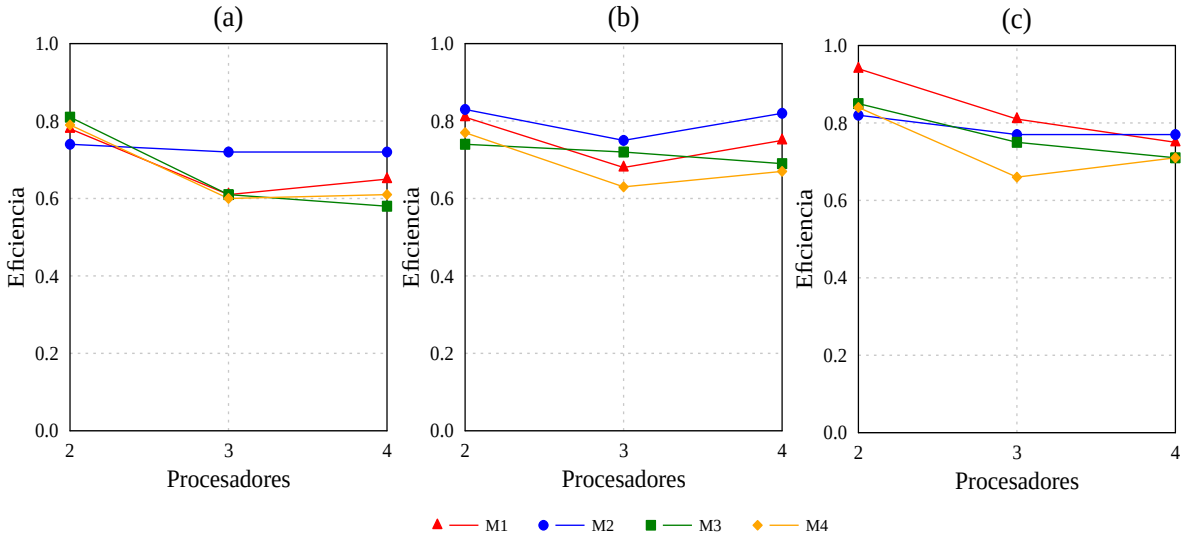

Figura 3.7: Comportamiento de la eficiencia de la versión paralela OpenMP en la configuración de hardware H-II (a) Pendiente (b) Rugosidad (c) Sombreado.

Por su parte la Tabla 3.5 muestra los tiempos de GPU alcanzados con la versión OpenCL propuesta, empleando las configuraciones de hardware H-III y H-IV descritas en la Tabla 3.2.

Los resultados reflejan que con la aplicación de la variante de procesamiento en GPU propuesta, los tiempos de procesamiento pueden ser mejorados de 34.220 a 3.843 segundos en el caso del cálculo del sombreado para el modelo M4 en la configuración de hardware H-IV; y de 34.220

Tabla 3.5: Tiempos de GPU de los algoritmos propuestos para diferentes muestras de MDE en las configuraciones de hardware III y IV.

Algoritmo	Modelo	Hilos generados	Tiempo (s)		
			Secuencial	Paralelo	
				H-III	H-IV
Pendiente	M1	3935	1,156	2.032	1.012
	M2	7872	4,422	2.641	1.312
	M3	11809	9,735	3.607	2.041
	M4	15746	30,253	4.479	3.109
Rugosidad	M1	3935	1,416	2.598	1.587
	M2	7872	4,871	3.038	2.002
	M3	11809	10,282	3.915	2.223
	M4	15746	31,433	4.929	3.579
Sombreado	M1	3935	2,949	2.882	1,048
	M2	7872	6,561	3.739	1,377
	M3	11809	11,605	4.110	2,922
	M4	15746	34,220	5.081	3,843

a 5.081 segundos para el mismo caso en la configuración de hardware H-III, representando estos valores una reducción del 89 % y el 85 % del tiempo de ejecución en correspondencia con el algoritmo secuencial respectivamente, con una ganancia de velocidad de hasta 9, 73 y 6, 75 según las gráficas de la Figura 3.8.

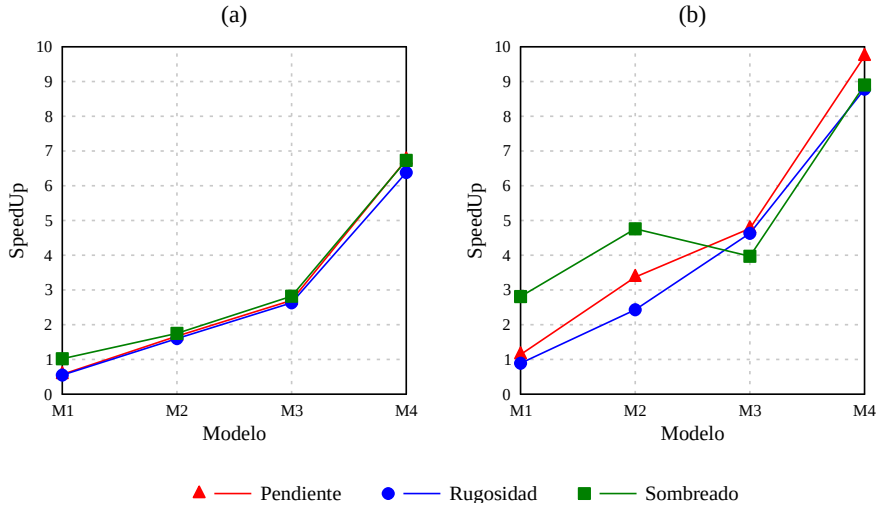

Figura 3.8: Comportamiento de la ganancia de velocidad de la versión paralela en GPU en las configuraciones de hardware (a) H-III y (b) H-IV.

Incluso, en comparación con la variante paralela que emplea OpenMP (Figura 3.9), en la mayoría de los casos, la ganancia de velocidad en la GPU es notoria. Los resultados obtenidos muestran mejor desempeño según los datos reportados en las gráficas de la Figura 3.9, con tiempos de procesamiento que pueden ser reducidos de 16,231 a 3,843 segundos (en caso de utilizar cuatro núcleos en el cálculo del sombreado para el modelo M4 en la configuración de hardware H-I y la GPU en la configuración de hardware H-IV), lo que se traduce en una disminución del 76 % del tiempo de ejecución aproximadamente. Esto demuestra que el desempeño de las GPU excede significativamente al de las CPU en términos de capacidad de procesamiento, aunque a costa de utilizar mayor cantidad de núcleos.

