UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Escuela de Ingeniería Informática

Análisis de rendimiento de un clúster HPC y, arquitecturas manycore y multicore.

Trabajo de Titulación previo a la obtención

del Título de Ingeniero en Sistemas

AUTOR:

Brayme Lino Guamán Rivera

CI: 0104965991

DIRECTOR:

Ing. Lizandro Damián Solano Quinde, PhD

CI: 0102428893

Cuenca - Ecuador

2017

RESUMEN

Actualmente la tendencia para obtener una gran cantidad de cómputo, es mediante la computación paralela, un claro ejemplo radica en el hecho que los computadores más rápidos del mundo son clústeres, formados por aceleradores y procesadores. Los clústeres HPC ofrecen una capacidad computacional para solventar requerimientos de alto costo computacional en áreas como: predicción climática y aprendizaje automático. Sin embargo para tener un rendimiento óptimo en aplicaciones que hacen uso del paralelismo, es necesario analizar requerimientos como: carga de comunicaciones, plataformas de paralelización, entre otras. El objetivo de este estudio es analizar la escalabilidad del modelo de predicción climática WRF en clústeres HPC en base a las comunicaciones y procesos MPI; y el rendimiento del algoritmo Horizontal Diffusion en aceleradores XeonPhi y TeslaKepler, usando OpenCL y CUDA. Los resultados muestran una dependencia de la escalabilidad del WRF con las comunicaciones, debido a que, se obtuvo una aceleración máxima para InfiniBand FDR de 25.9, QDR 21.42 y Ethernet 6.4 veces más con respecto a una ejecución secuencial. El acelerador TeslaK40m muestra un rendimiento 6 veces mayor que XeonPhi, debido a que el algoritmo no utiliza la vectorización eficientemente. OpenCL tiene una curva de aprendizaje mayor que CUDA debido a sus propiedades multiplataforma, en cuanto a rendimiento CUDA es un 6% mejor, debido a que está orientado a GPUs NVIDIA, y posee configuraciones que mejoran el desempeño, por otro lado OpenCL no es afectado en gran medida por ser multiplataforma, y es una opción cuando los requerimientos, están orientados hacia la portabilidad.

Palabras clave: HPC, Rendimiento, Escalabilidad, OpenCL, CUDA.

ABSTRACT

Currently the tendency to obtain a large amount of computing, is through parallel computing, a clear example lies in the fact that the fastest computers in the world are clusters for high performance computing, formed by accelerators and multiprocessors. HPC clusters offer an important computational capacity to solve high computational cost requirements, in areas such as: weather prediction, machine learning. However, to achieve optimal performance is necessary to analyze their requirements, such as: communications, parallelization platforms. The objective of this study is to analyze the scalability of the climate prediction model WRF in HPC clusters based on inter-node communication and MPI processes; and the performance of the Horizontal Diffusion algorithm on XeonPhi and TeslaKepler accelerators, using OpenCL and CUDA. The results show a dependence of the scalability of the WRF with communications, since; a maximum acceleration was obtained for InfiniBand FDR of 25.9, QDR 21.42 and Ethernet 6.4 times more than sequential execution. The accelerator Tesla K40m shows a performance 6 times greater than XeonPhi, since the algorithm does not efficiently use vectorization Intel property, in addition Intel OpenCL drivers for its architecture manycore, they are deprecated. OpenCL has a higher learning curve than CUDA due to its multiplatform properties, in terms of performance CUDA shows a 6% improvement since it is oriented to NVIDIA GPUs, and has configurations that improve the performance, on the other hand OpenCL is not very affected by its multiplatform property, and is an option to consider if the requirements are oriented towards portability.

Keywords: HPC, Performance, Scalability, OpenCL, CUDA.

.

Contenido

Índice de Tablas

Índice de Figuras

Universidad de Cuenca
Cláusula de Licencia y Autorización para Publicación en el Repositorio Institucional

Brayme Lino Guamán Rivera en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Análisis de rendimiento de un clúster HPC y, arquitecturas manycore y multicore", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 25 de octubre de 2017

Brayme Guamán

Brayme Lino Guamán Rivera

C.I: 0104965991

Universidad de Cuenca
Cláusula de Propiedad Intelectual

Brayme Lino Guamán Rivera, autor del trabajo de titulación "Análisis de rendimiento de un clúster HPC y, arquitecturas manycore y multicore.", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 25 de octubre de 2017

Brayme Lino Guamán Rivera

C.I: 0104965991

Agradecimientos

Quiero expresar un agradecimiento especial al Doctor Lizandro Solano, quien me ha ayudado de una manera desinteresada con sus conocimientos y su tiempo en la realización del presente trabajo de titulación, a pesar de sus ocupaciones.

A la RED CEDIA, por la prestación del clúster de alto rendimiento, para realizar las diferentes evaluaciones y análisis.

Dedicatoria

Este trabajo, no es únicamente un esfuerzo propio, sino de otras personas que me han apoyado emocionalmente a lo largo de toda mi carrera universitaria:

A mi padre Daniel, que me ha apoyado de manera incondicional en todo, sin importar el resultado. A pesar de tu trabajo, siempre estuviste y fuiste importante en los momentos más especiales de mi vida. Te quiero, te admiro y te respeto.

A mi madre Rosa, por ser una mujer que ha dado todo por sus hijos, sin esperar nada a cambio. Que si existe un amor y cariño puro, tú eres el mayor ejemplo, que yo he tenido.

A una persona especial Erika, quien me apoyo durante toda mi carrera, y fue una gran amiga, novia y compañera.

A mis hermanos, Jhonny, Carlos y Marlene, que a pesar de todo, siempre nos apoyamos entre nosotros; y espero que nunca nos separemos, ni cambie el cariño que nos tenemos.

A todos mis amigos que hice en esta etapa Universitaria, en especial a Christian (Chulla) y Xavier, siempre es un gusto compartir tiempo con ustedes. Una buena amistad es algo difícil de encontrar. Y de un libro recién leído quiero citar una frase, que me hizo recordar a ustedes dos; *“Que la Amistad es el Puerto sereno a que llega el alma fatigada, en sus días de Infortunio”*, una excursión hacia los indios ranqueles.

Todas estas personas son fundamentales en mi vida, de quienes he recibido un gran apoyo desinteresado, y lleno de cariño.

A Dios y a las circunstancias que han hecho que este trabajo se culmine de la mejor manera posible.

Capítulo 1 : Introducción

1.1 Introducción

Actualmente la Universidad de Cuenca desarrolla varios proyectos de investigación, muchos de estos del ámbito científico que requieren de un alto costo computacional. Los cálculos que se necesitan en estos proyectos deben ser obtenidos en tiempos reducidos, por lo que las arquitecturas de computación tradicionales no son apropiadas. La computación de alto rendimiento o HPC (High Performance Computing) por sus siglas en inglés, es una alternativa para este tipo de problemas.

Dentro de los sistemas de HPC se tienen los denominados clústeres de computadores, los cuales están compuestos por nodos que implementan procesadores o aceleradores de diferentes arquitecturas. Actualmente, existen varias arquitecturas de hardware desarrolladas por diferentes fabricantes. Estas arquitecturas implementan características que las hacen aptas para ciertas aplicaciones, las cuales determinan su rendimiento al momento de ejecutar un problema en específico. Un método para determinar el rendimiento de una arquitectura es a través de la ejecución de aplicaciones que permitan este propósito, y de esta manera determinar fortalezas y debilidades de la arquitectura. Los resultados obtenidos de la aplicación de estos programas a una arquitectura de hardware, permiten tener argumentos para poder seleccionar la arquitectura más apropiada para la ejecución de determinadas aplicaciones.

La Universidad de Cuenca y la red CEDIA disponen de clústeres de alto rendimiento, cuyos nodos de cómputo implementan tres tipos de arquitecturas, GPUs NVIDIA Tesla, Intel XEON Phi y procesadores con múltiples núcleos, conectados a través de una red interna basada en InfiniBand (IB), la cual provee velocidades de transporte de datos altas, con reducida latencia.

A través del presente trabajo se pretende obtener información acerca del rendimiento de los clústeres de la Universidad de Cuenca y CEDIA, de las arquitecturas manycore que los componen, y de los modelos de programación paralelos utilizados en estas arquitecturas. Estos datos serán obtenidos a partir de la ejecución del modelo de predicción climática (Weather Research and Forecasting), el algoritmo de difusión horizontal (Horizontal Diffusion) perteneciente al modelo WRF y los modelos de programación paralela CUDA y OpenCL, siendo analizados para cuantificar el rendimiento y de esta manera evaluar conceptos de escalabilidad y aceleración.

1.2 Trabajo Relacionado

Hasta la actualidad se han realizado varios trabajos de comparación entre las arquitecturas Tesla y Xeon Phi. En los estudios realizados por (Kaczmarek, Schmidt, Steinbrecher, & Wagner, 2014; Martín, 2016) se utilizó un algoritmo de resolución del método del gradiente conjugado, y simuladores neuronales en donde la arquitectura NVIDIA tuvo un mejor desempeño. Por el contrario en comparaciones realizadas por (Teodoro, Kurc, Kong, Cooper, & Saltz, 2014) el algoritmo del gradiente de estadísticas, presenta una mayor aceleración por Xeon Phi. Por lo tanto, cada algoritmo puede obtener un mayor grado de aceleración en diferentes arquitecturas, y no todo el hardware es adecuado para todas las aplicaciones.

En cuanto a los modelos de programación CUDA y OpenCL, se ha obtenido un mejor rendimiento por parte de CUDA (Komatsu et al., 2010; Su, Chen, Lan, Huang, & Wu, 2012), en arquitecturas GPU NVIDIA. Por el contrario, según (Fang, Varbanescu, & Sips, 2011) a partir de análisis exhaustivos para determinar el rendimiento entre estos dos modelos, se determina que los motivos para tener un rendimiento más bajo en OpenCL, son: programadores, usuarios y compiladores. Por lo que es esencial realizar pruebas de rendimiento, con los drivers y compiladores que ofrecen los proveedores; el desarrollo y paralelización de un algoritmo determinado.

Como trabajo relacionado en el área local, la Universidad de Cuenca ha utilizado el modelo WRF (Weather Research and Forecasting), para realizar análisis climático, pero también con el objetivo de obtener información de rendimiento de un clúster formado por procesadores Intel de 6 u 8 núcleos con 96 GB de RAM conectados mediante Ethernet de 1 Gbps; concluyendo que para una mayor escalabilidad es necesario mejorar la velocidad de comunicación de la red que une los nodos del clúster (Gualán & Solano-Quinde, 2014). En este mismo contexto se ha desarrollado una implementación del método de difusión horizontal del WRF para GPU desarrollado en CUDA, teniendo aceleraciones de hasta 19 veces, comparado con la versión secuencial (Gualán-Saavedra, Solano-Quinde, & Bode, 2015; Solano-Quinde, Gualán-Saavedra, & Zuñiga-prieto, 2016).

Sin embargo, en el presente trabajo se proponen realizar evaluaciones de rendimiento del modelo WRF y el algoritmo horizontal diffusion en clústeres HPC pertenecientes a la Universidad de Cuenca y CEDIA, compuestos por redes InfiniBand y aceleradores Xeon Phi, que en estudios pasados no se disponían. En adición, se realiza una medición de rendimiento y análisis de los modelos de programación CUDA y OpenCL, mediante la paralelización del algoritmo Horizontal Diffusion en OpenCL, basándose en la versión paralelizada en CUDA.

Por otro lado, al realizar el análisis sobre el modelo WRF y el algoritmo Horizontal Diffusion, se potencializa el resultado puesto que el pronóstico climático es una de las áreas más relevantes de investigación en el Austro Ecuatoriano.

1.3 Objetivos del trabajo

A continuación, se describen los objetivos de la presente tesis:

1.3.1 Objetivo General

Obtener información sobre el rendimiento de un clúster HPC, para analizar y evaluar conceptos de escalabilidad y aceleración.

1.3.2 Objetivos Específicos

1. Analizar las arquitecturas manycore y multicore disponibles.
2. Investigar sobre los lenguajes de programación paralela, CUDA y OpenCL.
3. Seleccionar e implementar una aplicación que permita evaluar las arquitecturas de hardware disponibles.
4. Evaluar los resultados obtenidos, para determinar los requerimientos de la aplicación.

1.4 Estructura de la tesis

En esta sección se presenta una descripción de los capítulos desarrollados en el presente documento.

El **Capítulo 2**, Análisis de rendimiento del modelo WRF sobre un clúster HPC, presenta un estudio sobre la escalabilidad de rendimiento del modelo de predicción climática (WRF), en dos clústeres de alto rendimiento, utilizando diferentes tecnologías de red interna.

El **Capítulo 3**, Descripción y comparación de rendimiento entre OpenCL y CUDA, detalla las similitudes y diferencias entre los modelos de programación CUDA y OpenCL, para realizar una paralelización del algoritmo de difusión horizontal en OpenCL, y medir el rendimiento de los dos modelos de programación sobre una GPU de NVIDIA.

El **Capítulo 4**, Descripción y comparación de rendimiento entre arquitecturas manycore, detalla las diferencias entre aceleradores manycore Xeon PHI y GPU, para así poder adaptar un algoritmo de difusión horizontal a cada una de estas arquitecturas y realizar una medición de rendimiento entre ellos.

El **Capítulo 5,** Conclusiones, presenta las conclusiones obtenidas durante el desarrollo del presente trabajo de titulación.

