UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Aceleración de una herramienta para la predicción de energía solar mediante arquitecturas masivamente paralelas

Rodrigo Bayá Crapuchett

Programa de Maestría en Informática
PEDECIBA Informática
Universidad de la República

Montevideo – Uruguay
Setiembre de 2018

UNIVERSIDAD DE LA REPUBLICA
URUGUAY

Aceleración de una herramienta para la predicción de energía solar mediante arquitecturas masivamente paralelas

Rodrigo Bayá Crapuchett

Tesis presentada al Programa de Maestría en Informática, PEDECIBA Informática de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Magister en Informática.

Director de tesis:
Dr. Pablo Ezzatti

Codirector:
Ms.C. Ing. Martín Pedemonte

Director académico:
Dr. Pablo Ezzatti

Montevideo – Uruguay
Setiembre de 2018

Agradecimentos

Quisiera agradecer a mi familia que me apoyó y motivó en todo mi proceso educativo y especialmente en la realización de esta tesis. Adicionalmente, quiero agradecer a mi novia Analía, que me acompañó en todo momento y me alienta diariamente a hacer lo que me gusta, y a mi hermana Claudia, que convivió mano a mano conmigo durante uno de los años de esta tesis y cuyos aportes del idioma inglés fueron de gran ayuda. Además, quiero dar las gracias a mis amigos, que me escucharon hablar de este proceso varias horas y siempre brindaron por mis logros y fracasos. Quisiera agradecer también a mis tutores, Pablo y Martin, por su apoyo y guía durante todo el proceso y principalmente en el desarrollo del presente documento.

Por último, agradezco a aquellas instituciones que brindaron soporte económico para realizar tareas en el marco de esta tesis de maestría. En particular, al centro ICT4V (Information and Communication Technologies for Verticals) por financiar el desarrollo del posgrado a través de la beca de maestría POS-ICT4V-2016-1-02, y al PEDECIBA Informática y a CSIC que financiaron parcialmente pasantías, asistencias a congresos y compras de equipamiento.

(Epígrafe:)*"¡Bueno, cerebro, yo no te agrado y tu no me agradas!... ¡Sácame de ésta y podré seguirte matando con cerveza! (Cerebro: ¡Trato hecho!)"*

Homero J. Simpson

RESUMEN

En la última década, Uruguay ha comenzado a incorporar fuertemente la energía eólica y solar a su matriz energética. La inclusión de este tipo de fuentes de energía para abastecer la red eléctrica presenta un gran desafío al momento de administrar su uso, principalmente debido a su flujo de carácter fluctuante. Considerando esta situación, y con el objetivo de simplificar el trabajo de despacho de carga (que se encarga de administrar eficientemente los recursos energéticos presentes en la matriz), desde la Facultad de Ingeniería se ha desarrollado una herramienta capaz de predecir la generación de energía solar fotovoltaica en el país para un horizonte de tiempo de 96 horas. Uno de los principales inconvenientes de dicha herramienta es su elevado costo computacional, lo que resulta en tiempos de ejecución restrictivos.

Esta tesis aborda el estudio de la herramienta mencionada, haciendo foco especialmente en la componente que más tiempo y recursos requiere, el modelo numérico de circulación general de la atmósfera Weather Research and Forecasting (WRF). En una primera fase de este trabajo se analiza el tiempo de ejecución de dicho modelo, concluyendo que una de las etapas más costosa es el cómputo de la radiación solar, debido, entre otras cosas, a la precisión numérica que se requiere en estos cálculos. A partir de esta situación, en el presente trabajo se propone una nueva arquitectura de software asincrónica que permita desacoplar y calcular de forma paralela la radiación solar con el resto de las propiedades atmosféricas presentes en el WRF, siguiendo un patrón de paralelismo de tipo pipeline. Adicionalmente, se aborda el portado de una porción del cálculo de la radiación a un coprocesador masivamente paralelo, concretamente una GPU (Graphics Processing Unit) y/o un procesador Xeon-Phi, con el objetivo de disminuir la demanda de cómputo sobre la CPU.