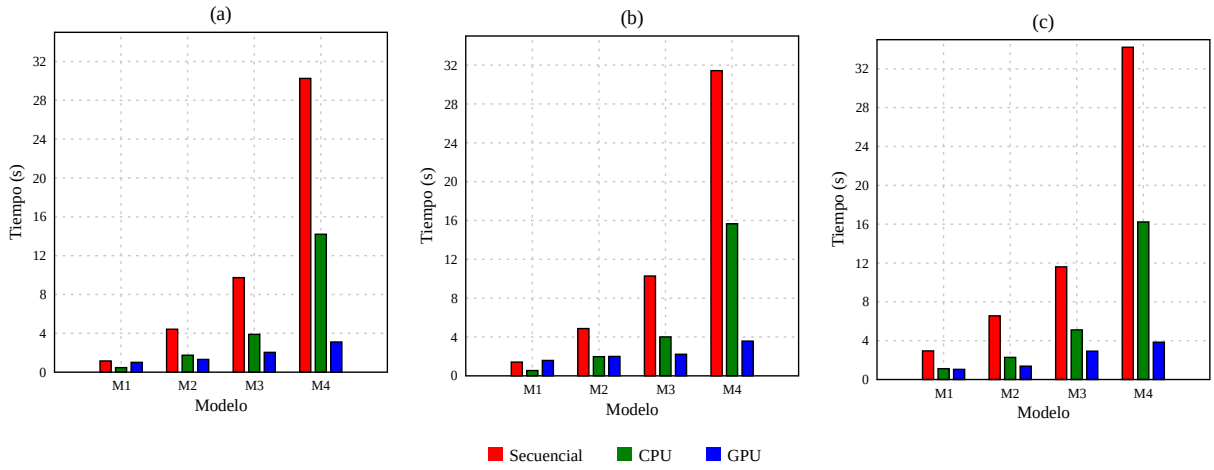

Figura 3.9: Comparación de los tiempos de GPU de los algoritmos paralelos propuestos con la versión secuencial y la versión para multinúcleos en las configuraciones de hardware H-I y H-IV (a) Pendiente (b) Rugosidad (c) Sombreado.

Sin embargo, cuando se trata del empleo de OpenCL debe medirse el tiempo para: (1) el envío de los datos a la memoria del dispositivo GPU y (2) el envío de los resultados hacia el *host*. Es por esto que cuando se trata del procesamiento en la GPU, es notable que la reducción del tiempo de ejecución del algoritmo para muestras de modelos pequeños[4] es despreciable, pues para la ejecución de los *kernels* en la tarjeta gráfica se requiere de un gasto extra en la transferencia de los datos a la memoria del dispositivo y viceversa, como muestra la Figura 3.10. No obstante, es notorio el elevado desempeño del algoritmo a medida que aumenta el tamaño del modelo a procesar.

[4]Modelos con menos de 8 millones de puntos.

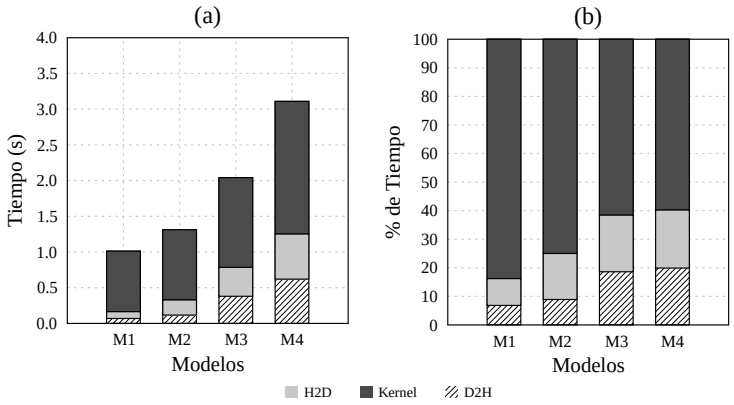

Figura 3.10: Tiempo de transferencia de datos CPU-GPU (H2D) y GPU-CPU (D2H) para el cálculo de la pendiente en la configuración de hardware H-IV. (a) En función del tiempo en segundos (b) En función del tiempo en %.

3.3.2. Experimentación en memoria distribuida

Con el objetivo de realizar una evaluación de los algoritmos propuestos para el análisis del terreno en entornos distribuidos se desplegó la Plataforma T-Arenal en 17 estaciones de trabajo no dedicadas del laboratorio 303 del Centro Geoinformática y Señales Digitales (GEYSED) de la Universidad de las Ciencias Informáticas (UCI). Todas las estaciones cuentan con el Sistema Operativo Ubuntu 14.04 LTS y están interconectadas mediante una red Ethernet no dedicada a una velocidad promedio de 100Mb/s. Del total de estaciones configuradas, 16 cuentan con la configuración de hardware H-I y una cuenta con la configuración de hardware H-IV, ambas descritas en las Tablas 3.1 y 3.2 respectivamente. Fue configurado un servidor central y un servidor de peticiones en una misma estación con las características de la configuración de hardware H-I y las estaciones restantes fueron configuradas como clientes del sistema. Se realizaron dos experimentos, en el primero una de las estaciones de trabajo estaba siendo usada por una persona en las tareas habituales, para un 6 % de utilización del total; y en el segundo, se incrementó el número de personas trabajando en las estaciones a nueve, para un 53 % de utilización.

Se utilizaron dos *datasets* con tamaños de 19 685 x 8 972 y 86 404 x 28 804 respectivamente. El *dataset* de menor tamaño cubre el área del archipiélago cubano con una resolución de 1m y el segundo es un *raster* de la región de Sudamérica generado por el proyecto SRTM con una resolución de 500m. Con respecto a este último, es importante destacar que la relevancia, en relación con los experimentos realizados, radica en el procesamiento de un modelo con un tamaño considerable y no se centra en la realización de un análisis a nivel global. Además, fueron utilizados varios valores de granularidad para ambos *datasets*, con un valor máximo de 10 000 filas para el primero y 30 000 filas para el segundo.

La Tabla 3.6 muestra el tiempo del algoritmo para el cálculo del sombreado a medida que aumenta el número de procesadores y varía la granularidad. Teniendo en cuenta que T-Arenal es un

sistema dinámico, es decir, no es posible determinar a priori la cantidad de estaciones disponibles que intervienen en el procesamiento, se incrementó gradualmente el número de clientes autorizados con el objetivo de variar la cantidad de procesadores. Para calcular el tiempo secuencial del algoritmo se tomó la máquina de mejores prestaciones visible a T-Arenal.

Tabla 3.6: Tiempos del algoritmo distribuido para el cálculo del sombreado empleando diferentes muestras de *datasets* y variando la granularidad.

MDE	Granularidad	Cantidad de procesadores/Tiempo (s)								
		1	2	4	6	8	10	12	14	16
Experimento 1: 6 % de utilización de las estaciones de trabajo										
19685 x 8972	200	241,8	201,1	176,3	132,4	120,2	88,5	68,4	61,5	55,3
	600		199,4	172,2	130,3	112,5	79,2	62,7	59,2	50,9
	1000		199,7	168,1	122,7	98,3	71,4	58,1	50,7	49,2
	3000		192,2	159,8	110,1	91,3	64,8	48,2	45,3	42,8
	5000		196,8	158,9	103,3	93,6	65,2	51,6	46,7	43,2
	10000		200,4	162,3	115,8	91,2	70,7	52,3	47,9	43,9
86404 x 28804	1000	1122,8	786,3	590,3	432,4	356,2	244,5	179,4	127,2	126,9
	3000		784,1	596,2	440,1	342,9	256,1	182,2	127,9	124,5
	5000		771,6	569,1	399,6	312,9	229,6	170,8	114,1	113,2
	10000		760,1	568,8	398,9	320,3	236,5	161,6	118,2	117,4
	20000		761,3	550,9	395,1	301,9	224,2	165,1	118,9	115,2
	30000		763,4	571,3	418,4	321,3	227,3	167,9	119,1	118,2
Experimento 2: 53 % de utilización de las estaciones de trabajo										
19685 x 8972	200	241,8	213,8	198,3	152,7	131,8	116,2	89.1	89.6	78.3
	600		211,9	195,2	155,1	137,3	108,9	85.2	80.2	76.2
	1000		202,7	182,9	143,9	121,3	95,1	79.7	75.3	73.9
	3000		199,1	178,2	130,7	117,4	87,8	80.3	69.9	67.7
	5000		205,9	177,9	139,5	104,2	84,2	73.7	68.1	65.1
	10000		206,2	184,8	149,2	122,1	94,6	87.1	78.4	69.2
86404 x 28804	1000	1122,8	897,7	752,6	624,4	497,1	402,7	368,3	315,3	273.9
	3000		853,1	712,7	601,9	503,6	387,9	362,1	276,7	266.5
	5000		833,6	701,3	582,6	460,1	366,1	315,3	262,1	224.5
	10000		850,8	660,2	572,8	477,8	350.,9	347,3	250,6	243.7
	20000		874,3	729,4	626,8	507,1	370,3	362,2	279,4	236.1
	30000		892,4	748,1	610,4	553,1	431,8	359,1	286,2	268.8

