UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
PROGRAMA DE DOCTORADO EN INGENIERÍA

Énfasis en Ciencia de la Información y el Conocimiento

ACELERACIÓN DE LA MICROFÍSICA DE LLUVIA PARA "MODELO AVANZADO DE PREDICCIÓN DEL ESTADO DEL TIEMPO WRF" UTILIZANDO COMPUTACIÓN HETEROGÉNEA PARALELA

ESTEBAN DE JESÚS HERNÁNDEZ BARRAGÁN

Tesis de grado para optar el título de Doctor en Ingeniería

Director
CARLOS ENRIQUE MONTENEGRO, Ph.D

Co-Director
CARLOS JAIME BARRIOS HERNÁNDEZ, Ph.D

Bogotá, 2017

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
PROGRAMA DE DOCTORADO EN INGENIERÍA

Énfasis en Ciencia de la Información y el Conocimiento

ACELERACIÓN DE LA MICROFÍSICA DE LLUVIA PARA "MODELO AVANZADO DE PREDICCIÓN DEL ESTADO DEL TIEMPO WRF" UTILIZANDO COMPUTACIÓN HETEROGÉNEA PARALELA

ESTEBAN DE JESUS HERNÁNDEZ BARRAGÁN

Tesis de grado para optar el título de Doctor en Ingeniería

Director
CARLOS ENRIQUE MONTENEGRO, Ph.D

Co-Director
CARLOS JAIME BARRIOS HERNÁNDEZ, Ph.D

Bogotá, 2017

Comisión de Doctorado

Esta tesis, titulada “Aceleración de la microfísica de lluvia para el Modelo Avanzado de predicción del estado del tiempo WRF utilizando computación heterogénea paralela”, escrita por Esteban de Jesús Hernández Barragán, ha sido aprobada en cuanto a estilo y contenido intelectual.

Hemos leído esta tesis y la aprobamos,

_____

, Dr
Jurado 1

_____

, Dr
Jurado 2

_____

, Dr
Jurado 3

_____

Carlos Enrique Montenegro Marín, Dr
Director de tesis

_____

Carlos Jaime Barrios Hernández, Dr
Codirector de tesis

Fecha de la defensa: abril 04 de 2017

Agradecimientos

A José Daniel, Samuel Felipe y Azeneth, por su paciencia en este proceso

A Carlos Jaime Barrios Hernández, por haberme “adoptado” sin contraprestación

A Carlos Montenegro y Rodolfo Cáliz, por su apoyo en momentos inciertos

A Ofelia y Eulises, por enseñarme a nunca rendirme.

RESUMEN

El pronóstico del estado del tiempo, actualmente es guiado principalmente por modelos numéricos que simulan la dinámica atmosférica, permitiendo establecer las condiciones futuras basadas en las condiciones iniciales de las variables atmosféricas. Dado el número de variables meteorológicas, los complejos sistemas de ecuaciones no lineales y los métodos numéricos utilizados, es necesario dividir el área de análisis en celdas de un tamaño determinado según el cual se establece la resolución del modelo. Debido a que existen fenómenos cuya dinámica es determinada a escalas globales y otros a escalas regionales o locales, cada modelo ajusta su dinámica para una determinada escala, entendiendo que algunos fenómenos requieren una mayor resolución (un tamaño de celda más pequeño) para tener una mejor probabilidad de acierto.

Aumentar la resolución significa disminuir el espaciado temporal de los puntos de malla y por tanto aumentar el poder computacional requerido por la dinámica del modelo; al aumentar el poder computacional se involucran aspectos como la comunicación entre nodos, la velocidad de acceso a memoria local y remota, las operaciones de lecto-escritura para acceder a los datos de entrada y la distribución de las cargas de trabajo para que la solución se pueda ejecutar en un marco de tiempo factible (que el tiempo de simulación sea mucho menor que el horizonte de pronóstico).

Varios estudios han establecido que aumentar la resolución del modelo en 2 veces requiere el aumento del poder computacional en 10 veces (lo cual aumenta no solo la complejidad de la infraestructura computacional sino también los costos económicos asociados). En la última década el uso de aceleradores gráficos y vectoriales, al igual que el uso de FPGAs, han permitido que se logre aumentar el poder de cómputo en configuraciones mucho más simples, eficientes, económicas y con modelos de programación

consistentes; Este tipo de arquitecturas que mezclan procesadores y aceleradores ha sido denominada computación heterogénea.

En la presente investigación se exponen diferentes técnicas utilizadas para acelerar el modelo de pronóstico del estado del tiempo WRF en plataformas de cómputo heterogéneas, analizando temas de infraestructura, modelo de programación y estructura interna del mismo modelo.

Palabras claves: Computación Heterogénea, Computación paralela, Cómputo de alto desempeño, HPC, Pronóstico Numérico.

ABSTRACT

The current weather forecast use mainly numerical models to resolve the atmospheric dynamics, allow to establish future conditions based on initial conditions of atmospheric variables. Given the number of meteorological variables, complex systems of nonlinear equations and numerical methods used, it is necessary to divide the analysis region in grids of a fixed size, the size of grid determines the resolution of the model and depending on the resolution, the results the prognosis may have chance of success.

Increasing the resolution means decreasing the temporal spacing of the grid points and thus increase the computing power required by the model dynamics; by increasing computational power aspects such as intercom nodes, access to local and remote memory access to data and distributing them throughout the involved solution and message passing it is involved to coordinate the work of distributed processing involved: for these reasons is that several studies have established that increasing the resolution 2 times requires increased computing power by 10 times, with the associated economic costs. In the last decade, the use of graphics and vector accelerators, like the use of FPGAs have allowed large computing power is achieved with much simpler configurations and models consistent

programming, such architectures that combine processors and accelerators has been called heterogeneous computing.

In the research presented different techniques used to accelerate the forecast model WRF in heterogeneous computing platforms are exposed.

Keywords: HPC, Heterogeneous architectures, Parallel programming, Weather Forecast.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE ECUACIONES

LISTA DE TABLAS

INTRODUCCIÓN

La demanda de cálculos computaciones complejos ha obligado que los modelos de programación paralela y distribuida deban cumplir un rol de vital importancia; promoviendo así, que tanto los investigadores de software como hardware enfoquen sus esfuerzos en lograr mejores niveles de desempeño en procesamiento, con menores niveles de consumo de energía, generación de lenguajes con soporte para estructuras paralelas y nuevos tipos de redes de comunicación con muy baja latencia. En los 10 últimos años la programación paralela ha utilizado los aceleradores gráficos para desarrollar las nuevas aplicaciones científicas en el área denomina GPGPU (General-purpose computing on graphics processing units), creando así, la llamada "era de las GPUs" (Nickolls & Dally, 2010). Uno de los mayores obstáculos en el proceso de adopción de este tipo de tecnología ha sido la dependencia de fabricantes que imponen lenguajes propietarios incompatibles entre sí (Podobas, Brorsson, & Faxén, 2010), al igual que los modelos de programación existentes que en gran medida no han evolucionado en los últimos 30 años (Brodtkorb, Hagen, & Sætra, 2013).

El pronóstico numérico del estado del tiempo es el área de la meteorología que tradicionalmente ha requerido computación de alto desempeño para su funcionamiento (Kimura, 2002; Wang, Huang, Huang, & Goldberg, 2011) y que de manera temprana ha intentado adoptar modelos de programación basada en GPUs (John Michalakes & Vachharajani, 2008), este tipo de computación requiere de grandes esfuerzos para volver a codificar algunas secciones o la totalidad de los programas. El costo asociado a la reescritura de los componentes de software (tanto monetarios como de esfuerzo en recodificación) aún no ha sido estudiado a fondo y en algunos casos los proyectos son abandonados o considerados esfuerzos perdidos (Heinecke, 2013).

Al utilizar arquitecturas basadas en sistemas de CPU multinúcleo (multicore, en adelante se utilizará indistintamente cualquiera de los dos términos) se han de considerar elementos tanto de hardware, software y datos, para determinar la viabilidad o no de reescribir código existente. Dentro de estos elementos podemos encontrar aquellos relacionados con los propios datos (tipos y tamaños de los datos, interdependencia, subdivisión o replicación requerida), los relacionados con el software (estructura y patrones de acceso a datos, modelos de acceso a memoria, uso de cachés, modelos de distribución y coordinación de tareas, tipos de mensajes y el uso de hilos) y los relacionados con el hardware (niveles, tamaño y coherencia de los cachés, tamaño y velocidad de la memoria, velocidades y tipo buses de datos, número de núcleos (cores, en adelante se utilizará indistintamente cualquiera de los dos términos) y afinidad entre ellos, tipo de clúster y tecnología de interconexión de nodos. (Jang, Schaa, Mistry, & Kaeli, 2011; Jia, 2014; Karsavuran, Akbudak, & Aykanat, 2015; McCool, Reinders, & Robison, 2012) Una vez efectuado este análisis se puede emitir juiciosos sobre la viabilidad de realizar la conversión de los códigos existentes para ser ejecutados en arquitecturas masivamente paralelas (GPU especialmente), no sin antes considerar que estas plataformas tiene requerimientos propios para lograr los niveles de desempeño esperado y que pueden afectar cualquiera de los elementos analizados (Cabezas, Jordà, Gelado, Navarro, & Hwu, 2015; Gupta, Xiang, & Zhou, 2013; Jia, 2014). De estos elementos, se debe prestar especial atención a la transferencia de datos entre la memoria de procesador y la coalescencia de la misma (Fauzia & Sadayappan, 2015).

En la siguiente propuesta se investiga las características que un algoritmo o programa debe poseer para que pueda ser un buen candidato para acelerar su ejecución en arquitecturas masivamente paralelas (Jia, 2014; Marowka, 2012; Mcintosh-smith, 2012), esta aceleración implica elementos como aumentos en el tiempo de ejecución, mejor uso de la plataforma de cómputo y además eficiencia energética. Se analizan además las mejores estrategias

a utilizar para convertir una aplicación escrita exclusivamente para CPUs en una que sea soportada por soluciones con arquitecturas heterogéneas (GPU y Aceleradores Vectoriales), analizando especialmente los efectos que tiene el uso de buses diferenciales de datos, las técnicas de uso de cachés y las mejores prácticas de sincronización de datos en memorias con velocidades de acceso diferencial. Se utiliza como caso de validación la microfísica de la lluvia de momento simple WSM3 y la microfísica de doble momento Morrison 2 para el modelo de pronóstico WRF3 (Hong, Dudhia, & Chen, 2004; John Michalakes, 2013; John Michalakes & Vachharajani, 2008; Mielikainen, Huang, Huang, & Goldberg, 2012; Shainer & Liu, 2009).

La presente investigación ha sido dividida de la siguiente manera:

En el Capítulo 1. se presentan los objetivos, la hip y la manera cómo fue posible obtener los resultados finales, a fin de presentar un proceso que resultó en etapas iniciales muy ambicioso, pero que requirió un enfoque y especialización en algunos elementos, para que fuera viable en tiempo, capacidad y elementos disponibles.

En el Capítulo 2. se presentan las bases teóricas del modelamiento atmosférico, incluyendo elementos que resuelven la dinámica del modelo y elementos que requieren ser parametrizados. Presenta de igual manera las generalidades de las ecuaciones de gobierno de la dinámica y la manera como los modelos resuelven estas ecuaciones. En la parte final de este capítulo se presenta las generalidades del modelo WRF, su composición y herramientas.

En el Capítulo 3. se realiza una introducción a la computación heterogénea, sus elementos y diferencias con la computación homogénea; se presentan 2 plataformas de cómputo heterogéneo, junto con sus modelos de programación e implementación, sus ventajas y especificidades.

En el Capítulo 4. se presentan las bases teóricas y los modelos existentes para realizar análisis detallado de las medidas de desempeño en arquitecturas paralelas y la manera cómo puede determinar la ganancia que presenta una implementación paralela respecto a otra en términos de tiempo de ejecución.

En el Capítulo 5. se realiza un análisis detallado de la arquitectura de software del modelo WRF, el framework de desarrollo que sigue, las operaciones que involucran intercambio de mensajes, actualizaciones periódicas de las condiciones y los tipos de configuración e implicaciones que tiene el uso de mensajes MPI o hilos OpenMP.

En el Capítulo 6. se presentan los elementos dentro del modelo que afectan el desempeño en tiempo de ejecución, tanto utilizando modelos de memoria compartida como memoria distribuida, centrado en las etapas de pre procesamiento y simulación.

En el Capítulo 7. se presentan los resultados de los procesos de aceleración incorporados dentro del modelo, las diferentes técnicas utilizadas y los resultados comparativos al utilizar cada una de ellas. Se presentan las generalidades de las simulaciones realizadas tanto en memoria compartida como en memoria distribuida.

Finalmente, en el Capítulo 9. se presenta una propuesta de futuros trabajos o trabajo alternativo que se puede realizar a partir del presente proyecto y se plantean además desafíos para que dichos trabajos tengan un grado de aplicabilidad al entorno nacional.

CAPÍTULO 1. PRELIMINARES

En este capítulo se presenta el proceso seguido al desarrollar la investigación, partiendo desde la hipótesis, los objetivos, el contexto de la investigación, los aportes realizados y los elementos de reproductibilidad desarrollados.

1.1. HIPOTESIS

Utilizando computación heterogénea paralela para el pronóstico de la lluvia en el modelo WRF, es posible acelerar la ejecución del modelo logrando un speedup superior 10%, comparado con arquitecturas homogéneas basadas en solo CPU.

1.2. OBJETIVO GENERAL:

Realizar el perfilamiento y adaptación de las microfísicas de lluvia (WSM3, WSM6 y Morrison de doble momento) para aprovechar las ventajas de plataformas de memoria compartida, memoria distribuida y sistemas de muchos cores (manycores en adelante) de tal manera que se pueda acelerar la corrida del modelo WRF.

1.3. OBJETIVOS ESPECÍFICOS:

1. Determinar las rutinas del modelo que presentan mayor consumo y que puedan ser afinadas para lograr mejores niveles de desempeño.
2. Construir un estado del arte sobre los trabajos en modelamiento del estado del clima utilizando computación científica y de alto desempeño en Colombia en los últimos 5 años.
3. Determinar las rutinas que exponen mejores características para ser vectorizadas logrando un incremento en el desempeño del modelo

4. Generar una propuesta de aceleración que incluya elementos de uso eficiente de la energía, logrando incrementar una aceleración energéticamente eficiente.
5. Generar una propuesta de análisis y perfilamiento del modelo siguiendo una metodología aceptada dentro de la computación científica.

1.4. CAMBIOS EN EL DESARROLLO DE LA INVESTIGACIÓN

El Cambio más significativo que ocurrió, fue el relacionado con arquitecturas masivamente paralelas que serán analizadas para acelerar el modelo, cambiando de arquitecturas GPUs hacia arquitecturas Vectoriales (Intel Xeon Phi KNC y KNL, en adelante MIC), por las siguientes razones.

a) El esfuerzo de portar microfísicas en el modelo WRF es mucho más alto en las GPUS que en los MIC, dada la compatibilidad en los modelos de programación legados; la base de código existente del modelo escrita en lenguaje fortran y la disponibilidad de compiladores que acepten API de alto nivel que se asemejen a la API OpenMP (John Michalakes, 2013; John Michalakes & Gill, 2016; John Michalakes, Iacono, Berthiaume, & Gokhale, 2014).
b) La capacidad de los nuevos compiladores Intel de poder realizar autovectorización y de esta manera evitar el problema presentado con cada nueva versión del modelo (2 veces por año), dada la gran variedad de arquitecturas de procesamiento y la diversidad de compiladores que son soportados tanto de esquemas comerciales como de esquemas libres (Heirman et al., 2014; Intel, 2013; Jeong, Kim, Lee, & Myung, 2012; Rahman, 2013).
c) Al momento de iniciar este proyecto los únicos compiladores existentes que soportaban el estándar OpenACC(Wolfe, 2013) eran

comerciales y salvo algún soporte muy experimental de algunas librerías de código abierto (R Reyes & López-Rodríguez, 2012; Ruymán Reyes, López, Fumero, de Sande, & Sande, 2012) se requería el pago de licencias costosas para poder utilizarlos. En una publicación previa se pudo observar que, aunque el panorama de OpenACC presentaba notables ventajas, la comparativa de rendimiento computacional presentaba comportamientos muy similares con la API OpenMP, la cual si es soportada por compiladores de código abierto como GCC (Hernández, Gaviria, & Montenegro, 2014). A partir del año 2013 el proyecto GCC inició una programa interno para soportar la especificación OpenACC, pero solo hasta el año 2016 logró tener una versión estable con soporte para las 2.0 del estándar como fue expuesto en la conferencia supercomputing 2016 (en adelante SC'16) y en la conferencia de usuarios de OpenMP año 2015 (Beyer, 2015; Foundation, 2017; Ilya & Yukhin, n.d.). Es de anotar que solo hasta la conferencia SC'16 Nvidia lanzó la versión comunitaria de su compilador con soporte para OpenACC.

1.5. CONTEXTO DE LA INVESTIGACIÓN

La presente investigación aunque utiliza un modelo meteorológico como elemento de análisis, no intentan determinar la calidad de los pronósticos generado por el mismo, o la capacidad de las ecuaciones para simular el estado actual de la atmósfera, por lo tanto los aspectos relacionados al modelo en sí solo se estudian desde la óptica de la computación y más específicamente en aquellos elementos relacionados con el uso y aprovechamiento de los recursos sobre arquitecturas heterogéneas, para lo cual se examinan los modelos programación involucrados (memoria compartida y memoria distribuida) utilizando diferentes tipos de infraestructuras. El enfoque principal se hace sobre los elementos de mensajería y modelos de distribución de hilos en sistemas de muchos cores, como se puede observar en la Figura 1. Contexto de la investigación

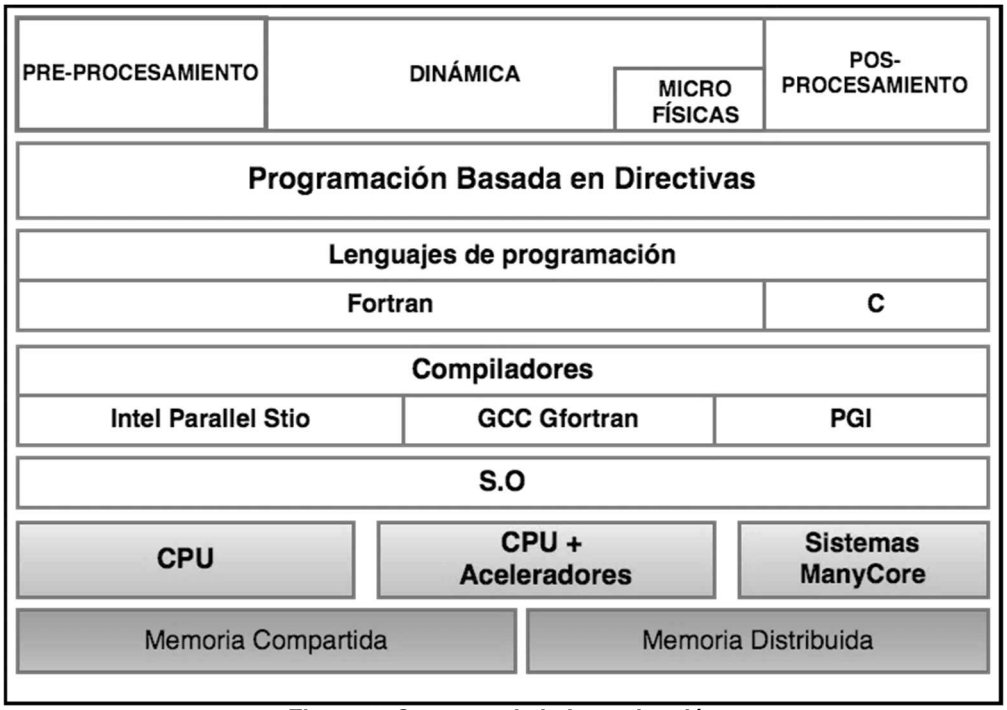


Figura 1. Contexto de la investigación
Fuente: Desarrollo propio

1.6. DIVULGACIÓN

Se presentó y aceptó el artículo "Parallel programming languages on heterogeneous architectures using openmpc, ompss, openacc and OpenMP" la revista Tecnura y fue publicado en la edición especial de doctorado 2014.

Se presentó la ponencia "Aceleración Enérgicamente Eficiente de Aplicaciones Científicas de Gran Escala Sobre Arquitecturas Heterogéneas" en conjunto con personal de grupo de supercomputación sc3 de la UIS y de procesamiento distribuido y paralelo de la Universidad Federal de Rio Grande Do Soul, en el Congreso Nacional de Ingeniería de Sistema e Informática, en febrero de 2016.

Se presentó el artículo enerGyPU and enerGyPhi Monitor for Power Consumption and Performance Evaluationon Nvidia Tesla GPU and Intel Xeon Phi, al 16th IEEE/ACM International Symposium on Clúster, Cloud and Grid Computing, junto con el Centro de supercomputación y calculo científico SC3 de la UIS y fue aceptado para ser defendido dentro del workshop de Fostering Collaboration in Latin America to address the Emerging Challenges by the growth of HPC for Academia and Industry en mayo de 2016.

Se presentó y defendió la propuesta de investigación “Aceleración de la microfísica de lluvia para el modelo de pronóstico del tiempo WRF utilizando computación heterogénea paralela” ante un foro nacional de doctores en el evento academia nVIDIA CUDA-SC3UIS de esa manera se que obtuvo retroalimentación valiosa para un reenfoque del proyecto y ajuste de algunos objetivos.

Se presentó el artículo y poster “Energetically Efficient Acceleration EEA-Aware For Scientific Applications of Large-Scale On Heterogeneous Architectures”, en la conferencia “The International Conference for High Performance Computing, Networking, Storage and Analysis 2016”, realizada entre el 14 y el 16 de noviembre de 2016.

Se presentó el artículo “The performance impact of Thread Affinity and memory distribution on Model WRF. A deep analyzing using Intel KNC and KNL processors”, a la revista “International Journal of Grid and High Performance Computing (IJGHPC)”

1.7. GENERACIÓN DE CONOCIMIENTO LOCAL

Se realizó el workshop “Parallel Programming and Optimization for Intel architectures”, en el 6 y 7 de septiembre de 2016, auspiciado por “Center for Scientific Computing of Universidade Stadual Paulista UNESP e Intel Brazil” y la universidad distrital Francisco José de Caldas.

Se aplicó los resultados de investigación dentro del proyecto “Sistema de Información para la Gestión, Monitoreo y Alertas Tempranas del Municipio de Ciénaga “**www.sigematcienaga.co**“, entre noviembre de 2015 y enero de 2016.

Se apoyó la adquisición de una plataforma de supercomputo con aceleradores vectoriales Intel Xeon Phi KNC 3120A, para el Centro de Cómputo de alto desempeño de la Universidad Distrital www.cecad.udistrital.edu.co

Se apoyó la adquisición e instalación del cluster equipado con procesadores Intel Xeon Phi KNL x200, con 4 nodos de procesamiento paralelo para el Centro de cómputo de alto desempeño de la Universidad Distrital Francisco José de Caldas www.cecad.udistrital.edu.co

Se apoyó la coordinación del evento Workshop Distrital de Supercomputación '17 que se realizará el 19 y 20 de septiembre del presente año con la presencia de la industria, la academia y la comunidad de desarrollo de software paralelo incluyendo OpenACC, Cray-Chapel y OpenMP.

1.8. REPRODUCTIBILIDAD

Un factor dentro del desarrollo del presente proyecto es la imposibilidad de reproducir los resultados presentados por diferentes autores al reportar resultados del performance o la escasa documentación sobre el tipo de prueba, la parametrización utilizada, los datos de entrada del modelo y las banderas utilizadas al momento de la compilación (J Michalakes, Dudhia, & Gill, 2004; Mielikainen et al., 2012; Ponder, Romanenko, & Snytnikov, 2014). Los argumentos que comúnmente se esgrimen es la financiación bajo recursos privados, federales o con restricción en los términos de confidencialidad, con lo cual se hace prácticamente imposible verificar

resultados o intentar aplicar técnicas diferenciales para comprobar la validez de los experimentos llevados a cabo.

Por estas razones se han propuesto una serie de iniciativas para permitir que mediante investigación reproducible y colaborativa se puedan mejorar los actuales modelos, evitando repetir esfuerzos y compartiendo los resultados sin restricciones (Hacker, Exby, & Gill, 2016).

Por las razones expuestas anteriormente, todas las parametrizaciones, resultados de perfilamiento, datos iniciales, logs y demás resultados obtenidos se dejan disponibles bajo la denominación "*Hernandez, Esteban (2017), "Dataset of WRF profiling", Mendeley Data, v2 http://dx.doi.org/10.17632/fjgjyj55wm.2*"

1.9. CONCLUSIÓN

El desarrollo de la presente investigación exigió un trabajo multidisciplinar debido al conjunto de elementos que intervienen en la modelación atmosférica y en la computación de alto desempeño de programación paralela y de las técnicas de perfilamiento de aplicaciones. El acelerar una aplicación específica implica conocer el modelado de los fenómenos involucrados, el conocimiento de la arquitectura de la aplicación, la determinación de los puntos calientes y cuellos de botella presentados. La experimentación con técnicas de aceleración a nivel de procesador, de nodo con memoria compartida y de soluciones de memoria distribuida, para indicar la verdadera capacidad de aceleración que la aplicación expone y que las herramientas pueden explotar.

En el siguiente capítulo se indica los parámetros, los procedimientos, las herramientas y los dataset utilizados con el fin que la presente investigación pueda ser reproducible para la comunidad investigativa.

CAPÍTULO 2. FUNDAMENTACIÓN TEÓRICA

En este capítulo se presentará la fundamentación teórica necesaria para entender los fenómenos meteorológicos que se tienen en cuenta en el modelamiento atmosférico, las ecuaciones de balance que están involucradas, los elementos que se parametrizan en el modelo y las generalidades de los modelos numéricos.

La simulación de sistemas físicos y en especial de modelos de pronóstico del estado del tiempo son una tarea que requiere un modelo numérico que represente el sistema físico, un conjunto de parametrizaciones de los mismos e infraestructura de cómputo especializada con gran poder de procesamiento (El-Askary et al., 2012). Las soluciones de Cómputo de Alto Desempeño (en adelante HPC) poseen un alto número de procesadores de propósito general (en adelante CPUs), redes de baja latencia (N. Jain, Bhatele, Ni, Wright, & Kale, 2014; Koibuchi, 2013) y librerías de paso de mensajes en ambientes de clusters; Estas soluciones por lo general presentan arquitecturas altamente acopladas para lograr el nivel de procesamiento deseado, sin embargo el costo de implementación resulta inalcanzable para muchas de las organizaciones de países que no pertenecen a las economías más prominentes (Snell, 2013), aumentando la brecha de desarrollo tecnológico, especialmente para economías como la de países en vía de desarrollo que no cuentan con un presupuesto significativo en investigación y desarrollo (Witte & Mannon, 2010). En la última década nuevas arquitecturas de HPC han emergido permitiendo que un nuevo tipo de “aceleradores” (Namyst, 2012) fueran ajustados para lograr cumplir los requerimientos de la computación científica, constituyendo un área alternativa denominada GPGPU (General-purpose computing on graphics processing units) la cual no solo resulta una solución económica sino que por su naturaleza paralela resulta ideal para procesos de simulación de modelos físicos (Keckler, Dally, Khailany, Garland, & Glasco, 2011).

Para realizar una simulación de un sistema físico como lo es un sistema atmosférico es indispensable usar soluciones de HPC y se considera incluso que su origen se debió a la necesidad de resolver un pronóstico de tipo atmosférico (Bougeault, 2008; Holton, 2004; Jacobson, 2005; Lynch, 2008), por tanto en los siguientes tópicos se intentará describir las necesidades de los modelos meteorológicos y especialmente los modelos numéricos, para luego describir la manera como la computación heterogénea paralela permite que estos se ejecuten de una manera eficaz, logrando ser acelerados en su ejecución mediante su uso.

2.1. ELEMENTOS DE LOS MODELOS METEOROLÓGICOS

En la construcción de un modelo de pronóstico meteorológico, se definen varios elementos que permiten determinar la calidad de los pronósticos generados, el tipo de parametrización e interacción de los fenómenos físicos incluidos, el área de cobertura que el modelo soportará (Kimura, 2002; Montoya, 2008; T. T. Warner, 2010) y la manera como se incorporarán aquellos fenómenos que sean de una escala menor a la cubierta por el modelo. Los elementos que afectan la calidad del modelo se enumeran a continuación:

1. La resolución del modelo
2. El tipo de coordenadas que utilizan
3. Las ecuaciones fundamentales que gobiernan del modelo
4. El tipo de parametrización involucrada
5. La solución numérica
6. Las condiciones iniciales y de frontera.

Dada la complejidad de las interacciones atmosféricas (Figura 1) que gobiernan la dinámica del modelo que son descritas por ecuaciones las fundamentales de gobierno, el área analizada (escala planetaria, regional o local) se divide en secciones de tamaño determinado según el tipo de

proyección cartográfica elegida en donde se resuelven las ecuaciones de gobierno. El tamaño de la división es denominado la resolución del modelo, de tal manera que una mayor resolución indica que el tamaño de la sección es más pequeño y por tanto el número de ecuaciones para un área determinada es mayor (T. Warner, 2011)

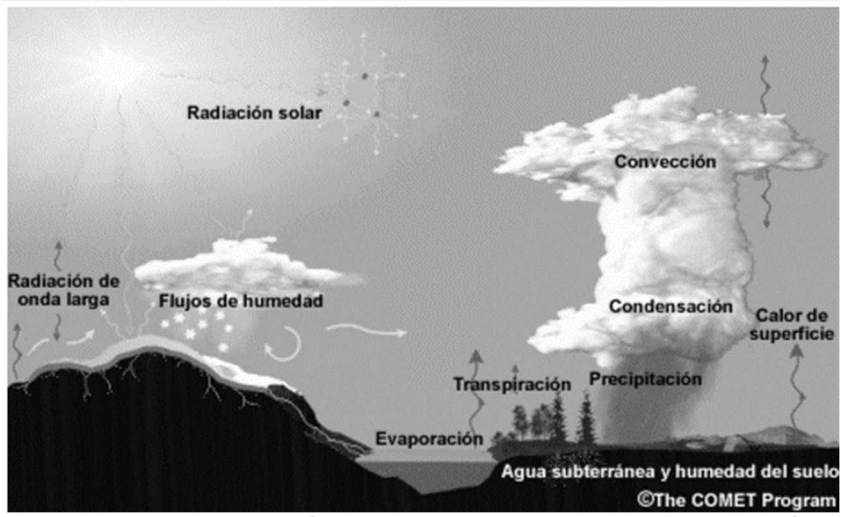


Figura 2. Procesos atmosféricos que afectan un modelo de pronóstico. Fuente: the comet program http://www.comet.ucar.edu/

2.1.1. Los modelos numéricos de pronóstico

Los modelos numéricos de pronóstico atmosférico (NWP en adelante) son útiles para la predicción del tiempo a corto y mediano plazo porque basan su predicción en un modelo matemático que representa el estado actual de las variables atmosféricas y mediante el uso intensivo de computación resolver las mismas de manera numérica para determinar las condiciones futuras, resolviendo la sección dinámica del esquema: la dinámica del modelo es descrita por un conjunto de ecuaciones de dinámica de fluidos que tomando unas condiciones iniciales para cada punto de la malla del área de análisis.

Hoy día, las salidas de los modelos atmosféricos de predicción numérica son las herramientas más comúnmente utilizadas por los pronosticadores en los servicios y agencias meteorológicas de orden nacional o internacional. Conocer las características físicas y dinámicas en las que se basan estos modelos, para evaluar su comportamiento, rendimiento y fiabilidad permite que se puedan proponer nuevos esquemas que pueden afectar la calidad del pronóstico y los recursos demandados para su ejecución.

2.1.2. Modelamiento Numérico

Es el proceso mediante el cual se obtiene un pronóstico objetivo del estado futuro de la atmósfera resolviendo un grupo de ecuaciones que describen la evolución de un conjunto de variables (temperatura, velocidad y dirección del viento, humedad, presión) que a su vez define el estado de la atmósfera en un determinado momento del tiempo.

El proceso inicia con el análisis del estado actual de la atmósfera utilizando un pronóstico a corto plazo y adicionando las observaciones disponibles (asimilación de datos) con el fin de lograr una mejor descripción del verdadero estado actual de la atmósfera. A partir de allí se ejecuta un modelo en computador para producir el pronóstico.

Todos los modelos numéricos de la atmósfera utilizan el mismo grupo de ecuaciones que gobiernan la atmósfera, las cuales son descritas en términos no-matemáticos (de advección, de conservación de la masa o de continuidad, hidrostática, termodinámica, de estado, de vapor de agua). Los modelos numéricos difieren entre sí en las aproximaciones y suposiciones hechas en la aplicación de estas ecuaciones y en como ellas son resueltas para la representación de los diferentes procesos físicos.