Capítulo 2 : Análisis de rendimiento del modelo WRF sobre un clúster HPC

2.1 Introducción

La predicción climática es de gran relevancia, ya que el clima tiene una gran influencia en campos que son pilares de la sociedad actual, tales como la gestión de recursos hídricos, la minimización de riesgo de desastre y la agricultura, entre otros. Los autores Michalakes, Loft, & Bourgeois, (2001); Shainer et al., (2009), sostienen que la predicción climática mediante la simulación numérica de la atmósfera, es una de las primeras aplicaciones en requerir un alto costo computacional. Los modelos numéricos para predicción climática cada vez aumentan su complejidad para mejorar su precisión, la granularidad es más fina para una mayor resolución y las escalas de tiempo son más grandes, lo que demanda una gran capacidad de cómputo. Actualmente la mayoría de pronósticos implican cientos de miles de millones de operaciones, por esto el progreso en las tecnologías de computación ha sido fundamental, lo cual ha permitido obtener predicciones más precisas en menor tiempo. Por estas razones es crucial tener una comprensión de los sistemas HPC (High Performance Computing), y así alcanzar porcentajes razonables de rendimiento del máximo teórico.

WRF es uno de los modelos más importantes y utilizados por un gran número de investigadores en el área de predicción climática. Así mismo, WRF puede ser ejecutado en entornos de memoria distribuida, con lo que se reduce los tiempos de ejecución de manera considerable. Por lo indicado, resulta importante determinar las ventajas y limitaciones que presenta el hardware de sistemas HPC, de manera que en las aplicaciones de software se pueda realizar las optimizaciones necesarias para obtener un rendimiento óptimo. En este contexto, para conocer y analizar el rendimiento de plataformas HPC, es necesario usar aplicaciones que cumplan con este objetivo.

El presente capítulo se estructura de la siguiente forma. Las Secciones 2.2, 2.3 y 2.4 describen brevemente el modelo WRF, componentes de los clústeres y el conjunto de datos, respectivamente. En la Sección 2.5 se proponen diferentes experimentos, con el fin de comparar, tecnologías de comunicación y procesos

MPI por procesador; además se describe el método y las versiones del software usadas.

2.2 WRF (Weather Research and Forecasting)

El modelo WRF permite realizar análisis numérico para predicción del clima y simulación atmosférica, con propósitos tanto de investigación, como de predicción operativa. El modelo fue desarrollado por varias instituciones, entre ellas, la National Center for Atmospheric Research's (NCAR) Mesoscale and Microscale Meteorology (MMM) Division, y la National Oceanic and Atmospheric Administration's (NOAA).

El Software del WRF tiene un alto nivel de organización, brindando así características como: descomposición de dominios para el manejo paralelo y de memoria compartida, separa el código científico del código de paralelización, soporta la ejecución en una amplia gama de plataformas de computación distribuida, vectorial, escalar y aceleradores como GPUs; lo que permite el uso eficiente de la computación paralela masiva y así poder aprovechar los avanzados sistemas HPC (Skamarock et al., 2008).

2.3 Descripción del clúster

En la actualidad, los clústeres se han convertido en la plataforma más usada para dar soporte aplicaciones informáticas que requieren un alto costo computacional. Un clúster es un conjunto de computadoras, llamados nodos computacionales, unidos mediante una red generalmente de alta velocidad, que a través de una plataforma de software especial funcionan como una única computadora; y un nodo front-end que es el encargado de recibir peticiones desde el exterior y gestionar las diferentes tareas entre los nodos. Según ("TOP 500 supercomputer," 2016), las supercomputadoras más rápidas del mundo están diseñadas con arquitecturas clúster, implementadas con un gran número de nodos de cómputo, cada uno con varios procesadores y aceleradores.

La tecnología de interconexión entre los nodos computacionales de un clúster es fundamental para maximizar su eficiencia. La arquitectura InfiniBand es una red estándar diseñada para proveer una tecnología de interconexión de alta velocidad, llegando hasta los 300 Gb/sec en EDR con 12 canales independientes. Su objetivo es proveer una latencia cero escalable, lo que significa que provee una latencia baja, independientemente de cuántos procesos se estén ejecutando en los nodos del clúster (IBTA, 2017), para esto provee acceso remoto directo a memoria (Remote Direct Memory Access, RDMA) que es crucial para alcanzar el mejor rendimiento en aplicaciones de alto rendimiento.

Para este estudio, se disponen de dos clústeres HPC, descritos en la Tabla 1; pertenecientes a la Universidad de Cuenca y al Consorcio Ecuatoriano para el Desarrollo de Internet Avanzado (CEDIA).

	Universidad de Cuenca	CEDIA
Nombre	UCUENCA	CEDIA
RAM por nodo	96 GB	96 GB
RED	InfiniBand 4x FDR 56 Gb/sec	InfiniBand 4x QDR 40 Gb/sec Gigabit Ethernet 1 Gb/sec
Front-end	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon E5620 2.4 GHz de 4 núcleos
Nodos	10	12
Nodo 1	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon X5650 2.67 GHz de 6 núcleos
Nodo 2	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon X5650 2.67 GHz de 6 núcleos
Nodo 3	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon X5650 2.67 GHz de 6 núcleos
Nodo 4	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos, y 2 Xeon Phi 7120p	2 Intel Xeon X5650 2.67 GHz de 6 núcleos
Nodo 5	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos, y 2 Xeon Phi 7120p	2 Intel Xeon X5650 2.67 GHz de 6 núcleos
Nodo 6	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon X5650 2.67 GHz de 6 núcleos
Nodo 7	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon E5-2650 2.00 GHz de 8 núcleos
Nodo 8	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos, y 2 NVIDIA Tesla K40m	2 Intel Xeon E5-2650 2.00 GHz de 8 núcleos
Nodo 9	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos, y 2 NVIDIA Tesla K40m	2 Intel Xeon E5-2650 2.00 GHz de 8 núcleos
Nodo 10	2 Intel Xeon E5-2630 v3 2.4 GHz de 8 núcleos	2 Intel Xeon E5-2650 2.00 GHz de 8 núcleos
Nodo 11	-	2 Intel Xeon E5-2650 2.00 GHz de 8 núcleos

Nodo 12	-	2 Intel Xeon E5-2660 2.20 GHz de 10 núcleos, y 2 NVIDIA Tesla K40m

Tabla 1. Descripción de los clústeres usados en nuestro estudio.

2.4 Conjunto de datos CONUS de 12Km

Continental United States (CONUS) es un conjunto de datos proporcionado por los desarrolladores del WRF y considerado una referencia para realizar análisis de rendimiento. Tiene un dominio con una resolución espacial de 12km, y un paso de tiempo de 72 segundos en un intervalo de 3 horas. El costo computacional para este dominio es de aproximadamente 28,5 mil millones de operaciones de punto flotante por cada 72 segundos (Shainer et al., 2009).

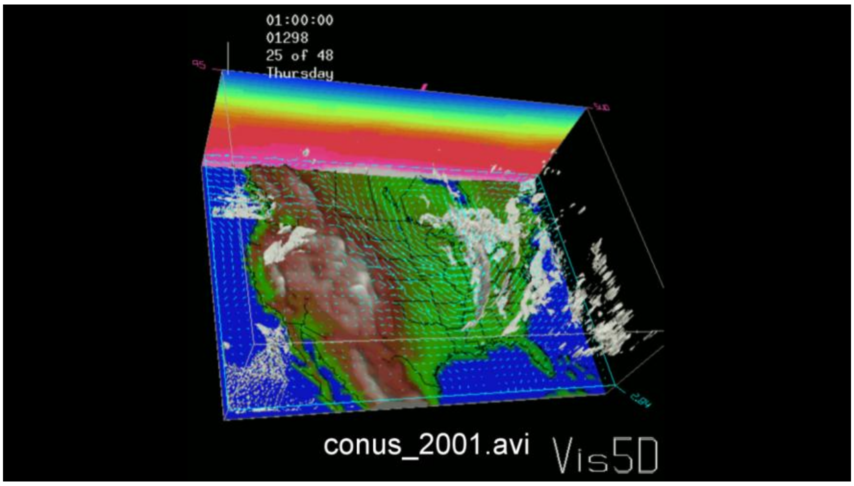

Figura 1. Captura de la animación CONUS 12KM (Eldred & Michalakes, 2008)

2.5 Análisis y escalabilidad del rendimiento

El programa de integración numérica *wrf.exe*, el cual realiza la simulación numérica del clima, es considerado el de mayor exigencia computacional dentro de los componentes del modelo WRF y puede ser ejecutado en paralelo en entornos de memoria distribuida, debido a que incorpora funciones MPI (Message Passing Interface), por esta razón, resulta idóneo para realizar análisis y evaluaciones de rendimiento de hardware de sistemas de alto rendimiento.

Con propósitos de evaluación de rendimiento, en este trabajo, se ejecutó el programa *wrf.exe* de la versión 3.8 del WRF, utilizando el conjunto de datos CONUS 12Km, en un entorno de memoria distribuida. Particularmente es de interés realizar: (i) Comparación de rendimiento usando diferentes tecnologías y velocidades de interconexión, InfiniBand QDR, FDR y Ethernet. (ii) Análisis de la utilización de la red por el WRF.

Para realizar las distintas evaluaciones de rendimiento, se midió el tiempo de ejecución de cada simulación, aumentando secuencialmente la cantidad de nodos computacionales. Para esto, el comando *time* de Linux fue usado (*Wall-clock time*) y está representado en el formato h:mm:ss. *Wall-clock time* es una medida del tiempo real que transcurre de principio a fin de una tarea, incluyendo tiempos como la espera de recursos disponibles o los retardos al disco. *Wall-clock time* es la suma de tiempo de la CPU, tiempo de llamadas al sistema, y el retardo de canal de comunicación en caso de que los datos se distribuyan en varias computadoras. Debido a estas influencias para la obtención de *wall-clock time*, es necesario realizar un promedio del tiempo de varias ejecuciones. Para este estudio se realizaron 12 ejecuciones, con lo que se obtuvo una desviación estándar de 1,293 segundos, un error estándar de 0,34 segundos, y un intervalo de confianza del 95% con 0.83 segundos, utilizando una distribución T de student.

En las siguientes sub secciones se detallan los resultados de las evaluaciones y su respectivo análisis.

2.5.1 Evaluación de rendimiento, sobre InfiniBand y Ethernet

En la presente sección, se evalúa el rendimiento y la escalabilidad del WRF en un entorno de memoria distribuida, mediante la medición del tiempo de ejecución de las simulaciones en diferentes escenarios. Para esto se ha utilizado, Intel MPI y una versión del WRF compilado usando los compiladores de Intel, dos estándares de redes distintas, InfiniBand y Ethernet; y dos velocidades en InfiniBand, QDR y FDR.

Para la selección del número de procesos MPI por nodo (ppn), se debe tener en cuenta el efecto de *oversubscription*, el cual se da cuando el número de procesos MPI es mayor al número de núcleos físicos (The Open MPI Project, 2016). En el caso del WRF, el *oversubscription* produce un deterioro en el rendimiento, que ha sido demostrado en diferentes escenarios (Gualán & Solano-Quinde, 2014). Por lo tanto para evitar este efecto, en este trabajo, se ha utilizado la misma cantidad de núcleos físicos que procesos MPI, dando un total de 12 ppn debido a que el procesador con menor número de núcleos es seis y cada nodo tiene dos procesadores.

La Figura 2 muestra el tiempo de ejecución del WRF usando ocho nodos del clúster, conectados a través de diferentes redes, InfiniBand QDR, FDR y Ethernet. Observando que en el caso de Ethernet no se puede escalar más allá de 24 procesos MPI (2 nodos), lo cual está acorde con lo indicado por (Gualán & Solano-Quinde, 2014). En este contexto es importante notar que, si bien Ethernet muestra una escalabilidad similar a InfiniBand, a partir de 24 procesos MPI, el rendimiento decae hasta en un 243%.

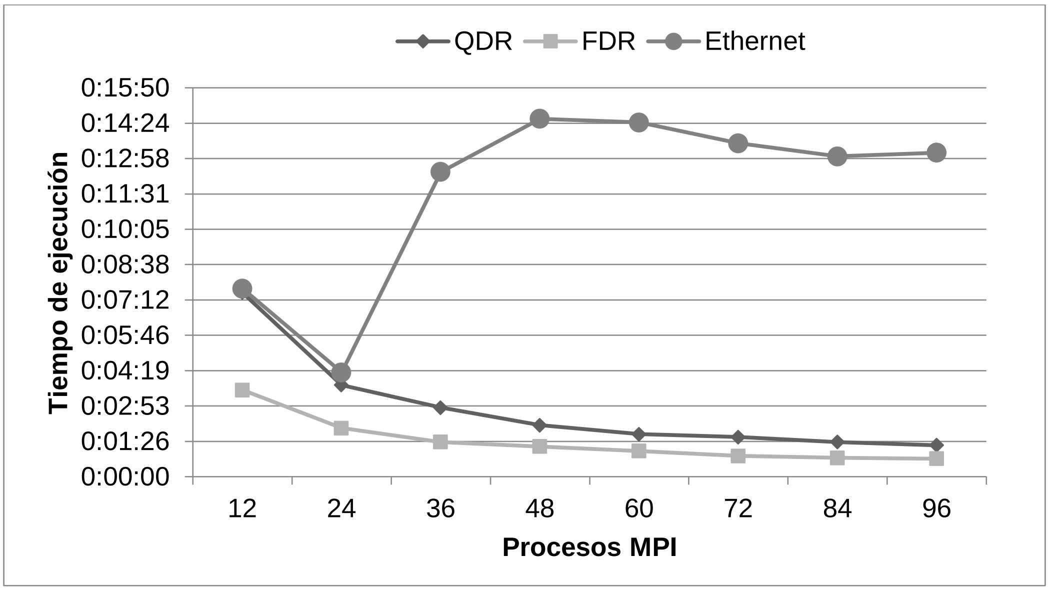

Figura 2. Tiempos de ejecución, sobre InfiniBand QDR, FDR y Ethernet

En contraste con Ethernet, la Figura 2 ilustra que InfiniBand puede escalar con más de dos nodos, así cada vez que se duplica la cantidad de procesos MPI, el tiempo de ejecución disminuye aproximadamente a la mitad.