Los resultados muestran una reducción considerable del tiempo de procesamiento a medida que se incrementan los procesadores en uso. En los mejores casos, según la ganancia de velocidad

obtenida en el primer experimento (cuando se utiliza el máximo de procesadores con $G = 3000$ y cuando se utiliza el máximo de procesadores con $G = 10000$ respectivamente), se observa que el tiempo secuencial fue reducido de 241,8 segundos (4,03 minutos) a 42.8 segundos para el primer *dataset* y de 1122,8 segundos (18,7 minutos) a 117,4 segundos (1,9 minutos) para el segundo *dataset*.

En el segundo experimento obviamente se obtuvieron mayores tiempos que en el primero. Esto se debe al incremento del uso de las estaciones clientes al momento de realizar el cálculo paralelo. Aún así, los tiempos fueron reducidos de 241,8 a 65,1 segundos (1,09 minutos) para $G = 5000$ en el caso del primer *dataset*, y de 1122,8 a 224,5 segundos (3,7 minutos) con $G = 10000$ para el segundo *dataset*, lo que refleja el beneficio claro de ejecutarse en un entorno no dedicado.

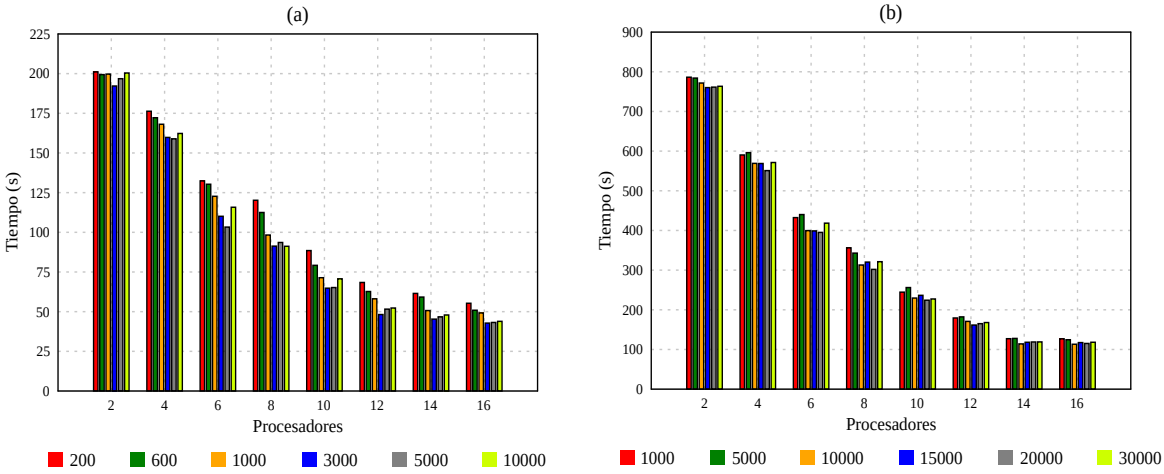

Figura 3.11: Comparación de los tiempos del algoritmo para el cálculo del sombreado en entornos distribuidos utilizando diferentes valores de granularidad en el experimento 1 (a) 19 685 x 8 972 (b) 86 404 x 28 804.

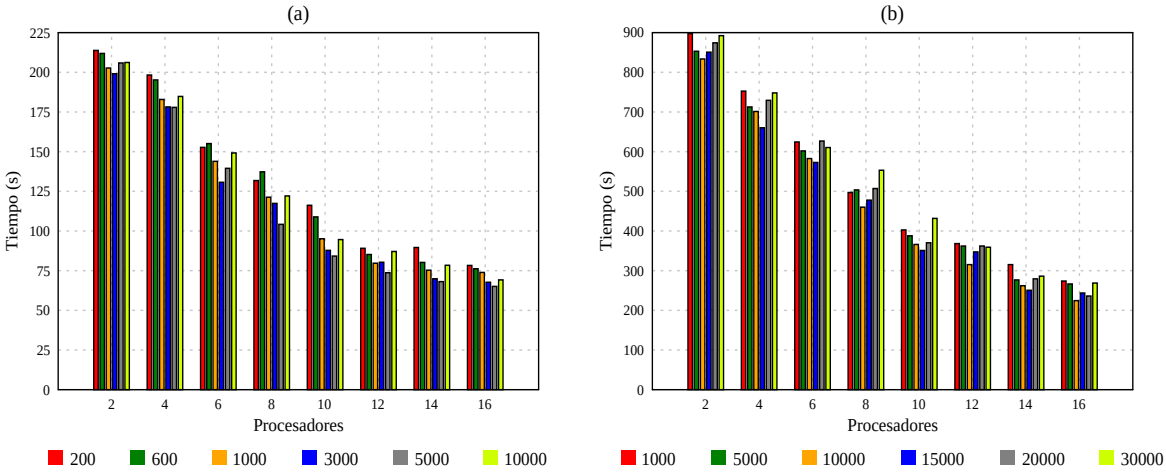

Figura 3.12: Comparación de los tiempos del algoritmo para el cálculo del sombreado en entornos distribuidos utilizando diferentes valores de granularidad en el experimento 2 (a) 19 685 x 8 972 (b) 86 404 x 28 804.

Tanto en la Tabla 3.6 como en las gráficas de las Figuras 3.11 y 3.12 se observa que para tamaños de granularidad pequeños (200, 600) el tiempo de procesamiento es mayor. Este comportamiento está dado por el aumento de las comunicaciones cuando se distribuyen bloques de filas pequeños. Pues aunque exista poca dependencia entre los datos, se incrementan las peticiones de trabajo y el envío de resultados por parte de los clientes, y en consecuencia la asignación de tareas por parte de los servidores.

En cuanto a la elección de la granularidad ideal para distribuir los datos, según los resultados alcanzados, se considera tener en cuenta que para valores de G muy pequeños aumentan las comunicaciones entre los clientes y los servidores, lo cual influye directamente en el tiempo total. Sin embargo, tampoco es recomendable utilizar valores de G muy grandes que impacten el costo de las operaciones de lectura/escritura en los clientes y que ocasionen desbalance de carga en el procesamiento. Por esta razón, G se concibe como un parámetro variable y configurable en los algoritmos.

En las gráficas de las Figuras 3.13 y 3.14 se muestra el comportamiento de la ganancia de velocidad para ambos *datasets* a medida que aumentan los procesadores. Por su parte en las Figuras 3.15 y 3.16 se muestra el comportamiento de la eficiencia. En ambos casos se observa que el comportamiento no es proporcional al número de procesadores utilizados. En este sentido, es importante destacar que aunque exista un número máximo de procesadores visibles a T-Arenal, se debe tener en cuenta que el sistema se caracteriza por ser dinámico, por lo que no en todo momento se cuenta con la cantidad máxima ideal de clientes. Por tanto, la ganancia de velocidad y la eficiencia, en ocasiones, deben estar lejos de su supuesto teórico.