2.1.3. Componentes de un modelo numérico

Un NWP emplea un conjunto de ecuaciones que describen el flujo de un fluido, combinado con la parametrización de otros procesos, aplicado a un dominio específico e integrado mediante un método numérico. La dinámica del modelo toma las condiciones iniciales y las condiciones de frontera y las hace evolucionar en el tiempo mediante códigos computaciones, con el fin de producir un pronóstico objetivo (Kimura, 2002; Knievel, 2006; Lynch, 2008; T. Warner, 2011; T. T. Warner, 2010)

Los elementos básicos de los modelos numéricos son:

- Las ecuaciones de gobierno
- Los métodos numéricos involucrados
- Las parametrizaciones físicas
- Los dominios utilizados
- La resolución numérica
- Condiciones Iniciales y de frontera

Todos estos elementos se juntan dentro de un marco general en donde unos datos procesados de manera previa (pre-procesamiento), permiten determinar las condiciones iniciales y de frontera de los dominios elegidos, con la resolución establecida, asimilando datos de entrada (asimilación) que permiten mejorar la precisión del pronóstico y que una vez resuelta la dinámica del sistema, utilizando las parametrizaciones físicas (y las microfísicas) permiten obtener un conjunto de resultados que permiten al pronosticador generar un pronóstico con un mayor grado de certeza posible (Figura 3)

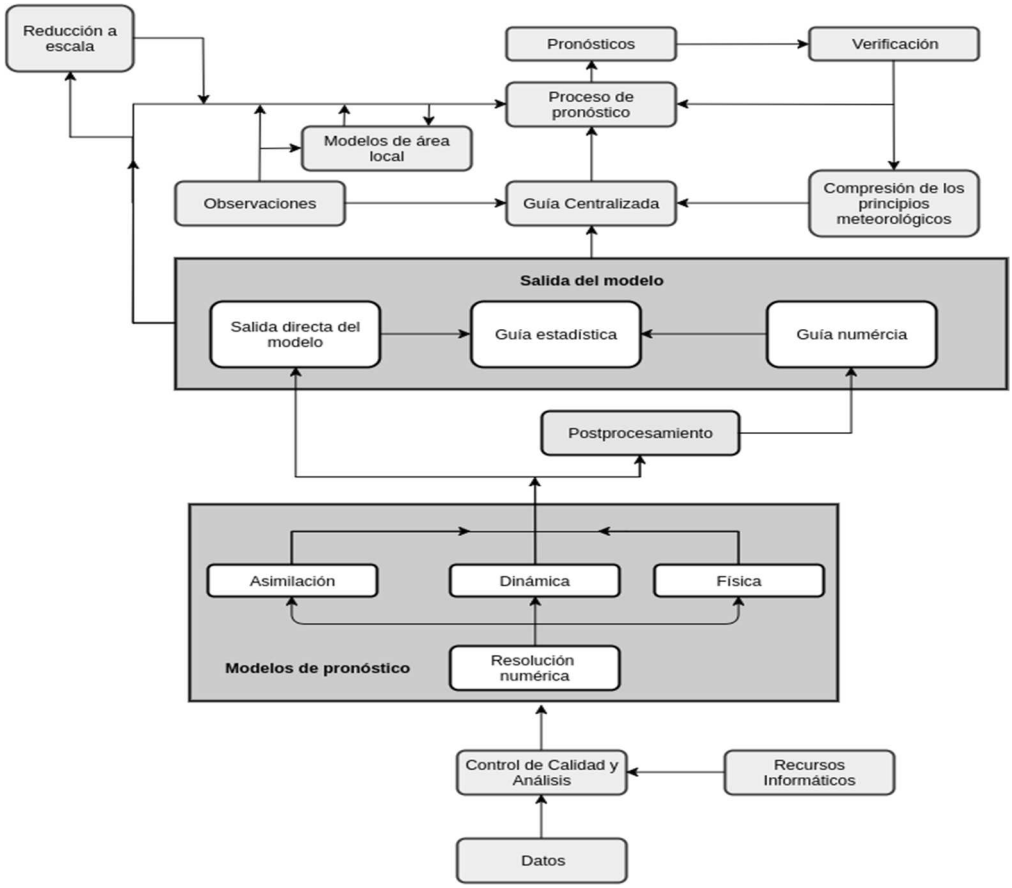


Figura 3. Elementos de un Modelo Numérico de Pronóstico (NWP). Fuente: The comet program http://www.comet.ucar.edu/

El primer paso es la asimilación de los datos que usualmente se realiza en forma secuencial. El modelo organiza y propaga la información desde observaciones previas. La información de nuevas observaciones es usada para modificar el estado del modelo y hacerlo lo más consistente posible. Luego, viene la parametrización de los diferentes procesos físicos de la atmósfera, tales como la radiación, la convección y los intercambios en los bordes o límites del modelo. Ahora, se necesita hacer que la dinámica del modelo evolucione en el tiempo para lo cual se integra numéricamente para encontrar las condiciones finales de las variables atmosféricas analizadas. Algunos fenómenos no pueden ser cubiertos por la resolución del modelo y requieren parametrizarse con el objetivo de ofrecer realismo al pronóstico realizado (Figura No 3)

Figura 4. Parámetros y procesos parametrizados.
Fuente the comet program http://www.comet.ucar.edu/

Los esquemas de parametrización pueden hacer suposiciones de acuerdo a restricciones computacionales involucradas en el proceso. El desarrollo de estos esquemas involucra una estrecha cooperación entre los diferentes grupos de investigación; los grupos de procesos atmosféricos ejecutan estudios de observaciones y modelamiento independientes que ayudan a entender los procesos físicos que ocurren en la atmósfera. Los elementos que se pueden parametrizar para un NWP se pueden observar en la figura No 3.

2.1.4. Ecuaciones fundamentales de los modelos

La atmósfera, es considerada un fluido y por tanto la dinámica que se presenta en ella se mantiene gobernada por 3 ecuaciones fundamentales:

- Conservación del momento
- Conservación de la energía
- Conservación de la masa.

También aparecen algunas fuerzas consideradas fundamentales:

- Fuerza del gradiente de presión
- Fuerza de gravedad
- Fuerza de tensión viscosa

y otras fuerzas consideradas aparentes

- Fuerza centrípeta
- Gravedad efectiva
- Equilibrio hidrostático
- Fuerza de Coriolis

A continuación, se presenta un listado simplificado de las principales ecuaciones que son derivadas de las leyes de conservación de momento, masa, energía y humedad:

Ecuaciones para el pronóstico del viento

$$\underbrace{\frac{\partial u}{\partial t}}_{\text{Cambio en el tiempo del viento de oeste a este}} = \underbrace{-u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - g\frac{\partial z}{\partial x}}_{\text{Advección horizontal del viento del oeste a este (momento)}} + F_x \underbrace{\overbrace{-\ \omega\ \frac{\partial u}{\partial p}}^{\text{Desviaciones del balance geostrófico del viento de sur a norte}}}_{\text{Advección vertical del viento del oeste a este (momento)}} + \underbrace{fv}_{\text{Fricción superficial y mezcla turbulenta que}}$$

Ecuación (1)

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \omega\frac{\partial v}{\partial p} + fu - g\frac{\partial z}{\partial y} + F_y$$

Ecuación (2)

Ecuación de continuidad

$$\underbrace{\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}}_{\text{Divergencia horizontal}} + \underbrace{\frac{\partial \omega}{\partial p}}_{\text{Divergencia vertical}} = 0$$

Ecuación (3)

Ecuación de pronóstico de temperatura

$$\underbrace{\frac{\partial T}{\partial t}}_{\text{Cambio en el tiempo de la temperatura}} = \underbrace{-u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}}_{\text{Advección horizontal de la temperatura}} - \underbrace{\omega\left(\frac{\partial T}{\partial p} - \frac{RT}{c_P p}\right)}_{\text{Diferencia entre la advección vertical de la temperatura y los procesos adiabáticos}} + \overbrace{\frac{H}{c_p}}^{\text{Otros procesos (radicación, mezcla, condensación)}}$$

Ecuación (4)

Ecuación de pronóstico de la humedad

$$\underbrace{\frac{\partial q}{\partial t}}_{\text{Cambio en el tiempo de la humedad}} = \underbrace{-u\frac{\partial q}{\partial x} - v\frac{\partial q}{\partial y}}_{\text{Advección horizontal de humedad}} - \underbrace{\omega\frac{\partial q}{\partial p}}_{\text{Advección vertical de humedad}} + \overbrace{E}^{\text{Evaporación y sublimación}} - \underbrace{P}_{\text{Condensación (Precipitación)}}$$

Ecuación (5)

Ecuación hidrostática

$$\underbrace{\frac{\partial z}{\partial p}}_{\text{Diferencia de altura entre las superficies isobáricas superior e inferior}} = -\underbrace{\frac{RT}{pg}}_{\text{Temperatura media de la capa}} \qquad \text{Ecuación (6)}$$

Todas las ecuaciones se presentan con coordenadas de presión x, y, p. La fuerza del gradiente de presión se expresa como el gradiente vertical z.

El estudio detallado de las ecuaciones fundamentales se explora en profundidad en un curso de dinámica atmosférica o en textos de modelamiento atmosférico como (G.I. Marchuk, n.d.; Holton, 2004; Jacobson, 2005; Mak, 2011; Montoya, 2008).

2.1.5. Resolución de los modelos de pronóstico

La resolución de los modelos está ligado a la escala de análisis de los fenómenos descritos (Jacobson, 2005), y se pueden clasificar en:

- Microescala de 2mm a 2 Kms
- Meso-escala de 2Kms a 2000 Kms
- Escala sinóptica de 500 a 10000 Kms
- Escala Planetaria mayor a 10000 Kms
-

Los modelos de microescala normalmente se han utilizado para resolver situaciones regionales como tormentas, vendavales, trombas y tornados; sin embargo dada su bajo alcance se prefiere utilizar modelos de mesoescala (también llamado regionales) con una alta resolución (50kms x 50kms o más) realizando ajustes con información de estaciones meteorológicas, aspectos

topográficos y demás (Ruiz Murcia, José Franklin Instituto de Hidrologia, 2010).

Una forma de aumentar la resolución de un modelo sin que las exigencias computaciones aumenten significativamente es limitar la solución a una región restringida (conocida como "dominio"), por ejemplo, alguna región o departamento de Colombia. Al ser una región reducida, el número de cálculos se puede mantener reducido también, pues existe una relación directa entre la cantidad de cómputo requerido y la resolución del modelo, tanto que duplicar la resolución del modelo equivale a aumentar 10 veces el recurso computacional (Posey, 2013).

Algunos de los modelos atmosféricos regionales más populares son:

- PSU/NCAR Mesoscale Model version 5 (MM5)
- ICTP Regional Climate Model (RegCM)
- Weather Research and Forecasting Model (WRF)
- Workstation Eta model
- Cosmo Climate Model
- Brazilian Regional Atmospheric Modeling System (BRAMS)

En especial el WRF (The Weather Research & Forecasting Model) (Kuo, Klemp, & Michalakes, 2004) ha sido diseñado tanto para investigación climática como para trabajo operativo de predicción del estado del tiempo además es uno de los modelos seleccionados por el IDEAM para realizar los pronósticos operativos para Colombia (Ruiz Murcia, José Franklin Instituto de Hidrologia, 2010)

2.2. EL MODELO WRF

El modelo de pronóstico WRF fue desarrollado por un conjunto de instituciones como la NCAR (National Center for Atmospheric Research), la

NOOA (National Oceanic and Atmospheric Administration), La Air Force Weather Agency (AFWA), Naval Research Laboratory, la universidad de Oklahoma, y la agencia federal de administración de la aviación (FAA). Desde sus inicios fue de carácter abierto, liberado bajo los términos de uso público y es sumamente flexible y tiene multitud de aplicaciones. No es únicamente una herramienta para las predicciones meteorológicas, también puede utilizarse para realizar estudios de pronóstico retrospectivo, estudios sobre la calidad del aire, modelización climática regional y previsión meteorológica a corto plazo. Los componentes del modelo WRF se presentan a continuación:

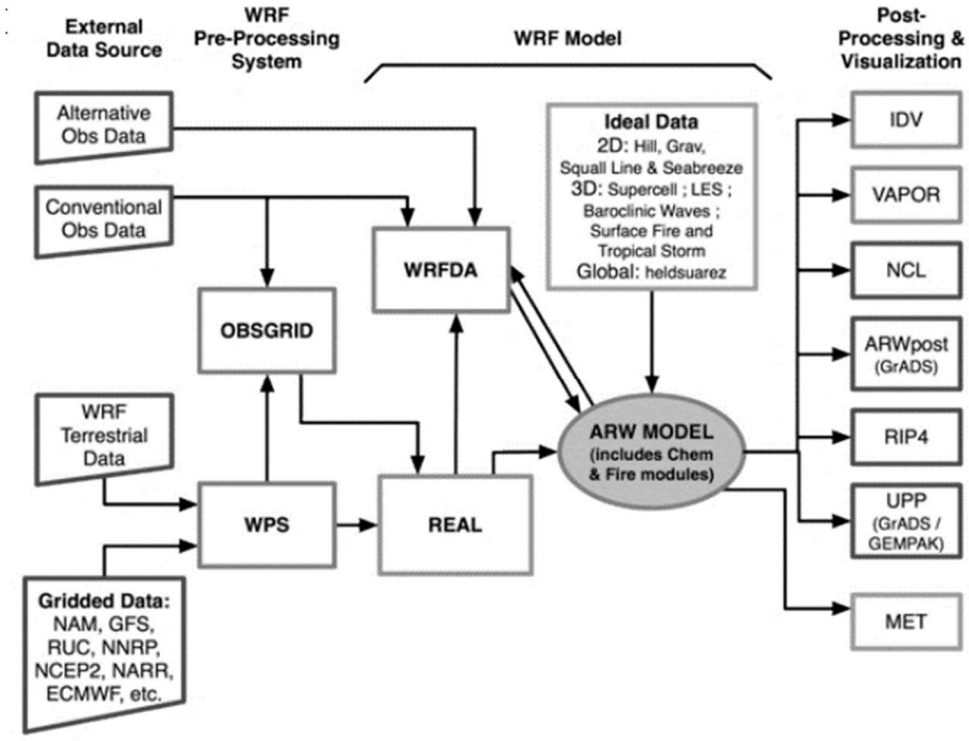


Figura 5. Componentes del modelo WRF.
Fuente: NCAR disponible en http://www2.mmm.ucar.edu/wrf/users/docs/user_guide_V3.1/users_guide_chap1.htm

2.2.1. Componentes del modelo WRF

Los modelos constan de 3 partes principales: Las herramientas de pre-procesamiento, las herramientas de simulación, las herramientas de post-procesamiento, como se observa en la Figura 6. Componentes generales del modelo WRF, cada una de estas se describe a continuación.

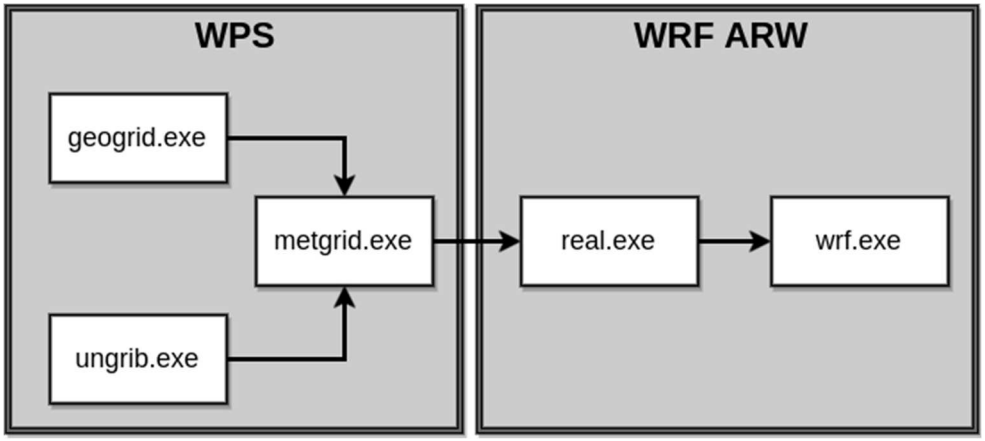


Figura 6. Componentes generales del modelo WRF
Fuente: desarrollo propio adaptado de WRF Tutorial

2.2.1.1. HERRAMIENTAS DE PREPROCESAMIENTO

WPS: WRF Preprocessing System: Permite definir la malla de WRF, generar mapas, tomar pronósticos de otros modelos, generar información de elevación, de la conformación del terreno y finalmente interpolar dichos datos dentro de las celdas de WRF. En su orden se conforma de 3 herramientas, geogrid.exe, ungrib.exe y metgrid.exe:

2.2.1.2. HERRAMIENTAS DE ASIMILACIÓN

WRF-Var: Este programa se utiliza para incorporar observaciones dentro de los datos interpolados por WPS. También permite actualizar las condiciones iniciales del modelo WRF cuando este corre en modos cíclicos.

OBSGRID: Objective Analysis, esta herramienta fue construida para mejorar el análisis meteorológico de primera aproximación al permitir ingresar datos de las observaciones. Obsgrid se ejecuta después de la herramienta metgrid.exe e incorpora datos de observaciones que estén en formato Little R.

2.2.1.3. HERRAMIENTAS DE SIMULACIÓN

ARW Model: Advanced Research WRF (ARW) dynamical solver: Encargado de las simulaciones atmosféricas, compuesto por algunos programas que permiten la simulación para situaciones ideales (investigación) o de datos reales (operacionales) y los programas de integración numérica. Es la pieza clave del sistema de modelamiento, dentro de los muchos elementos de este solver se incluye la completa opción de físicas de superficie terrestre, capa límite planetaria y atmosférica, radiación de la superficie y las microfísicas. Dentro de las microfísicas que soportan el fenómenos de lluvias tenemos la Morrison (opción 10) y la Kain-Fritsch (opción 1) (Cortes, 2012; Dudhia, n.d.)

2.2.1.4. HERRAMIENTAS DE POSTPROCESAMIENTO

NCL, ARWPost: Una vez generadas las salidas del modelo (wrfoutput), se debe crear mapas, procesos de análisis, animaciones y demás elementos necesarios. Estas dos herramientas permiten crear los tipos de gráficos necesarios basados en la información de la salida de la simulación del modelo. (Wei Wang, Cindy Bruyère, 2016)

2.3. CONCLUSIÓN

El modelamiento atmosférico requiere del entendimiento de los fenómenos involucrados, entender las ecuaciones que los describen, la manera como los modelos numéricos los resuelven en su parte dinámica y como se incorporan otros fenómenos no contemplados dentro de la dinámica, por último, entender la escala en la cual se resuelven los fenómenos nos permiten decidir qué modelo y que fases del mismo requieren ser utilizadas para que los pronósticos sean realistas y eficientes.

En el siguiente capítulo se presenta una visión general de la computación heterogénea paralela que en la actualidad ofrece los mejores niveles de desempeño para los modelos numéricos

CAPÍTULO 3. INTRODUCCIÓN A COMPUTACIÓN HETEROGÉNEA PARALELA

En este capítulo se hará una introducción al uso de la computación heterogénea paralela, como el tipo de arquitectura de procesadores que se usa en la presente investigación y que además explorar dos plataformas tradicionales existentes en el mercado y que fueron analizadas respecto a su arquitectura de software, modelos de programación y capacidades ofrecidas.

Cuando un sistema de cómputo combina diferente elementos de procesamiento que no presentan la misma estructura, método o arquitectura de procesamiento o método de almacenamiento de información, se considera heterogéneo. En un primer momento los sistemas de cómputo que combinaban diferentes arquitectura de procesadores fueron considerados heterogéneos, sin embargo el concepto ha cambiado de significado con la aparición de los sistemas multicore y los clústeres para procesamiento paralelo (Targhetta, 2008). Inicialmente se consideraba la heterogeneidad como la existencia de múltiples equipos con diferentes tipos de procesadores, diferentes velocidades conectadas por una red de transmisión (Marina & Antonio, 2004); más tarde apareció el concepto de Computación Heterogénea Paralela refiriéndose a aquella computación que se realiza en equipos de arquitectura de procesador similar pero con diferentes niveles de desempeño, mediante el uso de librerías de interconexión (Targhetta, 2008). En la actualidad se considera, que la heterogeneidad está determinada por los múltiples tipos de cores (CPU, GPUs, FPGA, Micro arquitecturas), elementos diferenciales de interconexión (Ethernet, Infiniband, Torus), tipo de memoria y capacidad y modelos de programación en paralelo (Ffoulkes & By, 2016; Mcintosh-smith, 2012).

3.1. UNIDADES DE PROCESAMIENTO GRÁFICO

Dentro de estos modelos heterogéneos se establece niveles de procesamiento diferencial en un mismo nodo, logrando especializar el tipo de trabajo realizado por cada tipo de procesador: los procesadores CPU se encargan del trabajo serial de manera muy eficiente y los aceleradores realizan el procesamiento masivamente paralelo. Estos últimos ocultan la lentitud de sus cores aumentando la cantidad de estos disponibles por procesador como se observa en la Figura 7.

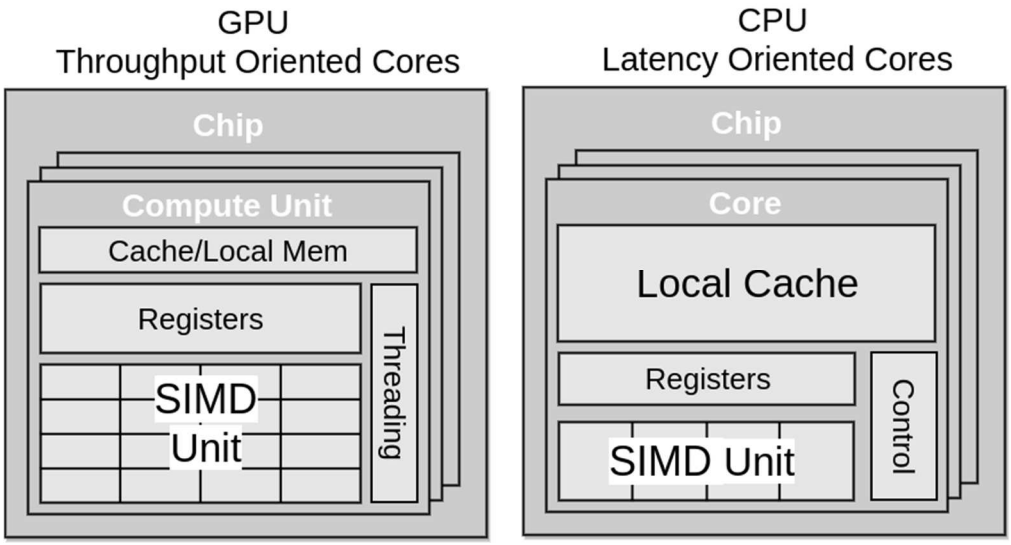


Figura 7. Comparación CPU vs GPU.
Fuente: Desarrollo propio adaptado de GPU Teaching Kit Computing

Las GPUs, presentan una arquitectura que se ha denominado Masivamente Paralela en esquemas multicore (D B Kirk & Hwu, 2012) que está especialmente diseñada para lograr un alto grado de paralelismo. De esta manera el diseño de las GPUS difiere de las CPUs y por tanto su utilización también tiene una orientación distinta. Las CPUs se consideran eficiente para procesamiento serie, mientras que las GPUs son eficientes cuando se logra un alto grado de paralelismo (ver Figura 5)

El modelo de ejecución dentro de las GPUs también cambia. En el caso específico de las GPUs Nvidia® con tecnología CUDA® los trabajos se lanzan

mediante kernels y se puede lograr 3 niveles de paralelismo siguiendo el modelo SIMT (Simple Instruction Multiple Thread) (Chandra, Menon, Dagum, & Kohr, 2000) como se muestra en la Figura 8. Niveles de paralelismo presentes en un GPU.

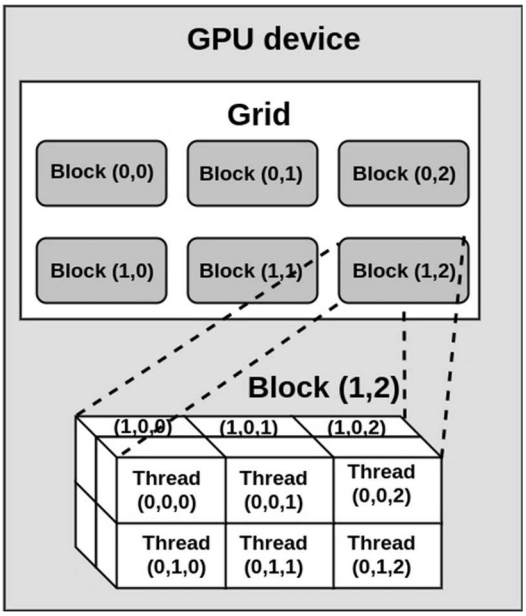


Figura 8. Niveles de paralelismo presentes en un GPU. Fuente GPU Teaching Kit Computing disponible en https://developer.nvidia.com/teaching-kit

La ejecución de un programa en arquitecturas heterogéneas, contempla las transferencias entre host y los dispositivos acelerados (GPU, Procesadores vectoriales, FPGA). El procesamiento inicia en el host equipado con una CPU, el cual se encarga de reservar la memoria requerida para inicializar las variables necesarias, luego se llega a la sección del código que presenta mayor nivel de paralelismo y que ha sido marcada previamente para ser ejecutada en un acelerador; en este punto, se requiere que los datos necesarios para procesar sean copiados de la memoria del host a la memoria del acelerador utilizando el bus de interconexión de dispositivos que actualmente por estándar es el bus PCI. El dispositivo requiere reservar su propio espacio de variables y alojarlos en los diferentes niveles de memoria que posee. Una vez termina el procesamiento en el acelerador los resultados

deben ser copiados al host realizando una transferencia por el mismo bus PCI y devolviendo el control al host que realizará la computación adicional necesaria como se observa en la Figura 9. Procesos de transferencia de datos entre CPU-GPU.

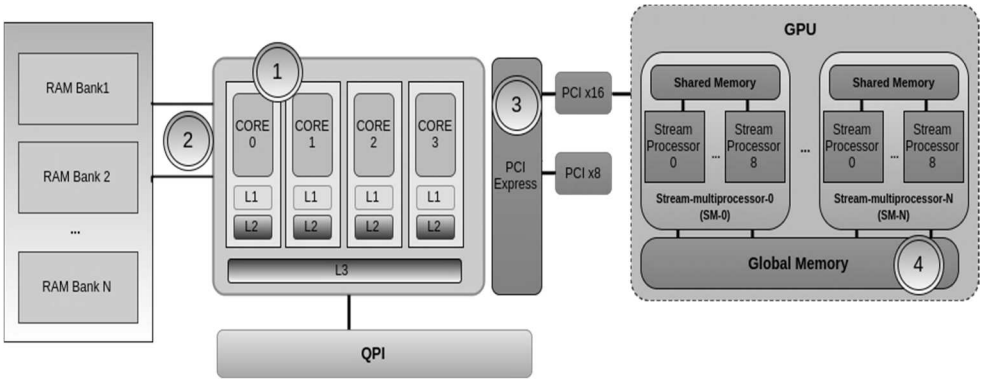


Figura 9. Procesos de transferencia de datos entre CPU-GPU.
Fuente: desarrollo propio

El modelo de ejecución en las GPU permite lanzar un programa en uno o múltiples celdas que se ejecutan sobre SM (Stream Multiprocessor), en diferentes agrupación de bloques de thread (Thread Block) (D B Kirk & Hwu, 2012). Lograr este nivel de paralelismo implica que se deberán realizar ajustes al software indicando explícitamente qué nivel de paralelismo se requiere, que tipos de memorias serán usadas y el grado de coalescencia que se tendrá en la ejecución de los códigos masivamente paralelos denominados kernels (Landaverde, Zhang, Coskun, & Herbordt, n.d.)

Para obtener el mejor desempeño posible dentro de la GPU, se debe explorar la utilizar patrones que garanticen el aprovechamiento de la localidad de la memoria y que evitan la transferencia constante de datos entre GPU y CPU, (Jang et al., 2011). Al aplicar técnicas eficientes de uso de memoria y niveles de paralelismo se puede lograr el aumento en el desempeño dependiendo el tipo de aplicación, la cantidad de código sujeto al cambio de la manera como se han segmentado los datos.

Para el caso específico del uso de GPUs en el modelo WRF, el tiempo de ejecución de las microfísicas ha mostrado una mejora importante que puede alcanzar un 25% de Speedup (John Michalakes & Vachharajani, 2008; Mielikainen et al., 2012), sin embargo en investigaciones de los dos últimos años se compara el grado de esfuerzo requerido para la reescritura del software contra la ganancia de desempeño logrado, colocando en tela de juicio la efectividad de la recodificación comparada con el uso de procesadores vectoriales que muestran desempeño semejantes con mucho menor esfuerzo de reescritura del software y sobre todo de una rama de soporte por parte de la comunidad (John Michalakes, 2013; Teodoro, Kurc, Kong, Cooper, & Saltz, 2013)

3.2. ACELERADORES VECTORIALES (INTEL Xeon Phi)

Los procesadores con soporte para operaciones vectoriales, intentan aprovechar dos características que han estado presentes en las diferentes familias de procesadores desde hace más de 15 años, pero que no han sido aprovechadas, debido a la dificultad en los modelos de programación o la manera cómo deben ser implementadas.

La resolución de un problema se convierte en la ejecución de un cierto número de instrucciones que deben de manera secuencial entrar en la ALU del procesador, para luego actualizar los registros del mismo y más tarde actualizar los diferentes niveles de memoria que estén disponibles. En operaciones de tipo matricial, las operaciones se vuelven ciclos de operaciones sobre un vector que realiza el mismo tipo de cálculos solamente variando el índice sobre el cual se opera en cada iteración del ciclo.

Si el procesador puede ofrecer instrucciones que no operen sobre una escalar sino sobre un vector, el desempeño de tales operaciones se incrementa ya que el número de ciclos requeridos para terminar una operación estaría limitado por el tamaño del vector que acepta el procesador para ser operado.

En el caso de procesadores x86 y más exactamente los fabricados por Intel ®, se han introducido instrucciones que inicialmente se llamaron MMX y que en la actualidad se llaman AVX (Advanced Vector Extensión). Este conjunto de instrucciones permite operar sobre vectores de 128, 256 y más recientemente 256 bits

Para aprovechar este tipo de instrucciones se debe utilizar marcadores o indicadores especiales al compilador, para que este pueda usar una instrucción vectorial en lugar de una escalar (Figura 10. Modos de operación vectorial y escalar en un procesador.)

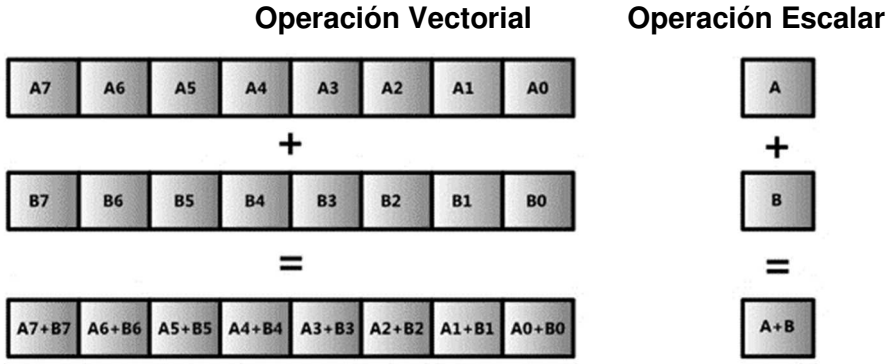


Figura 10. Modos de operación vectorial y escalar en un procesador. Fuente https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

Cada instrucción actúa sobre un CORE de procesamiento, sin embargo en las 2 últimas décadas los modelos se han introducido más de un core en el mismo procesador, logrando para aumentar el procesamiento y disminuir el consumo de energía, produciendo lo que se han denominado “la era multicore” (Bondhugula & Baskaran, 2008; Hill & Marty, 2008; Rauber & Rünger, 2010).

Intel ha introducido diferentes familias de procesadores multicore con soporte de 2, 4, 6, 8, 10, 12, y 16 cores por procesador, los cuales presentan un juego de instrucciones completas e instrucciones vectorizadas de hasta 256 bits.

Ante la necesidad de incrementar el rendimiento y poder ofrecer soluciones que puedan escalar a cientos de cores, se han introducido un nuevo esquema de procesadores que pueden escalar más allá de los 60 cores con 4 hilos de procesamiento por cada core, tal línea de procesadores ha sido denominada Intel Xeon Phi(Jeffers James;Reinders James, 2013).

En cuanto a arquitectura difiere de las GPUs en los siguientes aspectos:

- En primer lugar, cada core contiene soporte para instrucciones x86 de 64 bits completas, incluyendo operaciones SIMD. Dependiendo de la versión del dispositivo puede contener un juego de instrucciones P54c (Intel Pentium) hasta Compatible con Intel Xeon con soporte para instrucciones vectoriales AVX512.
- El dispositivo cuenta con su propio sistema operativo Linux, que permite verlo tanto como un dispositivo que se comunica a través del mecanismo de transferencia PCI tradicionales, como un nodo con una IP independiente, que se puede acceder por una emulación del protocolo IP sobre PCI.

3.2.1. Intel Xeon Phi Knights Corner (KNC)

El dispositivo internamente presenta un esquema basado en un anillo interno para interconexión de cores, mediante una arquitectura de acceso a memoria uniforme y con jerarquía de memoria L2. La estructura general se presenta en la (Figura 11. Microarquitectura para Intel xeon Phi KNC.)

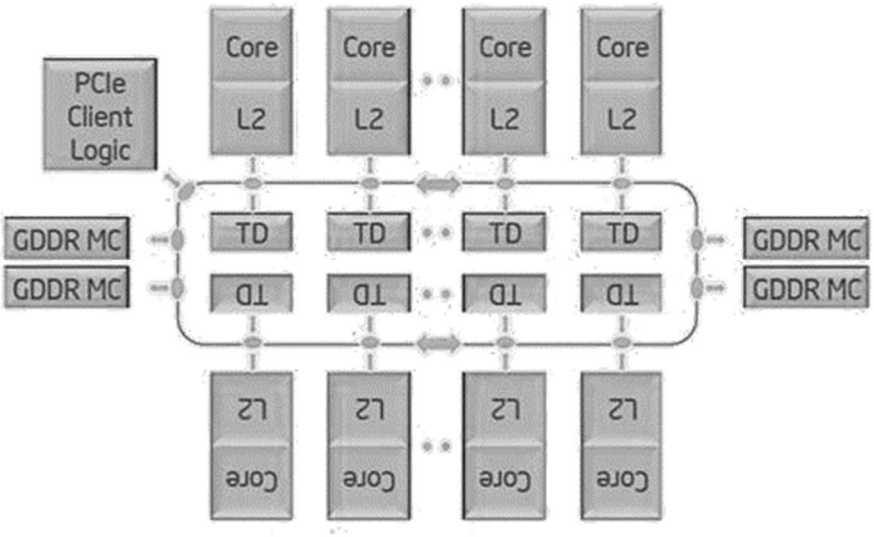


Figura 11. Microarquitectura para Intel xeon Phi KNC.
Fuente https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

Cada uno de los cores presenta soporte para registros escalares y para registros vectoriales, con sus respectivas unidades vectoriales y escalares de procesamiento. Para los dos tipos de unidades de procesamiento se ofrece un caché L1 de 32K para instrucciones y uno de 32 K para datos, también se ofrece una caché L2 de 512K (Figura 12. Micro-arquitectura de un core en Intel Xeon Phi KNC.). Este caché L2 se conecta a los cachés de los demás cores mediante un anillo

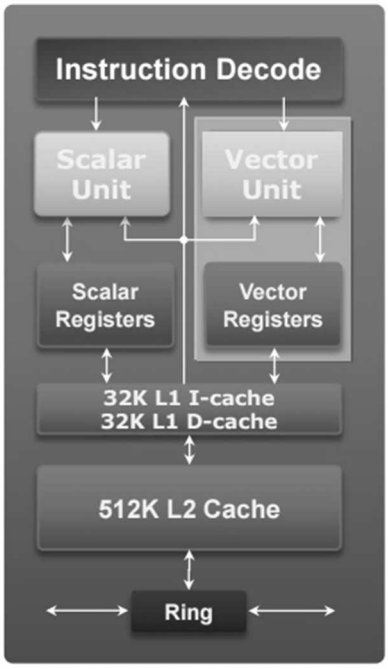


Figura 12. Micro-arquitectura de un core en Intel Xeon Phi KNC. Fuente Intel disponible en https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

3.2.1.1. Modos de ejecución en KNC

Como fue discutido anteriormente, un dispositivo KNC puede ser accedido mediante el uso de directivas PCI o mediante la emulación de TCP sobre PCI. De aquí se desprende los modos de ejecución, dado que el KNC puede ser visto tanto dispositivo independiente con un direccionamiento de red propio, como un dispositivo anexo al sistema accesible por PCI como se puede observar en la Figura 13. Modos de ejecución para Intel Xeon Phi KNC.