Para tener una visibilidad clara de la ganancia de rendimiento, al usar redes con mayor velocidad, la Figura 3 ilustra la aceleración obtenida con la aplicación, en clústeres que implementan diferentes tecnologías de red respecto a la ejecución secuencial del WRF. La ejecución seria fue medida en el clúster UCUENCA y CEDIA, obteniendo un tiempo de 19 y 27.5 minutos respectivamente. Las redes InfiniBand, muestran una aceleración incremental, a medida que aumenta el número de procesos MPI. FDR muestra un mayor grado de aceleración que QDR, mientras que Ethernet, empieza a desacelerar a partir de los 24 procesos MPI. Por lo tanto, las mejores capacidades en las tecnologías de red, muestran un claro beneficio en el rendimiento del WRF, ya que las aceleraciones máximas utilizando InfiniBand FDR es de 25.9, QDR 21.42 y Ethernet 6.4 veces más, comparado con la ejecución secuencial.

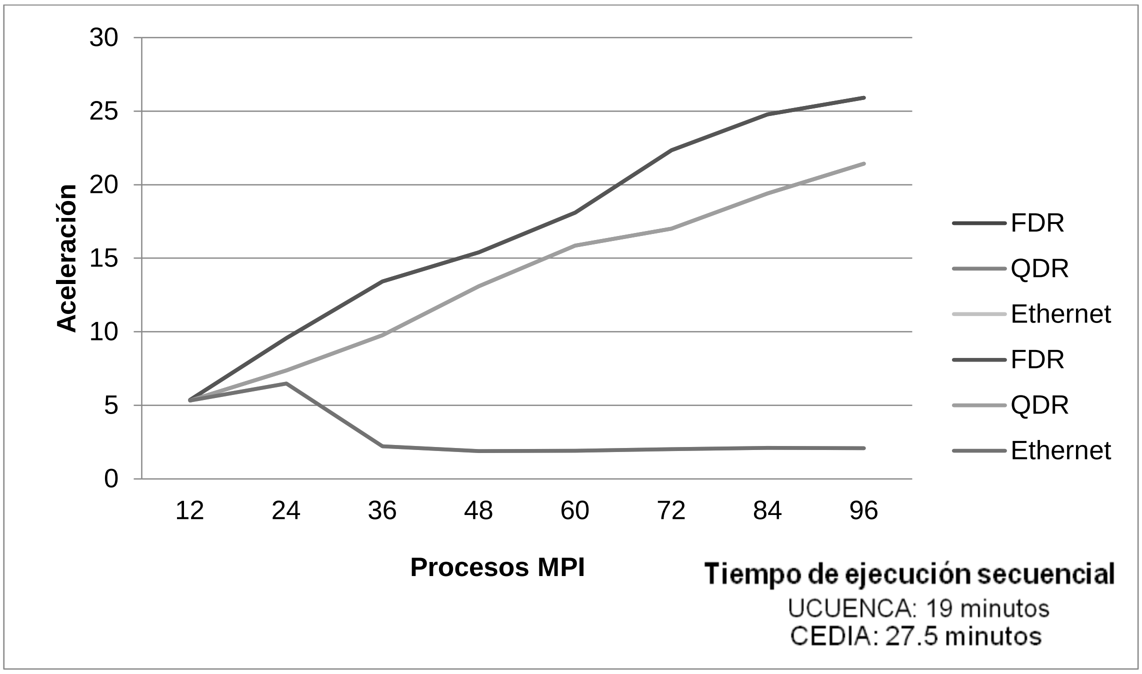

Figura 3. Aceleraciones de InfiniBand QDR, FDR y Ethernet, respecto a la ejecución serial del WRF

Los resultados obtenidos por las redes InfiniBand, muestran que existe una dependencia entre la escalabilidad del WRF y la red de conexión. Los análisis en la Sección 2.5.3, obtienen una mayor cantidad de información para analizar la dependencia entre la tecnología de red y la escalabilidad del WRF.

2.5.2 Evaluación del rendimiento, utilizando la máxima capacidad de cómputo disponible

La presente sección se centra en la evaluación del rendimiento del WRF, usando el máximo número de procesos posible en los nodos. El objetivo de estas ejecuciones, es obtener el mayor rendimiento posible usando toda la capacidad de cómputo disponible.

Por motivos de comparación de efectos de la red entre ambos clústeres y para no producir el efecto de oversubscription en los nodos del clúster de CEDIA que poseen 6 núcleos, en las evaluaciones anteriores solo se uso 12 ppn; a pesar que el clúster UCUENCA soporta hasta 16 ppn sin incurrir en oversubscription. En este caso, en el clúster de la Universidad de Cuenca se utilizan 16 procesos MPI por nodo.

El tiempo de ejecución en el clúster UCUENCA, utilizando 8 nodos con 16 procesos MPI por nodo, sobre la red InfiniBand FDR dio un menor tiempo, comparado con los 12 procesos usados en el análisis anterior. La Figura 4, muestra la aceleración obtenida respecto a una ejecución secuencial (19 minutos), con 12 y 16 procesos MPI por nodo, en donde la aceleración máxima es de 25 y 30 veces más respectivamente.

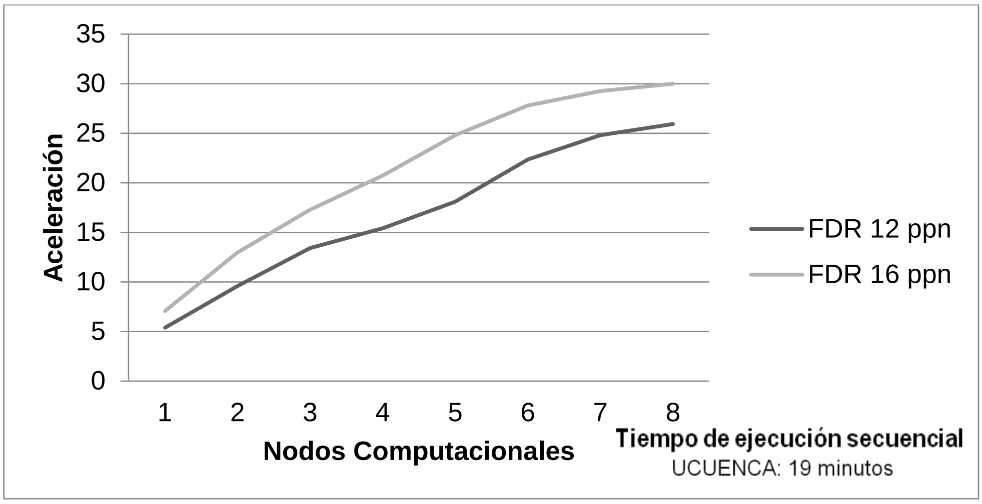

Figura 4. Aceleración de InfiniBand FDR con 16 y 12 procesos MPI por nodo.

2.5.3 Análisis de escalabilidad y uso de red de tecnologías Ethernet, InfiniBand QDR y FDR

Los escenarios anteriores se enfocan en mostrar información sobre la escalabilidad del rendimiento a través del incremento de procesos MPI y, por lo tanto, nodos computacionales. Sin embargo, no se ha analizado el efecto de enviar y recibir datos entre nodos. Este escenario obtiene información sobre los procesos MPI, que sirve para analizar y entender el comportamiento del modelo WRF en un entorno de memoria distribuida.

Cuando hay un entorno de memoria distribuida, el tiempo de ejecución total, se encuentra dividido mayormente en dos grupos: (i) tiempo de ejecución de la aplicación, (ii) tiempo usado por rutinas MPI. El tiempo de aplicación, es la duración de las tareas relacionadas a la aplicación como tal, en este caso *wrf.exe*. Mientras que el tiempo de rutinas MPI, hace referencia al tiempo usado por rutinas que cumplen tareas del paso de mensajes entre procesos, como por ejemplo, llamadas para transferir datos entre dos o más procesos, inicializar o finalizar comunicaciones, etc. La cantidad de datos transferida entre procesos está dividida en inter-nodo e intra-nodo, la inter-nodo son los datos transferidos a través de los nodos computacionales utilizando una red, mientras que los intra-nodo, son los datos transferidos dentro de cada nodo.

Para realizar el análisis propuesto, se ha usado la función *trace* de Intel MPI, el cual realiza una recolección de datos de una simulación. Posteriormente la herramienta ITAC (Intel ® Trace Analyzer and Collector) se usa para la visualización de los datos recolectados. En este contexto, es importante indicar que existe un *overhead* puesto por la función *trace*.

Las Figuras 5 y 6 muestran los resultados del tiempo de aplicación y rutinas MPI respectivamente. El tiempo de aplicación tanto de Ethernet como QDR es igual debido a que ambas están operando en el mismo clúster, en este caso CEDIA. El tiempo de aplicación, de manera general, muestra una mejora aproximadamente exponencial, que es el esperado al realizar procesos de paralelización. Por otro lado, con estos resultados se aclara que el aumento de rendimiento, mostrado en la Sección 2.5.1 de la red FDR no solo es debido a la velocidad de la red, sino a la mayor capacidad computacional de los procesadores.

Por otro lado la disminución de los Tiempos MPI es proporcional a las capacidades de la red, teniendo una escalabilidad a través de los nodos para las redes InfiniBand, por su alta tasa de transmisión de datos, obteniendo un comportamiento deseable, debido al overhead bastante bajo impuesto por las comunicaciones. Mientras que para la red Ethernet a partir de la inclusión de tres nodos, el tiempo de rutinas MPI crece rápidamente, debido a su baja capacidad para la transmisión de datos. Por lo tanto, el comportamiento de ethernet es totalmente indeseable debido a que el overhead impuesto es incluso más que el tiempo de aplicación.

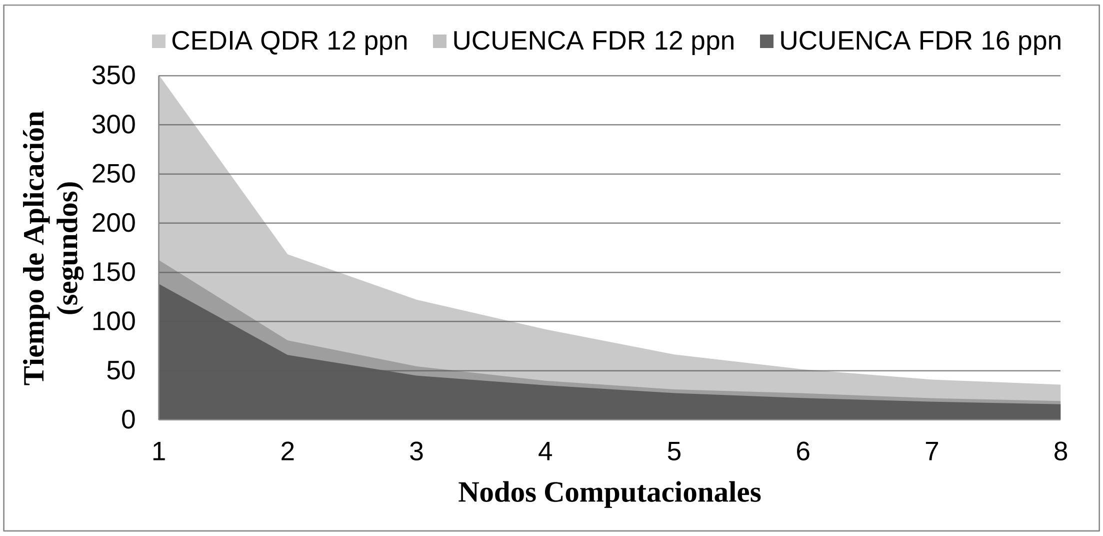

Figura 5. Tiempo de aplicación.

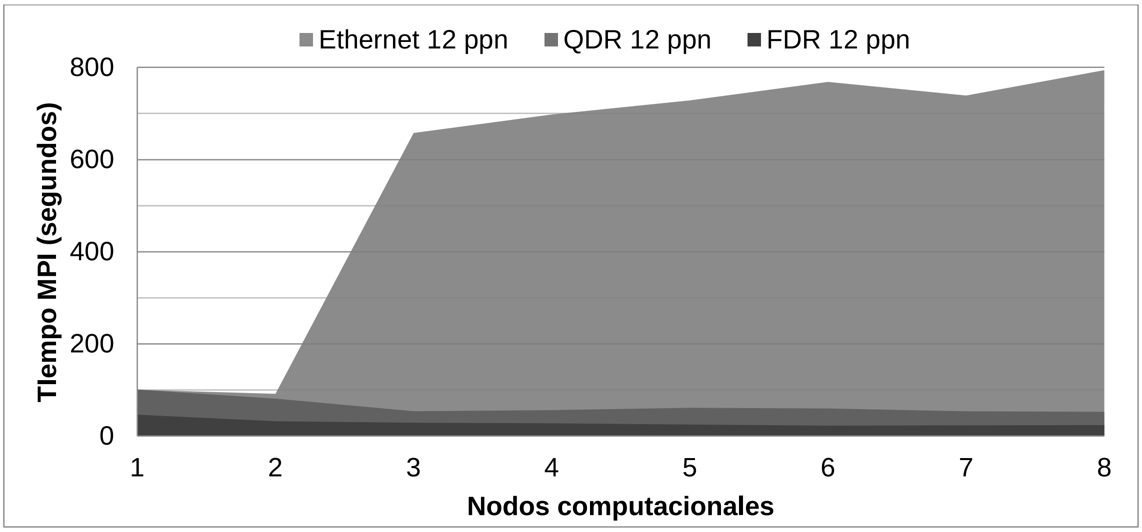

Figura 6. Tiempos de rutinas MPI.