La evaluación experimental de esta propuesta en un escenario de doce plantas fotovoltaicas en el territorio uruguayo permite concluir que la arquitectura asincrónica logra disminuir los tiempos de ejecución del modelo original en un 10 % aproximadamente, cuando se consideran equipos multicore con una gran

cantidad de núcleos. Adicionalmente, la extensión de esta arquitectura permite incorporar exitosamente la capacidad de cómputo de un coprocesador (GPU o Xeon-Phi), alcanzando mejoras de entre un 25 % a un 30 % en el tiempo total del modelo cuando se combinan ambas estrategias (asincronismo y uso de dispositivos de cómputo secundario).

Palabras claves:
Computación de alto desempeño, WRF, Energía solar, GPU, Xeon-Phi.

ABSTRACT

Over the last decade, Uruguay has begun a strong incorporation of eolic and solar energy to its energy matrix. The inclusion of this type of energy sources to supply the power grid poses a significant challenge at the moment of managing its use, mainly because of its variable flux. Considering this situation, and in order to simplify the power dispatch task (which efficiently manages the energy resources in the matrix), the Facultad de Ingeniería has developed a tool capable of predicting the generation of photovoltaic solar energy in the country for a 96-hour time horizon. One of the main drawbacks of said tool is its high computational cost, which results in restrictive runtimes.

This thesis addresses the study of the aforementioned tool, focusing especially on the most resource- and time-consuming component, the numerical weather prediction model Weather Research and Forecasting (WRF). In a first stage of this work, the runtime of said model is assessed, concluding that one of the most expensive steps is the solar radiation computation, because of, inter alia, the numerical precision required in these calculations. Starting from this situation, this work proposes a new asynchronous software architecture which enables decoupling computation of solar radiation and its parallel calculation with the remaining atmospheric properties in the WRF, following a pipeline parallel strategy. Additionally, offloading of a portion of the radiation calculation to a co-processor is addressed, specifically a GPU (Graphics Processing Unit) and/or a Xeon-Phi processor, in order to decrease the computation load on the CPU.

Experimental assessment of this proposal in a twelve-photovoltaic-facility scenario in Uruguayan land makes it possible to conclude that asynchronous architecture decreases runtimes of the original model by approximately 10 %, when considering multicore equipment with a large amount of cores. Furthermore, the extent of this architecture enables the successful incorporation of the computation ability of a co-processor (GPU o Xeon-Phi), reaching improvements of between 25 % and 30 % in the total execution time of the model when

both strategies are combined (asynchronism and use of secondary computation devices).

Keywords:
High performance computing, WRF, Solar energy, GPU, Xeon-Phi.

Tabla de contenidos

Capítulo 1

Introducción

En las últimas décadas, Uruguay ha comenzado a cambiar su matriz energética, priorizando la incorporación de energías de fuentes renovables. Siguiendo esta premisa, se busca construir a futuro una matriz sustentable y que no dependa de energías basadas en el petróleo o el gas natural, las cuales tienen un costo elevado tanto económico como ambiental. En particular, se ha impulsado la construcción de parques eólicos y fotovoltaicos a través de políticas públicas y programas estatales. En el caso de la energía eólica, pasó de ser una fuente utilizada con propósitos domésticos a producir más del 35 % de la energía eléctrica total de la matriz energética uruguaya en la actualidad [1]. En cambio, la energía fotovoltaica está en una etapa más temprana de su incorporación a la matriz energética, representando hoy en día aproximadamente un 2 % del total, aunque se proyecta su crecimiento en un futuro próximo al 5 % (229 MW de 4220 MW) [2, 3].

Debido al creciente uso de este tipo de energías renovables y considerando que no es posible asegurar su regularidad en el tiempo, resulta de interés el desarrollo de herramientas capaces de predecir su comportamiento y así poder utilizarlas de forma eficiente en el suministro de la red eléctrica.