Si la ejecución del programa inicia en el host (procesador) y al alcanzar secciones del código marcados especialmente para ser ejecutados en un dispositivo vectorial este código es transmitido vía PCI para ser ejecutado en el KNC en un proceso llamado “descarga de código” u code offload, usando las diferentes jerarquías de memoria existentes y luego de ser procesado el resultado es transmitido al host.

Si la ejecución considera al KNC como un nodo independiente, que se puede comunicar vía IP, utilizando los mecanismos de mensajería MPI y un ejecutable ha sido producido y copiado previamente al dispositivo, este modo se considera un modo híbrido, dada que la ejecución ocurre tanto en el host como en el KNC, pero se considera al KNC como un nodo independiente. Las transferencias presentadas aquí no son de secciones de código, sino del paso de mensajes MPI.

Si toda la ejecución del programa ocurre en el KNC, sin que existan transferencias con el host, se considera el modo nativo. En este caso es de especial atención el hecho que tanto la memoria como el almacenamiento disponible son limitados y por tanto se debe acudir a mecanismos de sistemas de archivos distribuidos que soporte el sistema operativo del KNC

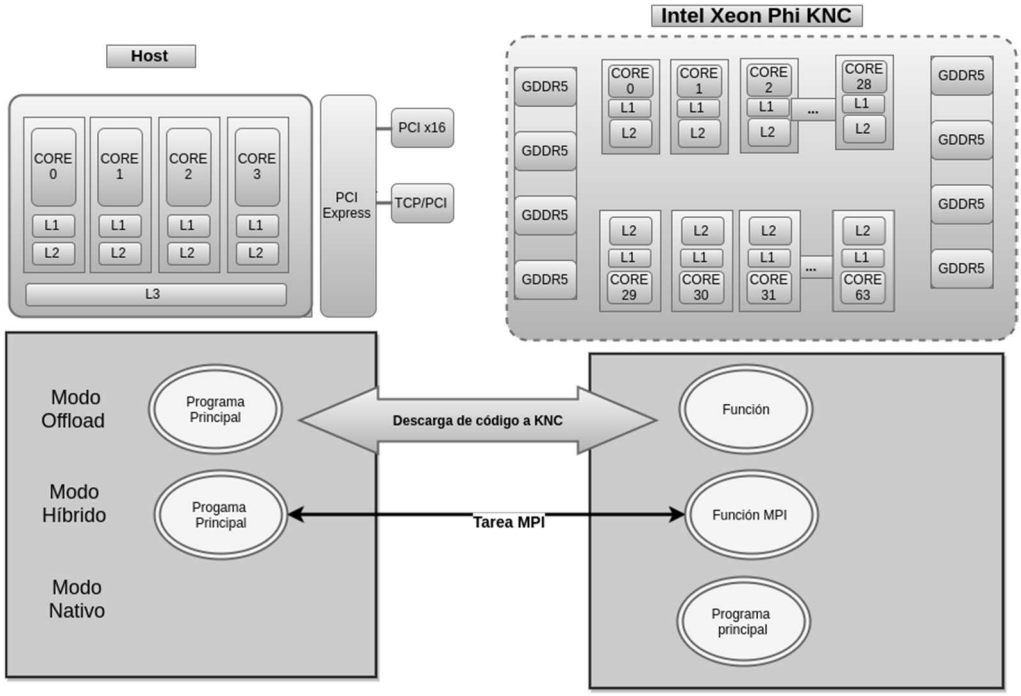


Figura 13. Modos de ejecución para Intel Xeon Phi KNC.
Fuente desarrollo propio

3.2.2. Intel Xeon Phi Knights Landing (KNL)

Dado los problemas de desempeño principalmente derivados de latencias introducidas por los procesos de transferencia de datos entre la memoria del host y del KNC (Rosales, 2014), se ha desarrollado una segunda generación de procesadores Intel Xeon Phi denominados KNL (Sodani, 2016) que presentan una arquitectura que pretende solucionar los problemas de transferencia de datos, acceso a memoria, compatibilidad binaria con procesadores Xeon.

Esta nueva serie de aceleradores se puede configurar tanto como un coprocesador (al estilo de KNC conectado al procesador central mediante buses PCI) o puede llegar a ser el procesador principal del sistema (Sodani, 2016), como se muestra en la Figura 14. Modos de configuración de un KNL.

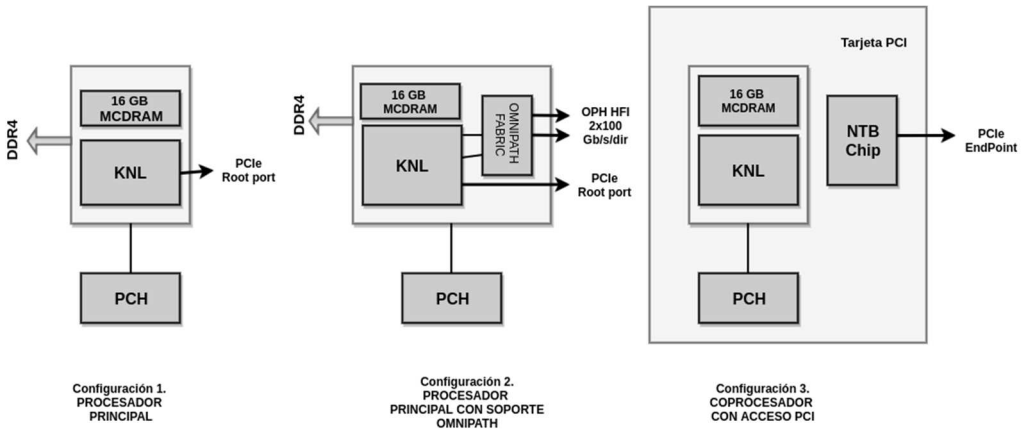


Figura 14. Modos de configuración de un KNL
Fuente: Desarrollo propio

Las configuraciones 1 y 2 presentan al KNL como procesador principal, sin embargo para la configuración 2, se adiciona un conector de Red directamente conectado al KNL denominado Intel® Omnipath (Birrittella et al., 2015). Esta red ofrece hasta 100Gb/s en cada uno de los sentidos de la comunicación, ofreciendo un ancho de banda total de 200Gb/s.

La configuración 3, presenta compatibilidad con la anterior generación (KNC), y permite que modelos basados en procesador de alto nivel de frecuencia (x86-64) usen el KNL como coprocesador y no como procesador principal.

3.2.2.1. Modos de configuración MCDRAM en KNL

Tomando como referencia la configuración 1 (KNL como procesador principal), esta nueva generación de procesadores también permite que la memoria MCDRAM disponible (Multi-Channel DRAM) sea usada como caché o como memoria de almacenamiento, dependiendo de las necesidades propias de escalamiento y localidad de cada algoritmo o programa (Asai, 2016), como se puede observar en la Figura 15. Modos de configuración MCDRAM en KNL.

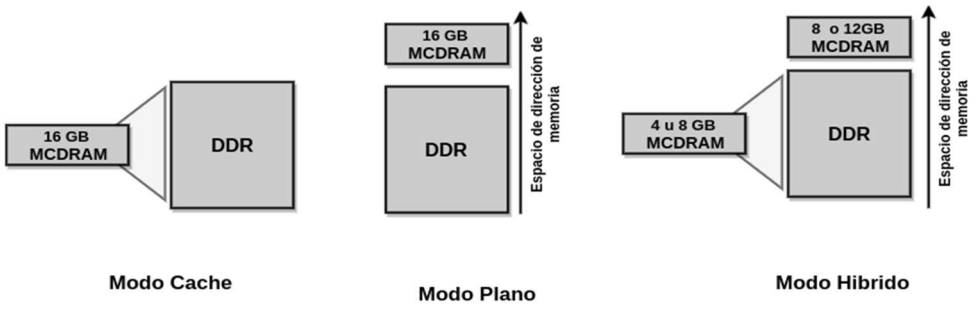


Figura 15. Modos de configuración MCDRAM en KNL.
Fuente Desarrollo propio

3.2.2.2. Procesamiento vectorial en KNL

El KNL está especialmente diseñado para procesamiento vectorial, en sus diferentes presentaciones de 64,68 o 72 Cores. Los cores se organizan en grupos de 2 más 2 unidades vectoriales (VPU) en una estructura denominada Tile, la cual tiene hasta 1MB de caché de nivel L2(Sodani, 2016). Cada VPU acepta hasta instrucciones AVX-512 que son operaciones de tipo SIMD de tamaño 512 bits (32 operaciones de precisión simple o 16 operaciones de

doble precisión) como se muestra en la Figura 16. Instrucciones vectoriales soportadas en KNL.

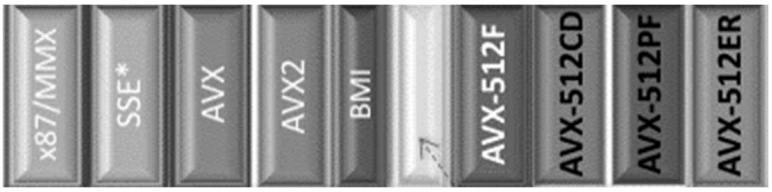


Figura 16. Instrucciones vectoriales soportadas en KNL.
Fuente https://software.intel.com/en-us/videos/knights-landing-an-overview-for-developers

3.3. RENDIMIENTO EN PLATAFORMAS HETEROGÉNEAS

Aunque las plataformas heterogéneas analizadas (GPUs e Intel MIC) presentan elementos que coinciden como soporte para instrucciones masivamente paralelas (ya sea a través de Kernels o instrucciones vectoriales), el tipo de problemas en que se enfocan en acelerar son diferentes y es por esta razón que no se puede decir a nivel general que una es mejor que otra, más bien cada tipo de plataforma ofrece mejor rendimiento para un conjunto específico de problemas.

Las GPUs por tener un número masivo de cores con un soporte limitado en cuanto a número de operaciones son especialmente eficientes en paralelismo de grano fino (Pratas, Trancoso, & Stamatakis, 2009) siempre que el problema pueda lograr aprovechar el throughput ofrecido por el dispositivo y que no se presenten divergencia en los caminos de ejecución (David B. Kirk & Hwu, 2013). Los algoritmos que expongan un alto grado de paralelismo pueden sacar ventajas de estas arquitecturas, siempre que las partes seriales sean muy pocas y que la ejecución no requiera de operaciones de registros complejas.

Los Intel MIC aunque comparativamente presentan un menor número de cores, pueden brindar un mayor nivel de rendimiento para aquellas

operaciones que exponen un paralelismo de grano grueso y que requiera el uso de instrucciones complejas (vectoriales) o que una porción del algoritmo o programa se ejecuten en gran medida serialmente (Reinders, 2012).

Además de las plataformas analizadas, han surgido con fuerza en los últimos 5 años, aquellas que añaden el factor de eficiencia energética, logrando el mejor desempeño por watt utilizado como es el caso de los procesadores de bajo consumo basados en arquitecturas ARM (Carrington, 2015).

3.4. CONCLUSIÓN

La computación heterogénea es el presente y sin duda el futuro de la computación científica y paralela, permitiendo que se pueda ganar en desempeño, capacidad de cómputo y eficiencia energética. El aprovechamiento de las diferentes plataformas existentes en el mercado y de nuevas que están en desarrollo de requiere que los equipos de desarrollado posean profundos conocimientos de la arquitectura de hardware y modelos de programación disponibles; para aquellas cargas de trabajo que contengan códigos desarrollados durante largos periodos de tiempo y que recodificarlos en nuevos API de programación brinde un desafío muy costoso en tiempo y esfuerzo, los aceleradores vectoriales ofrecen un magnífico escenario para acelerar su procesamiento sin reprogramar completamente la solución.

Para lograr la mejor eficiencia posible, se debe aprovechar los diferentes tipos de cargas de trabajo o las diferentes etapas que experimenta una carga en particular, de esta manera se puede agendar la ejecución dependiendo del tipo de paralelismo expuesto y del objetivo del desempeño (tiempo de ejecución, consumo de energía, eficiencia de uso de recursos) mediante el uso de distribuidores de carga automática o utilizando API que permitan que en tiempo de ejecución dichos parámetros se puedan determinar y así utilizar los recursos adecuados para la carga de trabajo especificada.

En el siguiente capítulo se explorará la manera como se analizan las ganancias en rendimiento y eficiencia, de tal manera que se pueda elaborar un análisis teórico de las posibilidades de mejora, los desafíos para aumentar el desempeño y las leyes que gobiernan los elementos claves para acelerar aplicaciones.

CAPÍTULO 4. ELEMENTOS TEÓRICOS PARA EL ANÁLISIS DE RENDIMIENTOS DE APLICACIONES.

Al intentar acelerar aplicaciones, un elemento fundamental es el rendimiento que podría ofrecer, al ser ejecutada en una plataforma paralela, sea esta heterogénea o no. Los análisis que se desarrollen permitirán estimar las ganancias que podrán ser obtenidas y contrastarlas con el esfuerzo requerido para alcanzarlas. Esta ganancia necesita tener en cuenta la naturaleza del problema, las condiciones de la plataforma y las capacidades de los algoritmos utilizados, de tal manera que se pueda alcanzar un mejor speedup, mejore la eficiencia de los algoritmos y alcance un mejor nivel de escalamiento cuando se incrementa el número de unidades de cómputo (PE) sobre la cual se ejecuta el mismo.

En la evaluación de sistemas paralelos se han utilizado tradicionalmente las métricas de speedup, eficiencia, y escalamiento, las cuales involucran tanto el algoritmo como la plataforma de ejecución del mismo; cada una de estas medidas presentan variaciones dependiendo del aspecto que se desea analizar. En la última década se han incorporado métricas adicionales como la eficiencia energética para tener una visión diferencial de la eficiencia computacional (Reams, 2012; Run Rules, Green500, 2014).

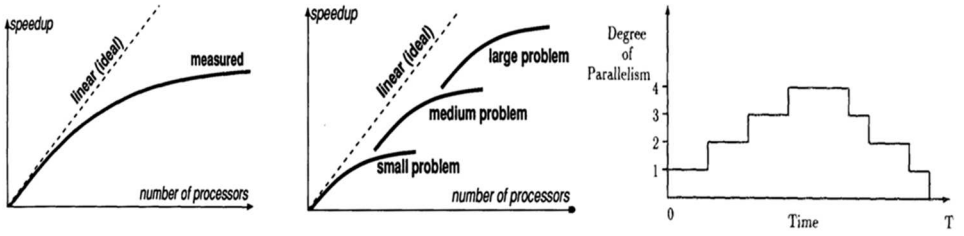
4.1. SPEEDUP

El speedup se define como la proporción de tiempo gastado cuando un programa se ejecuta en un sistema con un procesador frente a la ejecución del mismo programa en un sistema con n procesadores como es descrito en la Ecuación (7)(Amdahl, 1967; Gustafson, 1988; Sun & Gustafson, 1991). Como métrica es ampliamente utilizada para evaluar el rendimiento en

procesamiento paralelo (Malony et al., 2011; Sun & Gustafson, 1991; Tokhi, Hossain, & Shaheed, 2012), y ha sido la base para establecer algunas observaciones convertidas en “leyes”, como la así denominada “Ley de Amdahl”, y “Ley de Gustafson”, elementos que serán analizados en una sección más adelante en el presente capítulo.

Basados en las observaciones realizadas por John Gustafson a ley Amdahl, el análisis de speedup debió considerarse en tres situaciones distintas (Hill & Marty, 2008; R. Jain, 1991; Moreland & Oldfield, 2015; Tokhi et al., 2012), como muestra la Figura 17. Tipo de problemas para análisis de speedup.

- **Speedup para problemas de tamaño fijo**: En este enfoque se establece el tamaño del problema y se enfatiza que tan rápido el problema puede ser resuelto, normalmente el desempeño no incrementa linealmente cuando se incrementa el número de procesadores, por el fenómeno descrito en la ley de Amdahl (Hill & Marty, 2008).
- **Speedup para problemas de memoria fija**: En este enfoque se asume que la memoria es una barrera que limita la computación que se puede ejecutar y por tanto, debe crecer linealmente cuando incrementa el número de procesadores.
- **Speedup para problemas de tiempo fijo**: Este enfoque se especializa en cómo lograr un mayor “grado de paralelismo”, sin que el marco de tiempo de ejecución cambie radicalmente e intenta determinar el grado de eficiencia en el uso de plataformas paralelas.



Problemas de tamaño fijo Problemas de memoria fija Problemas de tiempo fijo

Figura 17. Tipo de problemas para análisis de speedup.
Fuente http://hint.byu.edu/documentation/Gus/FixedTime/FixedTime.html#4.1

En la presente investigación se siguió el primer enfoque dado que el tamaño del problema a analizar no podrá variar considerablemente y el principal objetivo es disminuir el tiempo de ejecución de las simulaciones para unas condiciones espaciales y temporales determinadas.

Formalmente el speedup relativo se definido como:

$$S(n) = \frac{Tiempo\ ejecución\ 1\ procesador}{Tiempo\ ejecución\ n\ procesadores} = \frac{T(1)}{T(N)} \qquad \text{Ecuación (7)}$$

Esta formulación supone un mismo tipo de paralelismo que se mantiene constante durante la ejecución de un programa, sin embargo como lo mostró Sun y Gustafson (Sun & Gustafson, 1991), el grado de paralelismo puede variar a lo largo del tiempo de ejecución de un programa dado que las arquitecturas exponen diferentes tipos y grados de paralelismo como paralelismo a nivel de instrucciones, el paralelismo a nivel de core y a nivel de nodo para hacer uso efectivo de los sistemas de cache y de las unidades de punto flotante, por tanto se puede deducir que el costo de operación de un trabajo (computación de las tareas de un programa) es dependiente de la arquitectura y no tan solo del número de unidades de cómputo existentes, por lo cual podemos expresar el tiempo de ejecución en función del trabajo como:

$$t_i(n) = \sum_{j=1}^{k} C_j W_{ij}$$ Ecuación (8)

En donde W_{ij} es el total de trabajo of tipo *j* ejecutado con el grado de paralelismo *i*, con un costo de operación C_j, asumiendo *k* diferentes tipos de trabajo, si asumimos un número *i* de procesadores entonces tenemos que el tiempo de ejecución sería:

$$t_i(i) = \frac{\sum_{j=1}^{k} C_j W_{ij}}{i} \; para \; 1 \leq i \leq m$$ Ecuación (9)

Asumiendo una arquitectura homogénea y despreciando el tiempo en procesos de comunicación, entrada/salida y sincronización, el tiempo total de ejecución en un infinito número de procesadores sería

$$T(\infty) = \sum_{i=1}^{m} \frac{\sum_{j=1}^{k} C_j W_{ij}}{i}$$ Ecuación (10)

Por tanto, el speedup sería

$$S\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{k} C_j W_{ij}}{\sum_{i=1}^{m} \frac{\sum_{j=1}^{k} C_j W_{ij}}{i}}$$ Ecuación (11)

Si consideramos un grado de paralelismo mayor que el número de procesadores disponibles $N < i$, entonces algunos procesadores deberán realizar el siguiente trabajo $W_i/i[i/N]$ y el resto de los procesadores realizarán el trabajo $W_j/i[i/N]$ y asumiendo que W_i y W_j no se pueden resolver simultáneamente para $i \neq j$ entonces el tiempo consumido sería:

$$t_i(N) = \frac{\sum_{j=1}^{k} C_j W_{ij}}{i} \left\lceil \frac{i}{N} \right\rceil$$ Ecuación (12)

y

$$T(N) = \sum_{i=1}^{m} \frac{\sum_{j=1}^{k} C_j W_{ij}}{i} \left\lceil \frac{i}{N} \right\rceil$$ Ecuación (13)

Por tanto, el cálculo del speedup sería:

$$S_N = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{k} C_j W_{ij}}{\sum_{i=1}^{m} \frac{\sum_{j=1}^{k} C_j W_{ij}}{i} \left\lceil \frac{i}{N} \right\rceil}$$ Ecuación (14)

El anterior enfoque ha sido considerado clásico para problemas de tamaño fijo, sin embargo (Gustafson, 1988; Sun & Gustafson, 1991) propone una revisión a estas técnicas poniendo a consideración que el speedup es relativo a las arquitecturas en las cuáles se ejecutan y, los parámetros utilizados para la afinación de una arquitecturas pueda llegar a producir perdida del speedup en otras.

Históricamente esta formulación se ha aplicado a arquitecturas que usan el mismo tipo de unidades de procesamiento, de elementos de comunicación y de memoria (arquitectura homogénea), por lo cual algunas de las premisas aplicadas no se pueden extrapolar directamente cuando se utiliza arquitecturas que tienen elementos de procesamiento y memoria no homogénea (arquitectura heterogénea). Problemas asociados a las comunicaciones, distribución de la memoria y latencias para copia de datos entre diferentes unidades de procesamiento hace que se deba analizar otros factores como el desequilibrio de carga, el retardo por transferencia y el nivel de paralelismo que se puede lograr en las diferentes unidades de procesamiento(Malony et al., 2011).

4.1.1. Ley de Amdahl

El argumento utilizado por el arquitecto de ordenadores Gene Amdahl, indica que el tiempo total de computación de un programa que corre sobre arquitecturas paralelas, se divide en 2 partes:

- El tiempo empleado para hacer trabajo serial (no paralelo)
- El tiempo empleado haciendo trabajo paralelo

Si llamamos a Ws al tiempo consumido en trabajo serial y Wp al tiempo consumido en trabajo paralelo, entonces el tiempo total de un trabajo para una arquitectura con 1 procesador y con P procesadores sería:

$$T_1 = Ws + Wp$$
$$T_p = Ws + \frac{Wp}{P} \qquad \text{Ecuación (15)}$$

De esta manera podemos indicar que el speedup (S) presentado cuando están presentes P unidades de procesamiento sería

$$S_P \leq \frac{Ws + Wp}{Ws + {Wp}/{P}} \qquad \text{Ecuación (16)}$$

Sin embargo, suponemos que existe una parte del código o programa que no puede ser paralelizada a la cual llamaremos f, tal que

$$Ws = fT_1$$
$$Wp = (1 - f)T_1 \qquad \text{Ecuación (17)}$$

Y sustituyendo esto en ecuación 16 tenemos

$$S_P \leq \frac{1}{f + {(1-f)}/{P}}$$ Ecuación (18)

Si suponemos un número infinito de elementos de cómputo entonces, el límite del speedup está determinado por la sección del código serial, como se muestra en

$$S_\infty \leq \frac{1}{f}$$ Ecuación (19)

Esta ley permite predecir que el máximo speedup estará determinado por la fracción de código serial que este posea (McCool et al., 2012; Tokhi et al., 2012) lo cual será más evidente en la medida que el número de elementos de procesamiento aumente como se muestra en la Figura 18.

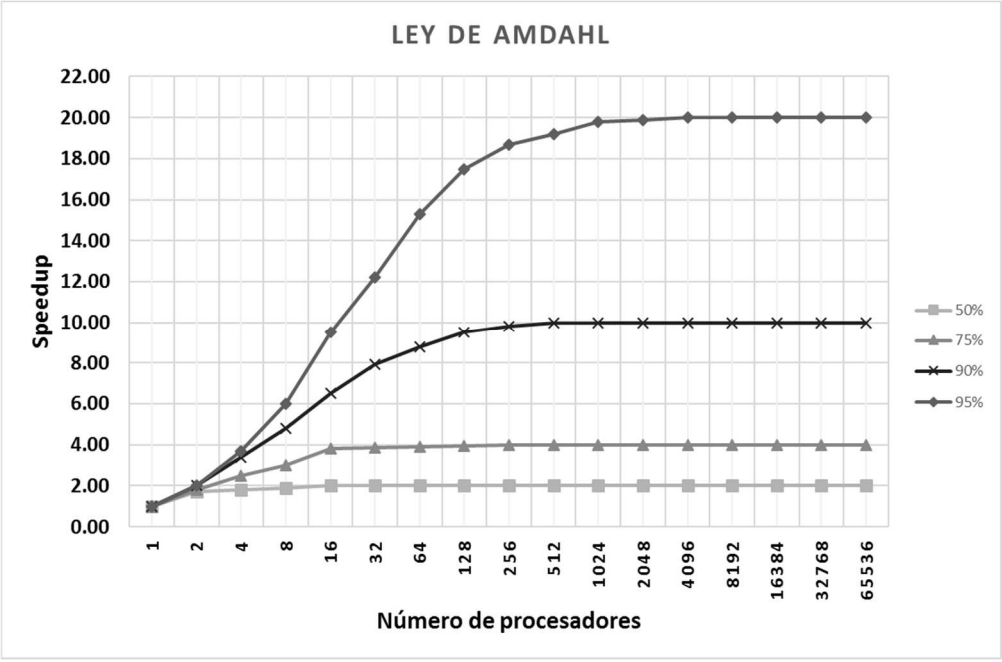


Figura 18. Nivel de speedup en relación con la cantidad de código paralelo
Fuente: desarrollo propio

4.2. EFICIENCIA

Aunque el speedup es una medida importante, en arquitecturas heterogéneas dado el diferencial de rendimiento de los elementos de procesamiento (PE), es necesario usar una segunda métrica denominada eficiencia, en virtud del speedup observado comparado con el speedup ideal, teniendo en cuenta el número de elementos de procesamiento (N). Esta medida se define como:

$$E(n) = \frac{S(N)}{N} = \frac{T(1)}{NT(N)} \qquad \text{Ecuación (20)}$$

En donde N indica el número de procesadores disponibles. Esta definición como se discutió tiene un enfoque aplicado en arquitecturas homogéneas, pues supone un speedup ideal alcanzado, sin embargo, en la mayoría de los escenarios el speedup no es lineal o presenta variaciones dependiendo el tipo de elemento de procesamiento disponibles y del tipo de paralelismo que expone el programa analizado. Podemos por tanto establecer una nueva medida denominada isoeficiencia, que es entendida como el incremento necesario de trabajo para que la eficiencia no disminuya (Tokhi et al., 2012).

Como objetivo del desempeño de aplicaciones, se establece el tiempo requerido para que un algoritmo pueda ejecutar todo el trabajo necesario; sin embargo, este objetivo se puede analizar en conjunto con el tamaño del problema a resolver y la cantidad de recursos disponibles para su solución. El establecimiento de un objetivo de desempeño ya sea tiempo, eficiencia o speedup hace que se fijen los demás elementos como fijos o con capacidad de crecimiento ilimitada. Desde el punto de vista algorítmico resulta interesante observar elementos aislados; sin embargo, desde el punto de vista computacional el tener en cuenta los recursos necesarios para resolver el problema, resulta fundamental.

Entendiendo las limitaciones del análisis de la eficiencia basada solo en la diferencia de speedup ganado a medida que aumenta el número de unidades de procesamiento, también es necesario analizar otra medida que nos permita calcular la manera como se realiza el trabajo computacional cuando

el tamaño del problema cambia o cuando cambia los recursos computacionales manteniendo fijo el tamaño del problema.

4.3. ESCALAMIENTO (Scaleup)

El scaleup puede definirse como la habilidad de mantener un tiempo de ejecución constante cuando la carga de trabajo o tamaño del mismo aumenta, mediante el aumento de los recursos computacionales (procesamiento, memoria y disco), se puede calcular como:

$$Scaleup = \frac{W_N}{W_1}$$ Ecuación (21)

Donde W_N representa el trabajo realizado en cierta cantidad de tiempo por N procesadores y W_1 representa el trabajo realizado por 1 procesador.

En una situación ideal, se puede mantener el tiempo de ejecución e incrementar la carga de trabajo y los recursos computacionales, de tal manera que presente un comportamiento lineal (Figura 19. Curva de scaleup ideal.).

Sin embargo el comportamiento real evidenciado en una larga cantidad de experimentos realizados por Gustafson, demostró que algunos problemas escalan bien cuando poseen más recursos disponibles manteniendo el tiempo de ejecución fijo y que algunos otros escalan manteniendo el tamaño del problema fijo y pero disminuyendo el tiempo de ejecución requerido (Gustafson, 1988; Sun & Gustafson, 1991)

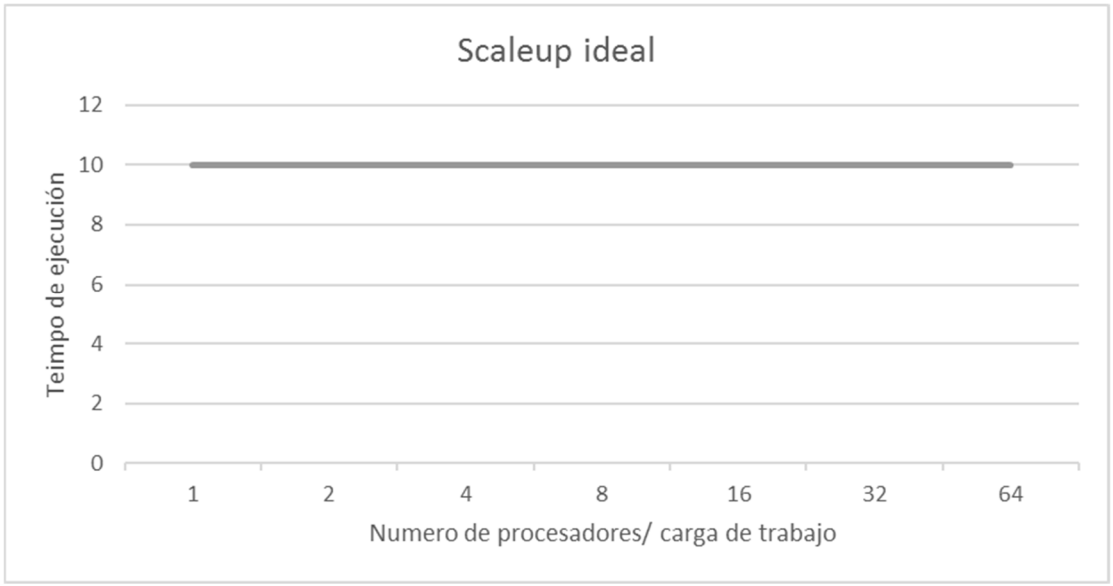


Figura 19. Curva de scaleup ideal.
Fuente: Desarrollo propio

Dependiendo del tipo de adaptación que presente, se pueden presentar dos tipos de escalamiento:

a) Escalamiento fuerte (Strong Scaling): En este caso el tamaño del problema se mantiene fijo y se incrementa el número de elementos de procesamiento (PE), de tal manera que se logre más trabajo por procesador. Al incrementar los PE se introduce latencias por procesos de división del trabajo, tareas de comunicación y recolección parcial de resultados siendo el mayor obstáculo la cantidad de código serial presente y tiene un efecto adverso como descrito por la ley de Amdahl (Hill & Marty, 2008; Woo & Lee, 2008). Se puede definir entonces como el speedup como:

$$Speedup(p,x) = \frac{t(1,x)}{t(p,x)} \qquad \text{Ecuación (22)}$$

En donde p es el número de procesadores y x el tamaño del problema.

b) Escalamiento débil (Weak Scaling): En este caso la carga de trabajo por procesador se mantiene constante, al igual que el tiempo de ejecución, pero el número de elementos de procesamiento y la carga total de trabajo se incrementan.

$$Speedup(p, x \times p) = \frac{t(1, x)}{t(p, x \times p)}$$ Ecuación (23)

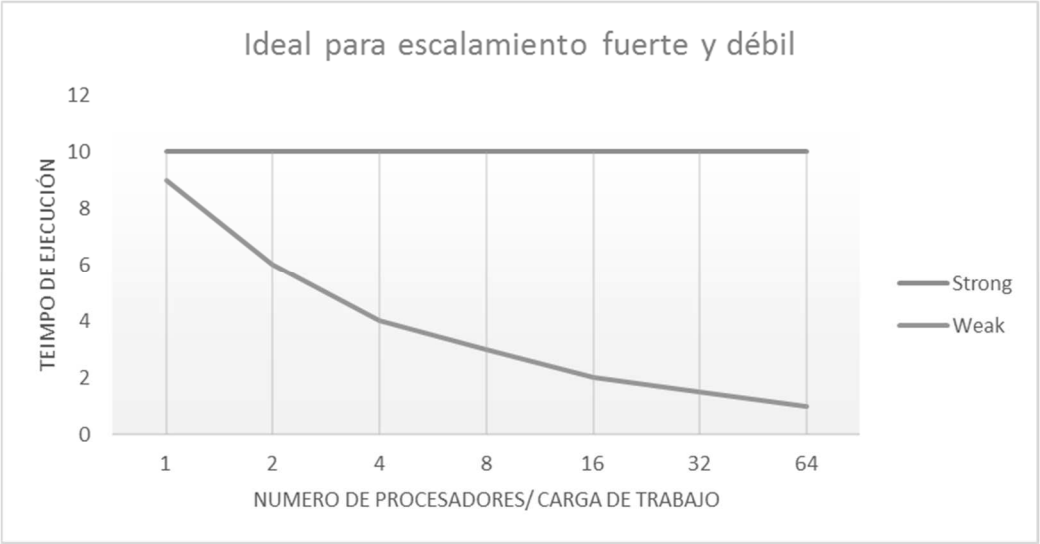


Figura 20. Comparación escalamiento fuerte vs escalamiento débil
Fuente: Desarrollo propio

4.4. ELEMENTOS QUE GENERAR SOBRECARGA EN EL PROCESAMIENTO

Hasta el momento se ha analizado el rendimiento de la aplicación teniendo en cuenta exclusivamente los elementos de procesamiento. Sin embargo, dentro de la ejecución de programas en esquemas de memoria distribuida aparecen algunos elementos que aumentan el tiempo de ejecución y que se relacionan con las operaciones de división y sincronización de tareas además de los procesos de comunicación y mensajería entre nodos. Cuando se utilizan arquitecturas heterogéneas aparecen tiempos adicionales por

transferencias de datos entre unidades de computación distintas, donde es necesario realizar copias entre diferentes jerarquías de memoria como lo muestran los análisis realizados para GPUs por (Gupta et al., 2013; Jia, 2014; Landaverde et al., n.d.; Panourgias, 2011) y los realizados para procesadores Intel por (Hackenberg, Molka, & Nagel, 2009; Majo & Gross, 2011; Molka, Hackenberg, Schöne, & Müller, 2009)

4.4.1. Estructuración de tareas:

La estructuración de tareas es uno de los mayores desafíos planteados al ejecutar un programa que escala a cientos o millones de procesadores, ya que debe contemplar tanto la complejidad como el tamaño de los datos de entrada, para producir tareas de un tamaño adecuado y agendar su procesamiento en las plataformas de cómputo suministrada. La estrategia que se escoja dividir las tareas resultará en un mejor o peor rendimiento de las aplicaciones o algoritmos, permitiendo una mejor o peor eficiencia.