Para reforzar el análisis, en la Figura 7 se presenta el porcentaje del tiempo de rutinas MPI con respecto al tiempo total, en donde el porcentaje de rutinas MPI aumenta según incrementan los procesos MPI, hasta llegar ocupar un 60 % del tiempo de ejecución. Esto se debe a que el incremento de la cantidad de datos transmitidos entre procesos aumenta acorde con la cantidad de nodos como se puede observar en la Figura 8, en donde para 3 nodos el volumen de datos de intercambio es de 16.8 GB siendo una carga bastante exigente para una red Ethernet. Con 8 nodos, la cantidad de datos transmitidos llega hasta 139 GB, donde 61.71 GB son inter-nodo, lo que ayuda a comprender las ganancias significativas mostradas en la Figura 6, al usar redes InfiniBand que soportan de manera eficiente toda la carga de comunicaciones.

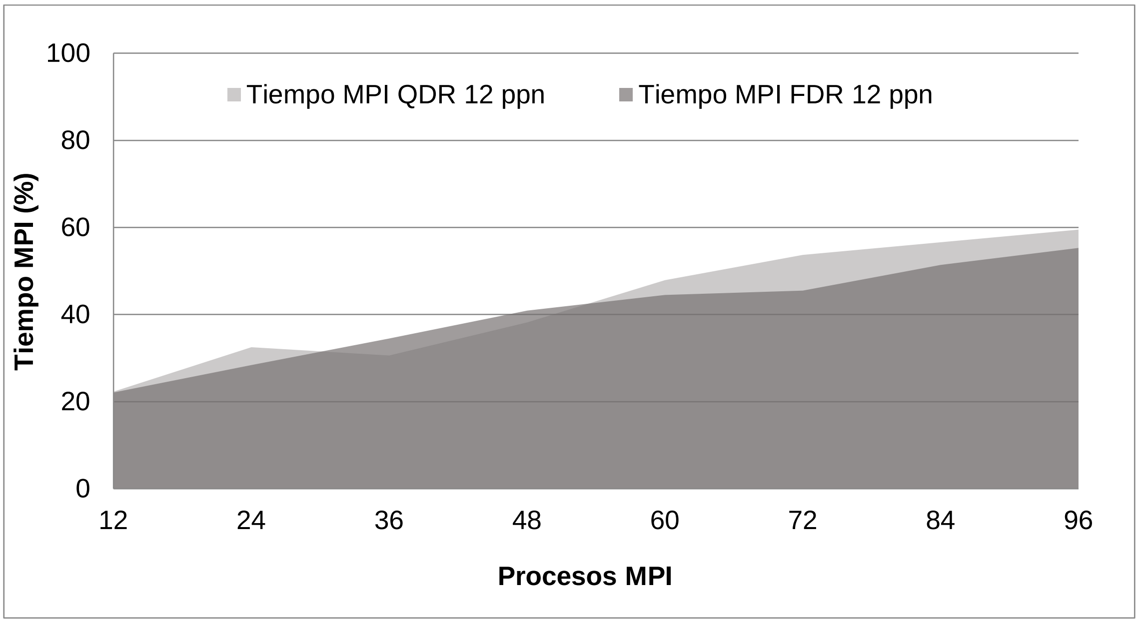

Figura 7. Porcentaje del tiempo MPI del tiempo total.

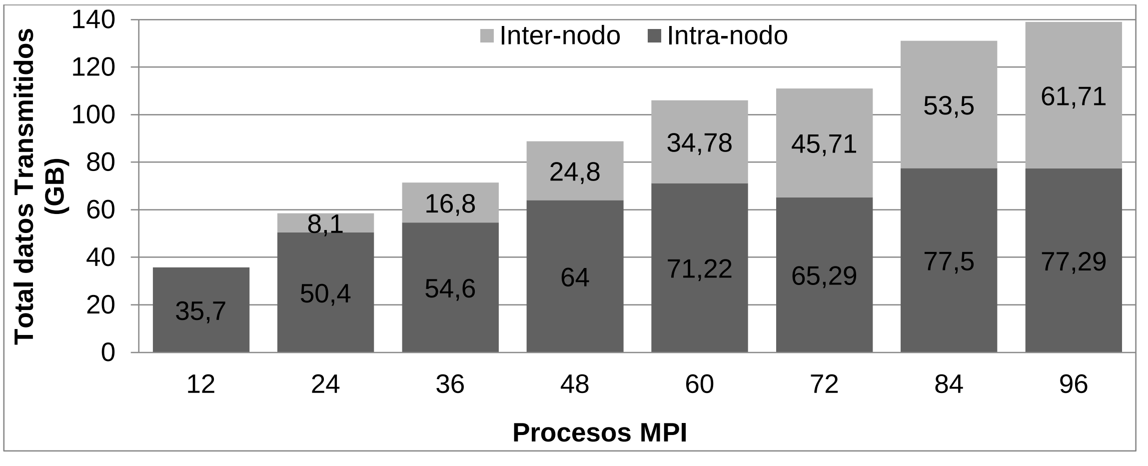

Figura 8. Total de datos transmitidos, con 12 procesos por nodo.

Otro aspecto que ayuda a comprender el aumento de rendimiento al usar redes InfiniBand, son las funciones MPI que mayor tiempo consumen en la ejecución, así podemos observar en la Figura 9 que la función *MPI_Wait* es la que más porcentaje del tiempo MPI total ocupa. Esta es una función que bloquea la ejecución hasta finalizar el envío y recepción de datos entre procesos, lo que implica que puede existir un retardo en la ejecución dependiendo de las capacidades de la red y de la eficiencia de la aplicación. Siendo este el caso de Ethernet, puesto que la cantidad de datos transmitidos entre procesos está entre 1 y 3 GB a partir de dos nodos computacionales. Por el contrario InfiniBand tiene la capacidad de solventar de manera eficiente estas cantidades de datos.

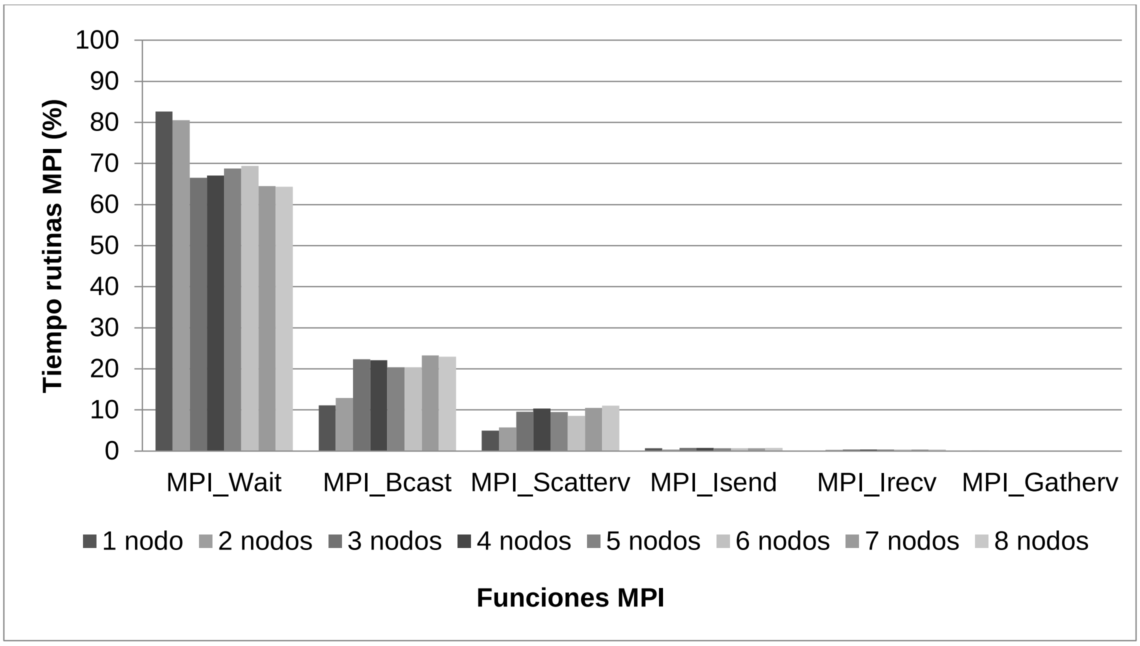

Figura 9. Tiempo de rutinas MPI en porcentaje para las funciones MPI más utilizadas, de 1 hasta 8 nodos.

Los resultados permiten concluir que la escalabilidad del WRF usando una red Ethernet no puede ir más allá de dos nodos debido a los grandes requerimientos de la aplicación, a nivel de comunicación, causando un cuello de botella. Esto no ocurre en las redes InfiniBand que logran tener una escalabilidad más allá de dos nodos, debido a sus altas tasas de transmisión de datos. Por lo tanto, también se concluye que al usar redes InfiniBand con una mayor tasa de transmisión de datos como es FDR respecto a QDR se logra un mayor rendimiento.

Capítulo 3 : Descripción y comparación de rendimiento entre OpenCL y CUDA

3.1 Introducción

Típicamente el incremento de la velocidad de reloj en los procesadores, fue el método usado para aumentar el rendimiento de los sistemas computacionales. Sin embargo, la tendencia actual para el aumento del rendimiento está enfocada en el paralelismo (Naffziger, 2011), arquitecturas manycore como los GPUs (Graphics Processing Unit) representan un claro ejemplo de esto. Los GPUs de hoy en día, poseen dos características principales: (i) miles de núcleos que ejecutan hilos en paralelo; (ii) un gran ancho de banda de memoria. Esto ha hecho que científicos y desarrolladores aprovechen el GPU para dar soporte a aplicaciones de propósito general (GPGPU), con el objetivo de mejorar el rendimiento de resolución de problemas con un alto grado de paralelismo (Kirk & Hwu, 2013).

CUDA (Compute Unified Device Architecture) y OpenCL (Open Computing Language) son modelos de programación y plataformas de computación paralela, que permiten utilizar el hardware de un GPU para propósitos generales, eliminando la necesidad de usar APIs gráficas como OpenGL o DirectX. Esto permite que la computación en GPU pueda ser usada de una manera más sencilla y se extienda entre desarrolladores e investigadores (Su, Chen, Lan, Huang, & Wu, 2012).

CUDA fue desarrollado por NVIDIA y lanzado en Junio del 2007; es software propietario, por lo que solo puede ser ejecutado en GPUs NVIDIA. El objetivo principal es facilitar la explotación del paralelismo de datos en un sistema formado por GPUs NVIDIA y CPUs, mediante un lenguaje simple de programación basado en C (NVIDIA Corporation, 2014).

OpenCL (Open Computing Language) es un estándar abierto no propietario desarrollado por Kronus Group, para la programación en paralelo en sistemas heterogéneos, y lanzada en Diciembre del 2008. Al ser un estándar, OpenCL puede ser compilado y ejecutado en diferentes arquitecturas, independiente del fabricante, mientras esta lo soporte. Entre las arquitecturas soportadas están, manycore como GPUs de NVIDIA o Coprocesadores de Intel; procesadores multinúcleo de Intel, AMD; y Cell Broadband Engine de IBM que han sido implementadas en PlayStation 3, entre otras arquitecturas (Khronos OpenCL WorkingGroup, 2017).

Este capítulo, tiene como objetivo comparar el rendimiento de OpenCL y CUDA mediante el desarrollo y ejecución de una aplicación escrita en dichos modelos de programación. Con el fin de analizar si la propiedad multiplataforma de OpenCL es una desventaja en cuanto a rendimiento, con respecto a CUDA que es un modelo de programación dedicado únicamente a GPUs NVIDIA. Además, Scarpino, (2011) enfatiza que los sistemas CPU + GPU son el futuro de la computación de alto rendimiento, esto es acertado ya que hoy en día los GPUs están siendo aplicados en campos importantes, como el análisis de tráfico aéreo, simulación de moléculas, identificación de placas ocultas en las arterias (NVIDIA Corporation, 2014), computación química, juegos, entrenamiento de redes neuronales; con el fin acelerar los tiempos de procesamiento.

De manera específica, en este trabajo se usó el algoritmo Horizontal Diffusion, ubicado entre los de más costo computacional del WRF. La implementación del algoritmo en OpenCL se baso en una versión escrita para CUDA (Gualan-Saavedra, Solano-Quinde, & Bode, 2015). En este contexto, es necesario realizar un análisis de las similitudes y diferencias de estos dos modelos de programación.

Este capítulo se organiza de la siguiente manera, la Sección 3.2 realiza una descripción de OpenCL y CUDA, con sus principales características; la Sección 3.3 describe el algoritmo utilizado para realizar la comparación, el conjunto de datos y las implementaciones seriales y paralelas tanto para CUDA como para OpenCL; y finalmente la Sección 3.4 muestra la comparación de rendimiento.

3.2 Descripción de OpenCL y CUDA

Con el objetivo de entender los conceptos y las diferencias entre CUDA y OpenCL, en esta sección se dan a conocer dichos modelos de programación, enfocado en las siguientes características: modelo de plataforma, modelo de ejecución y modelo de memoria. El término *dispositivo* hará referencia a un GPU; sin embargo, como se explicó en la sección anterior, OpenCL no está limitado a ejecutarse únicamente en GPUs.

3.2.1 Modelo de plataforma

El modelo de plataforma describe un sistema compuesto por un dispositivo anfitrión, y varios dispositivos secundarios. El dispositivo anfitrión es el encargado de gestionar y sincronizar todas las tareas entre los dispositivos. Un dispositivo en nuestro caso un GPU NVIDIA, está compuesto de varios Streaming MultiProcesssors (SMs), que son unidades de cómputo con independencia entre ellas y estas a su vez contienen un conjunto de núcleos escalares.