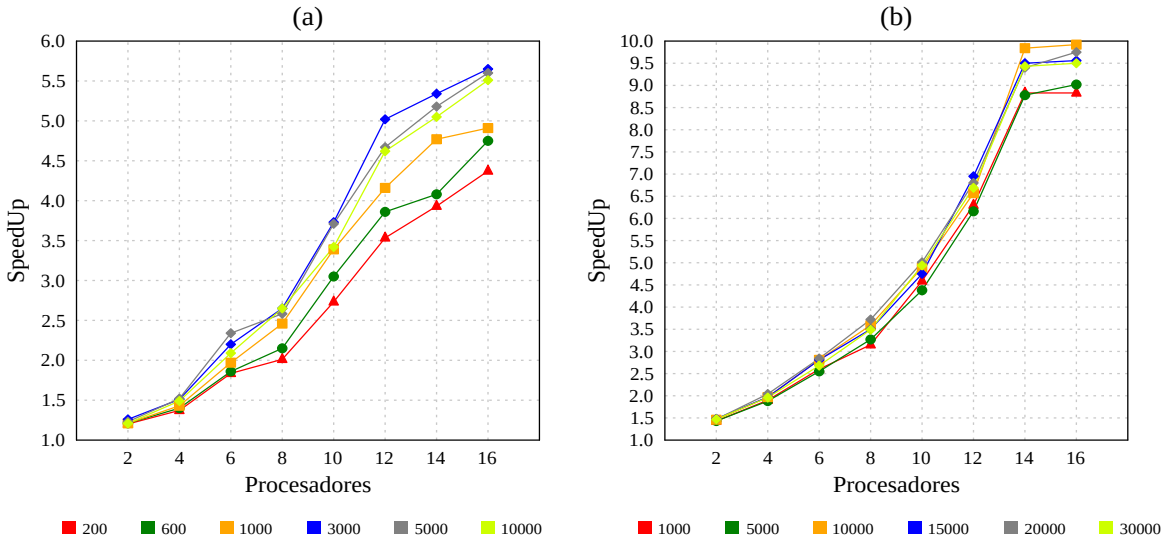

Figura 3.13: Comportamiento de la ganancia de velocidad del algoritmo para el cálculo del sombreado en entornos distribuidos en el experimento 1 (a) 19 685 x 8 972 (b) 86 404 x 28 804.

Se puede observar que en la mayoría de los casos la mayor ganancia de velocidad se alcanza

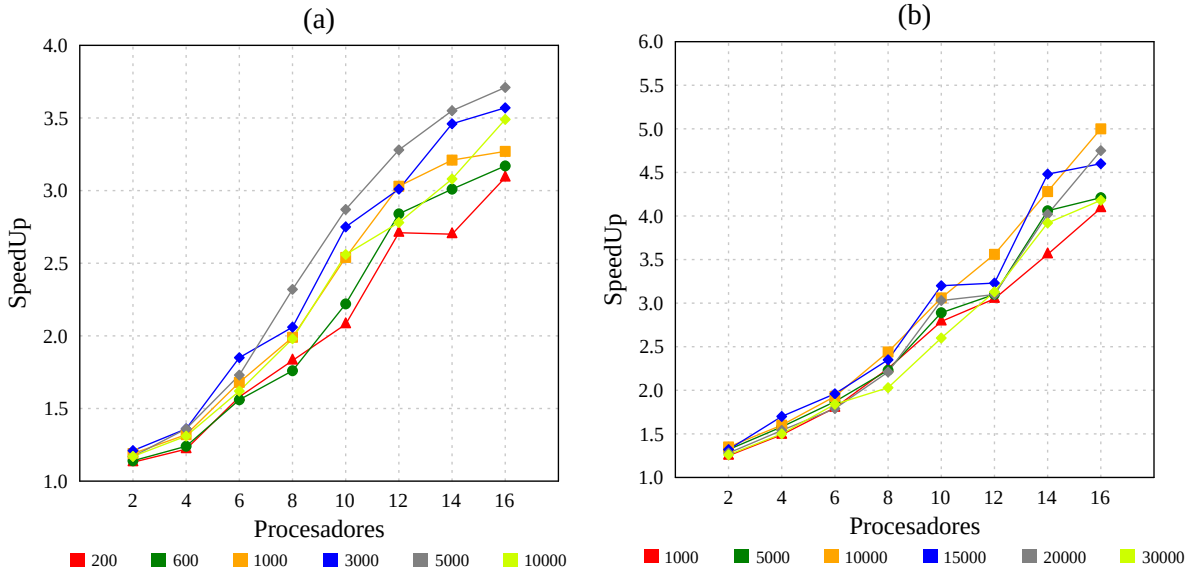

Figura 3.14: Comportamiento de la ganancia de velocidad del algoritmo para el cálculo del sombreado en entornos distribuidos en el experimento 2 (a) 19 685 x 8 972 (b) 86 404 x 28 804.

cuando se hace uso del número máximo de procesadores, sin embargo, para otros casos esto no significa que si se aumentan los clientes se seguirá ganando en tiempo, pues como muestra la gráfica (b) de la Figura 3.13, este valor comienza a comportarse de forma lineal a partir del uso de 14 clientes.

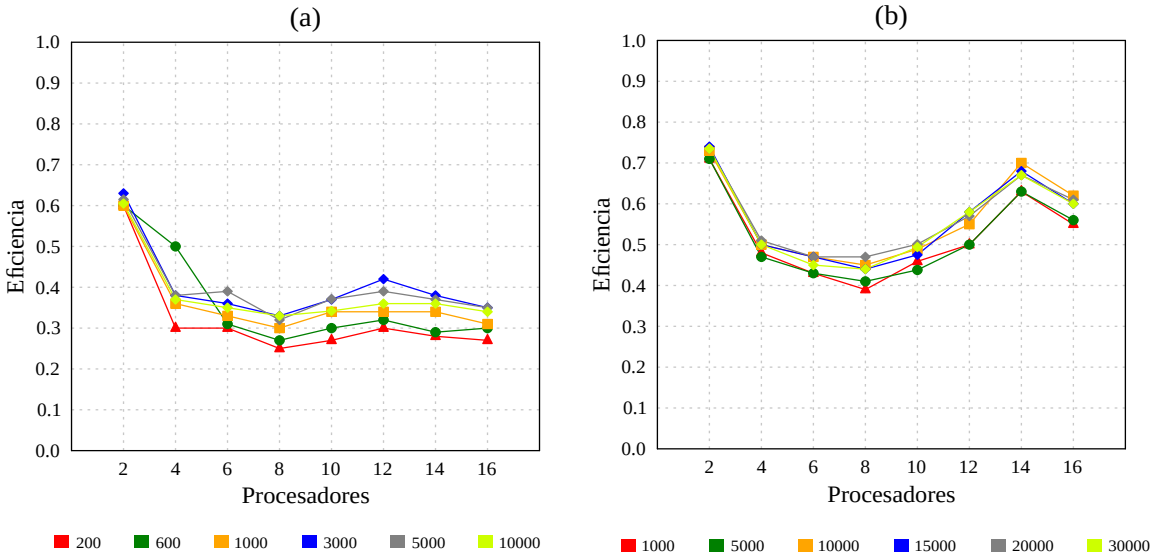

Figura 3.15: Comportamiento de la eficiencia del algoritmo para el cálculo del sombreado en entornos distribuidos en el experimento 1 (a) 19 685 x 8 972 (b) 86 404 x 28 804.