Cuando se estructuran tareas no homogéneas el desafío es mayor, ya que se encuentran un conjunto de N diferentes tipos de elementos de procesamiento que se ofrecen mejores desempeños para un tamaño de tarea ideal; ante este escenario se deben construir grupos de tareas de tamaño o comportamiento semejante que puedan ser lanzadas en conjunto, por lo cual el tamaño de la subtarea es función del tipo de procesador y del tamaño total del problema.

$$T(x) = f(tp, n) \qquad \text{Ecuación (24)}$$

Así por ejemplo para algunos tipos de PE es mejor tener T(x) más grandes (paralelismo de grano grueso) y para otros tipo de PE es mejor tener T(x) pequeños (paralelismo de grano fino)(John Michalakes & Vachharajani, 2008; Orr, Beckmann, Reinhardt, & Wood, 2014).

4.4.2. Sincronización de tareas:

La coordinación de tareas para se pueda conocer el estado de cada una de ellas en un determinado instante de tiempo, dentro de este proceso se debe determinar el grado de dependencias que existen entre las tareas, que tipo de prioridad se maneja entre ellas y que tipo de datos requieren que estén intercambiando de manera permanente. Para lograr que el programa escale de una manera adecuada, hay que mantener las tareas de sincronización lo más limitadas posibles y dependiendo el tipo de paralelismo expuesto por el problema o por el conjunto de datos, estas pueden presentar un costo mayor o menor en procesamiento. Este tipo de sincronización de tareas se ha agrupado por el grado de interacción entre las mismas y ha sido ampliamente expuesto por Kaltofen en el afamado artículo “Los 7 enanos de la computación simbólica” (Langer & Paule, 2011).

4.4.3. Comunicación entre tareas:

Una vez las tareas han sido divididas utilizando una estrategia que maximice el desempeño para un tipo de arquitectura específica y bajo un esquema de sincronización determinado, se requieren que ellas se comuniquen entre sí, utilizando mecanismos diferenciales si la comunicación ocurre en el mismo nodo (intranodo) o si ocurre entre tareas de se encuentran en nodos remotos (extranodo). Las tareas que se ejecutan en un mismo nodo pueden usar los diferentes mecanismos de jerarquías de memorias o incluso de transferencia entre ellas, sin acudir a sistema de paso de mensajes específicos, siendo el paso de mensaje, la comunicación interproceso, las tuberías y espacios de memoria compartida los mecanismos más comunes. Cuando la comunicación requiere ser entre diferentes nodos de cómputo interconectados por una red de datos, el tipo de intercomunicación utilizado es el paso de mensajes. Este paso de mensajes puede darse punto a punto, punto multipunto o comunicación colectiva; el uso intensivo de mensajería impacta de una

manera significativa el desempeño de una aplicación incluso produciendo curvas negativas de escalabilidad (Nokia Siemens Networks, 2009; Nupairoj & Ni, n.d.)

4.4.4. Control de hilos

En sistemas con procesadores de más de un core (multicore), se suele utilizar esquemas de hilos para aumentar el uso del procesador y disminuir el tiempo de procesamiento. Un hilo se diferencia de un proceso en que un este posee su propio espacio de direcciones de memoria, pila ejecución y estados de sus variables. Cada proceso está aislado de los demás que se encuentren ejecutándose en un sistema, mientras que los hilos pueden compartir el espacio de memoria, los estados de las variables y la pila de procesamiento, con lo cual es común que un proceso pueda iniciar un conjunto de hilos para aumentar la computación en una tarea específica (Herlihy, 2006; McCool et al., 2012; Pacheco, 2011). Mantener más de un hilo de procesamiento requiere que tanto los datos como las tareas ejecutadas por cada hilo estén bien definidas pues al estar accediendo a recursos compartidos se presentan problemas de abrazos mortales (deadlocks), condiciones de carrera (race condition) como es ampliamente abordado en el libro (Akhter & Roberts, 2006) y respecto al desempeño se presenta problemas de sobre-suscripción, es decir lanzar mucho más hilos de procesamiento que número de cores disponibles en el sistema, lo cual aunque aparentemente aumentaría el desempeño realmente tiende a degradar el rendimiento en la medida que el número de hilos aumenta. Este fenómeno es ampliamente analizado en la investigación del Laboratorio Nacional Lawrence Berkeley (Iancu, Hofmeyr, Blagojeviic, & Zheng, 2010)

4.5. CONCLUSION

El análisis de rendimiento de aplicaciones constituye el fundamento al momento de acelerar aplicaciones, ya que permite observar la ganancia que se podría obtener, los desafíos al adaptar los algoritmos a plataformas paralelas máxime si estas son heterogéneas y los problemas inherentes producidos por las copias que se deben realizar al interior de cada nodo, cuando se encuentran diferentes unidades de cómputo. Este elemento tiene mayor importancia cuando se trabaja en arquitecturas de memoria distribuida debido a la sobrecarga introducida por la mensajería, la estructuración de tareas y la sincronización de las mismas. Tener una visión clara sobre la real posibilidad de aceleración permite no sobreestimar el uso de herramientas y técnicas que prometen ganancias inmensas de desempeño con tan solo adoptar un determinado modelo de programación o herramienta de software.

En los siguientes capítulos se analizará a profundidad la manera como fue construido el modelo WRF, las estructuras internas de comunicación tanto para memoria compartida como para memoria distribuida, la manera como es capaz de aprovechar el tamaño del problema para realizar la división lógica de los datos y además cuáles secciones de su código son más relevantes para la presente investigación dado que aportan el mayor tiempo de computación en la ejecución de un pronóstico.

CAPÍTULO 5. ANÁLISIS DE LA ARQUITECTURA DE SOFTWARE DEL MODELO AVANZADO DE PRONÒSTICO WRF

Los modelos de pronóstico meteorológico han sido pioneros en el uso de computación de alto desempeño, debido a su naturaleza masiva tanto en datos como en cálculos (Lynch, 2008). El modelo WRF fue elegido como caso de análisis dada su capacidad de funcionar como modelo mixto tanto para investigación como para operación. El modelo WRF es de libre acceso al código fuente y de desarrollo comunitario (Wei Wang, Cindy Bruyère, 2016); un elemento fundamental para su elección es el hecho que es modelo numérico principal utilizado por la autoridad meteorológica nacional (Ruiz Murcia, José Franklin Instituto de Hidrologia, 2010).

El modelo WRF fue desarrollado en sus etapas iniciales en el año de 1998, basado principalmente en lenguaje fortran y lenguaje c. Dado el interés que despertó este modelo de mesoescala tanto a nivel de investigación como a nivel operativo rápidamente recibió aportes de múltiples agencias y autoridades ambientales inicialmente de Estados Unidos y luego de Europa y algunas universidades de Asia, como ya se ha mencionado en el capítulo 1.

5.1. MARCO DE TRABAJO DE SOFTWARE DEL MODELO WRF

Desde su creación dado su carácter de libre acceso y de múltiples aportantes al código, se enfatizó en un diseño flexible para ser ejecutado sobre un rango extenso de plataformas de hardware y clústeres de procesamiento paralelo. Para cumplir este objetivo de desarrolló un marco de trabajo denominado WRF software Framework (J Michalakes et al., 2004; John Michalakes & Gill,

2016). La arquitectura de WRF presenta por tanto una organización jerárquica, interfaces abstractas bien definidas para permitir la incorporación de múltiples núcleos y un sistema de plugins para las parametrizaciones físicas usadas.

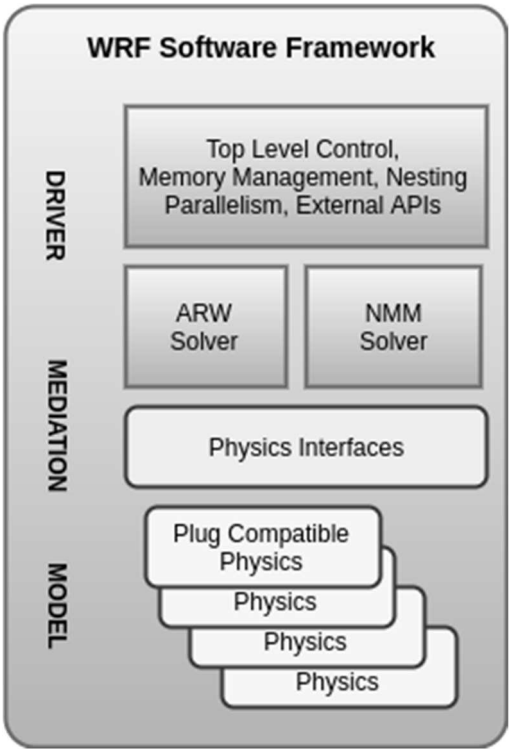


Figura 21. Marco de trabajo de software para el modelo WRF.
Fuente: Desarrollo propio

La capa de drivers maneja en tiempo de ejecución el alojamiento y descomposición de las estructuras de datos de los dominios del modelo, al igual que la organización, manejo, interacción sobre los dominios anidados, incluyendo el bucle de tiempo principal del modelo, las interfaces de alto nivel de entrada/salida y las interfaces de integración con otros componentes del modelo.

La capa de mediación engloba un paso de tiempo de un núcleo particular en un dominio simple. Las rutinas de los “solver” (Figura 21. Marco de trabajo de software para el modelo WRF.) contienen un grupo de invocaciones a las

rutinas de la capa del “modelo” como también la invocación de las rutinas de comunicación interprocesos (MPI) y multithreading (OpenMP). Las implementaciones actuales de WRF usan un esquema mixto de paralelismo que descompone los dominios horizontalmente en dos niveles. En el primero de ellos se eligen un número n de tareas MPI con dos factores, la descomposición en la dirección x y la descomposición en la dirección *y*; a este tipo de división se le denomina un “*patch*”. Sin embargo dado que la computación que se asigna a cada “*patch*” puede resultar grande, se crea otro nivel de descomposición asignando “*p*” cantidad de tareas en cada “*patch*”, que idealmente debería ser el equivalente al número de core físicos existentes en cada nodo y que se denomina un “*tile”* (Malakar et al., 2012; John Michalakes & Gill, 2016).

En cada una de las capas del modelo se realiza invocación a las rutinas a través de interfaces estándar y todos los estados son pasados como argumentos junto con los índices inicial y final en cada celda tridimensional de modo que un “*tile”* pueda ser computada.

Como se puede observar en Figura 21. Marco de trabajo de software para el modelo WRF. cada capa establece un contrato para asegurar que el código podrá ser ejecutado en cualquier plataforma de computación paralela soportada. Es responsabilidad del programador describir los tipos de comunicación y patrones que usa una parametrización particular mediante la adición de una entrada en el “Registro” del modelo además de insertar una notación para desarrollar la comunicación en la localización adecuada de las rutinas “solve” (Figura 22. Esquema de distribución de software en WRF.)

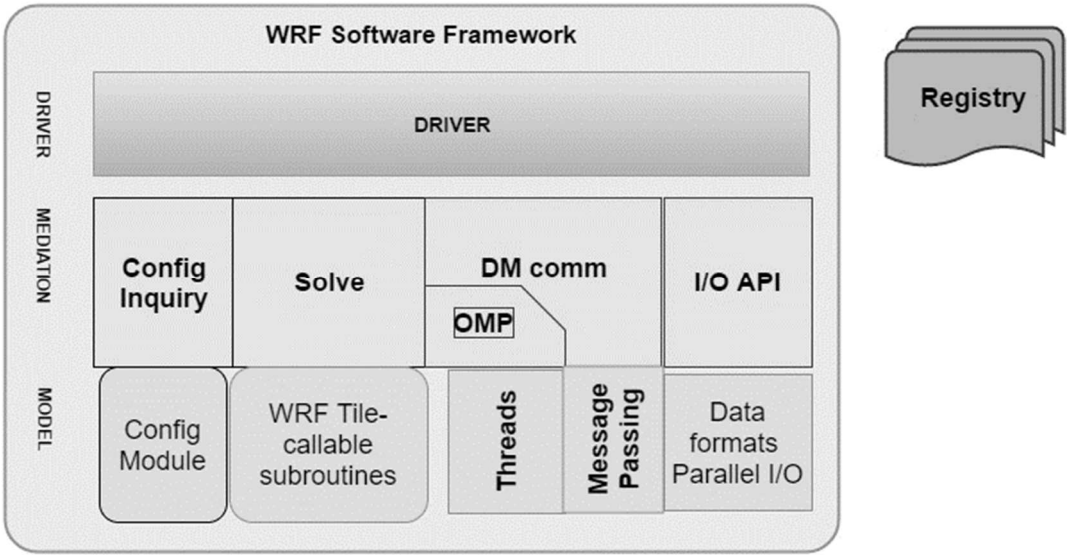


Figura 22. Esquema de distribución de software en WRF.
Fuente: Desarrollo propio basado en el tutorial de WRF 2016

El método principal para generar tareas para puntos de malla es dependiente del área del dominio y del espaciado horizontal y vertical elegido. Se puede generar n tareas MPI, donde n= k * m. Los valores de k y m son insertados por el usuario como parámetro en el archivo namelist.wps, en la etapa de pre-procesamiento, de igual manera el número de tiles por nodo es determinado por la variable numtiles del namelist.input, en caso de no ser definida se tomará de la variable de ambiente OMP_NUM_THREADS.

5.2. MANEJO DE PARALELISMO EN SISTEMA MULTICORE

El WRF fue diseñado para usar paralelismo de grano grueso manejado por la capa de mediación (ver Figura 22). Dentro del modelo este tipo de paralelismo está implementado mediante paso de mensaje MPI(John Michalakes & Gill, 2016), sin embargo con el rápido desarrollo de los sistemas multicore, de las GPUs y de los aceleradores vectorial Intel KNC y KNL, se dio soporte a un tipo de paralelismo de grado fino soportado inicialmente mediante el estándar OpenMP, además de algunos esfuerzos de implementarlo mediante el

estándar OpenACC (John Michalakes & Vachharajani, 2008; Ponder et al., 2014).

La combinación de estos dos tipos de paralelismo en arquitecturas heterogéneas permite utilizar MPI para trabajo entre nodos y el OpenMP en nodos multicore. Este acercamiento no está libre de problemas de rendimiento, pues a medida que se aumenta la necesidad de intercambio de mensajería disminuye la eficiencia del procesamiento, de igual manera a medida que aumentamos el número de hilos por nodo, se puede causar sobre-suscripción, lo cual disminuye le eficiencia de procesamiento como se muestra en la Tabla 1. Eficiencia en el procesamiento multi-hilo y multiproceso

No. de procesadores	Tamaño de malla	%Eficiencia OpenMP	%Eficiencia MPI
1	74x61	100	100
2	74x31	72	98
4	37x31	65	91
8	37x16	31	83
16	19x16	16	70
32	19x8	8	56
64	10x8	3	40

Tabla 1. Eficiencia en el procesamiento multi-hilo y multiproceso
Fuente: Adaptación WRF tutorial 2016

Al incrementar el número de procesos y disminuir el tamaño de malla, las actualizaciones de las condiciones de frontera en cada sección aumentan confirmando el comportamiento de malla estructurada propias de los códigos esténcil (Bader, 2013; Kaltofen, 2012). Este comportamiento ge genera dependencia de vecinos, pues se requiere calcular para cada patch n el estado de sus vecinos con índices n+1 y n-1 (ver Figura 22). Esta dependencia genera un costo de ejecución por las tareas de comunicación que realizan la actualización, es por esta razón que determinar el número de

tareas MPI adecuadas es de vital importancia para lograr un balance entre rendimiento y costo de la mensajería. El proceso de actualización de las condiciones de frontera con los vecinos es conocido como la actualización del contorno o “halo update”, al considerar una corrida del modelo para un dominio suficientemente grande con una gran cantidad de patch; una consideración necesaria es evitar el incremento de las actualizaciones “halo update”, que afectan el desempeño y tiempo de generación del pronóstico. La distribución de las tareas de MPI y de OpenMP según el tipo de procesamiento en el modelo WRF se puede observar en la Figura 24. Modelo de ejecución en paralelo de WRF. El impacto que los “halo update” tienen en el desempeño total del modelo, han sido analizados en detalle por (Malakar et al., 2012)

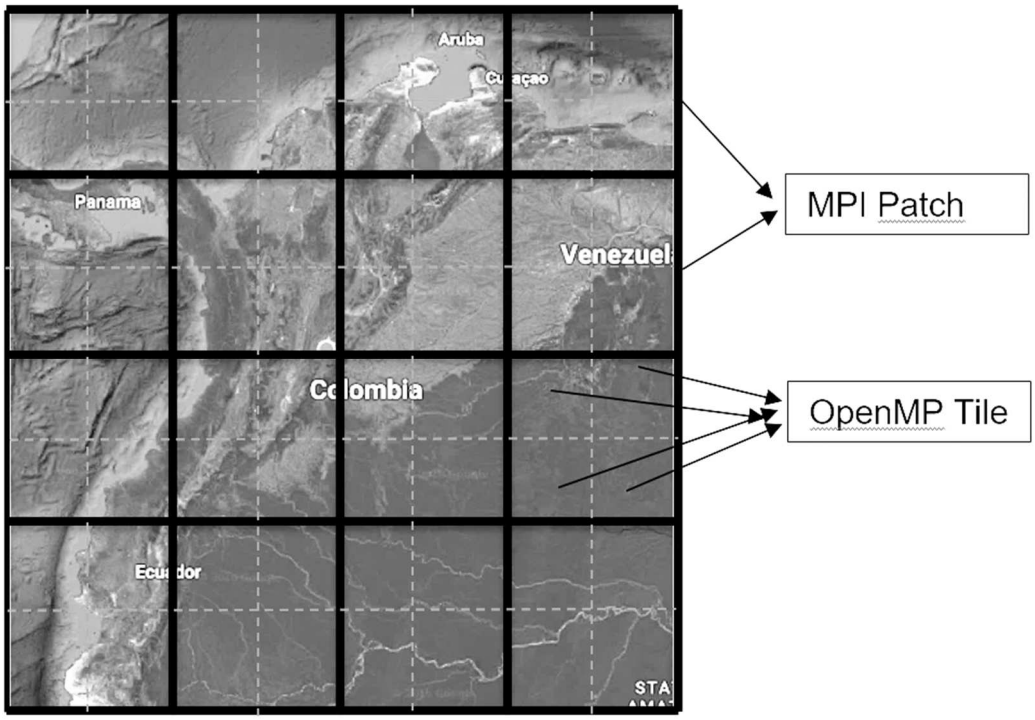


Figura 23. Modelo de descomposición de procesamiento en WRF.
Fuente: desarrollo propio

Dentro del procesos de ejecución del modelo, como ya se ha mencionado antes se presentan 2 tipos de paralelismo, pero solo dentro del paralelismo

de grano grueso (denominado patch dentro del modelo) se realizan los “halo updates”. Cada proceso MPI a la vez puede usar paralelismo de grano fino, generando regiones internas de procesamiento (denominadas en el modelo Tiles) las cuales son implementadas en OpenMP, la coordinación del procesamiento realizado por los tiles lo ejecuta la capa de mediación (Figura 22) y al finalizar la computación de todos los tiles el patch realiza un proceso de sincronización de memoria por proceso MPI. Este proceso es mostrado en la Figura 24. Modelo de ejecución en paralelo de WRF.

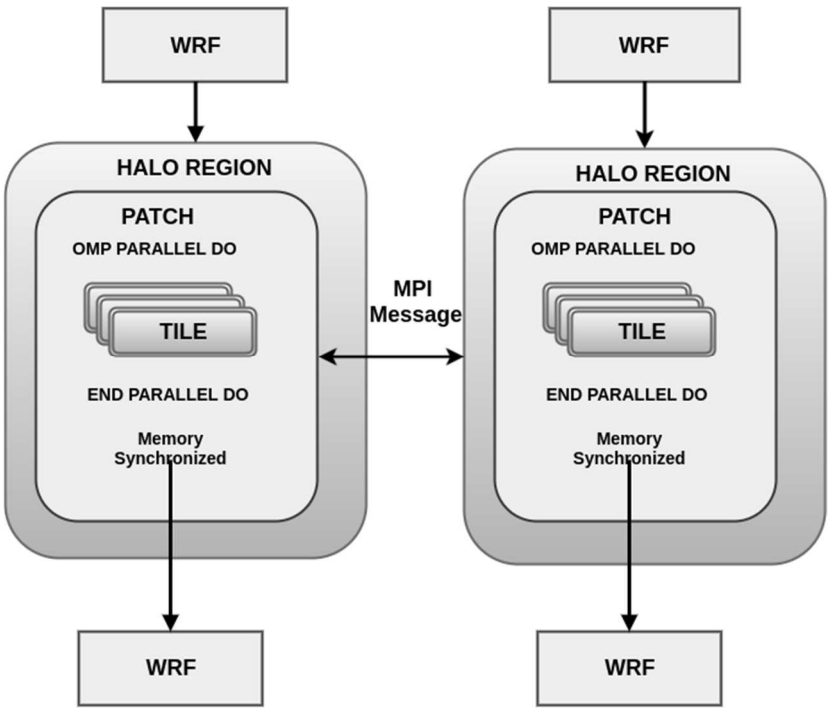


Figura 24. Modelo de ejecución en paralelo de WRF.
Fuente: Desarrollo propio

5.3. ESTRUCTURA DE CÓDIGO FUENTE DEL MODELO

El código fuente del modelo WRF se estructura acorde a sus funciones, a la capa del modelo que soporta y a tipo de servicio que provee. Esta estructura se puede observar en la Figura 25. Estructura general del código fuente del modelo WRF. El modelo está escrito principalmente en lenguaje Fortran77 y Fortran95, además de algunas secciones en lenguaje C y una pequeña

porción de archivos de configuración propios del modelo, como observa en la Figura 26. Distribución de la cantidad de archivos por su tipo. El modelo también distribuye su código de acuerdo a las capas del framework, ubicando aquellos artefactos que resuelven la dinámica y la física del modelo en carpetas independientes de acuerdo al tipo de núcleo que se utilice como se puede observar en la Figura 25.

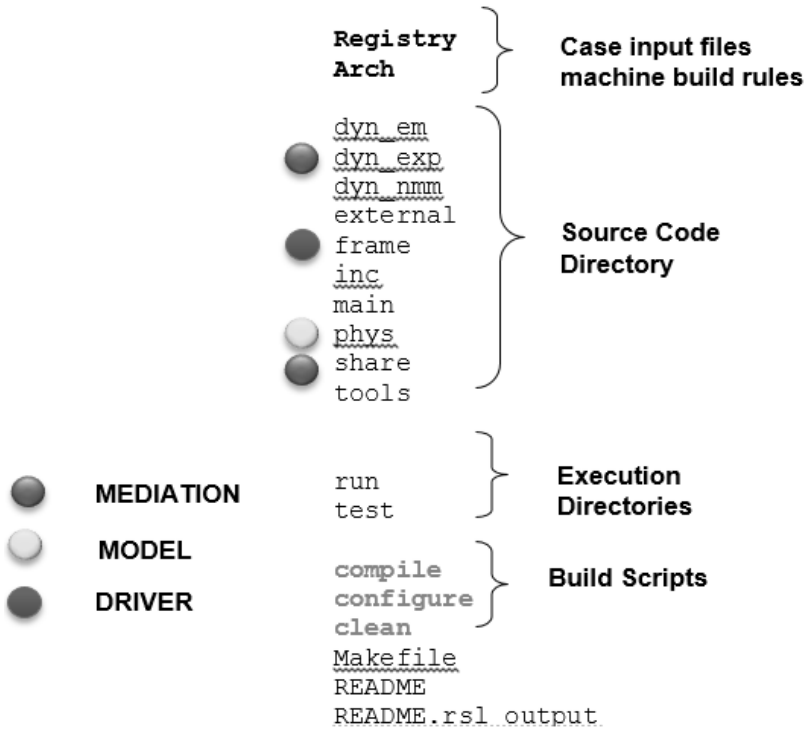


Figura 25. Estructura general del código fuente del modelo WRF
Fuente: Desarrollo propio adaptado del tutorial WRF 2016

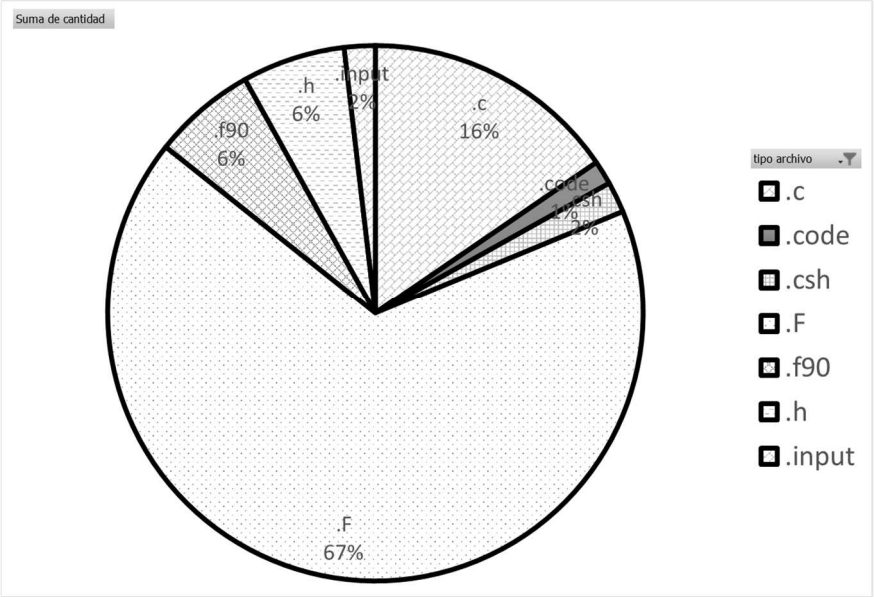


Figura 26. Distribución de la cantidad de archivos por su tipo
Fuente: Desarrollo propio

5.3.1. Capa de mediación

La capa de mediación realiza todos los procesos que tiene que ver con trabajo multicore y multinodo (tanto paralelismo de grano fino como paralelismo de grano grueso) por tanto, concentraremos el análisis en esta sección para ver las implicaciones que las rutinas de ciclos afectan el tiempo de ejecución del modelo. El proceso de invocación de dichas rutinas de acuerdo a los diferentes elementos del modelo se puede observar en la Figura 27. Proceso de invocación de rutinas en WRF. Las rutinas encargadas de realizar la integración en el tiempo y en el espacio (*x,y,z*) son denominadas solver y encuentran en la carpeta *solve_em* , en especial las rutinas encargas de inicializar los procesos de paralelismo (de memoria compartida y distribuida) se encuentran ubicadas en el archivo dym_em. Los procesos de paralelismo de grano se implementan en las rutinas de distribución en memoria distribuida y algunas rutinas de memoria compartida, que se encuentran ubicadas en la capa de mediación del modelo.

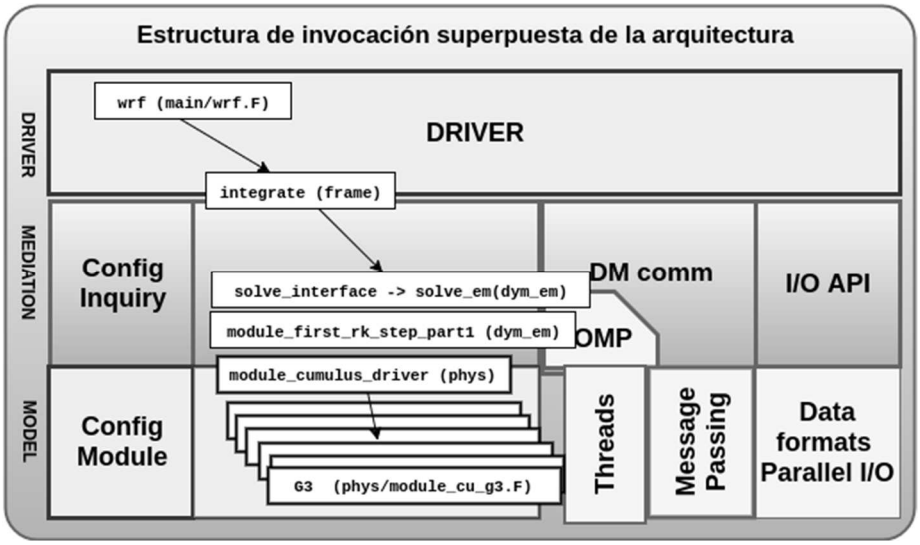


Figura 27. Proceso de invocación de rutinas en WRF.
Fuente: Desarrollo propio, adaptado del tutorial WRF2016

Las rutinas de manejo de multiproceso y multihilo, especialmente para los ciclos que definen la cantidad de patch y tiles son definidas en esta capa, observando una regla fundamental "El WRF no paraleliza todos los ciclos", solo paraleliza aquellas rutinas o ciclos que se pueden ser invocadas como como objetos completos para luego iterar sobre ellos en cada paso de tiempo, debido a que en esta resuelve la integración en tiempo para un dominio específico (John Michalakes & Gill, 2016; Wei Wang, Cindy Bruyère, 2016), siguiendo así un esquema de paralelismo de grano grueso.

En la Figura 28. Jerarquía de invocación de rutinas de mayor consumo en WRF, observa el modo como los ciclos son invocados dentro de un proceso de simulación y como existe un nivel de anidamiento en cada paso de tiempo; en esta gráfica podemos identificar la razón por la cual WRF usa paralelismo de grano grueso; las rutinas que más tiempo de procesamiento consumen están compuestas de una jerarquía de ciclos, estos ciclos de alto nivel pueden aprovechar el paralelismo de grano grueso mientras que rutinas intensivas aritméticamente como es el caso de las microfísicas (microphysics driver), pueden aprovechar mejor el paralelismo de grano fino(McCool et al., 2012).

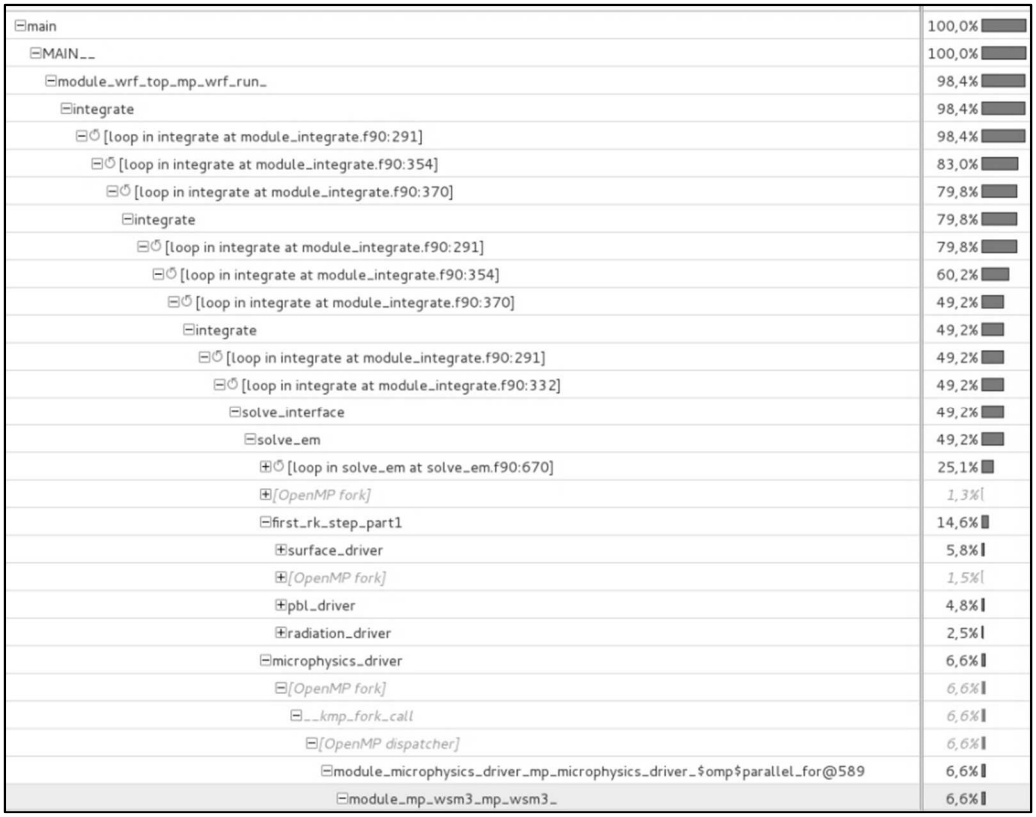


Figura 28. Jerarquía de invocación de rutinas de mayor consumo en WRF
Fuente: captura herramienta de perfilamiento

Es importante mencionar que la integración temporal que hace el solver debe cumplir el criterio de Courant el cual establece que para revolver numéricamente ecuaciones diferenciales en derivadas parciales el paso de tiempo no debe ser inferior a un valor determinado (ver Ecuación (25), de lo contrario la simulación producirá resultados incorrectos (Lewyt, 1967). El WRF realiza el proceso de integración temporal mediante el método de Runge-Kutta de tercer orden y requiere un valor de Courant Ud/dx < 1.73; las razones para este valor son analizadas en profundidad por (Skamarock & Dudhia, 2011)

$$U_t = L_{fast}(U) + L_{slow\,(U)} \qquad \text{Ecuación (25)}$$

5.4. CONCLUSION

El modelo WRF expone una sólida arquitectura para el aprovechamiento de sistemas de memoria compartida y ha incorporado las API de programación basada en directivas para el uso de CPU multicore, sin embargo, se ha sacrificado elementos de desempeño para mantener la coherencia y el bajo acoplamiento con el cuál fue diseñado. Realizar ajustes logrando explotar el paralelismo de grano fino, podría ofrecer mejores niveles de desempeño especialmente en los elementos de integración numérica de las microfísicas, pero comprometería el esquema de trabajo si no se establece un espacio en la capa de mediación que sea capaz de integrarlo de manera natural.