3.2.2 Modelo de ejecución

La ejecución de una aplicación OpenCL o CUDA en un sistema heterogéneo, en este caso CPU más GPU, es el conjunto de dos fases: la ejecución en paralelo de piezas de código en unos dispositivos separados como el GPU, y la ejecución del código principal en un host como el CPU. A partir de este proceso, el concepto de función Kernel es aquel que especifica el código a ejecutarse en paralelo.

Un thread o work-item es la unidad básica de procesamiento paralelo, denominada así por CUDA y OpenCL, respectivamente. Cada uno crea una instancia del Kernel, y ejecutan el mismo código, logrando así el paralelismo en las ejecuciones. OpenCL y CUDA agrupa los work-items y threads en work-groups y blocks (Figura 10). Para así asegurar que todos los work-items y threads dentro de un grupo sean ejecutados paralelamente en un solo SM, con el fin de: tener independencia entre grupos, compartir memoria y sincronización, ya que la sincronización queda definida solo dentro del grupo. Al tener independencia en la ejecución de blocks, se logra una escalabilidad automática de rendimiento, como se puede observar en la Figura 11. Por lo tanto, un mismo programa puede aprovechar la capacidad de cómputo de GPUs con pocos SMs dedicados a tareas comunes como videojuegos, hasta GPUs dedicados a la computación de alto rendimiento, con un mayor número de SMs.

En la fase de ejecución, un dispositivo ejecuta varios blocks con independencia entre ellos, en cada SM. La ejecución de los threads que componen el block son distribuidos entre los núcleos pertenecientes a un SM. En el caso de hardware de NVIDIA, los threads se agrupan en grupos de 32, denominadas como *warps* para ser ejecutados mediante un estilo de ejecución llamado SIMT (Single Instruction Multiple Thread).

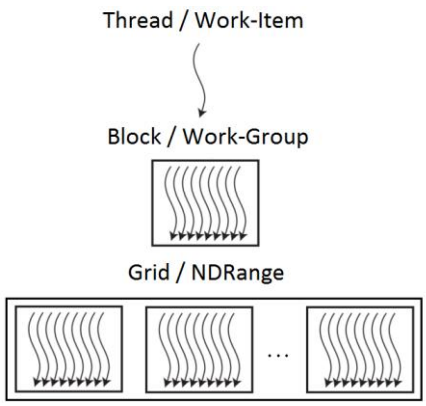

Figura 10. Agrupamiento de Threads o Work-Items. Elaboración propia.

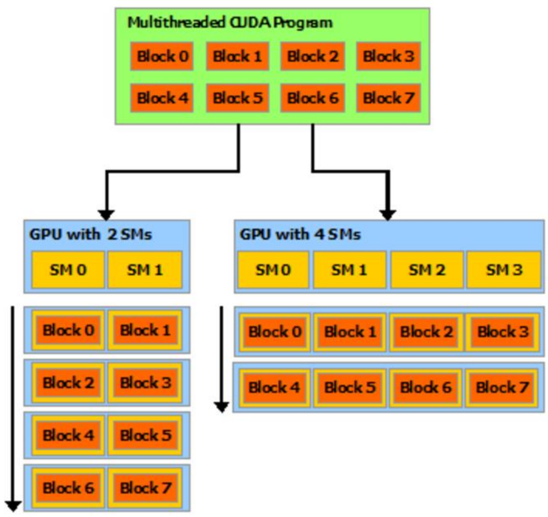

Figura 11. Escalabilidad automática en GPUs NVIDIA (NVIDIA, 2016).

Por otro lado, para explotar el paralelismo de datos CUDA y OpenCL permiten definir los threads, work-items, desde una dimensión hasta tres dimensiones. Dependiendo del tipo de datos que se vayan a procesar se escoge la dimensión; por ejemplo, si es una imagen la que va a ser procesada, lo más común es que los threads se organicen en grillas de dos dimensiones, como se observa en la Figura 12, y en el caso de que los datos sean espaciales la agrupación más adecuada es una grilla de tres dimensiones. En este contexto un thread puede ser visto como un punto en este espacio dimensional (Fraire, Ferreyra, & Marques, 2013), y puede ser identificado de manera única con un ID.

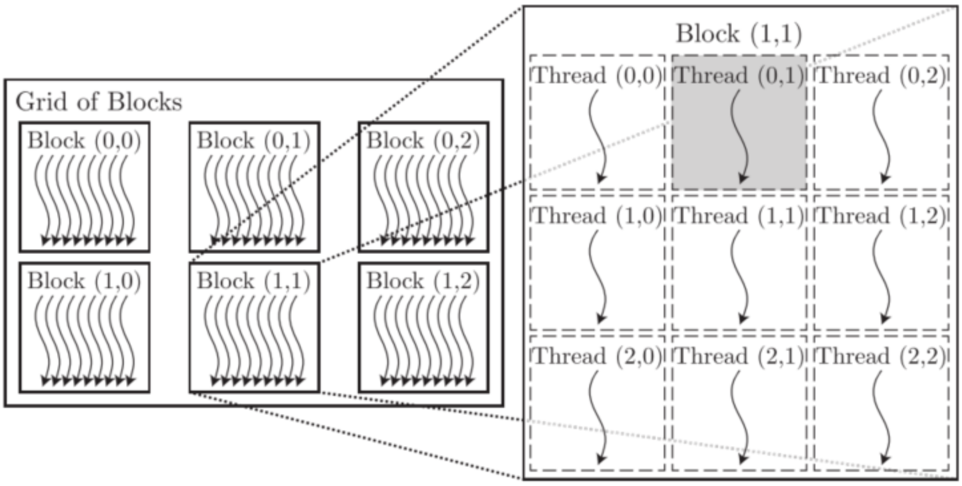

Figura 12. Jerarquía de una grilla con threads y bloques de 2 dimensiones (Caz, 2016)

En el modelo de ejecución, las diferencias de CUDA y OpenCL están más orientadas en la terminología, y la definición de tamaño de warp por parte de NVIDA, ya que OpenCL debido a su propiedad de portabilidad, no define ningún tamaño.

3.2.3 Modelo de memoria

El modelo de programación permite acceder de manera explícita a una jerarquía de memorias, diseñada con el fin de realizar una gestión eficiente de los datos. En la Tabla 2 se introducen las memorias, indicando las equivalencias en los dos modelos de programación, y las Figuras 13 y 14 muestran el nivel de acceso que tienen las diferentes unidades de procesamiento paralelo, hacia las memorias.

CUDA	OpenCL	Acceso
Register	Private Memory	Thread / Work-item
Shared Memory	Local Memory	Block / Work-group
Global Memory	Global Memory	Grid / NDRange
Constant Memory	Constant Memory	Grid / NDRange
Local Memory	Private Memory	Thread
Texture or Surface Memory	-	Grid

Tabla 2. Jerarquía de memorias de CUDA y OpenCL. Elaboración propia.

A continuación se detalla cada una de las memorias:

Register/Private Memory: Esta memoria es la de más rápido acceso, el ámbito en esta región de memoria es únicamente privada para un thread/work-item.

Shared/Local Memory: El ámbito es por block/work-group, así todos los threads/work-items, pertenecientes a un mismo block/work-group, pueden compartir datos en esta área de memoria. En el GPU, esta hace referencia a la memoria local que tiene cada SM. Esta memoria se emplea para reutilización de datos y disminución del número de los accesos a la memoria principal, el cuál es un método muy común para minimizar las penalizaciones de tiempo, ya que al ser una memoria compartida cuenta con una mayor velocidad de acceso.

Global Memory: Hace referencia a la memoria principal del dispositivo que por lo general es una del tipo DRAM (Dynamic Random Access Memory), la cual es de gran capacidad comparado con las demás, su característica principal es el gran ancho de banda, pero con alta latencia. Dependiendo de la arquitectura, esta memoria puede utilizar un caché de nivel L1 y L2 o solamente L2. Todos los threads/work-items existentes y el host pueden acceder a ella para realizar operaciones de lectura y escritura. Aquí, se trasladan los datos de la memoria del host para ser procesado por el GPU.

Constant Memory: Sirve para el almacenamiento de datos constantes, por lo que el dispositivo solo puede realizar lecturas, mientras que el host, puede realizar operaciones de lectura y escritura. Algunos dispositivos proveen una memoria específica para este tipo de datos, pero en la mayoría de casos se utiliza la misma región que la memoria global, y utiliza una caché para mejorar su rendimiento.

Local / Private Memory: Este concepto de memoria es utilizada cuando una variable o estructura supera el espacio de registros disponible, entonces los datos son transferidos a local memory, lo cual es conocido como register spilling. La memoria local reside en la memoria del dispositivo, en el mismo lugar que la memoria global, debido a esto, no es recomendable sobrepasar el espacio de los registros, ya que se obtiene retrasos en el acceso. Dependiendo de la arquitectura de hardware se utiliza la caché L1 y L2 o solamente la caché L2, de la misma manera que el acceso a la Memoria Global.

Texture Memory: Reside en la memoria principal del dispositivo, también utiliza un caché, pero de texturas, que está optimizada para aprovechar la localidad de datos de dos dimensiones.

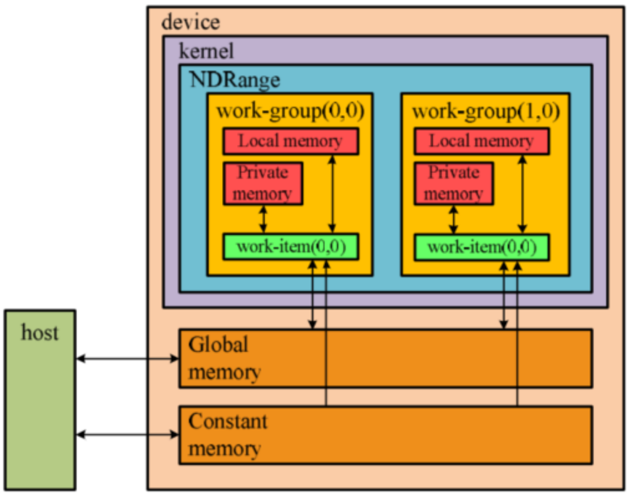

Figura 13. Jerarquía de memorias y acceso de work-Item en OpenCL (Su, Chen, Lan, Huang, & Wu, 2012)

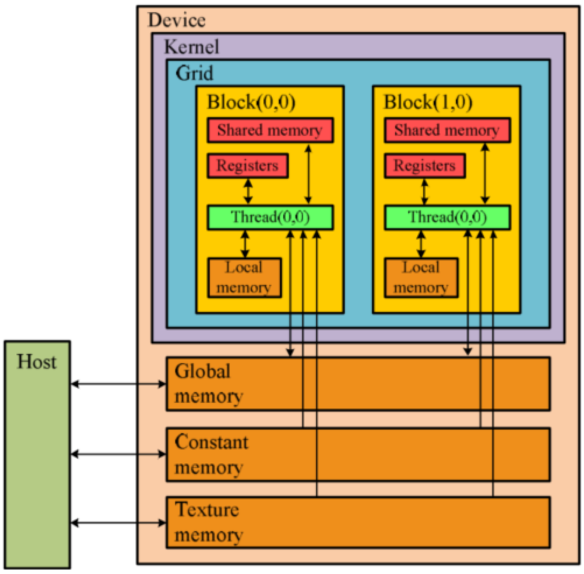

Figura 14. Jerarquía de memorias y acceso de thread en CUDA (Su et al., 2012).

CUDA tiene la capacidad de realizar configuraciones para especificar la cantidad de memoria de un chip usa como L1 y shared memory, para así poder adaptar de mejor manera un determinado algoritmo a un GPU; mientras que, en la documentación de OpenCL no existe este tipo de opciones de configuración (Khronos OpenCL WorkingGroup, 2017).

La diferencia entre el modelo de memoria de OpenCL y CUDA está en sus terminologías para cada función de memoria, y la definición de tipos de memoria adicionales por parte de CUDA, que están ligadas a las funciones de un dispositivo GPU, como por ejemplo la memoria de texturas. En adición CUDA tiene la ventaja de ofrecer configuraciones adicionales vinculadas directamente con el hardware NVIDIA, lo que da la opción de mejorar el rendimiento en determinados algoritmos. Sin embargo, gran parte del desempeño de un programa depende de un buen diseño, ya que una mala utilización de memoria puede dar un rendimiento totalmente indeseado.

3.3 Algoritmo: Horizontal Diffusion

Horizontal Diffusion por su nombre en inglés, es un algoritmo que pertenece al módulo de dinámicas ARW del modelo WRF. Un algoritmo de difusión o disipación espacial, en los modelos de clima numéricos sirve generalmente para las discretizaciones espaciales y temporales y controla problemas de origen numérico como el ruido indeseable debido a una mala inicialización y la estabilidad computacional. El módulo ARW tiene dos formulaciones para la difusión espacial explícita: el espacio físico (x,y,z) y en coordenadas de superficie; siendo el algoritmo Horizontal Diffusion parte de las coordenadas de superficie (Skamarock et al., 2008).

Este algoritmo es el utilizado en nuestro estudio, para realizar la comparación de rendimiento entre CUDA y OpenCL, debido a su alto costo computacional, ya que se encuentra sexto entre los diez métodos de mayor exigencia del WRF. Como se mencionó en la Sección 1.2 de trabajos relacionados, a día de hoy, los únicos estudios de paralelización realizados sobre Horizontal Diffusion, son hechos por Gualán-Saavedra, Solano-Quinde, & Bode, (2015) en la Universidad de Cuenca, y se desea continuar con el enfoque de rendimiento y aceleración.