En este contexto, en la Facultad de Ingeniería se ha desarrollado una herramienta para la predicción de la generación de la energía solar fotovoltaica, con el objetivo de asistir a la tarea de despacho de carga a nivel país [3, 4]. Dicha herramienta involucra distintas etapas para realizar el pronóstico, entre las cuales se encuentran: la obtención de datos globales públicos de pronósticos meteorológicos, la ejecución de un modelo de atmósfera regional, a partir de los datos antes mencionados, generando pronósticos de alta resolución de la

radiación solar sobre una región de interés, y ciertos ajustes estadísticos de estas salidas. Dentro de los procedimientos involucrados, el más costoso desde un punto de vista computacional es la ejecución del modelo de predicción de atmósfera Weather Research and Forecasting (WRF), sobre todo si se desean pronósticos con una alta precisión numérica y poca incertidumbre.

Dado que se intenta pronosticar la radiación solar en una región determinada (donde podría ubicarse una granja de paneles fotovoltaicos), el cálculo de esta propiedad en el modelo tiene que ser lo más exacta posible, de forma de poder tomar decisiones sobre dichos datos. Esta restricción hace que el tiempo de ejecución del modelo aumente considerablemente, lo que limita la frecuencia con la que se realizan los pronósticos y podría dificultar un correcto manejo de esta fuente de energía.

Considerando este inconveniente, resulta razonable intentar mejorar el tiempo de ejecución del modelo utilizando técnicas de computación de alto desempeño (HPC, por su sigla en inglés High Performance Computing), en particular para mejorar los tiempos de ejecución asociados con los cálculos de la radiación solar. Con este tipo estrategias, se puede explotar el gran poder de cómputo que presentan diversas arquitecturas de hardware, las cuales combinan el uso de CPUs multicore de gran porte, que poseen entre 12 y 64 núcleos, y coprocesadores masivamente paralelos, como las GPUs (por su sigla en inglés Graphics Processing Unit -Unidad de procesamiento gráfico-) o los procesadores *Intel* Xeon-Phi. Si bien el WRF ya utiliza estrategias de HPC basadas en un enfoque de paralelismo de datos, no generan mejoras acordes a las esperadas en muchos casos, debido a que se subutiliza el recurso CPU para los casos estudiados. Adicionalmente, el uso de coprocesadores para acelerar alguna sección de código del modelo ya fue abordada en otros trabajos [5, 6]. Estos trabajos logran acelerar notablemente ciertas rutinas del modelo pero presentan inconvenientes para aprovechar adecuadamente los beneficios que estas unidades de procesamiento brindan para acelerar el modelo en su globalidad, producto principalmente de sobrecostos asociados a la transferencia de datos para realizar los cálculos.

En el presente trabajo se plantea un enfoque distinto para abordar el cómputo de la radiación solar en el WRF, intentando cambiar el paradigma de cómputo de dicha propiedad. En concreto, se diseña, implementa y evalua una arquitectura asincrónica para el cálculo de la radiación, siguiendo una estrategia de paralelismo de tipo *pipeline* [7, 8]. Conceptualmente, la arquitectura

propuesta desacopla el cómputo de la radiación del modelo WRF, permitiendo solapar este cálculo con otras propiedades de la atmósfera, logrando así un paralelismo a nivel de tareas. Tomando como base dicho enfoque, se diseña una extensión de la arquitectura que permite incluir un coprocesador al cálculo de la radiación, de forma de aprovechar el asincronismo creado y posibilitar el ocultamiento de los tiempos de transferencia. La evaluación experimental del uso de la arquitectura asincrónica en solitario mostró resultados prometedores sin comprometer la precisión numérica de los mismos, utilizando de mejor forma la CPU que la implementación original del modelo WRF. Adicionalmente, los resultados de la arquitectura extendida mostraron beneficios superiores a los alcanzados anteriormente, logrando disminuir el tiempo de ejecución total del modelo entre un 25 % y un 30 %.