Acelerar una aplicación y en este caso específico el modelo WRF tiene implicaciones sobre la arquitectura de software establecida, los modelos de programación integrados y las arquitecturas de hardware soportadas, incluyendo infraestructuras de red y de almacenamiento compartido; aunque la mayor parte de la computación es realizada por los solver para resolver numéricamente la integración temporal, no es el único factor que debería considerarse como se ha hecho tradicionalmente(Sharma & Kulkarni, 2015). Acelerar el modelo requiere una visión integrada que cubra todo el proceso de generación de un pronóstico desde la obtención de los datos de condiciones iniciales hasta el post-procesamiento en los formatos adecuados para usuarios finales.

En el siguiente capítulo se realiza un análisis de aquellos factores que afectan el incremento en el desempeño de la ejecución del modelo en las diferentes fases y etapas que un pronóstico requiere. Se muestran además aquellos puntos claves que impiden que se pueda lograr un mayor speedup dependiendo de si se usa memoria compartida, memoria distribuida o una combinación de las dos.

CAPÍTULO 6. ANÁLISIS DE FACTORES COMPUTACIONALES QUE AFECTAN EL DESEMPEÑO DEL MODELO

El Modelo WRF puede ser compilado para correr en modo serial, en modo paralelo de memoria compartida y en modo paralelo de memoria distribuida (Wei Wang, Cindy Bruyère, 2016), en cada uno de estos tres escenarios existen factores que pueden afectar su rendimiento. En este capítulo se analizará dichos factores y el impacto que tienen sobre la ejecución en modo operativo en ambientes paralelos.

Como ya se mencionó en el capítulo 2, el modelo WRF consta de 3 componentes principales:

- Componentes de preprocesamiento,
- Componentes de Simulación (o resolución numérica)
- Componentes de posprocesamiento,

Al realizar el análisis del desempeño computacional solo se consideraron dos de ellos: El componente de preprocesamiento y el componente de simulación, dado que presentan el mayor tiempo computacional y es donde mejor se observa los diferentes tipos de paralelismo (paralelismo de grano grueso y paralelismo de grano fino)

6.1. ETAPA DE PREPROCESAMIENTO

En esta etapa se toman los datos obtenidos de los modelos globales para que establecer las condiciones iniciales en cada una de las mallas que se establecen al inicio de la configuración del modelo. Después de este paso se

inicia un preprocesamiento de los datos que incluye 3 fases (Wei Wang, Cindy Bruyère, 2016):

- La interpolación de los datos de información terrestre y la definición del dominio de simulación (geogrid)
- Trasformación del formato GRIB y formato intermedio de puntos de malla interpolados verticalmente (ungrib)
- La interpolación horizontal de los datos meteorológicos en el modelo de dominio (metgrid)

El flujo de ejecución de cada fase es mostrado en la Figura 29. Componentes del Sistemas de preprocesamiento WPS, permite observar que existe una cadena serial de ejecución previa a la simulación, la cual afecta el tiempo que se toma en preparar los datos para que la simulación wrf.exe pueda ser ejecutada.

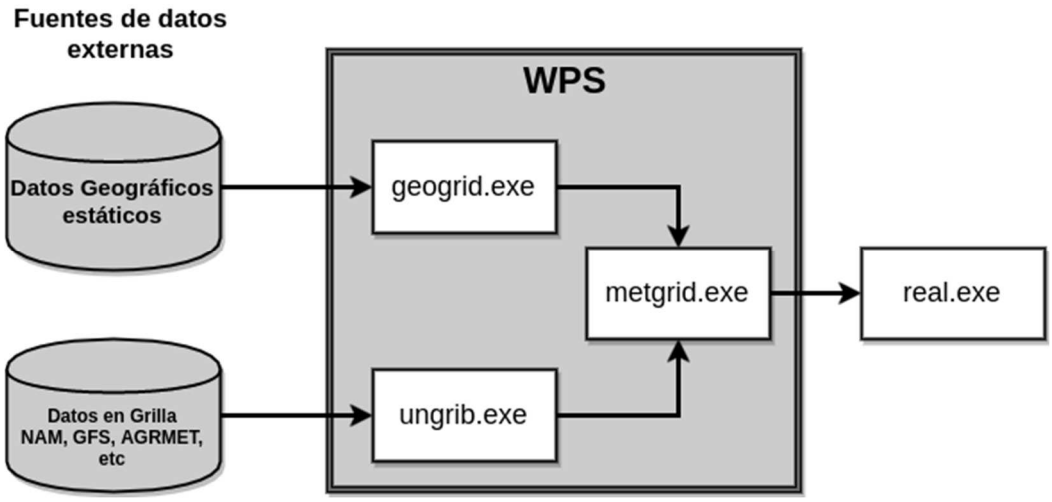


Figura 29. Componentes del Sistemas de preprocesamiento WPS
Fuente: desarrollo propio adaptación tutorial WPS

Tanto en memoria compartida, como en memoria distribuida el preprocesamiento, aporta un porcentaje considerable del tiempo de ejecución dado su naturaleza serial, y en especial la etapa de ungrib, ya que la lectura y escritura de datos en formato intermedio no puede ser ejecutada mediante

mpi (Wei Wang, Cindy Bruyère, 2016). De esta manera, geogrid y metgrib soportan el uso de múltiples procesos, aunque no aprovechan el uso memoria compartida, en caso de usar un dominio muy grande, se requiere usar soporte API de lectura/escritura paralela como parallel-NetCDF o pNetCDF, además de usar ajustes para escritura de archivos intermedios independientes.

El mayor problema de desempeño en esta etapa está ligado con el tamaño del dominio elegido, el número de subdominios y el espacio de tiempo para el pronóstico. Para ejecutar un pronóstico se requieren descargar los datos que contienen las condiciones iniciales provenientes de un modelo de escala global como el GFS o el ECMWF (Kanamitsu, 1989; Persson, 2015), el cual viene con resoluciones de 1 , ½ o ¼ de grado con pronósticos cada 3, 6 y 12 horas.

Al realizar un pronóstico con un horizonte de tiempo largo implica tener que usar mayor cantidad de datos de entrada en especial si se espera salida con mayor frecuencia (cada 3 horas). El tamaño del dominio elegido también tiene un efecto en el tiempo de procesamiento en especial cuando se elige un distancia de malla tanto horizontal como vertical (dx,dy) pequeña, pues con un dominio de gran tamaño y un tamaño de malla pequeño, aumenta el número de patch y por tanto la cantidad de mensajería entre procesos MPI también aumenta.

Adicional a los procesos de comunicación, están los procesos de interpolación vertical de datos que se deben realizar en cada uno de los niveles que contempla el modelo y para cada uno de los puntos de la malla, además de la escritura en una estructura intermedia (ungrib.exe) compatible en un formato que METGRID.exe pueda leer. Es de anotar que este proceso es de tipo serial y hasta el momento no permite aprovechar la ventaja de los sistemas multicore, por tanto en la medida que existan más datos iniciales el tiempo de procesamiento será mayor.

Leer y escribir datos entre formatos, cuando el tamaño de las escrituras aumenta tiene un efecto sobre el desempeño. Un estudio detallado de las implicaciones de la escritura de datos históricos fue realizado en (Porter, Ashworth, Gadian, & Burton, 2010), en donde se analizan los efectos de usar API de I/O paralela y la manera como se puede optimizar las escrituras designando nodos de escritura y nodos de procesamiento de datos. En caso de usar MPI para la interpolación vertical y horizontal, los tiempos de comunicación en mensajería deben ser considerados.

A pesar de los efectos analizados anteriormente, el tiempo de preprocesamiento es despreciable respecto del tiempo total de computación, correspondiendo a menos del 15% del miso utilizando APIs paralelas y hasta un 25% en caso de usar API exclusivamente seriales.

6.2. ETAPA DE SIMULACIÓN

Una vez los datos han sido preprocesados y se han generado los archivos en formato netcdf, se procede a inicializar e interpolar verticalmente los archivos generados por metgrid y establecer las condiciones de frontera y las condiciones iniciales, además de algunos chequeos de consistencia en la parametrización del modelo para casos de datos reales (real). En caso de compilar el modelo con soporte para memoria distribuida (dm) o para modelo híbrido memoria distribuida y memoria compartida (dm/sm), la capa de mediación ejercerá un rol al generar el paralelismo necesario (grano fino y grano grueso) requerido en cada patch y en cada tile dentro de patch.

En el proceso de integración numérica se resuelve tanto en tiempo (t) como en el espacio (x,y,z) las ecuaciones de balance (ver numeral 2.1.4. Ecuaciones fundamentales de los modelos).

Dentro del proceso de integración se involucra la parametrización de microfísicas, radiación de onda larga y corta, capas y modelos de superficie

y parametrización de las nubes. Múltiples ciclos de grano grueso se encargan de cada una de estas resoluciones como ya puede observar en la Figura 28. Jerarquía de invocación de rutinas de mayor consumo en WRF.

6.3. FACTORES QUE AFECTAN EL MODELO EN MEMORIA COMPARTIDA.

En sus dos componentes, al usar soporte para sistemas multihilo y aprovechar el modelo de memoria distribuida, se presentan los siguientes problemas que afectan el desempeño:

- Generar exceso de suscripción: Este problema se presenta cuando se generan más tiles que número de cores disponibles, asignando más de un tile por core generando continuos cambios de contexto de los procesos.
- Generar desbalance: Generar tiles OpenMP de diferente tamaño que generen trabajo homogéneo sobre todos los cores disponible en un nodo.
- Configuración de la pila de memoria por debajo de los límites permitidos en un nodo, de modo que no se puede aprovechar de manera eficiente por cada tile.
- En sistema manycores al no generar el suficiente paralelismo el desempeño obtenido es mucho menor que en procesadores tradicionales, ya que no se aprovechan las estructuras adicionales como instrucciones vectorizadas y gran cantidad de hilos disponibles.

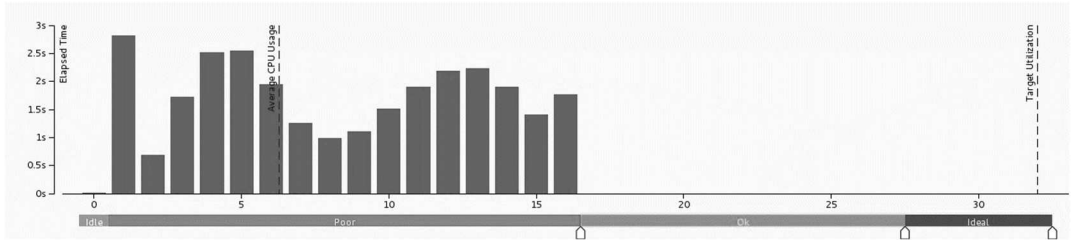


Figura 30. Eficiencia en el uso de CPU durante la corrida del modelo.
Fuente: Desarrollo propio, 6 horas de pronóstico, 8 threads y 8 CPU cores.

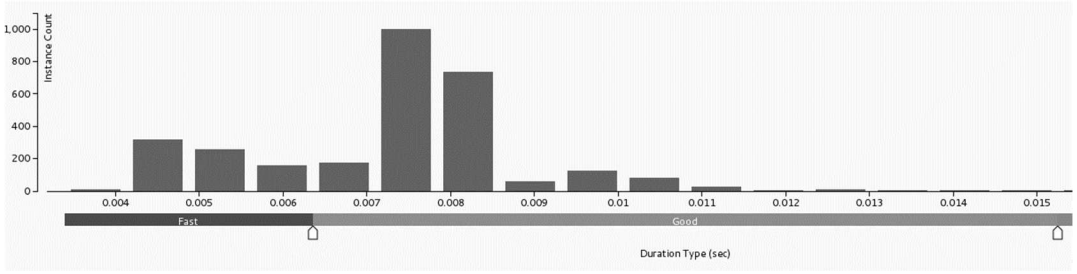


Figura 31. Duración de la computación en regiones OpenMP.
Fuente: Desarrollo propio: 6 horas de pronóstico, 8 threads en 8 CPU cores

En la Figura 32. Comparativo de rendimiento al usar 8 y 16 Threads OpenMP, se observar que al aumentar el número de Threads OpenMP, más allá de los cores físicos disponibles, se generan una degradación en el uso eficiente de las unidades de cómputo por efecto de la sobre suscripción.

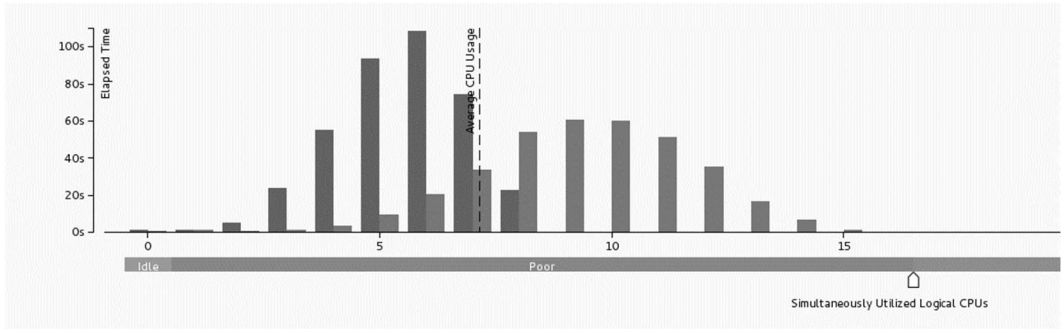


Figura 32. Comparativo de rendimiento al usar 8 y 16 Threads OpenMP
Fuente: Desarrollo propio. 6 horas de pronóstico, 8 CPU Cores.

6.4. FACTORES QUE AFECTAN EL MODELO EN MEMORIA DISTRIBUIDA.

Al tener múltiples nodos, el principal problema presentado es la división eficiente de la carga de trabajo en cada nodo, aprovechando los recursos que cada uno de ellos puede ofrecer. Este esquema de memoria distribuida sin embargo agrega elementos a la ecuación, pues al crecer el número de nodos, crece también el número de mensajes; las actualizaciones halo y las actualizaciones de condiciones de frontera, las cuales insertan tiempos adicionales dado que, durante los intercambios del mensaje, no se realiza computación haciendo que los procesos queden en espera hasta que la comunicación haya terminado (Malakar et al., 2012). Dentro de los procesos que se deben realizar esta las actualizaciones de condiciones de frontera, que ocurren para cada paso de tiempo, se deben actualizar las condiciones de las celdas de malla vecinas (códigos esténcil) tanto en la dirección x como en la dirección *y* , produciendo 4 actualización continua por celda.

El modelo compilado para soportar procesamiento distribuido (opción DM) presenta una sobrecarga por mensajería como se puede observar en la Figura 33. Distribución del tiempo de procesamiento en memoria distribuida, por su parte en la Figura 34 se puede observar para un periodo de 1.5 segundos de procesamiento las tareas de comunicación entre cada uno de los procesos MPI (líneas entre procesos), al igual que las operaciones propias de MPI (segmento rojo de cada proceso), con lo cual queda en evidencia el peso que las comunicaciones MPI tiene sobre un proceso de pronóstico en memoria distribuida.

En la figura Figura 33. Distribución del tiempo de procesamiento en memoria distribuida, se puede observar el impacto que tiene el aumentar el número de procesos MPI, cuando se correr sobre redes de alta latencia tipo Ethernet de 1GB, en caso de lanzar 8 ranks mpi el tiempo dedicado a la mensajería y

trabajo MPI puede equivaler al 38% del tiempo total de computación, mientras que para 2 ranks el tiempo involucrado está en el 10%. Este comportamiento se puede observar muy bien en la Figura 35, en donde se observa el intercambio de mensajes entre nodos y las latencias introducidas en tiempo de computación al tener que esperar que se realice la actualización del halo update antes de poder continuar con el procesamiento.

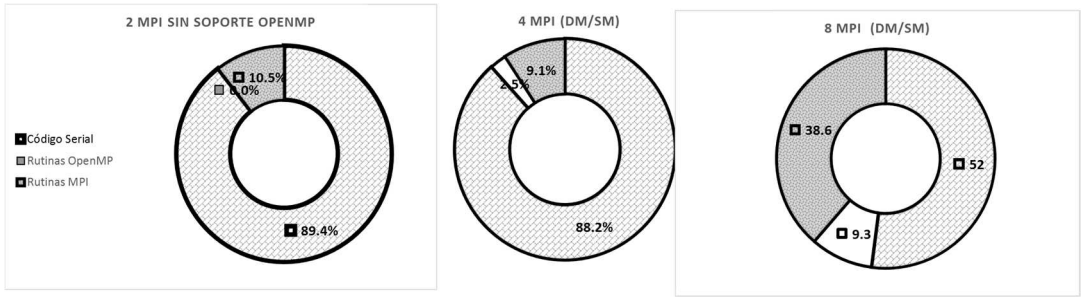


Figura 33. Distribución del tiempo de procesamiento en memoria distribuida
Fuente: Desarrollo propio, para 3 dominios, y 6 horas de pronóstico

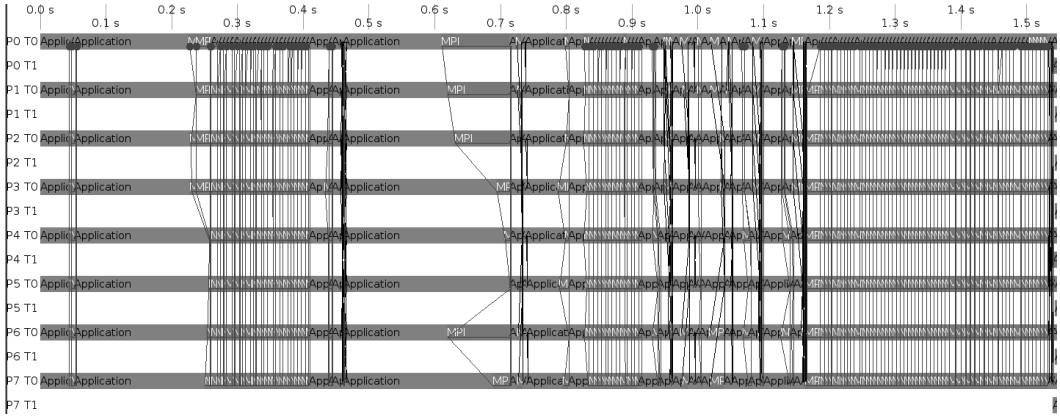


Figura 34. Inicialización de las cargas de trabajo para modelo WRF.
Fuente: Desarrollo propio para 8 procesadores y 8 MPI ranks.

En la Figura 35, al igual que en la Figura 34, se puede observar las actualizaciones que hace cada proceso (líneas negras entre proceso+) con sus vecinos para actualizar información además del tiempo exclusivo que requiere el manejo de mensajes MPI, podemos entonces observar que a medida que el número de patch aumenta, la mensajería aumenta y que

dependiendo del tipo de infraestructura de red que se posea esta puede llegar a ser una buena parte del tiempo total de computación.

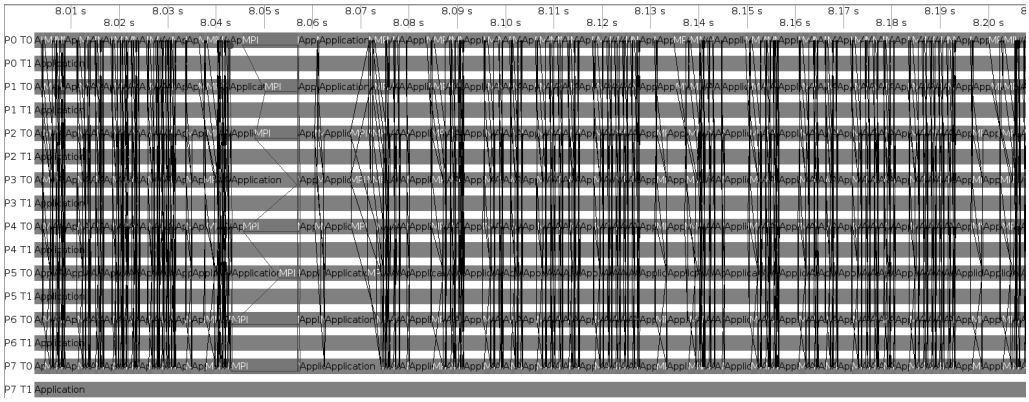


Figura 35. Intercambio de información en operaciones de halo update Fuente: Desarrollo propio, para corrida con 8 ranks para 6 horas de pronóstico

Las operaciones de I/O que se realizan también impactan de modo significativo el tiempo de procesamiento, dado que se deben leer y escribir continuamente las condiciones iniciales de los archivos interpolados y luego escribir los resultados parciales mediante archivos históricos para ir actualizando la salida del pronóstico en formato NetCDF.

En la Figura 36. Operaciones de I/O del modelo durante ejecución se puede observar las latencias introducidas por las escrituras a discos (que ocurren frecuentemente cuando se están actualizando las condiciones en la integración numérica) presentadas por las líneas azules. Las líneas moradas indican el tiempo que se está a la espera que una operación de I/O termine y en la parte inferior se puede observar la duración de la operación misma (más larga mayor duración), por tanto, una pronóstico de larga duración implicará que se utilicen técnicas que minimicen el impacto del I/O sobre el tiempo total de simulación.

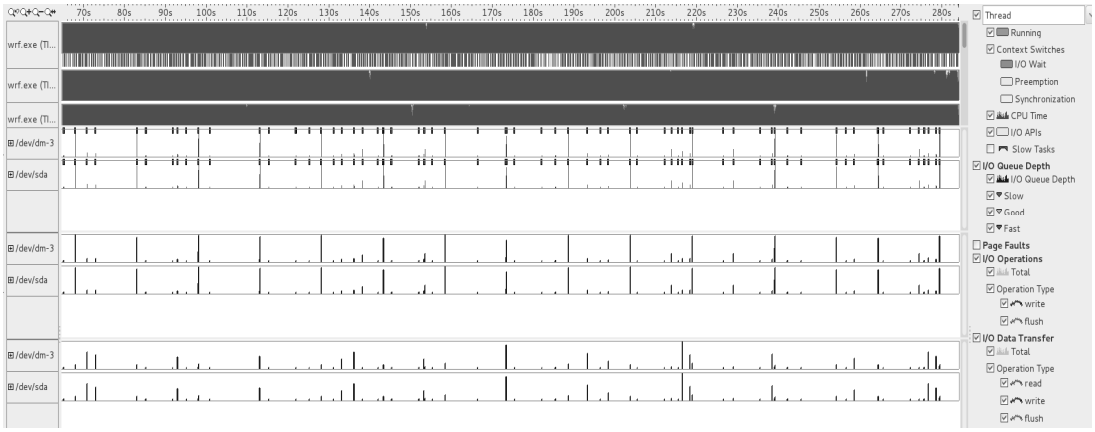


Figura 36. Operaciones de I/O del modelo durante ejecución
Fuente: Desarrollo propio: 6 horas de pronóstico y 8 ranks MPI

6.5. CONCLUSIÓN

El modelo WRF en sus diferentes aplicativos y etapas de generación del pronóstico, ofrece oportunidades para incrementar su desempeño siempre que sorteen exitosamente las barreras presentadas en cada una de las arquitecturas computacionales existentes dentro de esquemas heterogéneos paralelos, por lo tanto el mejor enfoque posible para poder escalar en una plataforma con múltiples nodos, múltiples procesadores por nodo, múltiples cores por procesador y múltiples registros vectoriales por core, resulta ser es un esquema balanceado donde se minimiza la mensajería entre nodos, aprovechando los esquemas de paralelismo de grano grueso, se divide las tareas de computación aprovechando la arquitectura del procesador en cada nodo (especialmente los temas de localidad, la adyacencia de hilos y fijación de cores), evitando la sobresuscripción de procesos y las transferencia de datos entre dispositivos internos.

En el siguiente capítulo se muestra cómo se utilizaron los conceptos teóricos, las herramientas, la experimentación de las diferentes técnicas exploradas para acelerar el pronóstico del estado del tiempo sobre un dominio que cubre el territorio colombiano y los elementos que permiten acelerar el modelo bajo unas condiciones propuestas.

Además, muestra la manera como el modelo se comporta en las arquitecturas de memoria compartida y distribuida, al igual que en sistemas con procesadores vectoriales.

CAPÍTULO 7. RESULTADOS DE LA ACELERACIÓN DEL MODELO

El proceso para acelerar la ejecución de componente de software, requiere analizar elementos de infraestructura, del modelo de programación y de los algoritmos que utilizan.

En este capítulo se mostrará cómo se abordó el proceso de análisis e implementación de técnicas de aceleración usadas para el modelo WRF, enfatizando en aquellos aspectos que permiten ofrecen el mejor speedup.

7.1. CARACTERIZACIÓN DEL HARDWARE UTILIZADO

Para el desarrollo de la presente investigación, se utilizaron 3 tipos de plataformas características:

- Un servidor muticore, configurado en modo NUMA con un acelerador vectorial, como prototipo de un esquema de memoria compartida.
- Un cluster de procesadores multicore como prototipo de un esquema de memoria distribuida basada en cpu.
- Un cluster de aceleradores manycores como prototipo de un esquema de memoria distribuida con aceleradores.

SERVIDOR INTEL XEON PHI KNC	
Tipo Procesador	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
No Socket	2
Cores por Socket	8
Threads por core	2
Coprocesador	Intel Xeon Phi KNC 3120A
Memoria	64GB – DDR4
Compiladores	Intel Parallel Studio 2016

Tabla 2. Caracterización Nodo Heterogéneo de memoria compartida

Este nodo presenta un diseño especial dado que, al tener 2 procesadores, uno de ellos se encarga de administrar los dispositivos de red y de almacenamiento, mientras el segundo procesador se especializa en administrar los dispositivos conectados mediante bus PCI. La importancia radica en que, precisamente el acelerador vectorial KNC se conecta mediante este tipo de bus. La descripción completa de la arquitectura de este nodo se encuentra detallada en el anexo No 1, Figura 61.

CLUSTER SABIO-I	
Tipo Procesador	Intel(R) Xeon(R) CPU X5570 @ 2.93GHz
No Socket	2
Cores por Socket	4
Threads por core	2
Coprocesador	No
Memoria	16GB – DDR3 x Nodo
No Nodos	10
Red	2Gb en Team 1Gbx2 en cada nodo
Compilador	Intel parallel Studio 2016
Numa config	0 (0,16)(1,17)(2,18)(3,19)(4,20)(5,21)(6,22)(7,23) 1 (8,24)(9,25)(10,26)(11,27)(12,28)(13,29)(14,30)(15,31)

Tabla 3. Caracterización cluster multicore de memoria distribuida

CLUSTER STAMPEDE-KNL	
Tipo Procesador	Intel(R) Xeon(R) Phi KNL x200

No Socket	1
Cores por Socket	72
Threads por Core	4
Coprocesador	No
Memoria	16GB – DDR4 x Nodo
No Nodos	12
Red	2Gb en Team 1Gbx2 en cada nodo
Compilador	Intel Parallel Studio 2016

Tabla 4. Caracterización cluster manycore en memoria distribuida

7.2. PARAMETRIZACIÓN UTILIZADA DEL MODELO

Para la ejecución del conjunto de experimentos en el modelo se utilizó un dominio que cubriera el territorio colombiano, un subdominio centrado en la región del altiplano cundiboyacense y un tercer dominio centrado en Bogotá.

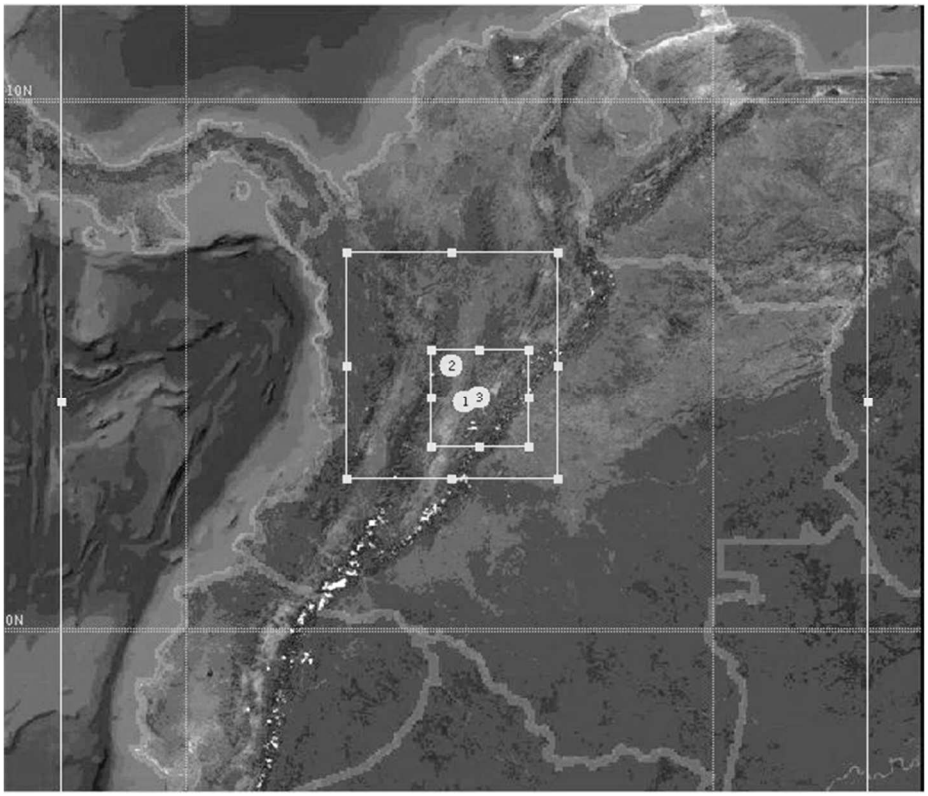


Figura 37. Configuración de los dominios utilizados en los experimentos Fuente: Desarrollo propio, salida obtenida por el Domain Wizard.

La configuración del modelo y la parametrización utilizada se detalla en la siguiente tabla.

Característica	Dominio 1	Dominio 2	Dominio 3
Resolución geográfica de los datos	5m	5m	30s
Tipo proyección	Mercartor		
Resolución del plano x	16900 (metros)		
Resolución del plano y	16900 (metros)		
Core para dinámica utilizado	ARW		

Datos para condiciones iniciales	**Modelo GFS resolución 0.5 grados (55km) aprox**

Tabla 5. Parámetros generales para el modelo WRF utilizados para el pronóstico

Característica	**Dominio 1**	**Dominio 2**	**Dominio 3**
Niveles verticales	**32**	**32**	**32**
Distancia de malla en x (mts)	**16900**	**5633**	**1877**
Distancia de malla en y (mts)	**16900**	**5633**	**1877**
Paso de tiempo	**101 segundos**		
Microfísicas	**3,14**	**3,14**	**3,14**
Core para dinámica utilizado	**ARW**		
Horas del pronóstico	**6,12,18,24,72**		

Tabla 6. Parametrización de cada dominio para simulación con WRF

Los archivos namelist.wps y namelist.input utilizados se encuentran publicados en el dataset bajo la denominación “Hernandez, Esteban (2017), “Dataset of WRF profiling”, Mendeley Data, v2 http://dx.doi.org/10.17632/fjgjyj55wm.2”

7.3. PROCESO DE OPTIMIZACIÓN Y ACELERACIÓN.

Se procedió a realizan un análisis tipo top-down para optimización de los esquemas de memoria distribuida (Hager & Wellein, 2011) y para los esquemas de memoria compartida se utilizó la técnica Roofline (Williams, Waterman, & Patterson, 2009), mediante la cual se compara la intensidad aritmética de las operaciones, el uso de las jerarquías de caché y el desempeño de las unidades de punto flotante.

7.3.1. OPTIMIZACIÒN DEL MODELO EN MEMORIA DISTRIBUIDA

Dada la alta dependencias de mensajería para procesas los diferentes patch en que se divide la malla del modelo, se procede mediante procesos de benchmarking usando el modelo para determinar la mejor distribución de ranks que se debe utilizar dado en redes de alta latencia (dado que son las únicas para la presente investigación disponibles).

Se generó un conjunto de experimentos sobre el cluster SabioI que permite observar el impacto que tiene el incremento del tiempo destinado a la mensajería sobre tiempo total de procesamiento, iniciando con un proceso por nodo hasta un máximo de 8 procesos por nodo. El límite ha sido impuesto para no violar la norma de 1 proceso por core disponible.