Actualmente existe una implementación en paralelo del algoritmo Horizontal Diffusion, implementado en CUDA para un GPU (Gualán-Saavedra et al., 2015) y múltiples GPUs (Solano-Quinde, Gualán-Saavedra, & Zuñiga-prieto, 2016). Este algoritmo diseñado y optimizado para CUDA fue ejecutado en esta evaluación y tomado como base para el desarrollo en OpenCL, teniendo en cuenta sus respectivas diferencias como la configuración de caché, el sincronismo a nivel de dispositivo, la definición de tamaño de work-group y configuraciones adicionales.

El algoritmo secuencial desarrollado en Fortran mostrado en la Figura 15, está formado principalmente por tres bucles que recorren un espacio tridimensional. Los cálculos son realizados por cada iteración de los bucles, con el fin de calcular el vector resultado *tendency*.

```
SUBROUTINE horizontaldiffusion ( !Parameters )
! . . .
DO j = j_start , j_end
DO k = kts , ktf
DO i = i_start , i_end
mkrdxm=(msftx(i-1,j)/msfty(i-1,j))*mu(i-1,j)*xkmhd(i-1,k,j)*rdx
mkrdxp =(msftx(i,j) / msfty (i,j))*mu(i,j) *xkmhd(i,k,j)*rdx
! . . .
tendency(i,k,j)=tendency(i,k,j)+( &
        mrdx*(mkrdxp*(field(i+1,k,j)-field(i   ,k,j))    &
                -mkrdxm*(field(i,k,j)-field(i-1,k,j)))  &
        +mrdy*(mkrdyp*(field(i,k,j+1)-field(i,k,j))     &
                -mkrdym*(field(i,k,j)-field(i,k,j-1))))
ENDDO
ENDDO
ENDDO
! . . .
END SUBROUTINE horizontaldiffusion
```

Figura 15. Esquema Horizontal Diffusion versión secuencial en Fortran.

El proceso de paralelización del algoritmo esta principalmente dado en el reemplazo de los dos bucles externos por, índices de threads representados en dos dimensiones. Para este espacio tridimensional la descomposición es de un thread por columna. Adicionalmente con el objetivo de mejorar el rendimiento, el algoritmo utiliza local memory para la reutilización de datos (Gualan-Saavedra et al., 2015). La Figura 16 y la Figura 17, muestran el resultado de este proceso de paralelización en CUDA y OpenCL respectivamente, las diferencias con mayor importancia en programación son las siguientes:

- La definición de los índices globales, CUDA no ofrece una función para obtener un índice global o identificador único de manera directa, por lo que se consigue mediante operaciones de multiplicación y suma entre el bloque y el identificador de thread por bloque. Por el contrario OpenCL provee una función para obtener este índice global directamente, esto se muestra en la línea 5 y 6 de la Figuras 16 y 17 para CUDA y OpenCL respectivamente.
- La barrera de sincronización de threads/work-items, para asegurar que la copia de datos de la *Global Memory* hacia la *Shared/Local Memory*, este completa. En la versión de CUDA, el condicional *IF* termina los threads que no están dentro del índice de datos mediante *return*, los threads restantes realizan la copia y esperan que todos los hilos del mismo bloque lleguen a la línea *__syncthreads,* mostrada en la Figura 16. En OpenCL para este caso, no puede utilizar la función *return* de la

misma manera que en CUDA, debido que para un correcto funcionamiento de las barreras, la documentación OpenCL dice: 1) La función *barrier* debe ser encontrada por todos los work-items de un work-group que ejecutan el kernel, 2) Si la barrera está dentro de una sentencia condicional, todos los work-items deben ingresar el condicional, y ejecutar la barrera (The Khronos Group Inc, 2009). En el código OpenCL mostrado en la Figura 17, la copia de datos esta dentro del condicional *IF* evitando que los *work-items* que no pertenecen al índice de datos, accedan a espacios de memoria fuera de rango; y debido a que la barrera de sincronización no puede ir dentro de un condicional, porque una parte de los work-items no está dentro del rango; esto es solucionado con la escritura del mismo condicional *IF* dos veces, uno para la copia en la *Local Memory* y otra para los cálculos de *tendency*, y así dejar fuera de un condicional a la barrera de sincronización *barrier(CLK_LOCAL_MEM_FENCE)* logrando que todos los work-items lleguen a esta barrera, como se puede ver en la Figura 17.

```
__global__ void horizontal_diffusion_cuda(/*parameter*/){
//...
__shared__ float mu_s[((blockDim.x+2)*(blockDim.y+2))];
//..
i = blockIdx.x*blockDim.x+threadIdx.x;
j = blockIdx.y*blockDim.y+threadIdx.y;
ti= threadIdx.x;
tj= threadIdx.y;
if (!(i>=i_start+4 && i<=i_end+4 && j>=j_start+4 && j<=j_end+4)) {
 return;
}
mu_s[S2(ti-1,tj-1)] = mu[P2(ti-1,tj-1)];
//...
__syncthreads();
for (k=kts-1; k<=ktf-1; k++) {
mkrdxm = (msftx_s[S2(ti-1,tj)]/msfty_s[S2(ti-1,tj)])*mu_s[S2(ti-1,tj)]*
    xkmhd[P3(ti-1,k,tj)]*rdx;
mkrdxp = (msftx_s[S2(ti,tj)]/msfty_s[S2(ti,tj)])*mu_s[S2(ti,tj)]*xkmhd[
    P3(ti,k,tj)]*rdx;
//...
tendency[P3(ti,k,tj)] = tendency[P3(ti,k,tj)]+(mrdx*(mkrdxp*(field[P3(
    ti+1,k,tj)]-field[P3(ti,k,tj)])-mkrdxm*(field[P3(ti,k,tj)]-field[P3
    (ti-1,k,tj)]))+ mrdy*(mkrdyp*(field[P3(ti,k,tj+1)]-field[P3(ti,k,tj
    )])-mkrdym*(field[P3(ti,k,tj)]-field[P3(ti,k,tj-1)])));
}
//...
}
```

Figura 16. Esquema Horizontal Diffusion versión CUDA.

```
__kernel void horizontal_diffusion_OpenCL(/*parameter*/){
//...
__local float mu_s[((get_local_size(0)+2)*(get_local_size(1)+2))];
//..
i = get_global_id(0);
j = get_global_id(1);
ti = get_local_id(0);
tj = get_local_id(1);
if(i>=i_start+4 && i<=i_end+4 && j>=j_start+4 && j<=j_end+4){
mu_s[S2((ti-1),(tj-1))] = mu[P2((i-1),(j-1))];
//...
}
barrier(CLK_LOCAL_MEM_FENCE);
if(i>=i_start+4 && i<=i_end+4 && j>=j_start+4 && j<=j_end+4){
for (k=kts-1; k<=ktf-1; k++) {
mkrdxm = (msftx_s[S2((ti-1),tj)]/msfty_s[S2((ti-1),tj)])*mu_s[S2((ti-1)
    ,tj)]*xkmhd[P3((i-1),k,j)]*(rdx);
mkrdxp = (msftx_s[S2(ti,tj)]/msfty_s[S2(ti,tj)])*mu_s[S2(ti,tj)]*xkmhd[
    P3(i,k,j)]*(rdx);
//...
tendency[P3(i,k,j)]=tendency[P3(i,k,j)]+(mrdx*(mkrdxp*(field[P3((i+1),k
    ,j)]-field[P3(i,k,j)])-mkrdxm*(field[P3(i,k,j)]-field[P3((i-1),k,j)
    ]))+ mrdy*(mkrdyp*(field[P3(i,k,(j+1))]-field[P3(i,k,j)])-mkrdym*(
    field[P3(i,k,j)]-field[P3(i,k,(j-1))])));
}
}
//...
}
```

Figura 17. Esquema Horizontal Diffusion versión OpenCL.

En adición, OpenCL requiere de programación adicional debido a configuraciones particulares necesarias, por su forma de manejo de varios dispositivos de diferentes tipos como se muestra en la Figura 18. Para lograr despachar un kernel desde el host hacia un dispositivo específico, se debe realizar las siguientes tareas: (i) creación de un Contexto que es el contenedor de todos los dispositivos, (ii) creación del programa que es el que contiene los diferentes kernels a ser despachados, (iii) obtención de un dispositivo específico, (iv) creación de una cola de comandos que es el medio para despachar un kernel desde un programa hacia un dispositivo específico.

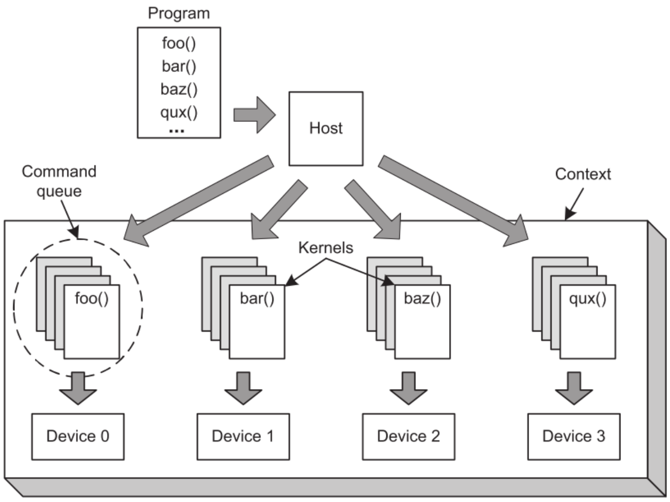

Figura 18. Distribución de kernels hacia dispositivos OpenCL.

Por último, debido a que los dos lenguajes de programación están basados en el estándar c99 o ISO/IEC 9899:1999, el esfuerzo para la traducción entre estos dos lenguajes una vez que se tienen todos los conceptos claros, no es tan elevado. Por otro lado, debido a que OpenCL tiene más características como, paralelismo de tareas, portabilidad, y el lanzamiento de kernels entre dispositivos de diferentes arquitecturas, hacen de su curva de aprendizaje más prolongada que CUDA.

3.4 Evaluación de rendimiento de CUDA y OpenCL sobre un GPU

Se plantea evaluar el rendimiento del algoritmo paralelizado Horizontal Diffusion, descrito en la Sección 3.3, utilizando dos modelos de lenguajes paralelos diferentes CUDA y OpenCL. Adicionalmente, se ejecutó el algoritmo en su versión secuencial en Fortran, que sirve como un valor de referencia para las aceleraciones obtenidas por OpenCL y CUDA. Para esto se utilizó, el conjunto de datos CONUS de 12 km, descrito en la sección 2.4.

Los algoritmos paralelizados son ejecutados sobre un GPU NVIDIA TESLA K40m y el algoritmo secuencial es ejecutado sobre un procesador Intel Xeon E5-2630 2.4 GHz

El método consiste en ejecutar el algoritmo Horizontal Diffusion, tanto para CUDA, OpenCL y Fortran, con un tamaño de block de 32 x 8, dando un total de

256 hilos por bloque, considerado la mejor configuración para el algoritmo según Gualan-Saavedra et al., (2015); sin embargo, se realizaron ejecuciones variando el tamaño del bloque de hilos, en la versión OpenCL con el objetivo de encontrar el mejor rendimiento. Para la medición del tiempo de ejecución del kernel se utilizaron los eventos de profiling provistos por CUDA y OpenCL.

Lenguaje	Descripción	Tiempo de Ejecución (milisegundos)
FORTRAN	Implementación secuencial	60
OpenCL	Bloque 32 x 8	1.31
CUDA	Bloque 32 x 8, configuración con preferencia L1	1.24

Tabla 3. Tiempo de ejecución Horizontal Diffusion.

La Tabla 3 muestra el tiempo de ejecución del programa secuencial en FORTRAN y el tiempo de ejecución del programa en versión paralelo en CUDA y OpenCL, dando un mayor rendimiento a CUDA con un 6% sobre OpenCL. CUDA muestra una mejor utilización de su propio hardware con su plataforma dirigida a GPUs y configuraciones adicionales que se pueden hacer, al contrario de OpenCL. Aunque OpenCL tiene un menor rendimiento, en general este no es afectado de una forma crítica por sus características multiplataforma.

Capítulo 4 : Descripción y comparación de rendimiento entre arquitecturas manycore

4.1 Introducción

Las arquitecturas manycore como Intel Xeon PHI 7120p o GPU NVIDIA Tesla K40m son utilizadas ampliamente para dar soporte ha aplicaciones que requieren gran capacidad de cómputo, tomando ventaja de la gran cantidad de núcleos de computación que posee. Sin embargo estas presentan diferencias en sus arquitecturas de hardware, a nivel de la jerarquía de memorias, y procesadores. Por lo tanto una arquitectura tiene características propias que la hacen más apropiadas para algunos algoritmos, mientras que para otros no.

Las arquitecturas de NVIDIA e Intel tienen como objetivo maximizar la cantidad de operaciones en punto flotante por segundo, mediante el uso del paralelismo. Sin embargo, las dos arquitecturas presentan deficiencias como la gran latencia que existe en los accesos a la memoria principal (Toledo & Barrientos, 2014). Existen dos diferencias principales entre estas dos arquitecturas, que son un sistema de cachés explícito, con una gran cantidad de procesadores escalares por parte de los GPUs NVIDIA y un sistema de caché de coherencia con unidades de procesamiento vectoriales y escalares por parte de Xeon Phi. Debido a las diferencias inherentes entre las dos arquitecturas, es necesario analizar el rendimiento en un algoritmo de interés. Este análisis permitirá determinar las fortalezas y debilidades que presentan las arquitecturas. De manera similar que en el capitulo anterior, en el presente trabajo se utilizará la implementación en OpenCL del algoritmo Horizontal Diffusion.