Las principales contribuciones de esta tesis son:

- El diseño, desarrollo y validación de una arquitectura de software asincrónica que permite calcular de forma paralela el cómputo de la radiación con el resto de las propiedades atmosféricas presentes en el WRF.
- Una extensión de la arquitectura asincrónica que permite incorporar al cómputo de la radiación el uso de coprocesadores masivamente paralelos.
- El portado a Xeon-Phi del módulo de onda larga RRTM.
- La evaluación experimental de la arquitectura asincrónica en un ambiente multicore de gran porte, mostrando que esta propuesta utiliza de mejor forma la plataforma que su contraparte original.
- La evaluación experimental de la arquitectura asincrónica con dos coprocesadores masivamente paralelos, concretamente una GPU y un Xeon-Phi, mostrando que esta alternativa es mejor para ambientes híbridos (CPU y un coprocesador) cuando la cantidad de núcleos es limitada.

La tesis presentada en este documento está compuesta por seis capítulos y tres apéndices, desarrollándose de la siguiente manera. Para comenzar, en el Capítulo 2, se relevan los distintos tipos de paralelismos que se pueden desarrollar en una aplicación, así como una clasificación de las arquitecturas de hardware según su grado de paralelismo. Adicionalmente, se profundiza en tres arquitecturas de hardware en concreto, las CPUs multicore, las GPUs, centrando el estudio en las fabricadas por la empresa *Nvidia*, y los coprocesa-

dores Xeon-Phi del fabricante *Intel*.

En el Capítulo 3, se presenta la situación de las energías renovables en Uruguay, considerando tanto la historia de su explotación en nuestro territorio como su estado actual. Además, se analizan las características principales de la energía solar, estudiando el comportamiento de ésta en nuestro planeta y las distintas formas de sacar provecho de ella. Posteriormente, se describe la herramienta de predicción de la generación de la energía solar desarrollada en la Facultad de Ingeniería, así como también se analiza experimentalmente el desempeño computacional individual del modelo WRF. Finalmente, se resume el estado del arte relacionado a la aplicación de técnicas de HPC al modelo WRF.

El Capítulo 4 describe en detalle el diseño de una arquitectura asincrónica, presentando un estudio teórico de su correcto funcionamiento. Posteriormente, se detalla su implementación y las variantes que surgen al momento de llevar a la práctica el diseño de esta novel propuesta. Finalmente, se evalúa experimentalmente la estrategia asincrónica planteada, analizando los resultados obtenidos.

En el Capítulo 5, se extiende el diseño de esta arquitectura, con el objetivo de incorporar al cómputo de la radiación el uso de una GPU o un procesador Xeon-Phi. Se describen los detalles de implementación de esta nueva arquitectura que la diferencian de la descripta previamente, así como el portado de una sección del módulo de la radiación a la GPU y al procesador Xeon-Phi por separado. Para cerrar este capítulo se evalúan estas implementaciones de forma experimental, analizando los resultados obtenidos.

Para concluir este documento, en el Capítulo 6 se resume el trabajo realizado, haciendo énfasis en los resultados y conclusiones que se desprenden de éste. Adicionalmente, se proponen posibles líneas de trabajo a desarrollar en futuras propuestas.

Relativo a los apéndices, en el primero de ellos, Apéndice 1, se presenta el trabajo "Paralelismo funcional en el WRF", expuesto en el Congreso sobre Métodos Numéricos y sus Aplicaciones (ENIEF) 2016, realizado en la

ciudad de Córdoba, Argentina. En el Apéndice 2 se exhibe el artículo de revista "Enhancing the performance of the WRF model with an asynchronous computation architecture", enviado a la revista Journal of Parallel and Distributed Computing. Esta publicación se encuentra actualmente en proceso de revisión. Finalmente, en el Apéndice 3 se expone un artículo de conferencia titulado "Task parallelism in the WRF model through computation offloading to many-core devices", presentado en la conferencia Parallel, Distributed and Network-Based Processing (PDP) 2018, llevado a cabo en la ciudad de Cambridge, Reino Unido.