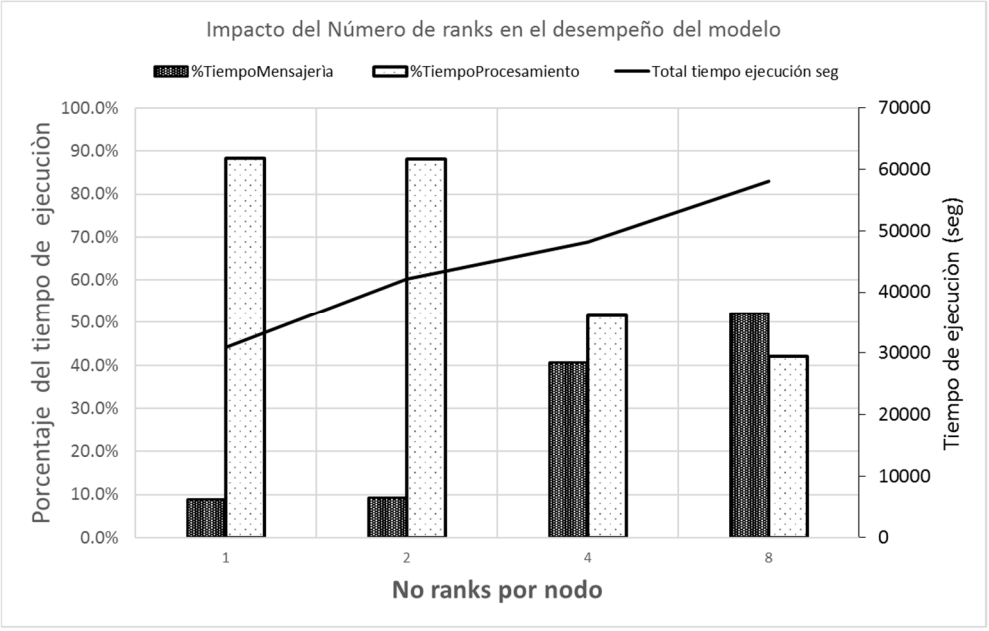


Figura 38. Impacto del número de ranks el desempeño del modelo. Fuente desarrollo propio

Este experimento permite observar que al generar mayor número de procesos MPI, el porcentaje del tiempo que se destina al procesamiento disminuye produciendo que tiempo total de la simulación aumente.

Al indagar sobre las razones por las cuales la mensajería aumenta de esta manera, debemos recordar la manera como el modelo divide su procesamiento basado en patch equivalente al número de procesos MPI y la manera en la que cada uno de estos patch requieren hacer halo-updates de manera continua, para actualizar las condiciones de manera regular. Además, tenemos los intercambios de las condiciones de contorno que necesitan ser comunicadas entre celdas siguiendo el modelo stencil(Kjolstad & Snir, 2010), y las condiciones de frontera que necesitan ser conocidas por los vecinos.

Mediante el perfilamiento de la mensajería encontramos por ejemplo que al utilizar 4 ranks por nodo y 4 thread por rank, el número de mensajes puede alcanzar en los nodos más activos 880.000 mensajes, haciendo que gran parte del tiempo se gaste en el proceso de envio/recepción, agregando retrasos acumulativos al tiempo de procesamiento total.

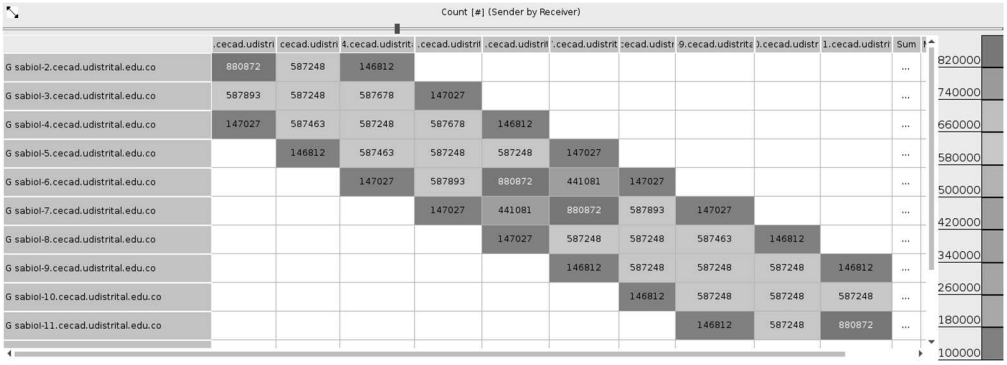


Figura 39. Cantidad de mensajes intercambiado por cada nodo
Fuente desarrollo propio, calculados para 6 horas de pronóstico

Este intercambio de mensajes además de ser numeroso, incrementa las comunicaciones entre cada nodo. Es así que, para 20 segundos de procesamiento el número de mensajes generadas entre cada uno de los

nodos lanzando 8 ranks MPI por nodo, puede alcanzar el 37% del total del tiempo de procesamiento; este porcentaje de deriva no solo del intercambio de mensajes, sino del tiempo que debe esperar cada nodo para que sus vecinos actualicen las condiciones de frontera como se puede observar en la Figura 40. Intercambio de mensajes MPI para 4 ranks por nodo. y en la Figura 41. Distribución de tiempo en cluster sabio I, para 8 ranks por nodo.

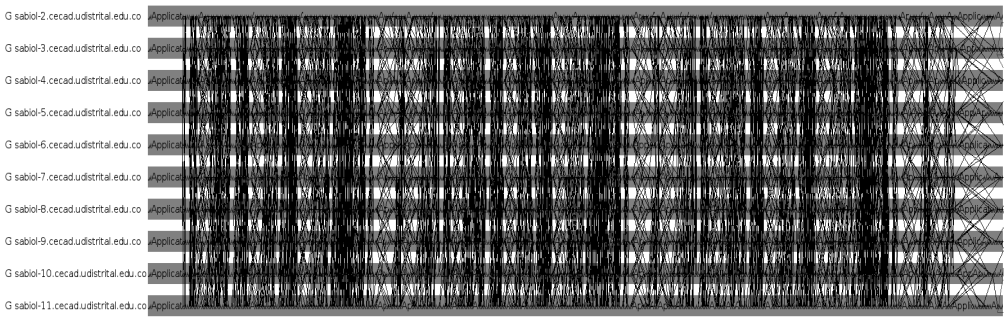


Figura 40. Intercambio de mensajes MPI para 4 ranks por nodo. Fuente desarrollo propio para 6 horas de pronóstico

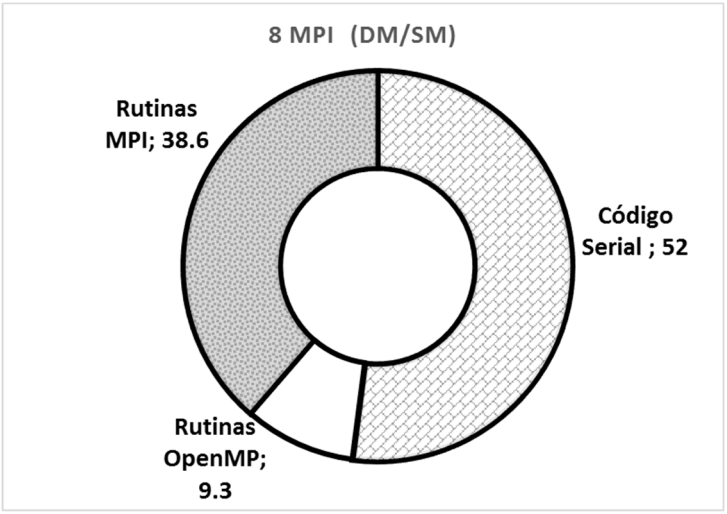


Figura 41. Distribución de tiempo en cluster sabio I, para 8 ranks por nodo. Fuente desarrollo propio

Además, el tráfico generado alcanza los límites existentes en la plataforma física de red que soporta la comunicación del cluster (1.5GB Efectivo), como

se muestra en la Figura 42. Consumo de Red para 4 procesos y 4 threads con 10 Nodos.

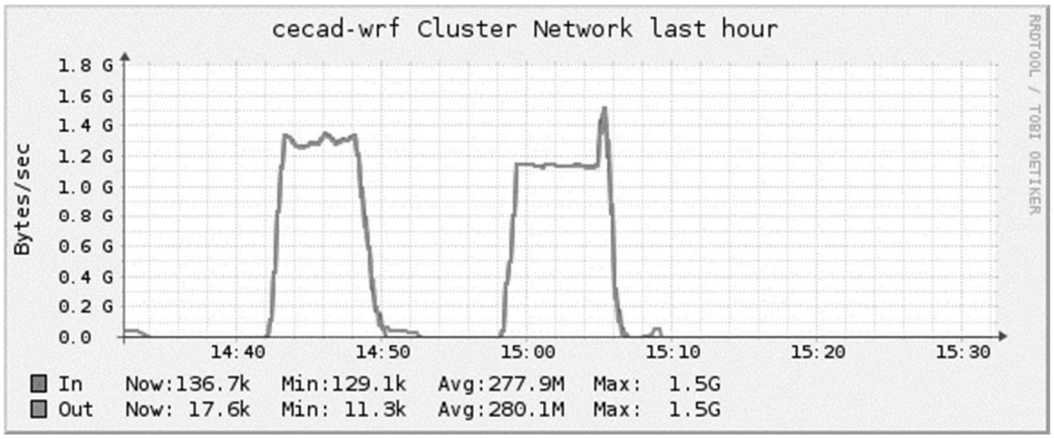


Figura 42. Consumo de Red para 4 procesos y 4 threads con 10 Nodos. Fuente desarrollo propio para 6 horas de pronóstico.

Si bien el número de mensajes y consumo de ancho de banda aumentan, el resultado es un mayor uso del procesador, lo significa un mejor aprovechamiento de los recursos (Figura 43. Consumo de CPU para 4 ranks por nodo., sin embargo, mayor uso del recurso de cómputo no significa mejor eficiencia, según lo expuesto por la ley de Amdahl expuesta en el capítulo 4, lo cual exige que se analice la naturaleza de la ejecución que ocurre en los procesos.

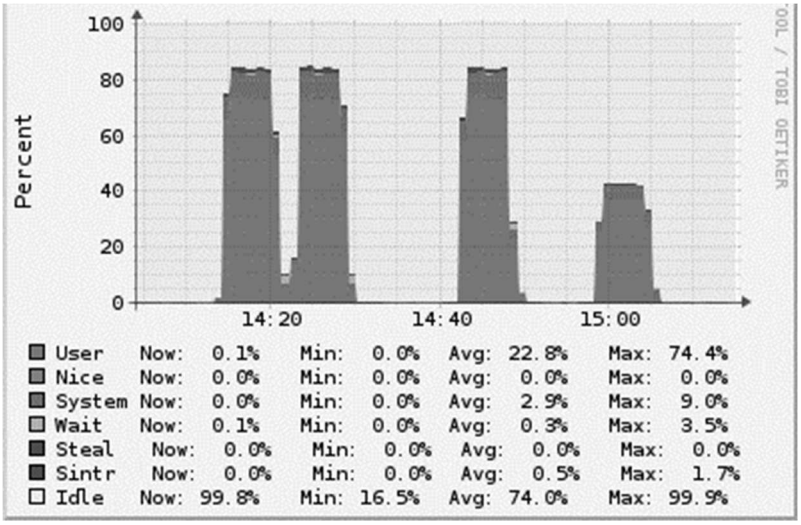


Figura 43. Consumo de CPU para 4 ranks por nodo.

Fuente desarrollo propio para 6 horas de pronóstico.

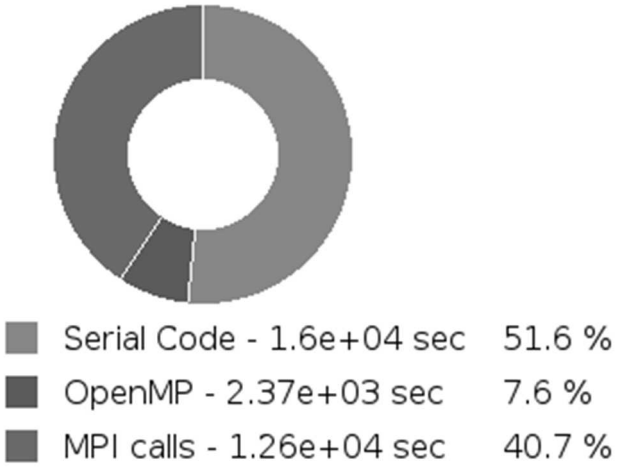


Figura 44. Análisis del tipo de ejecución para 8 ranks MPI.
Fuente Desarrollo propio para 6 horas de pronóstico

El análisis del tipo de ejecución nos muestra que gran parte de la ejecución es serial (51.6%) y solamente el 7.6% aprovecha de manera intensiva el paralelismo en cada nodo.

Si bien, al variar el número de ranks MPI el porcentaje de tiempo de computación y el tiempo de mensajería varía, disminuyendo el tiempo efectivo de computación, también es claro que ese tiempo en su mayoría corresponde a ejecución serial. Por la ley de Amdahl (ELEMENTOS TEÓRICOS PARA EL ANÁLISIS DE RENDIMIENTOS DE APLICACIONES.), entendemos que el speedup posible está limitado por la fracción de código serial que presente la ejecución de un algoritmo, por lo cual debemos analizar, si al incrementar el número de PE, presenta un escalamiento fuerte o débil e intentar incrementar el grado de paralelismo que se presenta dentro de la ejecución.

Como meta para lograr el modelo a nivel de infraestructura, debemos disminuir la sobrecarga producida por la mensajería al igual que el tiempo de ejecución del modelo, mediante el incremento del uso del recurso de

computación, que en nuestro experimento consiste en incrementar el número de procesos OpenMP (WRF Tiles) que se ejecutan por nodo y observar el tiempo de ejecución de la simulación, además del uso efectivo de los recursos.

Este análisis muestra que a partir de 4 threads por nodo el tiempo total de ejecución no presenta una mejoría notable (menos del 2%) de diferencia, y el porcentaje de tiempo de computación respecto del tiempo de mensajería presenta una diferencia considerable (entre el 35% y 40% respecto del uso de 2 hilos por nodo) como se puede observar en la Figura 45. Tiempo de ejecución vs número de hilos por nodo.

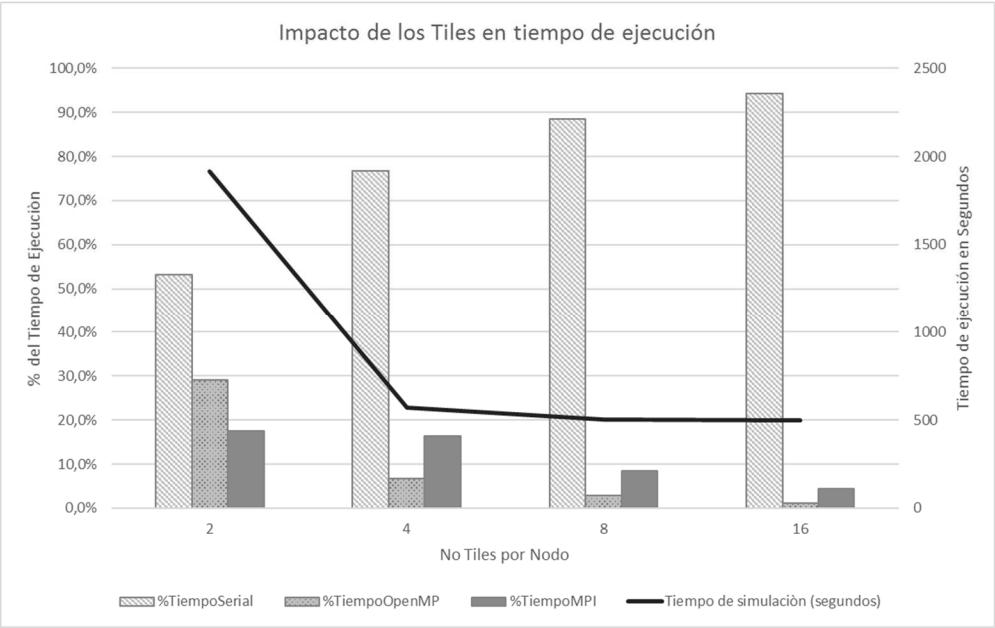


Figura 45. Tiempo de ejecución vs número de hilos por nodo
Fuente: Desarrollo propio para 6 horas de pronóstico.

Se puede inferir que no hay diferencia entre utilizar 4,8 o 16 Tiles por nodo en cuanto al tiempo de procesamiento del modelo se refiere, sin embargo, además de acelerar el modelo se debe tener en cuenta la eficiencia en el uso de los recursos como se observa en la Figura 46. Impacto de los tiles en el uso de recursos, en donde se puede observar que un considerable

incremento en el uso de los recursos (el doble de ellos) no representa una aceleración en el tiempo de ejecución del modelo, con lo cual se puede inferir que la mejor distribución de TILES por nodo debe ser entre 4 u 8, el uso de más TILES genera sobresuscripción de hilos sobre PE existentes, que no permiten mejorar el tiempo de ejecución.

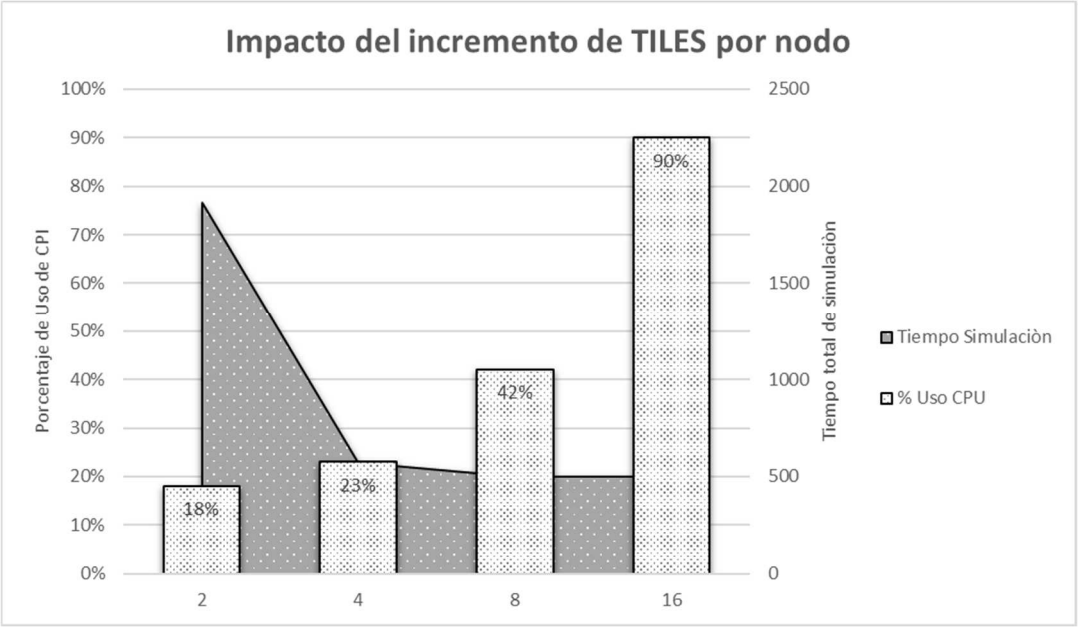


Figura 46. Impacto de los tiles en el uso de recursos
Fuente: Desarrollo propio

Tomando el análisis anterior se procede a ajustar el experimento haciendo variar el número de hilos y el número de ranks, para determinar la mejor distribución de recursos posibles que minimicen el tiempo de ejecución, mostrando la siguiente tabla de resultados:

No ranks MPI	1	2	2	4
No threads OpenMP	8	4	8	2
Tiempo de ejecución	08:25	07:16	07:29	09:19
%Tiempo MPI	8.5%	18.1%	9.4%	38.6%

%Tiempo OpenMP	2.9%	5.3%	2.3%	9.3%
%Tiempo procesamiento	88.4%	76.5%	88.2%	52%
%Uso de CPU	37.2%	36.24%	74.10%	29.61%

Tabla 7. Impacto de ranks y tiles sobre el tiempo de ejecución

De esta manera el menor tiempo de ejecución lo presenta una distribución de 2 ranks por nodo y 4 tiles por rank, junto con el menor promedio de uso de CPU.

En adelante esta será tomada como la distribución de referencia para acelerar el rendimiento del tiempo de ejecución del modelo.

7.3.2. OPTIMIZACIÓN BASADA EN LOCALIDAD

Como se puede observar en el Anexo 1, **¡Error! No se encuentra el origen de la referencia.**, el nodo tiene una configuración NUMA (Lameter, 2013). Esta configuración establece tiempos diferenciales de acceso a memoria dependiendo de la cercanía que este tenga con sus bancos de memoria y duplicando el tiempo de acceso para las memorias no locales. El objetivo es siempre hacer que los cores utilicen sus memorias locales pues resulta en un mejor tiempo de acceso. También se establece jerarquías de caché que son compartidas por algunos grupos de cores, de tal manera que, los datos que son accedidos desde esos cores pueden aprovechar las ventajas de tener diferentes niveles de caché. Cada jerarquía de caché tiene diferentes tamaños, configuraciones y velocidades de acceso, como se puede ver en la Tabla 8. Niveles de agrupación en core para uso de memorias cachés. Para la presente investigación es de especial atención los cachés de nivel L3, ya que, al colocar múltiples hilos de procesamiento, se intenta que la adyacencia pueda aprovechar el hecho de poder tener un espacio de alta velocidad compartida por los cores. De esta manera se siguió la estrategia de hacer fijación de procesador a cada proceso mpi (core pinning) y luego utilizar la

afinidad de los hilos de cada proceso para que puedan compartir el caché de nivel L3.

Cache	Tamaño	Core de procesamiento
L1	32 KB	(0,8)(1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)
L2	256 KB	(0,8)(1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)
L3	8 MB	(0,2,4,6,8,10,12,14)(1,3,5,7,9,11,13,15)

Tabla 8. Niveles de agrupación en core para uso de memorias cachés

Entendiendo que en la configuración de referencia se usan 2 ranks, la distribución lógica de caché es aquella que permita usar de manera efectiva nivel de cache L3, en donde aparecen 2 dominios de agrupación. Intentar utilizar una distribución de dominios, es decir que aproveche el caché de nivel, crea tiempos adicionales de coordinación y sincronización que introducen latencias considerables haciendo que el tiempo total aumente considerablemente como se observar en la Tabla 9. Impacto de la distribución de dominios de hilos, también se puede observar en la Figura 47. Distribución del tiempo de CPU (usuario/sistema), las 2 primeras ejecuciones, el tiempo de sistema es pequeño respecto del tiempo de usuario, mientras que en las últimas 2, el tiempo de sistema supera el tiempo de usuario, lo cual produce un aumento del tiempo de ejecución del modelo.

Método de agrupación	core	numa	cache L2
Tiempo de ejecución (min)	12.04	07:15	12:06
%Uso de CPU (usuario)	7.93%	39.24%	7.8%

Tabla 9. Impacto de la distribución de dominios de hilos

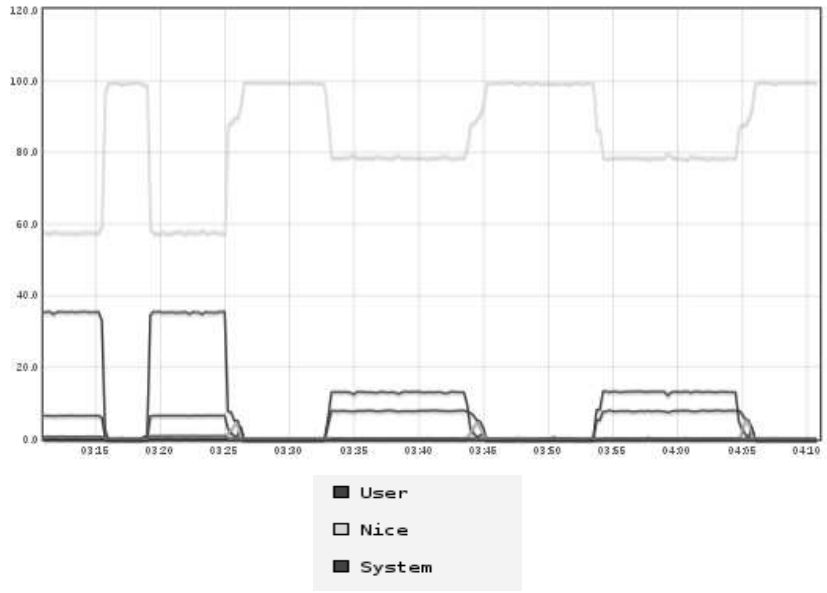


Figura 47. Distribución del tiempo de CPU (usuario/sistema)
Fuente: Desarrollo propio

7.3.3. CONCLUSION 1:

Acelerar una aplicación de memoria distribuida implica el afinamiento de los parámetros que permitan disminuir el tiempo de computación, disminuir la mensajería intercambiada entre nodos, aumentar el uso de los recursos de procesamiento y aprovechar de la mejor manera aquellos elementos de localidad de memoria.

En la presente investigación los resultados demuestran que la mejor distribución coloca 1 rank por cada procesador físico con que cuente el nodo, y el número de threads óptimos equivale al número de cores que cada procesador ofrece. Utilizar técnicas de Hiperthreading resulta en una desmejora del tiempo de ejecución originada en la sobre-suscripción de hilos en cada procesador.

7.3.4. OPTIMIZACIÓN DEL MODELO EN MEMORIA COMPARTIDA

Una vez analizados los elementos que ofrecen mejor potencial de aceleración dentro de una configuración de memoria distribuida y habiendo observado que gran parte del procesamiento es serial, se procede a analizar el desempeño del modelo por nodo, de tal manera que se pueda lograr aumentar el speedup local y por tanto el speedup general.

En el anterior numeral se determinó que la mejor distribución posible era 1 rank por procesador y con un número de tiles equivalente al número de cores disponibles (sin HyperThreading), con lo cual se procede a realizar el análisis para el nodo Xeon Phi descrito en la Tabla 10. Rendimiento por configuración en nodo memoria compartida, utilizando las siguientes configuraciones:

No Tiles	MPI Ranks	Tiempo medio Simulación (min)
4	2	5:54
8	2	4:29
16	1	6:51
16	0	6:50
4	4	7:39

Tabla 10. Rendimiento por configuración en nodo memoria compartida

Se considera correr el modelo con y sin soporte para MPI, dado que se analiza un único nodo y que se puede incrementar el número de TILES para que genere mayor ocupación computacional; en la serie de ejecuciones realizadas el principal factor analizado es el tiempo ejecución del modelo por lo cual los cambios en la distribución en las cargas de trabajo buscan alcanzar a este fin.

Se puede observar los siguientes elementos principales.

1) La sobrecarga introducida por el uso, de 1 proceso MPI, en relación con la no utilización es despreciable. Sin embargo, no utilizar MPI impediría que el modelo corra en una arquitectura de memoria distribuida impidiendo su escalamiento horizontal.
2) El tiempo promedio de la simulación fue de 6:20 minutos para un solo nodo, lo cual resulta menor comparada con el rendimiento obtenido por 10 nodos de la primera sección que fue de 08:07 minutos.

Esta diferencia de rendimiento por nodo, nos obliga a indagar aquellas características específicas de la arquitectura que presentan cada una de las plataformas y explotarlas para lograr un mejor nivel de speedup por nodo, según la siguiente tabla:

	Nodo-xeon-phi	Nodo-cluster-sabio
Nombre	Intel(R) Xeon(R) CPU E5-2630 v3	Intel(R) Xeon(R) CPU X5570
Microarquitectura del procesador	Haswell-EP	Nehalem
Procesadores	2	2
Cores x procesador	8	4
Threads x core	2	2
Soporte instrucciones vectorizadas	AVX2-256 bits	No
Velocidad del Bus	8 GT/s QPI	6.4 GT/s QPI
Cache-L3	20 MB	8 MB
Frecuencia	2.4 Ghz	2.9 Ghz
Memoria	64GB DDR4 2.1 Mhz	16GB DDR3 1.3 MHz

Tabla 11. Comparativa nodos memoria distribuida vs memoria compartida

Algunas diferencias tecnológicas son determinantes para lograr un incremento en el desempeño cuando se resuelven problemas de diferencias

finitas usando códigos stencil(Kjolstad & Snir, 2010), dentro de las cuales tenemos:

- Capacidad de soportar instrucciones vectorizadas.
- Tamaño y tipo de cachés utilizados
- Cantidad de cores por procesador.

El impacto que estos elementos tienen en el desempeño de una aplicación es analizado a fondo por (Zumbusch, 2012). A partir de estos elementos el análisis se centrará en la manera como el modelo puede aprovechar dichas características para lograr un mejor nivel de speedup.

7.3.4.1. ANÁLISIS DE CONCURRENCIA

En este proceso se analizó la manera como el modelo puede aprovechar la arquitectura multicore por nodo, es decir, como usa de manera paralela cada uno de los 16 cores existentes, con los resultados mostrados en la Figura 48. Concurrencia obtenida con 2 MPI y 4 threads por Rank. Se puede observar la baja concurrencia con 5 threads de 16 disponibles, por tanto, se procede a incrementar el número de threads, sin incrementar los ranks para lograr incrementar la concurrencia Figura 48. Concurrencia obtenida con 2 MPI y 4 threads por Rank

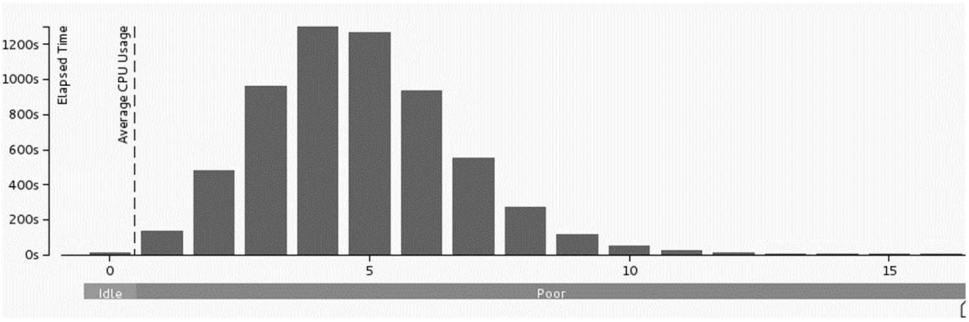


Figura 48. Concurrencia obtenida con 2 MPI y 4 threads por Rank
Fuente desarrollo propio.

Aunque aumentó la concurrencia como se observar en la Figura 49. Comparación concurrencia 2x4 threads vs 2x18 threads, el tiempo de simulación mejoró solamente en un 32% como se puede observar en la Figura 49. Comparación concurrencia 2x4 threads vs 2x18 threads

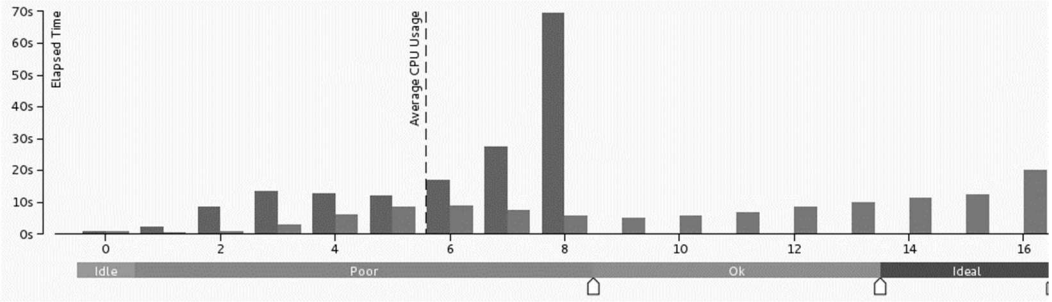


Figura 49. Comparación concurrencia 2x4 threads vs 2x18 threads
Fuente: Desarrollo propio

Al realizar el análisis de las razones por la cuales se presenta este desempeño el diferencial se observa que tiene su origen desbalance por espera en tareas OpenMP, por esperas en secciones del código serial y por la sobrecarga introducida en un número mayor de threads manejados por OpenMP como se muestra en la Tabla 12. Comparación rendimiento 2x8 vs 2x16 en memoria compartida.

Característica	**Variación de resultados**
Elapsed Time:	124.878s - 165.258s = -40.380s
CPU Time:	1941.862s - 1288.271s = 653.591s
Effective Time:	1104.568s - 921.441s = 183.127s
Spin Time:	621.808s - 277.714s = 344.094s
Imbalance or Serial Spinning (OpenMP):	601.655s - 250.913s = 350.742s
Lock Contention (OpenMP):	0.671s - 0.382s = 0.289s
Communication (MPI):	16.713s - 17.149s = -0.436s
Other:	2.769s - 9.270s = -6.501s
Overhead Time:	215.486s - 89.116s = 126.370s
Creation (OpenMP):	0.713s - 0.369s = 0.344s
Scheduling (OpenMP):	3.593s - 1.731s = 1.862s
Reduction (OpenMP):	Not changed, 0s
Atomics (OpenMP):	Not changed, 0s

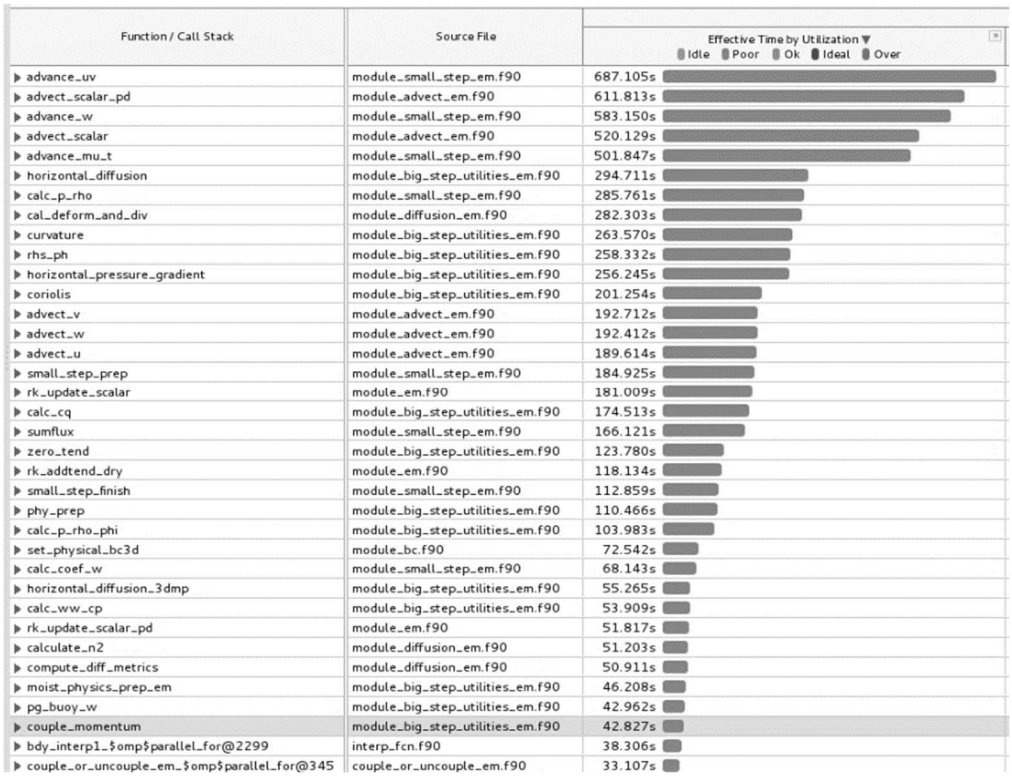
Tasking (OpenMP):	Not changed, 0s
Other:	211.180s - 87.016s = 124.164s
Instructions Retired:	3,902,604,000,000 - 2,973,484,800,000 = 929,119,200,000
CPI Rate:	1.420 - 1.347 = 0.073
CPU Frequency Ratio:	1.189 - 1.295 = -0.106
Total Thread Count:	18 - 8 = 10

Tabla 12. Comparación rendimiento 2x8 vs 2x16 en memoria compartida

Se realiza un análisis de las razones por las cuales se presenta el desbalance y de las secciones del modelo donde las regiones seriales pueden ofrecen alguna característica que permitan ser paralelizada, para lo cual, se realiza un análisis de las funciones que dentro de la ejecución del modelo consumen mayor tiempo de computación; una vez identificadas las rutinas se procederá a realizar un análisis en aquellas que más tiempo de computación consumen en ciclos para ver la ganancia al paralelizarlos mediante ciclos OpenMP o cambiar la manera como se realiza el agendamiento de tareas con el fin de mejorar su desempeño. El listado de rutinas que consumen más tiempo de procesamiento se puede observar en la Figura 50. Funciones con mayor consumo de CPU.