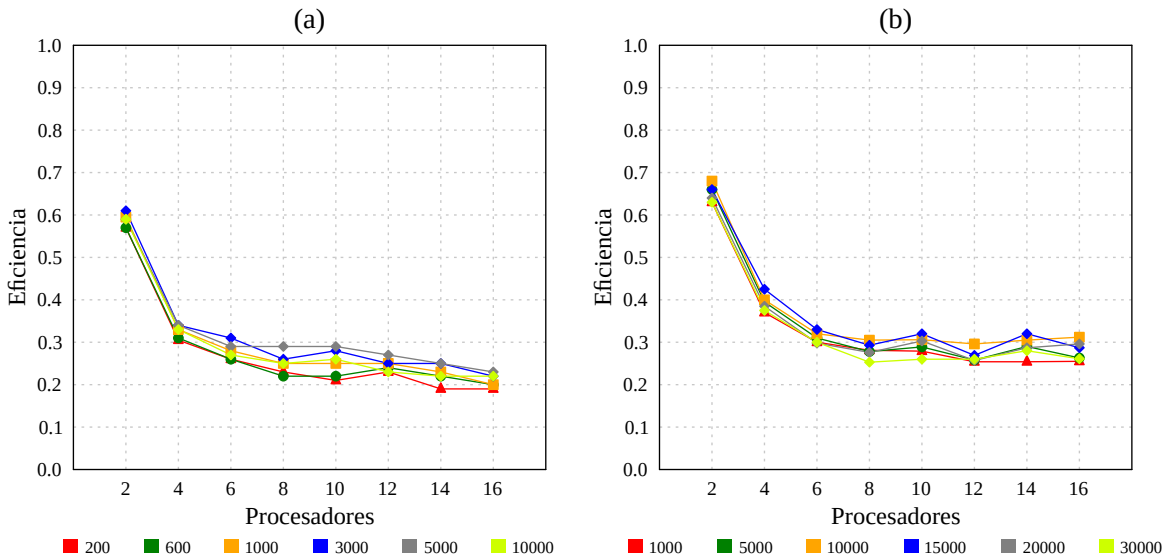

Figura 3.16: Comportamiento de la eficiencia del algoritmo para el cálculo del sombreado en entornos distribuidos en el experimento 2 (a) 19 685 x 8 972 (b) 86 404 x 28 804.

3.3.3. Experimentación en un entorno híbrido

La implementación híbrida de los algoritmos se realizó utilizando en un nivel global la distribución mediante T-Arenal y en un nivel local se utilizó OpenCL u OpenMP, en correspondencia con las características de las arquitecturas paralelas de cada cliente. Las pruebas experimentales se realizaron utilizando las mismas configuraciones de hardware descritas en la experimentación para entornos distribuidos (epígrafe 3.3.2) teniendo en cuenta que todos los clientes configurados cuentan con un procesador multinúcleo.

Las gráficas de las Figura 3.17 y 3.18 muestran el comportamiento del tiempo, la ganancia de velocidad y la eficiencia respectivamente. Para cada *dataset* se utilizó el valor de granularidad que mejores resultados reportó en el experimento anterior, $G = 3000$ para el *dataset* más pequeño y $G = 10000$ para el *dataset* de mayor tamaño. Esto no significa que en el entorno híbrido estos valores reportarán mejores resultados al igual que en el distribuido, para ello es recomendable utilizar algoritmos de planificación que permitan explorar valores de granularidad convenientes según el tipo de arquitectura utilizada.

Como se puede observar en las gráficas de la Figura 3.18, con el algoritmo híbrido se alcanzan mayores valores para la ganancia de velocidad y la eficiencia (con valores más cercanos al óptimo teórico) que con el algoritmo distribuido. Precisamente esto se debe a que además de realizar la distribución de los datos, a su vez, los clientes están explotando el paralelismo a un nivel más fino.

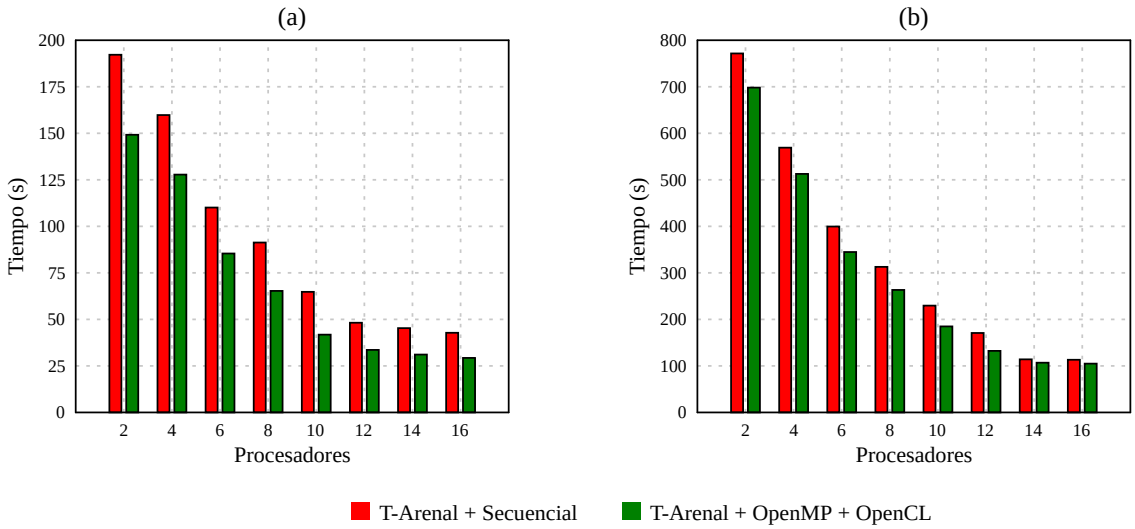

Figura 3.17: Comparación de los tiempos del algoritmo para el cálculo del sombreado entre el entorno distribuido T-Arenal y el entorno híbrido T-Arenal + OpenMP + OpenCL (a) 19 685 x 8 972 con $G = 3000$ (b) 86 404 x 28 804 con $G = 10000$.