Capítulo 2

Computación de alto desempeño

La computación de alto desempeño es un concepto que surge casi simultáneamente con la aparición del computador de propósito general a fines de la década del 60' y principios de los 70'. Es una disciplina que se focaliza en desarrollar tanto algoritmos como arquitecturas de hardware que utilizan, entre otras estrategias, la computación paralela para mejorar el desempeño computacional. De esta forma, la computación de alto desempeño se encarga de desarrollar, o encontrar, secciones de cómputo que puedan explotar las diferentes características de las arquitecturas subyacentes de las plataformas de hardware. En particular, utilizando técnicas de paralelismo que utilicen eficientemente los múltiples recursos de cómputo disponibles, mejorando así el desempeño de los algoritmos.

Esta disciplina se usa típicamente para resolver problemas de solución compleja, fuertemente orientada a áreas relacionadas con la investigación, como modelado, simulación y análisis de grandes volúmenes de datos. Sin embargo, debido al importante desarrollo tecnológico producido en las últimas décadas, estas estrategias se están usando ampliamente en la industria del software, con el fin de dar soporte a aplicaciones modernas que trabajan sobre una gran cantidad de información y usuarios, causado entre otras razones por el masivo acceso a Internet.

El concepto de computación de alto desempeño, en sus orígenes, estaba relacionado directamente con el desarrollo de las llamadas supercomputadoras. La primera computadora considerada de procesamiento paralelo, llamada Colossus 2, apareció en el año 1946 y era utilizada mayormente para criptoanálisis [9]. Aunque esta máquina fue la primera a la cual se le atribuye esta

propiedad, la primera supercomputadora que se utilizó para fines relacionados con la investigación fue la Cray-1 [10], desarrollada en el año 1975. Esta computadora poseía procesadores vectoriales que operaban a una velocidad de 80 MHz, 8 MB de memoria RAM y tenía un pico teórico de desempeño de 160 *MFlops* (millones de operaciones de punto flotante por segundo).

A comienzos del siglo XXI, el crecimiento de la frecuencia de los procesadores (y por lo tanto el poder de cómputo de las CPU) se estancó, producto de inconvenientes térmicos y del incremento del consumo energético a la hora de aumentar la velocidad del reloj [11]. Por esta razón, las empresas diseñadoras de procesadores decidieron cambiar de estrategia y construir procesadores con más unidades de cómputo, o núcleos, como forma de aumentar el poder de cálculo de los computadores. Esta situación potenció el estudio de técnicas y arquitecturas paralelas en las últimas décadas, siendo en la actualidad algo de uso cotidiano tanto a nivel de computadoras personales como en grandes centros de cómputo.

Hoy en día, según la lista Top 500[1] de fecha Noviembre de 2017 [12], el supercomputador más potente es el Sunway TaihuLight del National Supercomputing Center en Wuxi, China. Esta computadora cuenta con procesadores Sunway SW26010 260C de 1.45GHz, alcanzando un total de 10.649.600 núcleos, y 1.310.720 GB de memoria RAM. Se estima que esta computadora posee un pico teórico de 125.436 *TFlops*, consumiendo una cantidad de energía aproximada de 15.371 kW.

En este capítulo se introducen distintos enfoques para comprender la computación de alto desempeño, ya sea desde las técnicas utilizadas para ejecutar de forma paralela una aplicación, así como los distintos recursos de hardware que participan y permiten la ejecución de este tipo de técnicas. Concretamente, este capítulo se desarrolla de la siguiente manera. En primer lugar, se presenta una clasificación de los distintos niveles donde se puede aplicar técnicas de paralelismo. Posteriormente, se describe una clasificación de arquitecturas paralelas, profundizando en las más utilizadas en la actualidad. Finalmente, se describen en detalle las tres arquitecturas que se utilizan en este trabajo, las cuales son: las CPUs, las tarjetas gráficas (GPUs) y los procesadores Xeon-Phi.

[1] Top 500: sitio dedicado a listar, en forma semestral, las 500 máquinas más potentes del mundo.