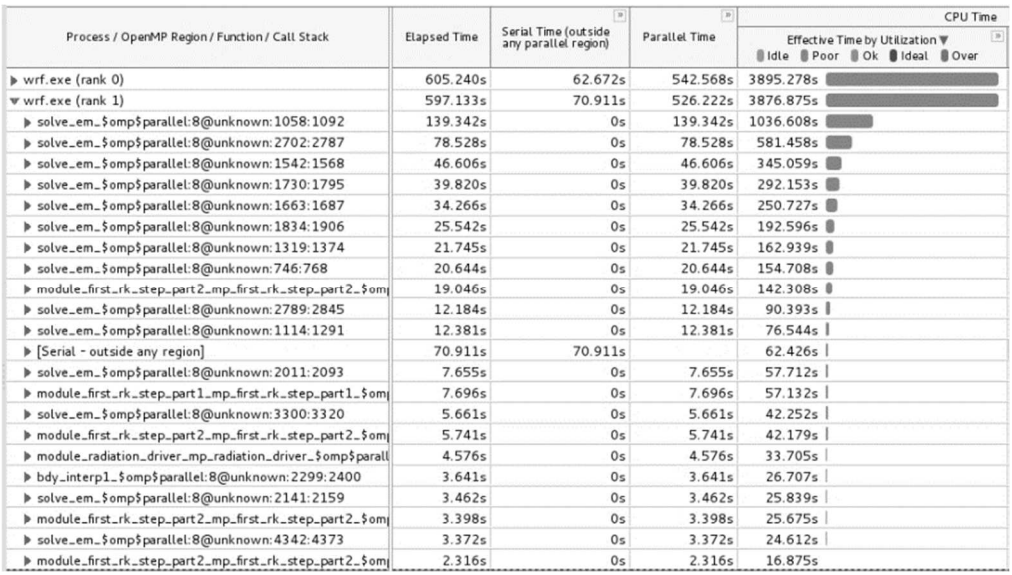
7.3.4.2. ANÁLISIS DE RUTINAS DE MAYOR CONSUMO

Además de las rutinas de mayor consumo de CPU, se deben tener en cuenta los ciclos que mayor consumo y el árbol de llamado dentro de la simulación de tal manera que se pueda atacar un ciclo de alto consumo de CPU que exponga un potencial de paralelismo adecuado y que, además contribuya al tipo de módulo que se esté utilizando dentro del modelo como se observa en la Figura 51. Estructura de ciclos con mayor consumo de CPU .



Function / Call Stack	Source File	Effective Time by Utilization ▼ (Idle Poor Ok Ideal Over)
advance_uv	module_small_step_em.f90	687.105s
advect_scalar_pd	module_advect_em.f90	611.813s
advance_w	module_small_step_em.f90	583.150s
advect_scalar	module_advect_em.f90	520.129s
advance_mu_t	module_small_step_em.f90	501.847s
horizontal_diffusion	module_big_step_utilities_em.f90	294.711s
calc_p_rho	module_small_step_em.f90	285.761s
cal_deform_and_div	module_diffusion_em.f90	282.303s
curvature	module_big_step_utilities_em.f90	263.570s
rhs_ph	module_big_step_utilities_em.f90	258.332s
horizontal_pressure_gradient	module_big_step_utilities_em.f90	256.245s
coriolis	module_big_step_utilities_em.f90	201.254s
advect_v	module_advect_em.f90	192.712s
advect_w	module_advect_em.f90	192.412s
advect_u	module_advect_em.f90	189.614s
small_step_prep	module_small_step_em.f90	184.925s
rk_update_scalar	module_em.f90	181.009s
calc_cq	module_big_step_utilities_em.f90	174.513s
sumflux	module_small_step_em.f90	166.121s
zero_tend	module_big_step_utilities_em.f90	123.780s
rk_addtend_dry	module_em.f90	118.134s
small_step_finish	module_small_step_em.f90	112.859s
phy_prep	module_big_step_utilities_em.f90	110.466s
calc_p_rho_phi	module_big_step_utilities_em.f90	103.983s
set_physical_bc3d	module_bc.f90	72.542s
calc_coef_w	module_small_step_em.f90	68.143s
horizontal_diffusion_3dmp	module_big_step_utilities_em.f90	55.265s
calc_ww_cp	module_big_step_utilities_em.f90	53.909s
rk_update_scalar_pd	module_em.f90	51.817s
calculate_n2	module_diffusion_em.f90	51.203s
compute_diff_metrics	module_diffusion_em.f90	50.911s
moist_physics_prep_em	module_big_step_utilities_em.f90	46.208s
pg_buoy_w	module_big_step_utilities_em.f90	42.962s
couple_momentum	module_big_step_utilities_em.f90	42.827s
bdy_interp1_ompparallel_for@2299	interp_fcn.f90	38.306s
couple_or_uncouple_em_ompparallel_for@345	couple_or_uncouple_em.f90	33.107s

Figura 50. Funciones con mayor consumo de CPU
Fuente: Desarrollo propio



Process / OpenMP Region / Function / Call Stack	Elapsed Time	Serial Time (outside any parallel region)	Parallel Time	CPU Time: Effective Time by Utilization ▼ (Idle Poor Ok Ideal Over)
wrf.exe (rank 0)	605.240s	62.672s	542.568s	3895.278s
wrf.exe (rank 1)	597.133s	70.911s	526.222s	3876.875s
solve_em_ompparallel:8@unknown:1058:1092	139.342s	0s	139.342s	1036.608s
solve_em_ompparallel:8@unknown:2702:2787	78.528s	0s	78.528s	581.458s
solve_em_ompparallel:8@unknown:1542:1568	46.606s	0s	46.606s	345.059s
solve_em_ompparallel:8@unknown:1730:1795	39.820s	0s	39.820s	292.153s
solve_em_ompparallel:8@unknown:1663:1687	34.266s	0s	34.266s	250.727s
solve_em_ompparallel:8@unknown:1834:1906	25.542s	0s	25.542s	192.596s
solve_em_ompparallel:8@unknown:1319:1374	21.745s	0s	21.745s	162.939s
solve_em_ompparallel:8@unknown:746:768	20.644s	0s	20.644s	154.708s
module_first_rk_step_part2_mp_first_rk_step_part2_$om	19.046s	0s	19.046s	142.308s
solve_em_ompparallel:8@unknown:2789:2845	12.184s	0s	12.184s	90.393s
solve_em_ompparallel:8@unknown:1114:1291	12.381s	0s	12.381s	76.544s
[Serial - outside any region]	70.911s	70.911s		62.426s
solve_em_ompparallel:8@unknown:2011:2093	7.655s	0s	7.655s	57.712s
module_first_rk_step_part1_mp_first_rk_step_part1_$om	7.696s	0s	7.696s	57.132s
solve_em_ompparallel:8@unknown:3300:3320	5.661s	0s	5.661s	42.252s
module_first_rk_step_part2_mp_first_rk_step_part2_$om	5.741s	0s	5.741s	42.179s
module_radiation_driver_mp_radiation_driver_ompparall	4.576s	0s	4.576s	33.705s
bdy_interp1_ompparallel:8@unknown:2299:2400	3.641s	0s	3.641s	26.707s
solve_em_ompparallel:8@unknown:2141:2159	3.462s	0s	3.462s	25.839s
module_first_rk_step_part2_mp_first_rk_step_part2_$om	3.398s	0s	3.398s	25.675s
solve_em_ompparallel:8@unknown:4342:4373	3.372s	0s	3.372s	24.612s
module_first_rk_step_part2_mp_first_rk_step_part2_$om	2.316s	0s	2.316s	16.875s

Figura 51. Estructura de ciclos con mayor consumo de CPU
Fuente: Desarrollo propio

Con las rutinas candidatas identificadas y teniendo en cuenta el framework de trabajo del modelo expuesto en el capítulo 4 (ESTRUCTURA DE CÓDIGO FUENTE DEL MODELO), se procede a evaluar la ganancia que podría tener introducir mayor nivel de paralelismo en los ciclos identificados.

Las siguientes rutinas fueron identificadas como las mayores generadoras de consumo y por tanto candidatas a ser reescritas para aprovechar mejor el paralelismo existente en la plataforma (un listado completo de las rutinas se puede encontrar publicado en la url *Hernandez, Esteban (2017), "Dataset of WRF profiling", Mendeley Data, v2 http://dx.doi.org/10.17632/fjgjyj55wm.2*"

Dentro de las rutinas de mayor consumo tenemos solve_em; esta rutina es la encargada de inicializar la integración numérica e invoca todas las parametrizaciones físicas que se establezcan. Aparecen también las rutinas module_first_rk_step_part1 y module_first_rk_step_part2, encargas localizan los datos en la estructura de integración en tiempo al igual que establecer las dimensiones de los "tile" que son computadas y pasadas a los elementos de driver (Figura 22. Esquema de distribución de software en WRF.)

Dada lo estructural de estas rutinas anteriormente mencionadas, las microfísicas resultan ser un lugar propicio para la optimización, dado su alto consumo de procesamiento y su baja ramificación estructural.

7.3.4.3. ANÁLISIS DE RENDIMIENTO DE MICROFÍSICAS

Para determinar el impacto de la microfísica dentro del rendimiento del modelo, se eligieron aquellas que podrían representar mejor las condiciones meteorológicas colombianas, de acuerdo a la siguiente tabla

Nombre	Opción No	Descripción
Kessler Scheme	1	Tiene en cuenta fenómenos de lluvia cálida sin hielo, normalmente se usa para fenómenos idealizados.
WRF Single–moment 3–class	3	Esquema que predice el radio mixto de 3 especies (vapor de agua, agua de las nubes y lluvia o nieve)
WRF Single–moment 6–class Scheme	6	Esquema que predice el radio mixto de 6 especies (vapor de agua, liquido de nubes, hielo, lluvia, nieve y graupel)
Thompson Scheme	8	Esquema de bulk, que predice 6 especies: (vapor de agua, liquido de nubes, hielo, lluvia, nieve y graupel)
Morrison 2–moment Scheme	10	Esquema de bulk, de doble momento que predice el radio mixto y la concentración numérica de 6 especies: (vapor de agua, liquido de nubes, hielo, lluvia, nieve y graupel)

Tabla 13. Microfísicas utilizadas para medir rendimiento
Fuente: Adaptación de
http://www2.mmm.ucar.edu/wrf/users/phys_references.html

Sin embargo el análisis detallado de desempeño se realizó sobre la microfísica WRF Single–moment 3–class (Hong et al., 2004), debido a que los estudios realizados por el departamento de meteorología de la Universidad Nacional de Colombia, demuestran que predicen con mayor precisión los fenómenos de lluvia (Cortes, 2012)

En este análisis se puede concluir que para la microfísica No 3 (WRF Single–moment 3–class), el mejor rendimiento se logra utilizando 2 ranks y 8 Tiles por rank, como se puede observar en la Figura 52 Rendimiento de simulación por microfísica.

Se aclara que el rendimiento computacional no tiene una relación directa con la calidad del pronóstico de lluvia, este tipo de análisis son propios de la meteorología con técnicas como tablas de contingencia, re análisis y más recientemente técnicas de aprendizaje de máquina. Sobre este tópico a nivel nacional se han realizado análisis a profundidad como las tesis de maestría de Cortes y de Jiménez (Cortes, 2012; Jiménez, 2014). En la presente propuesta, estos análisis están fuera de los objetivos planteados, por tanto no fueron abordados.

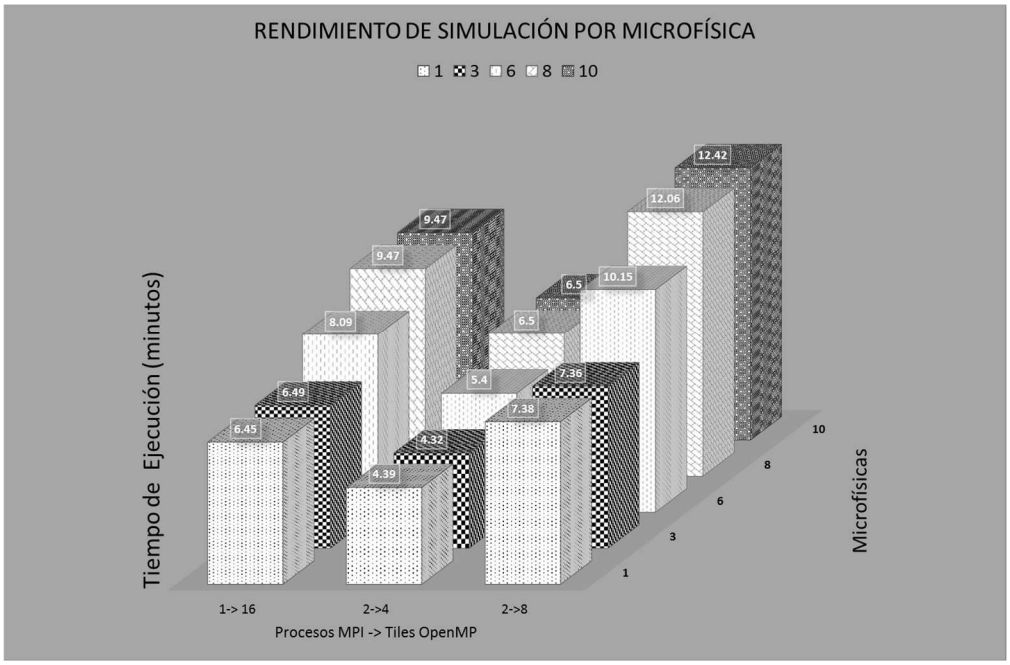


Figura 52 Rendimiento de simulación por microfísica
Fuente: desarrollo propio

7.3.4.4. CONCLUSIÓN 2

La aceleración del modelo WRF requiere conocer en profundidad sus componentes, especialmente el marco de trabajo y sus 3 capas constitutivas, que permiten orientar los esfuerzos en las rutinas adecuadas. Las posibilidades de optimización deben tener en cuenta la distribución de los hilos por procesador para aprovechar la localidad, el uso de la afinidad de

threads ajustando los procesos e hilos dentro de un mismo nodo NUMA y la distribución de cargas de trabajo para lograr el mejor aprovechamiento de los recursos disponibles; sin embargo, dichas estas técnicas no pueden desconocer el hecho que las secciones seriales del código serán el limitante para lograr aumentar el speedup.

Partiendo de un nivel más grueso (distribución de la carga de trabajo por nodos), se ha enfocado en el rendimiento por nodo (a nivel de threads por core) y en la siguiente sección se analizará el rendimiento a nivel de core, de tal manera que se puedan aprovechar las instrucciones propias de cada tipo de procesador para incrementar el rendimiento por cada ciclo de instrucción

7.4. OPTIMIZACIÓN DEL MODELO A NIVEL DE INSTRUCCIONES

Aprovechar el paralelismo a nivel core, requiere que se pueda utilizar instrucciones especializadas para que en cada ciclo de reloj puedan realizar una instrucción sobre un conjunto de datos (un vector), lo que normalmente se conoce como una operación SIMD, que están presentes en procesadores x86 como en otras arquitecturas de procesadores (Mitra, Johnston, Rendell, McCreath, & Zhou, 2013).

Para el desarrollo de la presente investigación las arquitecturas exploradas están basadas en x86 de 64 bits en Intel MIC, las cuales ofrecen diversas capacidades de registros vectoriales dependiendo de la microarquitectura y la generación del procesador, como se muestra en la Tabla 14. Características Vectoriales de procesadores utilizados

Procesador	**Capacidad SIMD**	**Tamaño**
Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz	AVX2	256 Bits

Intel(R) Xeon(R) CPU X5570 @ 2.93GHz	SSE-4.2	128 Bits
Intel Xeon Phi KNC 3120A	VPUD	512 Bits
Intel(R) Xeon(R) Phi KNL x200	AVX-512	512 Bits

Tabla 14. Características Vectoriales de procesadores utilizados

Cada una de estas características ofrecen ventajas al utilizar su conjunto de instrucciones vectorizadas sobre ciclos de procesamiento que cumpliendo con los requisitos de no dependencias ofrecen niveles superiores de desempeño (Jeong et al., 2012)

El uso de instrucciones SIMD puede ser implementadas explícitamente, mediante un profundo conocimiento de los tipos de registros y la manera como estos deben ser invocados en rutinas utilizando tipos de datos especializados o mediante lenguaje ensamblador (Bik, 2004; Mitra et al., 2013), también se puede hacer uso de lenguajes o extensiones del lenguaje que ocultan la complejidad de implementación por medio de estructuras o directivas especializadas (McCool et al., 2012; Reinders, 2007).

El modelo WRF ha sido construido utilizando OpenMP como modelo de paralelismo de memoria compartida y ha dejado a discreción de quien compila el modelo la posibilidad o no de utilizar instrucciones vectorizadas.

En el desarrollo del presente trabajo se decidió utilizar los compiladores de Intel y explotar las instrucciones vectorizadas AVX, AVX2 y AVX-512 según la disponibilidad en cada plataforma utilizada.

Una exploración inicial del rendimiento teórico con uso de instrucciones vectoriales, utilizando la microfísica no WSM3.

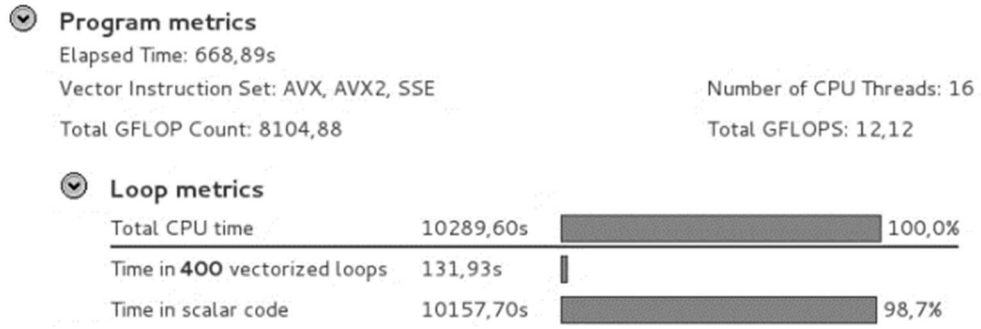


Figura 53. Rendimiento en GFOPs y ciclos vectorizados
Fuente: Resultado de perfilamiento aplicación

Una revisión detallada utilizando el análisis de roofline (Williams et al., 2009), permite observar que las funciones que presenta alta intensidad aritmética están usando al límite las instrucciones de adiciones de vectores simples y que casi ninguna rutina hace uso intensivo de las operaciones FMA (Fused-Multiply Add) que permiten realizar operaciones aditivas de punto flotante en un solo paso con redondeo simple (Ambler, 2007), por tanto esas rutinas presentan potencial de lograr un mejor desempeño de la simulación.

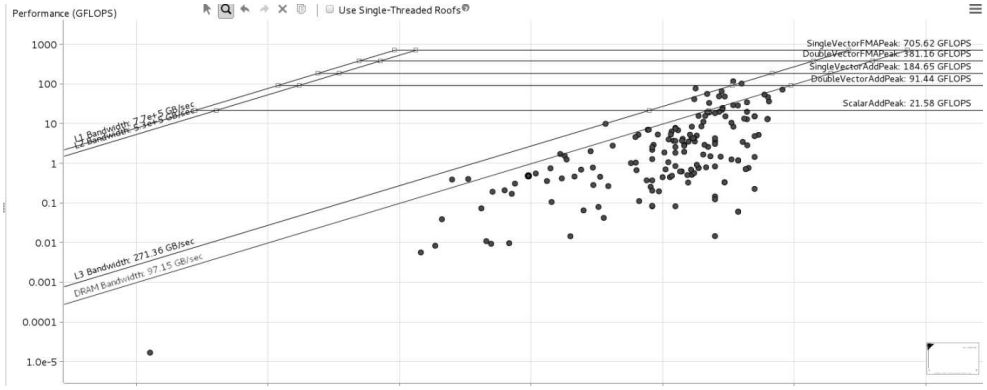


Figura 54. Análisis Roofline para modelo compilado con vectorización.
Fuente: Captura herramienta de perfilamiento

Se procedió a revisar las rutinas de mayor uso y su posibilidad de vectorizar las partes críticas, especialmente podemos observar las rutinas de la microfísica de Morrison (número 10), que presenta 2 ciclos con potencial de ser vectorizados

Loop	Self Time	Total Time
[loop in advect_scalar_pd at module_advect_em.f90:7609]	23,692s	23,692s
[loop in module_mp_morr_two_moment_mp_morr_two_moment_micro_ at ?]	18,306s	31,882s
[loop in sintb_ at ?]	17,110s	17,110s
[loop in module_mp_morr_two_moment_mp_morr_two_moment_micro_ at ?]	9,015s	13,790s
[loop in advect_scalar_pd at module_advect_em.f90:6454]	7,375s	7,375s

Figura 55. Loops de mayor consumo con posibilidad de vectorizar secciones Fuente: Captura pantalla herramienta profiling. Nota: Color Azul de la izquierta, sin vectorizar, color naranja ya vectorizado pero con problemas

Al observar el árbol de llamado podemos encontrar esta rutina como una de las que no utiliza vectorización

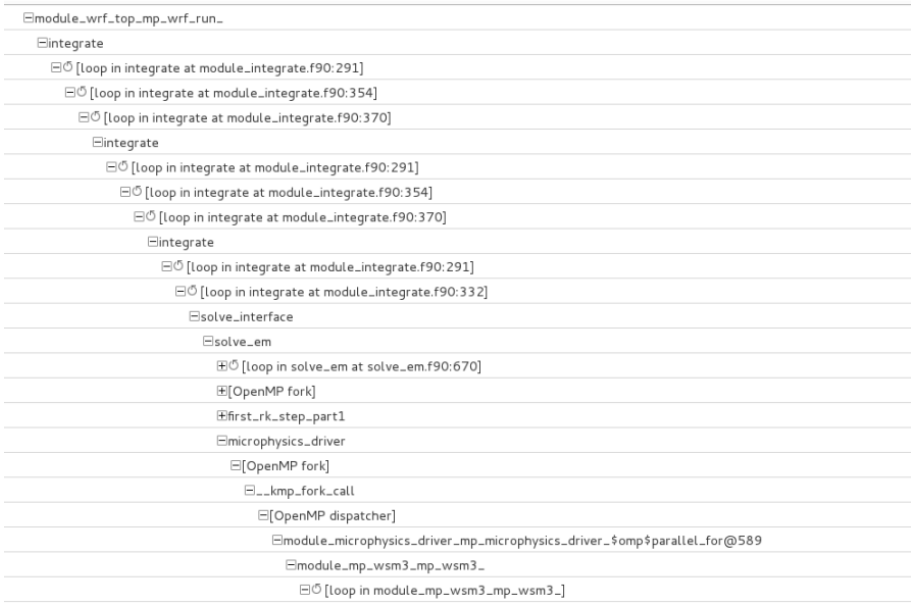


Figura 56. Árbol de ciclos para invocación de la microfísica de WSM3. Fuente: Captura de pantalla herramienta de perfilamiento

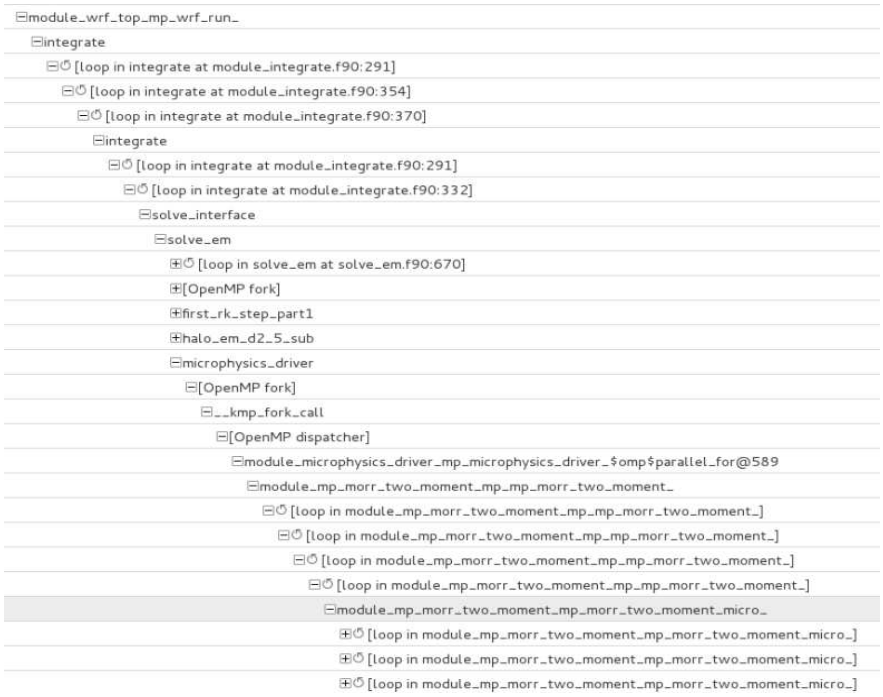


Figura 57. Ciclos para invocación de la microfísica de dos momentos Morrison
Fuente: Captura de pantalla herramienta de perfilamiento

Identificados los ciclos de mayor consumo se procede a realizar ajustes para crear regiones paralelas.

Una vez se realizan estos ajustes se presentan los siguientes resultados:

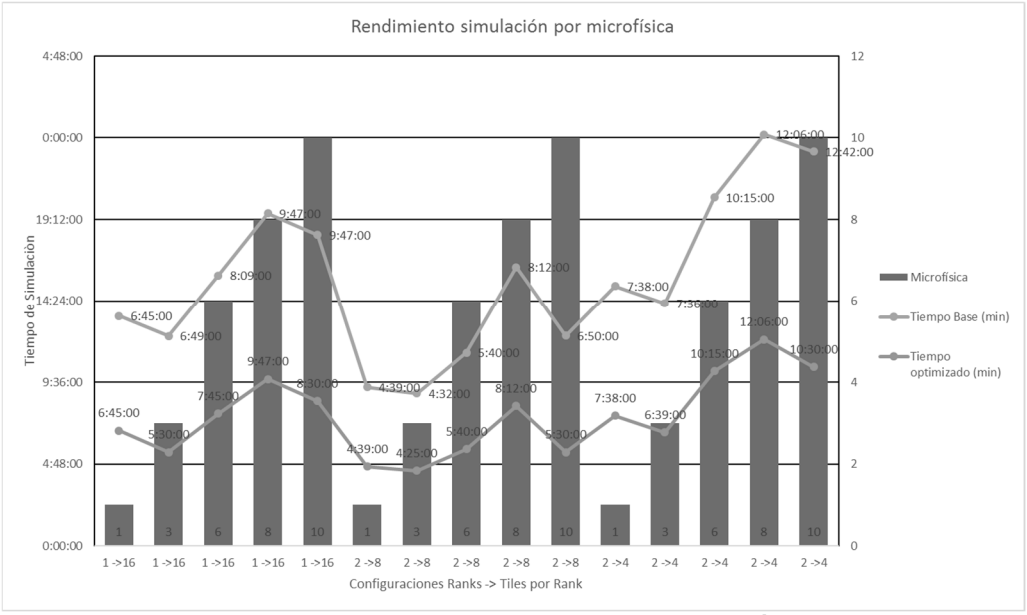


Figura 58. Tiempo de rendimiento vs configuración base.
Fuente: Desarrollo propio

7.4.1. CONCLUSIÓN 3.

Para acelerar el tiempo de ejecución por core se requiere observar el comportamiento de las de uso ciclos e instrucciones para determinar cuáles se de ellas pueden ser vectorizadas. Al analizar el comportamiento de ciclos, se debe buscar que no existen dependencias entre ciclos embebidos (para usar esquema de multiproceso como OpenMP y sacar ventajas de las operaciones que no tienen dependencias y que pueden sacar provecho de cachés grandes para aumentar la localidad. Determinar cada uno de estos puntos calientes y funciones candidatas requieren del uso de las herramientas adecuadas para instrumentar y perfilar las aplicaciones.

7.5. PROPUESTA PARA ACELERAR APLICACIONES CIENTÍFICAS

Partiendo de los resultados obtenidos en la aceleración de las microfísicas del modelo WRF, se puede proyectar una propuesta común para acelerar aplicaciones científicas y que se resumen en la Figura 59. Propuesta para acelerar aplicaciones científicas.

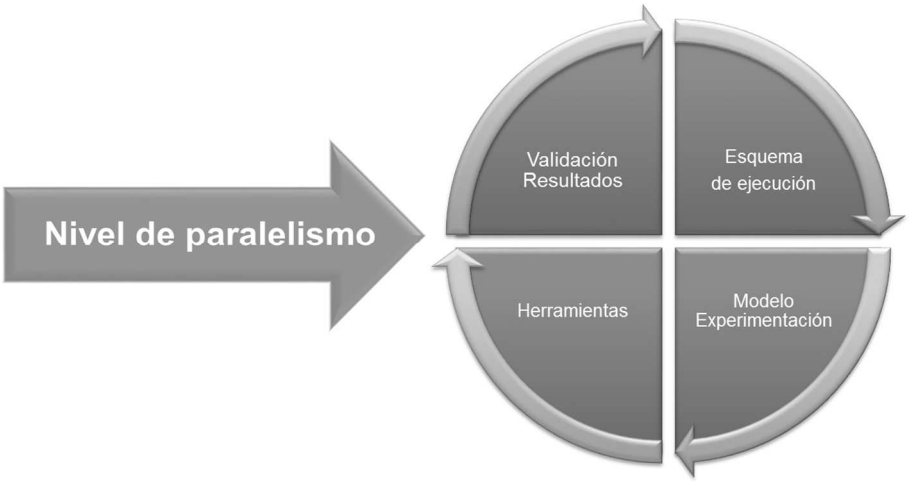


Figura 59. Propuesta para acelerar aplicaciones científicas
Fuente: Desarrollo propio

Dado que toca aplicación que desea escalar de un (1) procesador a cientos o miles de cores debe exponer algún nivel de paralelismo ya sea paralelismo a nivel de datos o paralelismo a nivel de tareas (McCool et al., 2012). Con este tipo de paralelismo identificado se inicia un ciclo de análisis que incluye los siguientes pasos:

a) Elegir un esquema de ejecución. En este paso dado se procede a elegir entre un esquema de memoria compartida, un esquema de memoria distribuida o un esquema mixto, intentando maximizar las variables planteadas (tiempo de ejecución, uso de los recursos, datos analizados) o buscar esquemas de uso eficiente de los mismos.
b) Construcción de un modelo de experimentación: Se construye un conjunto de datos y configuraciones requeridos por el modelo que permitan analizar el comportamiento de las variables analizadas, y que además puedan ser ajustados dentro del esquema de ejecución seleccionado.

c) Elección de herramientas: En este paso dado el modelo de experimentación y el esquema de ejecución se elijan un conjunto de herramientas que permitan medir las variables planteadas, dentro del esquema de ejecución, sin que afecten de manera significativa el modelo de experimentación y que puedan brindar flexibilidad para analizar los resultados desde diferentes enfoques.

d) Una vez se han ejecutado la experimentación siguiendo un modelo establecido, recopilando y analizando el comportamiento de la aplicación en tiempo de ejecución, se procede a validar los objetivos y observar si corresponde a los objetivos planteados inicialmente. Si son satisfactorios se logra una parametrización adecuada, de otra manera se repite el ciclo haciendo variar el esquema de ejecución, el modelo de experimentación o las herramientas seleccionadas hasta que se puedan llegar a los objetivos de las variables de análisis propuestas.

CAPÍTULO 8. CONCLUSIONES

- Acelerar una aplicación es un proceso iterativo que requiere entender el problema o algoritmo, el modelo de ejecución de la aplicación y la plataforma de cómputo sobre la cual se ejecuta. Cada aplicación presentará una mejor o peor aceleración dependiendo de un preciso ajuste de los parámetros en cada uno de sus elementos.

- Cada aplicación expone diferentes niveles de paralelismo (paralelismo a nivel de instrucciones, paralelismo a nivel de hilos y paralelismo a nivel de tareas), que es necesario ajustar cuidadosamente para lograr una ganancia en speedup. Optimizar una aplicación sin aprovechar cada tipo de paralelismo expuesto algunas veces culmina en esfuerzos perdidos.

- Las aplicaciones que pueden aprovechar los esquemas de procesadores vectoriales deben usar intensivamente este tipo de instrucciones, para ocultar la latencia producida por una menor velocidad de reloj y la no existencia optimización del I/O.

- Los modelos numéricos de pronóstico del estado del tiempo y especialmente aquellos que usan códigos stencil con mallas estructuradas, ofrecen la mejor aceleración en una configuración mixta que distribuye las operaciones seriales a procesadores tradicionales de alta frecuencia y las operaciones masivamente paralelas en aceleradores vectoriales.

- Cuando se ejecutan aplicaciones en esquemas de memoria distribuida, es esencial disminuir la mensajería entre nodos para lograr mejores niveles de speedup. Los mejores rendimientos se obtienen usando esquemas mixtos de memoria distribuida y memoria compartida.