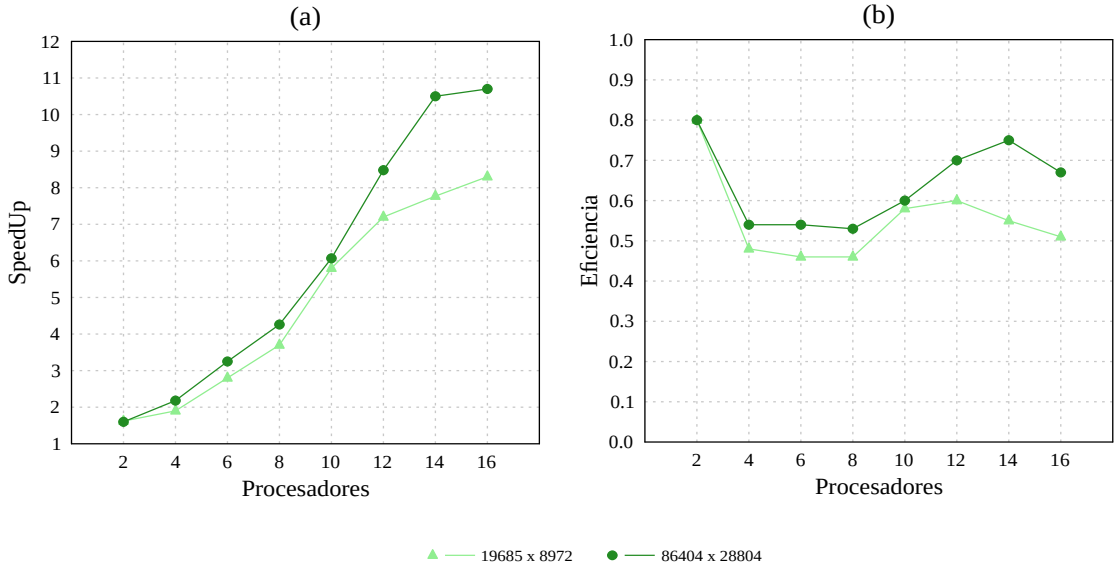

Figura 3.18: Comportamiento de la ganancia de velocidad y la eficiencia del algoritmo híbrido acorde a los resultados mostrados en las gráficas de la Figura 3.17 (a) Ganancia de velocidad (b) Eficiencia

3.4. Conclusiones parciales

Las pruebas realizadas a los algoritmos paralelos propuestos para el cálculo de los parámetros del terreno muestran de manera general resultados favorables. Como consideraciones finales del

presente capítulo se relacionan las siguientes:

- A pesar de que el uso de OpenCL tanto en CPU como en GPU brinda mayor portabilidad en la implementación, resultó más conveniente utilizar OpenMP en la CPU debido a la complejidad de las optimizaciones de OpenCL para CPU.
- El desempeño de los algoritmos en la GPU muestra el nivel de paralelismo de datos que implementan las mismas y que puede ser bien aprovechado, logrando el procesamiento simultáneo de n filas, lo que normalmente en el algoritmo secuencial se ejecuta en n iteraciones.
- La puesta en práctica del algoritmo distribuido e híbrido mediante T-Arenal, muestra que con el aprovechamiento de los recursos computacionales no dedicados y disponibles en las infraestructuras, es posible obtener buenos resultados en cuanto a ganancia de velocidad en el procesamiento de *datasets* con altos niveles de resolución.
- Aunque los resultados reportados por los algoritmos para entornos distribuidos e híbridos son favorables, el uso de estos enfoques no siempre resulta ventajoso, debido a que el tiempo de comunicación y la latencia de la red degradan el tiempo total de procesamiento. Por lo tanto, cuando se trata de modelos que por su tamaño pueden ser cargados en la memoria del sistema, resulta más eficiente realizar el procesamiento en arquitecturas multinúcleos o en GPU.

CONCLUSIONES GENERALES

- El tiempo empleado por los algoritmos secuenciales para el análisis del terreno a través de la extracción de sus atributos, en ocasiones, y teniendo en cuenta el nivel de resolución de los datos, no satisface las necesidades de los usuarios de SIG en entornos computacionales no dedicados. Por otro lado, los algoritmos paralelos propuestos en la bibliografía requieren contar con una infraestructura dedicada a este fin.

- Los algoritmos propuestos en la presente investigación permiten realizar el procesamiento en arquitecturas paralelas de tipo heterogéneas y no dedicadas, demostrando la factibilidad de uso de la potencia de cálculo de las GPU y de las estaciones interconectadas en instituciones a través de una red local sin recursos destinados a la realización de esta tarea en específico.

- Los resultados reportados en las pruebas realizadas demostraron la factibilidad de uso de las arquitecturas multinúcleos y de las GPU para modelos pequeños con una ganancia de velocidad de hasta $9,73$ y de los sistemas distribuidos para modelos con altos niveles de resolución que por su tamaño no pueden ser almacenados en la memoria del sistema.

- Como aporte práctico se realizó la implementación de una herramienta de análisis del terreno para la biblioteca GDAL que complementa la existente y permite realizar el procesamiento de forma más eficiente aprovechando los recursos computacionales disponibles en el entorno.

- Aunque se realizó el diseño y la implementación de algoritmos para el cálculo de tres parámetros esenciales identificados en GDAL: la pendiente, la rugosidad y el sombreado del terreno, la propuesta es aplicable a otros algoritmos de la familia de funciones de análisis focal.

RECOMENDACIONES

La propuesta de algoritmos realizada en la presente investigación aborda la utilización de arquitecturas de cómputo paralelo de tipo heterogéneas y no dedicadas en el análisis del terreno mediante la extracción de sus parámetros esenciales. A partir de la evaluación de los algoritmos y de los resultados alcanzados, se recomienda:

- Implementar un mecanismo de procesamiento por bloques para evitar las limitaciones de memoria en las arquitecturas multinúcleos.
- Optimizar la implementación de OpenCL para garantizar mayor rendimiento en el uso de la CPU como plataforma de cómputo, para de esta forma lograr mayor portabilidad en la implementación mediante el empleo de una sola tecnología en la paralelización.
- Investigar sobre el uso de algoritmos de planificación en entornos heterogéneos, que permitan explorar valores de granularidad diferentes y convenientes según el tipo de arquitectura utilizada.
- Teniendo en cuenta la influencia del tiempo de las operaciones de lectura/escritura de GDAL en la ganancia de velocidad de los algoritmos, se considera factible incorporar un proveedor propio para la lectura/escritura de los datos, externo a las operaciones implementadas por la API de GDAL.

REFERENCIAS BIBLIOGRÁFICAS

[Anderson y Fedak, 2006] Anderson, D. P. y Fedak, G. (2006). The computational and storage potential of volunteer computing. En *Cluster Computing and the Grid. CCGRID 06. Sixth IEEE International Symposium on*, volumen 1, pp. 73–80. IEEE.

[Armstrong y Densham, 1992] Armstrong, M. P. y Densham, P. J. (1992). Domain decomposition for parallel processing of spatial problems. *Computers, Environment and Urban Systems*, 16:497–513.

[Beberg y col., 2009] Beberg, A. L., Ensign, D. L., Guha, J., Khaliq, S., y Pande, V. S. (2009). Folding@ home: Lessons from eight years of volunteer distributed computing. En *Parallel & Distributed Processing. IPDPS 2009. IEEE International Symposium on*, pp. 1–8. IEEE.

[Biesemans y col., 2000] Biesemans, J., Meirvenne, M., y Gabriels, D. (2000). Extending the rusle with the monte carlo error progagation technique to predict long-term average off-site sediment accumulation. *Journal of Soil and Water Conservation*, 55(1):35–42.

[Calkin y col., 1994] Calkin, R., Hempel, R., Hoppe, H. C., y Wypior, P. (1994). Portable programming with the parmacs message-passing library. *Parallel Computing*, 20:615–632.

[Cameron Hughes, 2003] Cameron Hughes, T. H. (2003). *Parallel and Distributed Programming Using C++*. Addison Wesley.

[Chapman y Huang, 2007] Chapman, B. y Huang, L. (2007). Enhancing openmp and its implementation for programming multicore systems. *Parallel Computing: Architectures, Algorithms and Applications. John von Neumann Institute for Computing, Julich, NIC Series*, 38:3–18.