El presente capítulo tiene como objetivo realizar una comparación de rendimiento y de arquitectura de los aceleradores Intel Xeon Phi y NVIDIA Tesla Kepler. Si bien OpenCL es un modelo de programación portable entre diferentes arquitecturas, el rendimiento obtenido es muy dependiente de la arquitectura en la que se ejecuta. En este contexto, las aplicaciones deben ser optimizadas con el objetivo de sacar el mayor provecho de la arquitectura sobre la que se ejecuta (Fang, Varbanescu, & Sips, 2011). Debido a esto es necesario entender las arquitecturas existentes, con el fin de poder realizar optimizaciones en el algoritmo desarrollado en OpenCL.

4.2 Descripción de Intel Xeon Phi 7120p

Intel ha diseñado coprocesadores orientados a aplicaciones de alto rendimiento, que poseen una gran cantidad de núcleos, denominado MIC (Many Integrated Core), entre los que se encuentra el coprocesador Intel Xeon Phi 7120p, el cual posee 60 núcleos y un tamaño de memoria principal de 16 GB del tipo GDRAM, con ocho controladores de memoria.

Como se observa en la Figura 19, cada núcleo del acelerador tiene una unidad de procesamiento escalar y una vectorial (VPU), con un ancho de 512 bits, que da la capacidad de ejecutar 16 operaciones de precisión simple u ocho de doble precisión de punto flotante a la vez. También, posee una caché de segundo nivel de 512 KB privada inclusiva (L2); es decir, el contenido de la caché de primer nivel (L1) está replicado en la L2. Las cachés de primer nivel tienen una capacidad de 32-KB para datos y 32-KB para instrucciones. Las cachés L2 son coherentes y pueden intercambiar datos con las cachés de los otros núcleos a través del anillo de bus bidireccional. La coherencia se mantiene gracias a un directorio de etiquetas distribuido denominado TD (Tag Directory) (Intel Corporation, 2013).

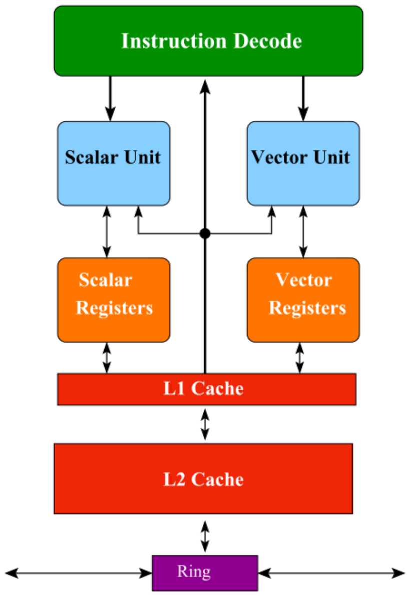

Figura 19. Esquema de un núcleo de in coprocesador Intel Xeon Phi (Toledo & Barrientos, 2014).

En la Figura 20, se puede observar un esquema completo de la arquitectura Intel Xeon Phi.

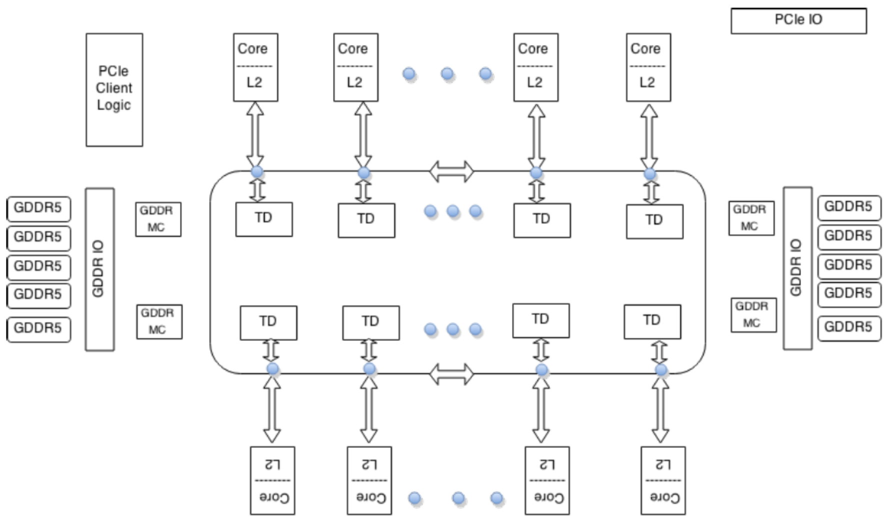

Figura 20. Esquema de un coprocesador Intel Xeon Phi (Theses & Shareef, 2015).

Para la ejecución de aplicaciones en OpenCL en aceleradores Xeon Phi, los work-groups son ejecutados por cada núcleo y son independientes entre ellos, cada work-group es asignado a un thread y este ejecuta a todos los work-items dentro del work-group mediante SIMD (Single Instruction Multiple Data). Por lo tanto, existe paralelismo a nivel de work-group mediante instrucciones vectoriales y paralelismo entre work-group mediante threads.

4.3 Descripción de GPU NVIDIA Tesla K40m

NVIDIA tiene varias arquitecturas GPU, orientadas al alto rendimiento. Dentro de estos tenemos el modelo Tesla, diseñada con una arquitectura Kepler. El GPU que fue utilizado para el presente trabajo es el NVIDIA Tesla K40m, la cual está compuesta por una GPU Kepler GK110B, 15 SMs (Streaming Multiprocessors), y 6 controladores de memoria de 64 bits con una interfaz de 384 bit, que soportan una memoria de GDDR5 DRAM de 12 GB; como se puede observar en la Figura 21 (a) (NVIDIA Corporation, 2013).

Cada SM como se observa en la Figura 22, está compuesto por: 192 núcleos de precisión simple, 64 unidades de doble precisión; 4 planificadores con 2 despachadores de instrucciones, lo que permite la ejecución de 4 warps al mismo tiempo; 32 unidades para la carga y almacenamiento de datos; una caché L1 de 64 KB más un área de cache de 48KB para la lectura de datos. Adicionalmente tiene una cache L2 de 1536 Kb, que es compartida por todos los SM (NVIDIA Corporation, 2012) como se observa en la Figura 21 (b).

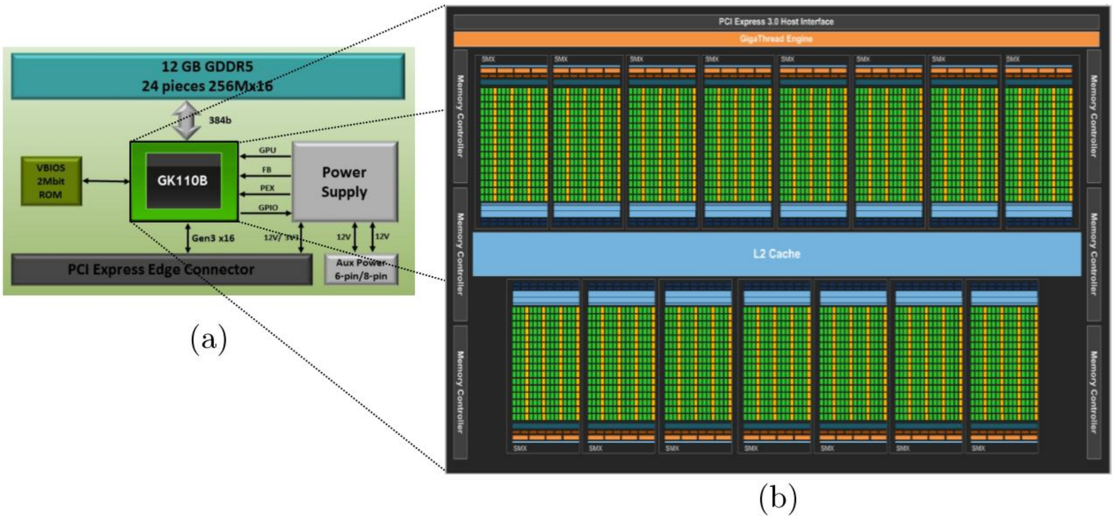

Figura 21. (a) Esquema de una NVIDIA TESLA K40m (b) Esquema de una GPU GK110B (NVIDIA Corporation, 2013).

Figura 22. Esquema de un SM (NVIDIA Corporation, 2012).

Para la ejecución de aplicaciones en OpenCL en un GPU NVIDIA, se asigna un work-group a cada SM, y este divide en grupos de 32 work-items (warps), para ser planificados y despachados, entre los núcleos de cada SM. El tamaño máximo aceptado por este GPU es de 1024 work-items por work-group.

4.4 Optimización de Horizontal Diffusion, para Xeon Phi

El algoritmo Horizontal Diffusion descrito en la Sección 3.3 está desarrollado y optimizado para una arquitectura GPU de NVIDIA. Un coprocesador Intel Xeon Phi tiene una arquitectura diferente, por lo cual es necesario incorporar modificaciones y optimizaciones a la aplicación. Como se mencionó antes, si bien OpenCL es portable entre arquitecturas, es necesario hacer cambios en el código del algoritmo, con el fin de obtener el máximo rendimiento. Con el objetivo de aprovechar el mayor desempeño del coprocesador Intel Xeon Phi 7120p para el algoritmo Horizontal Diffusion, Intel hace las siguientes recomendaciones (Intel Corporation, 2013):

- No utilización de local memory: El GPU permite el uso de esta memoria de alta velocidad de manera explícita, para disminuir los accesos a la memoria principal, dando total libertad al programador de decidir qué datos copiar a esta memoria. En contraste, Xeon Phi implementa un sistema de cachés coherentes mediante TD, en consecuencia, no se debe realizar copias a la local memory, ya que se vuelve a copiar los datos en la memoria principal, provocando una caída de rendimiento, causada por las copia de datos innecesaria provocando redundancia. Por lo tanto, se ha eliminado el uso de las variables que utilizan local memory, como se observa en la Figura 23.
- Modificación del tamaño de work-group: Intel recomienda tener una cantidad total de work-groups 10 veces más que los núcleos, para así lograr una ocupación completa de todos los núcleos. Además, recomienda utilizar tamaños en múltiplos de 16, para lograr un acceso contiguo a la memoria principal. Para encontrar el mejor tamaño de work-group se utilizó la opción de determinación automática y también se realizaron varias pruebas manuales con diferentes tamaños, teniendo como mejor resultado la selección manual de 32 work-items por work-group, con la dimensión de 16 por 2 work-items.

Por otro lado, existe la Vectorización implícita que es una mejora de rendimiento realizada por Intel. Esta es una optimización automática llevada a cabo por el módulo de vectorización de Intel, que empaqueta work-items contiguos o vectoriza la dimensión cero; es decir, los bucles más internos de un Kernel los ejecuta mediante instrucciones SIMD. Cabe mencionar que el grado de vectorización siempre está dado por los compiladores de Intel, sin embargo Intel recomienda la utilización de valores del tipo float, double, integer, y Kernels con un flujo simple (Intel Corporation, 2014).

```
__kernel void horizontal_diffusion_OpenCL(/*parameter*/){
//...
i = get_global_id(0);
j = get_global_id(1);

if( !( i>=i_start+4 && i<=i_end+4 && j>=j_start+4 && j<=j_end+4 ) ){
return;
}

for (k=kts-1; k<=ktf-1; k++) {
mkrdxm = (msftx[P2((i-1),j)]/msfty[P2((i-1),j)])*mu[P2((i-1),j)]*xkmhd[P3((i-1),k,
     j)]*(rdx);
mkrdxp = (msftx[P2(i,j)]/msfty[P2(i,j)])*mu[P2(i,j)]*xkmhd[P3(i,k,j)]*(rdx);
//...

tendency[P3(i,k,j)]=tendency[P3(i,k,j)]+(
        mrdx*(mkrdxp*(field[P3((i+1),k,j)]-field[P3(i,k,j)])
        -mkrdxm*(field[P3(i,k,j)]-field[P3((i-1),k,j)]))
        + mrdy*(mkrdyp*(field[P3(i,k,(j+1))]-field[P3(i,k,j)])
        -mkrdym*(field[P3(i,k,j)]-field[P3(i,k,(j-1))])));
}
}
```

Figura 23. Kernel en OpenCL optimizado para Intel Xeon Phi.

4.5 Evaluación de rendimiento de aceleradores, Intel Xeon Phi 7120p y GPU NVIDIA Tesla K40m

Esta Sección describe la comparación de rendimiento entre los aceleradores manycore Intel Xeon Phi 7120p y el GPU NVIDIA Tesla K40m mediante la ejecución del algoritmo Horizontal Diffusion. El método para llevar a cabo esta evaluación, consiste en medir el tiempo de ejecución del algoritmo Horizontal Diffusion, mediante los eventos de profiling de OpenCL. El algoritmo está implementado en OpenCL, pero con optimizaciones realizadas para cada uno de los aceleradores, en concordancia con las Secciones 3.3 y 4.4. Así mismo el conjunto de datos usado fue el CONUS 12 km, descrito en la Sección 2.4. Con estos resultados se puede ver cuál de estos dos aceleradores ofrece un mejor rendimiento.

Acelerador	Tamaño work-group	Descripción	Tiempo de Ejecución (milisegundos)
GPU Tesla K40m	32 x 8	Optimizado para GPU	1.31
Xeon Phi 7120p	32 x 8	Optimizado para GPU	15.18
Xeon Phi 7120p	32 x 8	Sin Local Memory	11.18
Xeon Phi 7120p	--	Ajuste de bloque automático	10.22
Xeon Phi 7120p	16 x 2	Ajuste de bloque manual	6.32

Tabla 4. Tiempo de ejecución Horizontal Diffusion en un Tesla K40m y Xeon Phi 7120p.