8.1. CONTRASTACIÓN

Planteados	Logrados
Realizar el perfilamiento y adaptación de las microfísicas de lluvia (WSM3, WSM6 y Morrison de doble momento) para aprovechar las ventajas de plataformas de memoria compartida, memoria distribuida y sistemas de muchos cores (manycores en adelante) de tal manera que se pueda acelerar la corrida del modelo WRF	Se crearon 5 esquemas de ejecución, de los cuales el que evidenció una mejor aceleración fue el de uso completo de la infraestructura (sin sub o sobre utilizar los recursos de computo). Las fuentes que implementan el modelo WRF deben ser compilados específicamente para aprovechar la vectorización propias de cada plataforma.
Determinar las rutinas del modelo que presentan mayor consumo y que pueden ser afinadas para lograr mejores niveles de desempeños.	Las rutinas de mayor consumo identificada son: Integración numérica y Microfísicas. Que pudieron ser optimizadas mediante la vectorización en cada una de las plataformas. El tipo de Microfísica afectan el nivel de optimización.
Determinar las rutinas que exponen mejores características para ser vectorizadas logrando un incremento en el desempeño del modelo.	Son rutinas que no dependen entre ciclos anidados y las que permiten alinear alineación de instrucciones, entre estas se encuentran: microfísica wsm3 y parametrización bl_ysu_mp_ysu2d.
Generar una propuesta de aceleración que incluya elementos de uso eficiente de la energía, logrando incrementar una	Se realizaron análisis y se crearon herramientas para perfilar el uso eficiente de la energía como EnergyPhi.

aceleración energéticamente eficiente	
Generar una propuesta de análisis y perfilamiento del modelo siguiendo una metodología aceptada dentro de la computación científica.	Se desarrolló una “Propuesta para acelerar aplicaciones científicas”. El contexto en que se aplicó esta propuesta fue el pronóstico del estado del tiempo usando modelo WRF con mallas estructuradas y Computación Heterogénea Paralela.

CAPÍTULO 9. RECOMENDACIONES Y FUTUROS TRABAJOS

EL modelamiento atmosférico es un proceso que involucra trabajo interdisciplinario entre las geociencias, las ciencias de la computación, las matemáticas y áreas técnicas especializadas en arquitecturas de hardware y software.

Las investigaciones futuras deberán tomar los esquemas planteados para adicionar mecanismos de agendamiento automáticos de carga de trabajo que permita en tiempo de ejecución decidir que partes de la computación se ejecutan en qué tipos de plataformas, basados en métricas especializadas como aumento del speedup, menor consumo de recursos, uso intensivo de operaciones de punto flotante y cualquier otra requerida.

Análisis de alertas tempranas sobre eventos extremos, como lluvias torrenciales, pueden ser analizados procurando maximizar la cantidad de datos analizados y minimizar el tiempo del pronóstico, incluyendo modelos de ensamble. Dado la variedad de parametrizaciones que pueden ser usadas dentro del modelo especialmente para pronóstico de lluvia, es posible involucrar el uso de Procesadores gráficos dentro de los futuros esquemas planteados, de tal manera que la sección de la dinámica del modelo pueda aprovechar el paralelismo de grano fino que este tipo de procesadores ofrece.

BIBLIOGRAFIA

Akhter, S., & Roberts, J. (2006). *Multi-core Programming: Increasing Performance Through Software Multi-threading*. Intel Press. Retrieved from https://books.google.com.co/books?id=E9qySgAACAAJ

Ambler, T. (2007). *Floating-Point Fused Multiply-Add Architectures Committee :*

Amdahl, G. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 18-20, 1967, Spring Joint …*, *30*, 483–485. Retrieved from http://dl.acm.org/citation.cfm?id=1465560

Asai, R. (Colfax). (2016). *MCDRAM AS HIGH-BANDWIDTH MEMORY (HBM) IN KNIGHTS LANDING PROCESSORS: DEVELOPER'S GUIDE*.

Bader, M. (2013). HPC – Algorithms and Applications Dwarf # 5 – Structured Grids Dwarf # 5 – Structured Grids, 1–45.

Beyer, J. (Nvidia). (2015). Comparing OpenACC 2.5 and OpenMP 4.1. In *OpenMP User Conference 2015*.

Bik, A. J. C. (2004). *The Software Vectorization Handbook: Applying Multimedia Extensions for Maximum Performance*. Intel Press. Retrieved from https://books.google.com.co/books?id=DzMUNAAACAAJ

Birrittella, M. S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., … Zak, R. C. (2015). Intel Omni-path Architecture: Enabling Scalable, High Performance Fabrics. *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 1–9. https://doi.org/10.1109/HOTI.2015.22

Bondhugula, U., & Baskaran, M. (2008). Towards effective automatic parallelization for multicore systems. *… , 2008. IPDPS 2008. …*, (1), 1–5. https://doi.org/10.1109/IPDPS.2008.4536401

Bougeault, P. (2008). High Performance Computing and the Progress of Weather and Climate Forecasting. In J. L. Palma, P. Amestoy, M. Daydé, M. Mattoso, & J. Lopes (Eds.), *High Performance Computing for Computational Science - VECPAR 2008* (Vol. 5336, p. 349). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-92859-1_31

Brodtkorb, A. R., Hagen, T. R., & Sætra, M. L. (2013). Graphics processing unit (GPU) programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing*, *73*(1), 4–13.

https://doi.org/10.1016/j.jpdc.2012.04.003

Cabezas, J., Jordà, M., Gelado, I., Navarro, N., & Hwu, W. (2015). GPU-SM: shared memory multi-GPU programming. In *Proceedings of the 8th Workshop on General Purpose Processing using GPUs - GPGPU 2015* (pp. 13–24). New York, New York, USA: ACM Press. https://doi.org/10.1145/2716282.2716286

Carrington, L. (2015). Performance and energy-efficiency analysis of ARM processors for HPC workloads. In *Proceedings of the 2nd International Workshop on Hardware-Software Co-Design for High Performance Computing - Co-HPC '15* (pp. 1–1). New York, New York, USA: ACM Press. https://doi.org/10.1145/2834899.2834900

Chandra, R., Menon, R., Dagum, L., & Kohr, D. (2000). *Parallel programming in OpenMP*. Retrieved from http://books.google.com/books?hl=en&lr=&id=18CmnqIhbhUC&oi=fnd&pg=PR7&dq=Parallel+Programming+in+OpenMP&ots=sVm-_FO1kV&sig=hhMnLx1TggVkLeC2WEv9eXXc-e4

Cortes, A. U. (2012). Estimacion de la capacidad del modelo WRF para pronosticar eventos extremos asociados con altas precipitaciones en la region on Andina Colombiana.

Dudhia, J. (n.d.). Microphysics Options in WRF Microphysics.

El-Askary, H., Allali, M., Rakovski, C., Prasad, A., Kafatos, M., & Struppa, D. (2012). Computational methods for climate data. *Wiley Interdisciplinary Reviews: Computational Statistics*, *4*(4), 359–374. https://doi.org/10.1002/wics.1213

Fauzia, N., & Sadayappan, P. (2015). Characterizing and Enhancing Global Memory Data Coalescing on GPUs. *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, 12–22. Retrieved from http://dl.acm.org/citation.cfm?id=2738600.2738603

Ffoulkes, P., & By, Y. O. U. (2016). *A Trusted Approach for High Performance Networking*. Retrieved from http://insidehpc.com/white-paper/ihpc-special-report-high-performance-networking/

Foundation, G. (2017). *GNU Offloading and Multi Processing Runtime Lib*.

G.I. Marchuk. (n.d.). *Numerical Methods in Weather Prediction*.

Gupta, S., Xiang, P., & Zhou, H. (2013). Analyzing locality of memory references in GPU architectures. *… of the ACM SIGPLAN Workshop on Memory …*, 1–2. https://doi.org/10.1145/2492408.2492423

Gustafson, J. (1988). Reevaluating Amdahl's law. *Communications of the*

ACM, *31*(5), 4–6. Retrieved from http://dl.acm.org/citation.cfm?id=42415

Hackenberg, D., Molka, D., & Nagel, W. E. (2009). Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42* (p. 413). https://doi.org/10.1145/1669112.1669165

Hacker, J., Exby, J., & Gill, D. (2016). Collaborative WRF-based research and education with reproducible numerical weather prediction enabled by software containers. In *WRF Workshop 2016* (pp. 1–18).

Hager, G., & Wellein, G. (2011). *Introduction to High Performance Computing for Scientists and Engineers*. *Book*. https://doi.org/10.1201/EBK1439811924

Heinecke, A. (2013). Accelerators in scientific computing is it worth the effort? *2013 International Conference on High Performance Computing & Simulation (HPCS)*, (May), 504–504. https://doi.org/10.1109/HPCSim.2013.6641460

Heirman, W., Carlson, T. E., Van Craeynest, K., Hur, I., Jaleel, A., & Eeckhout, L. (2014). Automatic SMT threading for OpenMP applications on the Intel Xeon Phi co-processor. *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers - ROSS '14*, 1–7. https://doi.org/10.1145/2612262.2612268

Herlihy, M. (2006). *The art of multiprocessor programming*. *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing - PODC '06*. https://doi.org/10.1145/1146381.1146382

Hernández, E., Gaviria, G. de J. M., & Montenegro, C. E. (2014). Parallel programming languages on heterogeneous architectures using OPENMPC, OMPSS, OPENACC and OPENMP. *Revista Tecnura*, *18*(Edición especial doctorado), XX–XX. https://doi.org/10.14483/udistrital.jour.tecnura.2014.DSE1.a05

Hill, M., & Marty, M. (2008). Amdahl's law in the multicore era. *Computer*, *41*(7), 33–38. https://doi.org/10.1109/MC.2008.209

Holton, J. R. (2004). *An Introduction to Dynamic Meteorology*. Elsevier Science. Retrieved from http://books.google.com.co/books?id=zzExWj94NloC

Hong, S.-Y., Dudhia, J., & Chen, S.-H. (2004). A Revised Approach to Ice Microphysical Processes for the Bulk Parameterization of Clouds and Precipitation. *Monthly Weather Review*, *132*(1), 103–120. https://doi.org/10.1175/1520-0493(2004)132<0103:ARATIM>2.0.CO;2

Iancu, C., Hofmeyr, S., Blagojeviic, F., & Zheng, Y. (2010). Oversubscription

on multicore processors. *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010*. https://doi.org/10.1109/IPDPS.2010.5470434

Ilya, K. V., & Yukhin. (n.d.). OpenMP 4 Offloading Features implementation in GCC.

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

Jacobson, M. (2005). *Fundamentals of atmospheric modeling*. New York: Cambridge University Press. Retrieved from http://books.google.com/books?hl=en&lr=&id=FrHcZmwj7JQC&oi=fnd&pg=PP1&dq=Fundamentals+of+Atmospheric+Modelling&ots=P2GUyx6n_K&sig=v7zOUpsNdAHlJPGfTk4ixaMOThY

Jain, N., Bhatele, A., Ni, X., Wright, N. J., & Kale, L. V. (2014). Maximizing Throughput on a Dragonfly Network. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 336–347). IEEE. https://doi.org/10.1109/SC.2014.33

Jain, R. (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley. Retrieved from http://www.bibsonomy.org/bibtex/2fec32c393af648375f02cde200e33b99/dblp

Jang, B., Schaa, D., Mistry, P., & Kaeli, D. (2011). Exploiting memory access patterns to improve memory performance in data-parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, *23*(1), 105–118. https://doi.org/10.1109/TPDS.2010.107

Jeffers James;Reinders James. (2013). *Intel Xeon Phi Coprocessor High Performance Programming, 1st Edition* (1st ed.). Reinders. Retrieved from http://store.elsevier.com/product.jsp?isbn=9780124104143&utm_source=publicity&utm_medium=pressrelease&utm_campaign=IntelXeonPhi

Jeong, H., Kim, S., Lee, W., & Myung, S.-H. (2012). Performance of SSE and AVX Instruction Sets, 7. Retrieved from http://arxiv.org/abs/1211.0820

Jia, W. (2014). *Analysis and Optimization Techniques for Massively Parallel Processors*. Princeton. Retrieved from http://dspace.princeton.edu/jspui/bitstream/88435/dsp01wm117r22g/1/Jia_princeton_0181D_11168.pdf

Jiménez. (2014). *Validación de la capacidad de modelo WRF "Weather Research and forecasting" para pronósticar lluvia intensa, usando el método orientado a objetos y tablas de contingencia (tesis de maestria)*. Universidad Nacional de Colombia. Retrieved from

http://www.bdigital.unal.edu.co/49615/7/Libro_Tesis_Mauricio.pdf

Kaltofen, E. (2012). The “seven dwarfs” of symbolic computation. *Numerical and Symbolic Scientific Computing*, 1–11. https://doi.org/10.1007/978-3-7091-0794-2

Kanamitsu, M. (1989). Description of the NMC Global Data Assimilation and Forecast System. *Weather and Forecasting*. https://doi.org/10.1175/1520-0434(1989)004<0335:DOTNGD>2.0.CO;2

Karsavuran, M. O., Akbudak, K., & Aykanat, C. (2015). Locality-Aware Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication on Many-Core Processors. *IEEE Transactions on Parallel and Distributed Systems*, *6*(1), 1–1. https://doi.org/10.1109/TPDS.2015.2453970

Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., & Glasco, D. (2011). GPUs and the Future of Parallel Computing. *IEEE Micro*, *31*(5), 7–17. https://doi.org/http://doi.ieeecomputersociety.org/10.1109/MM.2011.89

Kimura, R. (2002). Numerical weather prediction. *Journal of Wind Engineering and Industrial Aerodynamics*, *90*(12–15), 1403–1414. https://doi.org/10.1016/S0167-6105(02)00261-1

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

Kirk, D. B., & Hwu, W. W. (2012). *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier Science. Retrieved from http://books.google.com.co/books?id=E0Uaag8qicUC

Kjolstad, F. B., & Snir, M. (2010). Ghost Cell Pattern. *Proceedings of the 2010 Workshop on Parallel Programming Patterns - ParaPLoP ’10*, *1*, 1–9. https://doi.org/10.1145/1953611.1953615

Knievel, J. (2006). Numerical Weather Prediction (NWP) and the WRF Model, (August).

Koibuchi, M. (2013). Future Low-Latency Networks for High Performance Computing. In *2013 First International Symposium on Computing and Networking* (pp. 22–23). IEEE. https://doi.org/10.1109/CANDAR.2013.11

Kuo, Y., Klemp, J., & Michalakes, J. (2004). Mesoscale numerical weather prediction with the WRF Model. *… Numerical Weather …*. Retrieved from http://nldr.library.ucar.edu/repository/assets/osgc/OSGC-000-000-001-311.pdf

Lameter, C. (2013). NUMA (Non-Uniform Memory Access): An Overview.

Queue, *11*(7), 40. https://doi.org/10.1145/2508834.2513149

Landaverde, R., Zhang, T., Coskun, A. K., & Herbordt, M. (n.d.). An Investigation of Unified Memory Access Performance in CUDA. https://doi.org/10.1109/HPEC.2014.7040988

Langer, U., & Paule, P. (2011). *Numerical and Symbolic Scientific Computing: Progress and Prospects*. Springer Vienna. Retrieved from https://books.google.com.co/books?id=uG0FzuObcXIC

Lewyt, H. (1967). Courant-Friedrichs-Lewy, (March).

Lynch, P. (2008). The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, *227*(7), 3431–3444. https://doi.org/10.1016/j.jcp.2007.02.034

Majo, Z., & Gross, T. R. (2011). Memory management in NUMA multicore systems. *ACM SIGPLAN Notices*, *46*(11), 11. https://doi.org/10.1145/2076022.1993481

Mak, M. (2011). *Atmospheric Dynamics*. Cambridge University Press. Retrieved from http://books.google.com.co/books?id=0kp6HfWTAKAC

Malakar, P., Saxena, V., George, T., Mittal, R., Kumar, S., Naim, A. G., & Husain, S. A. B. H. (2012). Performance evaluation and optimization of nested high resolution weather simulations. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7484 LNCS*, 805–817. https://doi.org/10.1007/978-3-642-32820-6_80

Malony, A. D., Biersdorff, S., Shende, S., Jagode, H., Tomov, S., Juckeland, G., … Lamb, C. (2011). Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs. In *Proceedings of the 2011 International Conference on Parallel Processing* (pp. 176–185). Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/ICPP.2011.71

Marina, L. U. Z., & Antonio, M. D. E. (2004). *Computación paralela hetereogenea*.

Marowka, A. (2012). Extending Amdahl's Law for Heterogeneous Computing. *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 309–316. https://doi.org/10.1109/ISPA.2012.47

McCool, M. D., Reinders, J., & Robison, A. D. (2012). *Structured Parallel Programming: Patterns for Efficient Computation*. Elsevier/Morgan Kaufmann. Retrieved from http://books.google.com.co/books?id=zpaHa5cjLwwC

Mcintosh-smith, S. (2012). *Trends in Heterogeneous Systems Architectures*.

Michalakes, J. (2013). Code restructuring to improve performance in WRF model physics on Intel Xeon Phi WRF and Accelerators.

Michalakes, J., Dudhia, J., & Gill, D. (2004). The weather research and forecast model: Software architecture and performance. *… of High Performance …*, (June). Retrieved from http://www.wrf-model.org/wrfadmin/docs/ecmwf_2004.pdf

Michalakes, J., & Gill, D. (2016). WRF Software : Code and Parallel Computing. In *WRF Tutorial 2016*. Boulder Colorado.

Michalakes, J., Iacono, M., Berthiaume, D., & Gokhale, I. (2014). Performance-related developments in WRF. *WRF Workshop 2014*.

Michalakes, J., & Vachharajani, M. (2008). Gpu Acceleration of Numerical Weather Prediction. *Parallel Processing Letters*, *18*(4), 531–548. https://doi.org/10.1142/S0129626408003557

Mielikainen, J., Huang, B., Huang, H.-L. A., & Goldberg, M. D. (2012). GPU Implementation of Stony Brook University 5-Class Cloud Microphysics Scheme in the WRF. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *5*(2), 625–633. https://doi.org/10.1109/JSTARS.2011.2175707

Mitra, G., Johnston, B., Rendell, A. P., McCreath, E., & Zhou, J. (2013). Use of SIMD vector operations to accelerate application code performance on low-powered ARM and intel platforms. *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013*, 1107–1116. https://doi.org/10.1109/IPDPSW.2013.207

Molka, D., Hackenberg, D., Schöne, R., & Müller, M. S. (2009). Memory performance and cache coherency effects on an intel nehalem multiprocessor system. In *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT* (pp. 261–270). https://doi.org/10.1109/PACT.2009.22

Montoya, G. de J. (2008). *Lecciones de meteorología Dinámica y Modelamiento atmosférico* (2008th ed.). Universidad Nacional de Colombia, Unibiblos.

Moreland, K., & Oldfield, R. (2015). Formal Metrics for Large-Scale Parallel Performance. In J. M. Kunkel & T. Ludwig (Eds.), *High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings* (pp. 488–496). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-20119-1_34

Namyst, R. (2012). Programming heterogeneous, accelerator- based multicore machines: a runtime's perspective.

Nickolls, J., & Dally, W. (2010). The GPU computing era. *Micro, IEEE*, *30*(2), 56–69. https://doi.org/10.1109/MM.2010.41

Nokia Siemens Networks. (2009). The impact of latency on application performance. *White Paper*, 1–14. https://doi.org/10.5663/aps.v1i1.10138

Nupairoj, N., & Ni, L. M. (n.d.). Performance Metrics and Measurement Techniques of Collective Communication Services. https://doi.org/10.1.1.50.3913

Orr, M., Beckmann, B., Reinhardt, S., & Wood, D. (2014). Fine-grain Task Aggregation and Coordination on GPUs. *Research.cs.wisc.edu*. Retrieved from http://research.cs.wisc.edu/multifacet/papers/isca14-channels.pdf

Pacheco, P. S. (2011). *Introduction to Parallel Programming*. *Elsevier Inc.* https://doi.org/10.1007/978-1-4471-2736-9

Panourgias, I. (2011). *NUMA effects on multicore, multi socket systems*. *The University of Edinburgh*. Retrieved from https://www.epcc.ed.ac.uk/sites/default/files/Dissertations/2010-2011/IakovosPanourgias.pdf

Persson, A. (2015). User guide to ECMWF forecast products. *Ecmwf*, 127.

Podobas, A., Brorsson, M., & Faxén, K. (2010). A comparison of some recent task-based parallel programming models, 1–14. Retrieved from http://soda.swedish-ict.se/3869/

Ponder, C., Romanenko, A., & Snytnikov, A. (2014). Progress Porting WRF to GPU using OpenACC. In *NCAR 4 multicore workshop*. Retrieved from https://www2.cisl.ucar.edu/heterogeneous-multi-core-5-workshop/2015

Porter, A. R., Ashworth, M., Gadian, A., & Burton, R. (2010). WRF code Optimisation for Meso-scale Process Studies (WOVIPS) dCSE Project Report, (January), 1–26.

Posey, S. (2013). *GPU Progress and Directions for Earth System Modeling*.

Pratas, F., Trancoso, P., & Stamatakis, A. (2009). Fine-grain Parallelism using Multi-core , Cell / BE , and GPU Systems : Accelerating the Phylogenetic Likelihood Function. *Processing*, 9–17. https://doi.org/10.1109/ICPP.2009.30

Rahman, R. (2013). *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress. Retrieved from https://books.google.com.co/books?id=wZ-9AAAAQBAJ

Rauber, T., & Rünger, G. (2010). *Parallel programming: For multicore and cluster systems*. Retrieved from http://books.google.com/books?hl=en&lr=&id=wWogxOmA3wMC&oi=fnd&pg=PR5&dq=Parallel+Programming+For+Multicore+and+Cluster+Systems&ots=0BiMUZ8v87&sig=s41eyxFU7q-InNFkDotuNoq4Udw

Reams, C. (2012). *Modelling energy efficiency for computation*.

Reinders, J. (2007). Intel threading building blocks: outfitting C++ for multi-core processor parallelism. *Journal of Computing Sciences in Colleges*, 334. https://doi.org/10.1145/1559764.1559771

Reinders, J. (2012). An overview of programming for Intel Xeon processors and Intel Xeon Phi coprocessors, 1–20. Retrieved from http://software.intel.com/sites/default/files/blog/337861/reindersxeonandxeonphi-v20121112a.pdf%5Cnhttp://software.intel.com/en-us/blogs/2012/11/14/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi

Reyes, R., & López-Rodríguez, I. (2012). accULL: An OpenACC implementation with CUDA and OpenCL support. *Euro-Par 2012 Parallel …*, *2*(228398), 871–882. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-32820-6_86

Reyes, R., López, I., Fumero, J., de Sande, F., & Sande, F. De. (2012). A comparative study of openacc implementations. *Jornadas Sarteco*. Retrieved from http://www.jornadassarteco.org/js2012/papers/paper_150.pdf

Rosales, C. (2014). Porting to the intel Xeon Phi: Opportunities and challenges. *Proceedings - 2013 Extreme Scaling Workshop, XSW 2013*, 1–7. https://doi.org/10.1109/XSW.2013.5

Ruiz Murcia, José Franklin Instituto de Hidrologia, M. y E. A. I. (2010). ¿COMO INTERPRETAR LOS MODELOS DE PRONÒSTICO DEL ESTADO DEL TIEMPO? Retrieved October 9, 2012, from http://bart.ideam.gov.co/wrfideam/GUIA_MODELOS.pdf

Run Rules, Green500. (2014). Energy Efficient High Performance Computing Power Measurement Methodology. Version: 1.2RC2.

Shainer, G., & Liu, T. (2009). Weather Research and Forecast (WRF) Model Performance and Profiling Analysis on Advanced Multi-core HPC Clusters. *… High-Performance …*. Retrieved from http://www.linuxclustersinstitute.org/conferences/archive/2009/PDF/Shainer_64557.pdf

Sharma, Y., & Kulkarni, D. B. (2015). GPU Based Acceleration of WRF Model : A Review. *International Journal of Science and Research*, *4*(7), 2013–2016.

Skamarock, B., & Dudhia, J. (2011). The Advanced Research WRF (ARW) Dynamics Solver. *Power Point Slides Presented at the November 2011 NCAS Tutorial.*, (July), 1–42. Retrieved from http://www.mmm.ucar.edu/wrf/users/tutorial/201111/07_tut_dyn_arw_201111.pdf

Snell, A. (2013). Cost Models for HPC and Supercomputing Intersect360 Research.

Sodani, A. (2016). Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor. In *2015 IEEE Hot Chips 27 Symposium, HCS 2015* (pp. 1–24). IEEE. https://doi.org/10.1109/HOTCHIPS.2015.7477467

Sun, X., & Gustafson, J. (1991). Toward a better parallel performance metric. *Parallel Computing*, 1093–1109. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167819105800286

Targhetta, A. (2008). *Heterogeneous Parallel Computing*. *nitte.ac.in*. Retrieved from https://cs.nmt.edu/~cs451/lectures/grad/targhetta.pdf

Teodoro, G., Kurc, T., Kong, J., Cooper, L., & Saltz, J. (2013). Comparative Performance Analysis of Intel Xeon Phi, GPU, and CPU, 11. https://doi.org/10.1109/IPDPS.2014.111

Tokhi, M. O., Hossain, M. A., & Shaheed, M. H. (2012). *Parallel Computing for Real-time Signal Processing and Control*. Springer London. Retrieved from https://books.google.com.co/books?id=NmwKBwAAQBAJ

Wang, J., Huang, B., Huang, A., & Goldberg, M. D. (2011). Parallel Computation of the Weather Research and Forecast (WRF) WDM5 Cloud Microphysics on a Many-Core GPU. *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, 1032–1037. https://doi.org/10.1109/ICPADS.2011.160

Warner, T. (2011). *Numerical weather and climate prediction*. Retrieved from http://www.newbooks-services.de/MediaFiles/Texts/0/9780521513890_Intro_001.pdf

Warner, T. T. (2010). *Numerical Weather and Climate Prediction*. Cambridge University Press. Retrieved from http://books.google.com.co/books?id=6RQ3dnjE8lgC

Wei Wang, Cindy Bruyère, M. et all. (2016). User's Guide for the Advanced Research WRF (ARW) Modeling System Version 3.8. *Book*, (January), 408. https://doi.org/10.5065/D68S4MVH

Williams, S., Waterman, A., & Patterson, D. (2009). Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, *52*(4), 65–76. https://doi.org/10.1145/1498765.1498785

Witte, J. C., & Mannon, S. E. (2010). *The Internet and Social Inequalities (Google eBook)*. Taylor & Francis. Retrieved from http://books.google.com/books?hl=en&lr=&id=MQPnleAu2UIC&pgis=1

Wolfe, M. (2013). The OpenACC Application Programming Interface, *2*.

Woo, D., & Lee, H. (2008). Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer*. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4712496

Zumbusch, G. (2012). Tuning a finite difference computation for parallel vector processors. *Proceedings - 2012 11th International Symposium on Parallel and Distributed Computing, ISPDC 2012*, 63–70. https://doi.org/10.1109/ISPDC.2012.17

ANEXOS

9.1. ANEXO No 1.

Caracterización del hardware utilizado para desarrollo de proyecto.

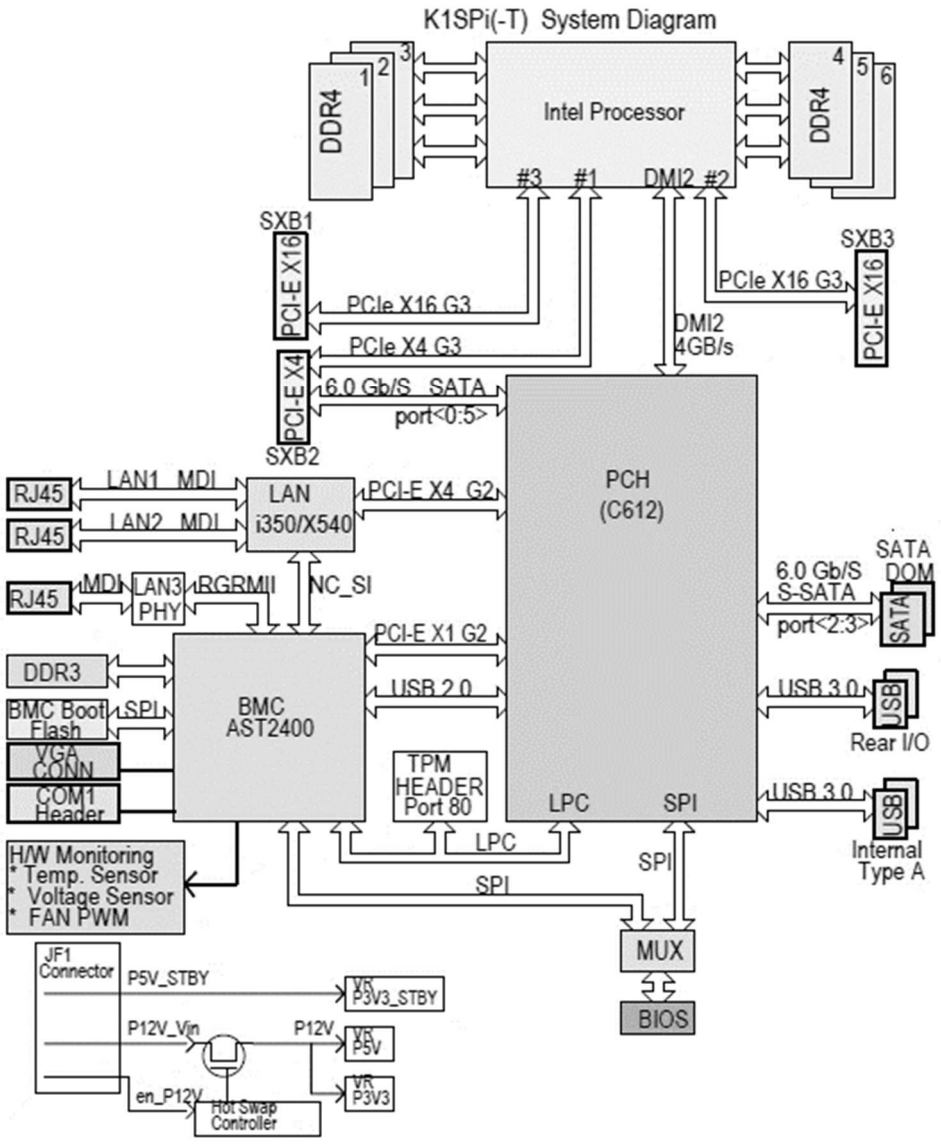


Figura 60. Arquitectura nodo manyCore:
Fuente: Manual del fabricante

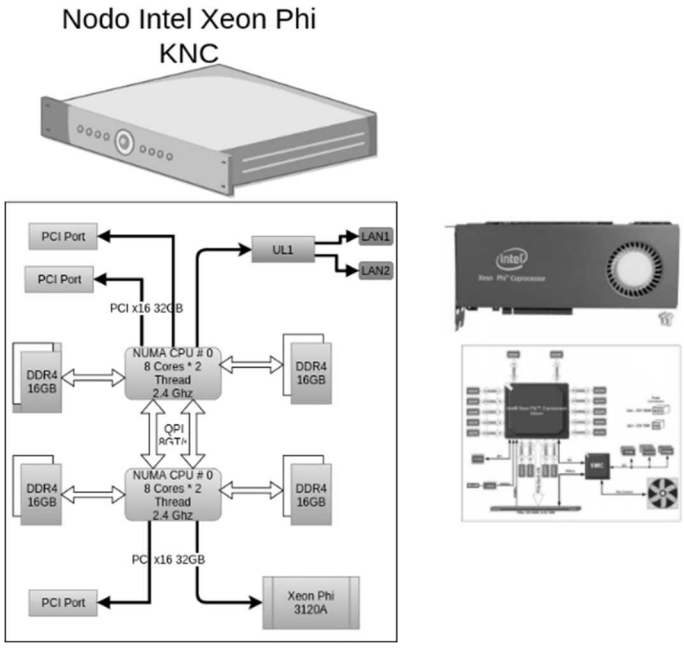


Figura 61. Arquitectura de nodo multicore.
Fuente Manual de fabricante

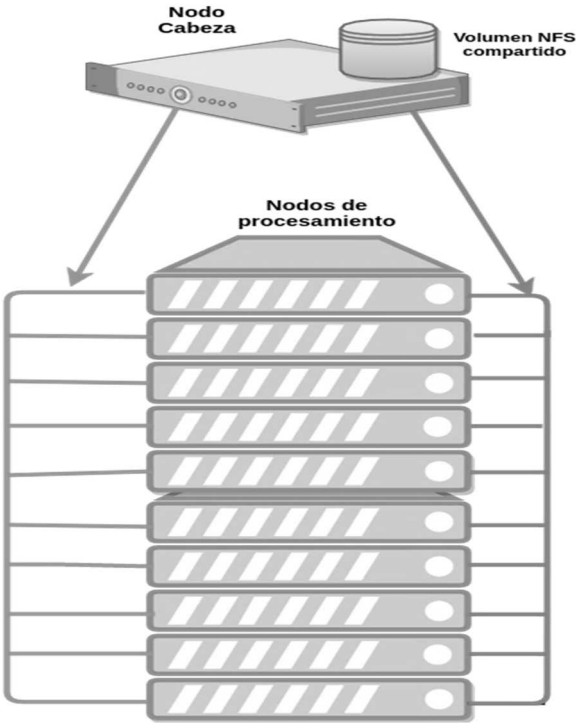


Figura 62. Arquitectura clúster multicore Sabio I.
Fuente CECAD.

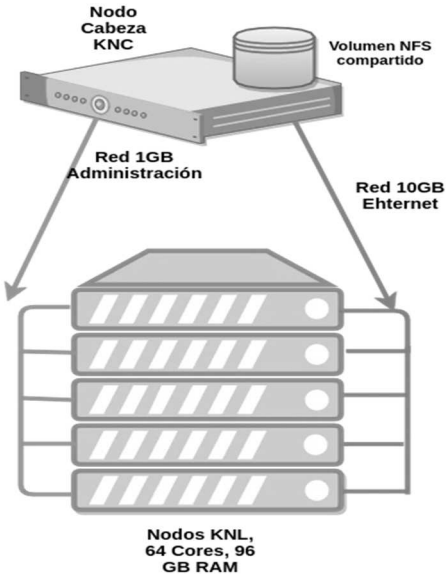


Figura 63. Arquitectura cluster manycore Sabioll.
Fuente CECAD