[Chapman y col., 2008] Chapman, B., Jost, G., y van der Pas, R. (2008). *Using OpenMP. Portable Shared Memory Parallel Programming.* The MIT Press Cambridge. Massachusetts Institute of Technology. London, England.

[Contreras y Martonosi, 2008] Contreras, G. y Martonosi, M. (2008). Characterizing and improving the performance of intel threading building blocks. *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium*, pp. 57 – 66.

[Daga y col., 2011] Daga, M., Aji, A. M., y chun Feng, W. (2011). On the efficacy of a fused cpu+gpu processor (or apu) for parallel computing. Technical report, Dept. of Computer Science Virginia Tech Blacksburg, USA.

[Dagum y Menon, 1998] Dagum, L. y Menon, R. (1998). Openmp: An industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 6:46–55.

[Dartnell, 2000] Dartnell, P. (2000). Applying remote sensing techniques to map seafloor geology/habitat relationships. Tesis de máster, San Francisco State University.

[ESRI, 2012] ESRI (2012). *ArcGIS Resource Center. [En línea]* http://help.arcgis.com/es/arcgisdesktop/10.0/help/index.html.

[Farr y col., 2007] Farr, T. G., Rosem, P. A., Caro, E., Crippen, R., Duren, R., y Hensley, S. (2007). The shuttle radar topography mission, reviews of geophysics. 47.

[Fleming y Hoffer, 1979] Fleming, M. y Hoffer, R. (1979). Machine processing of landsat mss data and lars. Technical report, Technical Report 062879, Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana.

[Fraire y col., 2013] Fraire, J. A., Ferreyra, P., y Marques, C. (2013). Opencl overview, implementation, and performance comparison. *IEEE Latin America Trasactions*, 11(1).

[Gao y col., 2013] Gao, Y., Yu, H., Liu, L., y Guo, X. (2013). General neighborhood analysis model for grid dem on cuda. Technical report, Institute of RS & GIS, Peking University, Beijing, China.

[GDAL Development Team, 2014] GDAL Development Team (2014). *GDAL - Geospatial Data Abstraction Library, Version 1.10*. Open Source Geospatial Foundation.

[Gebali, 2011] Gebali, F. (2011). *Algorithms and Parallel Computing*. John Wiley & Sons, Inc., Hoboken, New Jersey.

[Geist y col., 1994] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., y Sunderam, V. (1994). *PVM: Parallel Virtual Machine: a users' guide and tutorial for networked parallel computing*. MIT Press, Cambridge, MA, USA.

[González, 2010] González, A. G. (2010). Sistema paralelo de apoyo al proceso de exploración del níquel. Tesis de máster, Universidad de las Ciencias Informáticas.

[Gove, 2011] Gove, D. (2011). *Multicore Application Programming. For Windows, Linux, and Oracle® Solaris*. Addison Wesley.

[Grama y col., 2003] Grama, A., Gupta, A., Karypis, G., y Kumar, V. (2003). *introduction to Parallel Computing. Second Edition*. Addison Wesley, second edition edición.

[Guan, 2008] Guan, Q. (2008). *Parallel algorithms for geographic processing*. Tesis doctoral, University of California, Santa Barbara.

[Guan, 2009] Guan, Q. (2009). prpl: an open-source general-purpose parallel raster processing programming library. *Newsletter SIGSPATIAL Special*, 1:57–62.

[Guan y Clarke, 2010] Guan, Q. y Clarke, K. (2010). A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science*, 24:695–722.

[Guan y col., 2013] Guan, Q., Zeng, W., y Liu, S. (2013). prpl 2.0 improving the parallel raster processing library. En *Proceedings of the 12th International Conference on GeoComputation, LIESMARS, Wuhan University, Wuhan, China*.

[Hall y Leahy, 2008] Hall, G. B. y Leahy, M. G. (2008). *Open Source Approaches in Spatial Data Handling*. Springer.

[Herlihy, 2007] Herlihy, M. (2007). *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, capítulo The Multicore Revolution, pp. 1–8. Springer Berlin Heidelberg.

[Horn, 1981] Horn, B. (1981). Hill shading and the reflectance map. *Proceedings of IEEE*, 69(1):14–47.

[Jacas, 2011] Jacas, C. R. G. (2011). *Front-end de la Plataforma de Tareas Distribuidas. Interfaz de Escritorio. T-arenal v2.0*.

[Jacas y col., 2014] Jacas, C. R. G., Mendoza, L. A., Pérez, R. G., Ponce, Y. M., Martínez, L. A., Barigye, S. J., y Avdeenko, T. (2014). Multi-server approach for high-throughput molecular descriptors calculation based on multi-linear algebraic maps. *Molecular Informatics*, 34:60–69.

[Jenny, 2001] Jenny, B. (2001). An interactive approach to analytical relief shading. *CARTOGRAPHICA*, 38.

[Jiang y col., 2013a] Jiang, L., an Tang, G., Liu, K., y yi Yang, J. (2013a). Extraction of drainage network from grid terrain datasets using parallel computing. *Geomorphometry.org*.

[Jiang y col., 2013b] Jiang, L., Tang, G., Liu, X., Song, X., Yang, J., y Liu, K. (2013b). Parallel contributing area calculation with granularity control on massive grid terrain datasets. *Computers & Geosciences*, 60:70 – 80.

[Jost y col., 2003] Jost, G., Jin, H., an Mey, D., y Hatay, F. F. (2003). Comparing the openmp, mpi, and hybrid programming paradigms on an smp cluster. Technical report, NAS Technical Report NAS-03-019.

[Karniadakis y Kirby, 2002] Karniadakis, G. E. y Kirby, R. M. (2002). *Parallel Scientific Computing in C++ and MPI A seamless approach to parallel algorithms and their implementation*. Cambridge University Press.

[Karypis y Amin, 1994] Karypis, G. y Amin, M. B. (1994). *Parallel programming library (ppl) - user's guide.*

[Komatsu y col., 2010] Komatsu, K., Sato, K., Arai, Y., Koyama, K., Takizawa, H., y Kobayashi, H. (2010). Evaluating performance and portability of opencl programs. Technical report, Cyberscience Center, Tohoku University Sendai Miyagi 980-8578, Japan.

[Kwesi y Amoako-Yirenkyi, 2008] Kwesi, A. G. N. y Amoako-Yirenkyi, P. (2008). Volunteer computing: Application for african scientist. *ICT AFRICA.*

[Lastovetsky, 2008] Lastovetsky, A. (2008). *Parallel Computing on Heterogeneous Networks.* John Wiley & Sons.

[Lewis y Berg, 1997] Lewis, B. y Berg, D. J. (1997). *Multithreaded Programming With PThreads.* Prentice Hall PTR.

[Llopis, 2008] Llopis, J. P. (2008). Sistemas de información geográfica aplicados a la gestión del territorio. entrada, manejo, análisis y salida de datos espaciales. teoría y práctica general para esri arcgis 9. universidad de alicante : Editorial club universitario.

[Lv y col., 2012] Lv, M. H., Wei, X., y Lei, C. (2012). A gpu-based parallel processing method for slope analysis in geographical computation. *Advanced Materials Research*, 538:625–631.

[Martínez, 2011] Martínez, Y. V. (2011). *Modelo de Representación de Superficies de Terreno para su Visualización en Tres Dimensiones.* Tesis doctoral, Universidad de las Ciencias Informáticas, La Habana, Cuba.

[Max, 2005] Max, N. (2005). Progress in scientific visualization. *The Visual Computer 21*, p. 979–984.