La Tabla 4 muestra el tiempo de ejecución de la aplicación en el acelerador y el tamaño de work-group usado, resultando en un mejor desempeño Tesla K40m, siendo este 4.82 veces más rápido que Xeon Phi 7120p. La selección de tamaño de work-group que obtuvo un mejor rendimiento, se realizó de forma empírica, tomando en cuenta las recomendaciones de Intel, mencionadas en la Sección 4.4. La razón para que Tesla K40m obtenga un mejor rendimiento, es por el alto grado de paralelización del algoritmo Horizontal Diffusion, ya que genera un total de 134850 threads lo que hace a este acelerador idóneo por poseer 2880 núcleos. Mientras que Intel Xeon Phi 7120p, no está orientado totalmente a la paralelización, sino también al procesamiento vectorial, debido a las VPU que contienen cada núcleo. Por otro lado, el software que ofrece Intel para OpenCL de sus arquitecturas manycore no se encuentra actualizado, así que es posible obtener un mejor rendimiento, al usar otras herramientas como OpenMP, Cilk o TBB (Threading Building Blocks) de Intel.

Capítulo 5 : Conclusiones

El presente trabajo evaluó el rendimiento de clústeres HPC, aceleradores manycore y modelos de programación paralela CUDA y OpenCL, a través del modelo WRF y el algoritmo Horizontal Diffusion que se encuentra entre los de mayor exigencia computacional dentro del WRF.

Para la evaluación del rendimiento y la escalabilidad del WRF en clústeres de alto rendimiento se seleccionó el programa *wrf.exe*, debido a su implementación en paralelo. Se utilizó tres escenarios para obtener información de rendimiento, en dos clústeres HPC. El primer escenario, midió el tiempo de ejecución de una simulación con 12 procesos MPI por nodo y ocho nodos sobre redes InfiniBand FDR, QDR y Ethernet. El segundo, buscó obtener la aceleración máxima con el incremento de 12 a 16 procesos por nodo, sobre el clúster UCUENCA. El tercero, consistió en separar el tiempo de ejecución en tiempo de rutinas MPI y tiempo de aplicación, usando las redes InfiniBand FDR, QDR y Ethernet, para analizar la relación entre las comunicaciones y la escalabilidad del WRF.

Los escenarios planteados, mostraron una clara dependencia entre el WRF y las comunicaciones entre los nodos del clúster HPC, debido a: (i) el incremento del volumen de datos transmitidos a medida que se utilizan mas nodos, (ii) el tiempo de utilización de la función MPI_Wait, la cual bloquea la ejecución mientras se realiza el envío de datos. Estas dos razones producen un cuello de botella en la red Ethernet, debido a que MPI_Wait produce un retardo por el gran volumen de datos, que se produce a partir de la utilización de 3 nodos. Para el caso InfiniBand, no existe el cuello de botella, ya que posee baja latencia, por lo que puede soportar grandes volúmenes de datos. Por otro lado, si bien el rendimiento del WRF mejoró al aumentar de 12 a 16 procesos por nodo, el mayor rendimiento se obtuvo al usar una red InfiniBand FDR que mejoró un 55% con respecto a InfiniBand QDR y un 87% con respecto a Ethernet; debido a esto se hace hincapié en que no es útil aumentar la capacidad computacional de los procesadores, si no hay disponibilidad de una red que pueda soportar las exigencias de comunicación del WRF.

Para la comparación de rendimiento de CUDA y OpenCL, se requirió. Primero, la descripción y comparación general de cada modelo de programación. Segundo la implementación del algoritmo Horizontal Diffusion en OpenCL basado en una versión de CUDA. La evaluación se realizó con la medición del tiempo de ejecución de los dos modelos, sobre una GPU NVIDIA. Con el fin de determinar si el rendimiento de OpenCL, es afectado por su propiedad multiplataforma.

Las diferencias de modelo de memoria de CUDA y OpenCL, son las configuraciones y tipos de memoria adicionales que posee CUDA, por estar diseñado exclusivamente para arquitecturas GPUs, ayudando en algunos casos a mejorar la aceleración de algoritmos. En la etapa de implementación del algoritmo en OpenCL no demandó dificultad, ya que están basados en el estándar C99. Las diferencias se dieron, en la obtención de los identificadores globales de thread o work-item, la sincronización de hilos dentro de un block o work-group, y el lanzamiento del Kernel. Además, OpenCL por ser multiplataforma, y soportar paralelismo de tareas entre diferentes dispositivos, necesita la comprensión de conceptos y configuraciones adicionales, por lo que su curva de aprendizaje es más prolongada.

El rendimiento de CUDA es 6% mejor respecto a OpenCL, debido a su modelo de programación que está completamente orientado a GPUs NVIDIA. A nivel general, el rendimiento de OpenCL no se ve afectado de manera drástica por ser multiplataforma, concordando con estudios realizados por (Fang, Varbanescu, & Sips, 2011; Karimi, Dickson, & Hamze, 2010; Su et al., 2012), sobre las diferencias de rendimiento e implementación entre OpenCL y CUDA. En consecuencia OpenCL es una alternativa viable con respecto a CUDA, en caso de que las necesidades o requerimientos estén apuntando hacia la portabilidad.

La comparación de rendimiento del algoritmo Horizontal Diffusion en los aceleradores NVIDIA Tesla K40m e Intel Xeon Phi 7120p se realizó mediante la medición del tiempo de ejecución del algoritmo implementado en OpenCL, debido a que este modelo de programación es portable entre estos dos aceleradores. Se requirió de un análisis de cada arquitectura, para optimizar el código del algoritmo para cada acelerador..

Las diferencias en la arquitectura Tesla Kepler y Xeon Phi, son el sistema de cachés y sus unidades de procesamiento. NVIDIA da la posibilidad de utilizar las cachés de manera explícita en los GPUs; es decir, el programador puede copiar datos a la local memory para reutilizarlos según sea su conveniencia. Por otro lado, Intel en Xeon Phi integra un sistema de cachés tradicional de coherencia, así la reutilización de datos es implícita y es gestionada directamente por el acelerador. En cuanto al rendimiento máximo teórico, el GPU Tesla k40m presenta 1.43 Tflops y 4.29 Tflops para precisión doble y simple respectivamente, y posee 2880 núcleos de precisión simple, dándole una gran cantidad de paralelismo. Mientas que Xeon Phi 7120p presenta 1.208 Tflops para precisión doble, posee 61 núcleos con una unidad de procesamiento escalar y vectorial, que dan un total de 4 threads por unidad alcanzando hasta los 244 threads en total.

Mediante la medición del tiempo de ejecución del Horizontal Diffusion optimizado. Tesla k40m obtuvo un rendimiento 4.82 veces más rápido que Xeon Phi 7120p, debido principalmente a que el algoritmo Horizontal Diffusion genera 134850 threads, lo que demandan un alto grado de paralelismo, haciendo idóneo a un GPU como el Tesla K40m para su ejecución. Por otro lado para que Xeon Phi 7120p, tenga un mejor rendimiento, el algoritmo no solamente debe tener un alto grado de paralelismo, sino también un alto grado de vectorización, la cuál es una de las propiedades principales de Intel.

Referencias

Caz, T. Del. (2016). Análisis teórico del contacto plasma superficie y sus aplicaciones industriales.

Eldred, C. (Pittsburgh S. C., & Michalakes, J. (National C. for A. R. (2008). WRF V3 Parallel Benchmark Page. Retrieved May 10, 2017, from http://www.mmm.ucar.edu/wrf/WG2/benchv3/

Fang, J., Varbanescu, A. L., & Sips, H. (2011). A comprehensive performance comparison of CUDA and OpenCL. *Proceedings of the International Conference on Parallel Processing*, *arXiv*(1), 216–225. https://doi.org/10.1109/ICPP.2011.45

Fraire, J. A., Ferreyra, A., & Marques, C. (2013). OpenCL Overview, Implementation, and Performance Comparison. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, *11*(1), 274–280. https://doi.org/10.1109/TLA.2013.6502816

Gualan-Saavedra, R. M., Solano-Quinde, L. D., & Bode, B. M. (2015). GPU Acceleration of the Horizontal Diffusion Method in the Weather Research and Forecasting (WRF) Model. *2015 Asia-Pacific Conference on Computer Aided System Engineering*, 284–289. https://doi.org/10.1109/APCASE.2015.57

Gualán, R., & Solano-Quinde, L. (2014). Análisis de rendimiento y profiling del modelo WRF en un clúster HPC. *Maskana*, 151–162.

IBTA. (2017). InfiniBand® Trade Association. Retrieved October 16, 2017, from http://www.infinibandta.org/

Intel ®. (n.d.). Intel® Trace Analyzer and Collector | Intel® Software. Retrieved June 16, 2017, from https://software.intel.com/en-us/intel-trace-analyzer

Intel Corporation. (2013). Intel ® Xeon Phi ™ Coprocessor System Software Developers Guide, 163.

Intel Corporation. (2014). OpenCL* Design and Programming Guide for the Intel® Xeon Phi™ Coprocessor | Intel® Software. Retrieved October 16, 2017, from https://software.intel.com/en-us/articles/opencl-design-and-programming-guide-for-the-intel-xeon-phi-coprocessor

Kaczmarek, O., Schmidt, C., Steinbrecher, P., & Wagner, M. (2014). Conjugate gradient solvers on Intel Xeon Phi and NVIDIA GPUs, *2*(2), 1–6. Retrieved from http://arxiv.org/abs/1411.4439

Karimi, K., Dickson, N. G., & Hamze, F. (2010). A Performance Comparison of CUDA and OpenCL. *ArXiv E-Prints*, *arXiv*(1), 1005.2581. https://doi.org/10.1109/ICPP.2011.45

Khronos OpenCL WorkingGroup. (2017). The OpenCL Specification.

Kirk, D. B., & Hwu, W. M. W. (2013). *Programming massively parallel processors: A hands-on approach, second edition*. *Programming Massively Parallel Processors: A Hands-on Approach, Second Edition*. https://doi.org/10.1016/B978-0-12-415992-1.00022-5

Komatsu, K., Sato, K., Arai, Y., Koyama, K., Takizawa, H., & Kobayashi, H. (2010). Evaluating Performance and Portability of OpenCL Programs.

Science And Technology, *2*(January), 52. Retrieved from http://vecpar.fe.up.pt/2010/workshops-iWAPT/Komatsu-Sato-Arai-Koyama-Takizawa-Kobayashi.pdf

Martín, J. P. Y. (2016). *Adaptación de Neurite a Intel Xeon Phi*. Universidad Politécnica de Madrid.

Michalakes, J., Loft, R., & Bourgeois, A. (2001). Performance-Portability and the Weather Research and Forecast Model. *HPC Asia 2001*.

Naffziger, S. (2011). Technology Impacts from the New Wave of Architectures for Media-rich Workloads. *2011 Symposium on VLSI Technology - Digest of Technical Papers*, *3*, 6–10.

NVIDIA. (2016). NVIDIA CUDA C, Programming Guide. *Compare A Journal Of Comparative Education*, (June). Retrieved from http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

NVIDIA Corporation. (2012). Kepler GK110. Retrieved from www.nvidia.com

NVIDIA Corporation. (2013). TESLA K40 GPU ACCELERATOR, (July), 16.

NVIDIA Corporation. (2014). Parallel Programming and Computing Platform | CUDA. Retrieved August 16, 2017, from http://www.nvidia.com/object/cuda_home_new.html

Scarpino, M. (2011). *OpenCL in Action*. Retrieved from papers2://publication/uuid/69731F95-2EF6-4DAA-93D3-3E101997D299

Shainer, G., Liu, T., Michalakes, J., Liberman, J., Layton, J., Celebioglu, O., … Cownie, D. (2009). Weather Research and Forecast (WRF) Model: Performance Analysis on Advanced Multi-core HPC Clusters. *The 10th LCI International Conference on High Performance Clustered Computing*, 1–14.

Skamarock, W. C., Klemp, J. B., Dudhi, J., Gill, D. O., Barker, D. M., Duda, M. G., … Powers, J. G. (2008). A Description of the Advanced Research WRF Version 3. *NCAR Tech*, 113. https://doi.org/10.5065/D68S4MVH

Solano-quinde, L., Gualan-saavedra, R., & Zuñiga-prieto, M. (2016). Multi-GPU implementation of the Horizontal Diffusion method of the Weather Research and Forecast Model. *In Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycore*. https://doi.org/10.1145/2883404.2883407

Su, C. L., Chen, P. Y., Lan, C. C., Huang, L. S., & Wu, K. H. (2012). Overview and comparison of OpenCL and CUDA technology for GPGPU. *IEEE Asia-Pacific Conference on Circuits and Systems, Proceedings, APCCAS*, 448–451. https://doi.org/10.1109/APCCAS.2012.6419068

Teodoro, G., Kurc, T., Kong, J., Cooper, L., & Saltz, J. (2014). Comparative Performance Analysis of Intel Xeon Phi, GPU, and CPU: A Case Study from Microscopy Image Analysis. *IEEE Transactions on Parallel and Distributed Systems : A Publication of the IEEE Computer Society*, *2014*, 1063–1072. https://doi.org/10.1109/IPDPS.2014.111

The Khronos Group Inc. (2009). Barrier. Retrieved August 27, 2017, from https://www.khronos.org/registry/OpenCL/sdk/1.0/docs/man/xhtml/barrier.html

The Open MPI Project. (2016). FAQ: Running MPI jobs. Retrieved May 10, 2017, from https://www.open-mpi.org//faq/?category=running#oversubscribing

Theses, M., & Shareef, B. (2015). Monte Carlo Simulations on Xeon Phi : Offload and Native Mode Monte Carlo Simulations on Xeon Phi : Offload and Native Mode by A thesis submitted to the Graduate College.

Toledo, C. M., & Barrientos, R. J. (2014). Búsqueda exhaustiva utilizando el coprocesador Intel Xeon Phi.

TOP 500 supercomputer. (2016). Retrieved from https://www.top500.org/