[Munshi y col., 2012] Munshi, A., Gaster, B. R., Mattson, T. G., Fung, J., y Ginsburg, D. (2012). *OpenCL Programming Guide.* Addison Wesley.

[Nvidia, 2011] Nvidia (2011). Nvidia cuda c programming guide version 4.0. [En línea] http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA C Programming Guide.pdf.

[Osterman, 2012] Osterman, A. (2012). Implementation of the r.cuda.los module in the open source grass gis by using parallel computation on the nvidia cuda graphic cards. *ELEKTROTEHNISKI VESTNIK. ENGLISH EDITION*, 79:19–24.

[Ouyang y col., 2013] Ouyang, L., Huang, J., Wu, X., y Yu, B. (2013). Parallel access optimization technique for geographic raster data. *Geo-Informatics in Resource Management and Sustainable Ecosystem. Communications in Computer and Information Science. Springer Berlin Heidelberg*, 398:533–542.

[Petersen y Arbenz, 2004] Petersen, W. P. y Arbenz, P. (2004). *Introduction to parallel computing. A practical guide with examples in C.* Oxford University Press Inc., New York.

[Qin y col., 2014a] Qin, C., Zhan, L. J., y Zhu, A. X. (2014a). How to apply the geospatial data abstraction library (gdal) properly to parallel geospatial raster i/o? *Transactions in GIS*.

[Qin y Zhan, 2012] Qin, C.-Z. y Zhan, L. (2012). Parallelizing flow-accumulation calculations on graphics processing units-from iterative dem preprocessing algorithm to recursive multiple-flow-direction algorithm. *Computers & Geosciences*, 43:2012.

[Qin y col., 2014b] Qin, C.-Z., Zhan, L.-J., Zhu, A.-X., y Zhou, C.-H. (2014b). A strategy for raster-based geocomputation under different parallel computing platforms. *International Journal of Geographical Information Science*, pp. 37–41.

[Rodríguez y Suárez, 2010] Rodríguez, J. L. G. y Suárez, M. C. G. (2010). Comparison of mathematical algorithms for determining the slope angle in gis enviroment. *Aqua-LAC*, 2.

[Romero, 2006] Romero, F. S. (2006). La teledetección satelital y los sistemas de protección ambiental. *AquaTIC. Revista científica de la Sociedad Española de Acuicultura*, 24:13–41.

[Seitavuopio y col., 2005] Seitavuopio, P., Rantanen, J., y Yliruusi, J. (2005). Use of roughness maps in visualisation of surfaces. *European Journal of Pharmaceutics and Biopharmaceutics*, 59(2):351–158.

[Serrano y col., 1998] Serrano, F. S., López, R. T., y de la Viña, J. B. (1998). Análisis de la variabilidad del relieve a partir de modelos digitales del terreno. *Rev. Sol. Geol. España*, 11(1-2):139 – 149.

[Shary y col., 2005] Shary, P., Sharaya, L., y Mitusov, A. (2005). The problem of scale-specific and scale-free approaches in geomorphometry. *Geografia Fisica e Dinamica Quaternaria*, 28(1):81–101.

[Smith y col., 2013] Smith, M. J., Goodchild, M. F., y Longley, P. A. (2013). *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools.* Troubador Publishing.

[Steinbach y Hemmerling, 2012] Steinbach, M. y Hemmerling, R. (2012). Accelerating batch processing of spatial raster analysis using gpu. *Computers & Geosciences*, 45:212–220.

[Tesfa y col., 2011] Tesfa, T. K., Tarboton, D. G., DanielW.Watson, Schreuders, K. A., Baker, M. E., y Wallace, R. M. (2011). Extraction of hydrological proximity measures from dems using parallel processing. *Environmental Modelling & Software*, 26:1696–1709.

[Tyler, 2005] Tyler, M. (2005). *Web mapping illustrated: using open source GIS toolkits*. O'Reilly Media, Inc.

[Wagner y Scott, 1995] Wagner, D. F. y Scott, M. S. (1995). Improving the performance of raster gis: A comparison of approaches to parallelization of cost volume algorithms. *AUTOCARTO-CONFERENCE*, pp. 164–176.

[Wallis y col., 2009] Wallis, C., Watson, D., Tarboton, D., y Wallace, R. (2009). Parallel flow-direction and contributing area calculation for hydrology analysis in digital elevation models. *PDPTA'09, The 2009 International Conference on Parallel and Distributed Processing Techniques and Applications. Las Vegas, Nevada, USA*.

[Wang y col., 2012] Wang, X., Li, Z., y Gao, S. (2012). Parallel remote sensing image processing: Taking image classification as an example. *Computational Intelligence and Intelligent Systems. Communications in Computer and Information Science. Springer Berlin Heidelberg*, pp. 159–169.

[Warmerdam, 2008] Warmerdam, F. (2008). The geospatial data abstraction library. *Brent H and Michael LG. Open Source Approaches in Spatial Data Handling. Springer, Berlin*, pp. 87–104.

[Wilson y col., 2007] Wilson, M. F. J., O'Connell, B., C.Brown, Guinan, J. C., y Grehan, A. J. (2007). Multiscale terrain analysis of multibeam bathymetry data for habitat mapping on the continental slope marine geodesy. *Marine Geodesy. Taylor & Francis Group*, 30:3 – 35.

[Xia y col., 2011] Xia, Y., Kuang, L., y mei Li, X. (2011). Accelerating geospatial analysis on gpus using cuda. *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)*, 12:990–999.

[Zevenbergen y Thorne, 1987] Zevenbergen, L. y Thorne, C. (1987). Quantitative analysis of land surface topography, earth surface processes and landforms. 12:47 – 56.

[Zhan y Qin, 2012] Zhan, L.-J. y Qin, C.-Z. (2012). Parallel geospatial raster processing by geospatial data abstraction library (gdal) — applicability and defects. *Transactions in GIS*.

[Zhao y col., 2010] Zhao, Y., Huang, Z., Chen, B., y Fang, Y. (2010). Local acceleration in distributed geographic information processing with cuda. *Geoinformatics. 18th International Conference. IEEE Xplore Digital Library*.

[Zhou y Liu, 2004] Zhou, Q. y Liu, X. (2004). Error analysis on grid-based slope and aspect algorithms. *Photogrammetric Engineering & Remote Sensing*, 70(8):957–962.

ACRÓNIMOS

API Interfaz de Programación de Aplicaciones (del término en inglés API: *Application Program Interface*)

CPU Unidad Central de Procesamiento (del término en inglés CPU: *Central Processing Unit*)

GDAL Biblioteca de Abstracción de Datos Geoespaciales (del término en inglés GDAL: *Geospatial Data Abstraction Library*)

GEYSED Geoinformática y Señales Digitales

GPU Unidades de Procesamiento Gráfico (del término en inglés GPU: *Graphics Processing Unit*)

GPGPU *General Purpose GPU*

MDE Modelo Digital de Elevación

MPI Interfaz de Paso de Mensajes (del término en inglés MPI: *Message Passing Interface*)

OpenCL *Open Computing Language*

OpenMP *Open Multi - Processing*

PC Computadora Personal (del término en inglés PC: *Personal Computer*)

PVM Máquina Virtual Paralela (del término en inglés PVM: *Parallel Virtual Machine*)

SIG Sistema de Información Geográfica

SRTM *Shuttle Radar Topography Mission*

UCI Universidad de las Ciencias Informáticas