2.1. Programación paralela

Una forma de entender la computación paralela es explicarla a través del nivel dónde se ejecute el paralelismo en la aplicación. En este aspecto, existen cuatro posibles enfoques distintos de paralelismo en cuanto al software se refiere:

- A nivel intra-instrucción.
- A nivel inter-instrucción.
- A nivel de procedimientos o tareas.
- A nivel de programas o trabajos.

El paralelismo intra-instrucción es el de más bajo nivel en cuanto a computación se refiere y sucede a nivel de hardware o de lenguaje de máquina. Una estrategia comúnmente utilizada en este nivel (aunque también puede ser implementada en otros de los niveles vistos) es el *pipeline* de instrucciones, el cual consiste en aplicar los mismos principios que utilizó Henry Ford en la cadena de ensamblaje, a principios del siglo XX [13]. Para esto, se separa el procesamiento de cada instrucción en una cantidad definida de etapas, donde la salida de cada una de ellas es la entrada de la etapa siguiente. De esta forma, se pueden ejecutar distintas etapas de distintas instrucciones en paralelo, aprovechando mejor cada ciclo de la CPU.

Subiendo un nivel de abstracción, en el caso del paralelismo inter-instrucción, el objetivo es reordenar las instrucciones de un programa para ejecutar varias de éstas al mismo tiempo (por ejemplo, mediante el uso de operaciones vectoriales). Este tipo de paralelismo se puede generar a nivel de sistema operativo, de compilador u optimizadores de código. Un ejemplo de este caso son las optimizaciones con la opción `oX` en los compiladores de los lenguajes C/Fortran.

La estrategia de paralelismo más común hoy en día es el paralelismo de tareas. Este paralelismo consiste en ejecutar simultáneamente funciones o porciones de código que no presentan una dependencia de datos entre sí o conceptualmente no están relacionadas. Esta opción es muy popular entre los programadores ya que no requiere un conocimiento específico de un lenguaje o arquitectura en particular, sino que depende intrínsecamente del problema a ser resuelto. Este tipo de programación paralela suele asociarse con el concepto

de hilos, ya que a cada porción independiente de código se la suele ejecutar de forma paralela utilizando estas estructuras lógicas.

Finalmente, el paralelismo de programas, como su nombre lo indica, consiste en ejecutar de forma simultánea programas que cooperan entre sí para procesar grandes volúmenes de datos o servicios. Esta forma de programación es la más escalable, ya que potencialmente cada uno de estos programas puede estar ejecutándose en distintas computadoras, las cuales pueden contar con diversos recursos de cómputo. Sin embargo, estas técnicas presentan un reto a la hora de ser implementadas, ya que es necesario tener protocolos de comunicación y sincronización entre las distintas aplicaciones, los cuales deben de ser sumamente eficientes si se quiere explotar de forma adecuada los recursos disponibles.

Dentro del marco de esta tesis, si bien se explotan técnicas pertenecientes a las cuatro categorías, las propuestas y el trabajo se centra en las últimas dos, ya que se hace hincapié en mejorar algorítmicamente el modelo independientemente de la arquitectura o lenguaje utilizado, buscando una solución que ofrezca la mayor escalabilidad posible.

2.2. Arquitecturas paralelas

En el ambiente de la computación existen distintas arquitecturas de hardware, las cuales se pueden clasificar de acuerdo a la taxonomía de Flynn [14]. Esta taxonomía categoriza las arquitecturas según el manejo de las instrucciones y su relación con los datos a procesar. De esta forma, la categorización está formada por cuatro grupos:

- SISD (*Single Instruction, Single Data*): Se aplica una instrucción sobre un único juego de datos.
- SIMD (*Single Instruction, Multiple Data*): Se aplica una instrucción sobre múltiples datos.
- MISD (*Multiple Instruction, Single Data*): Se aplican múltiples instrucciones sobre un único juego de datos.
- MIMD (*Multiple Instruction, Multiple Data*): Se aplican múltiples instrucciones sobre varios juegos de datos.

De las cuatro categorías, el presente proyecto se focaliza en dos de ellas, SIMD y MIMD, las cuales son de interés debido a su capacidad de computar sobre diferentes datos en forma simultánea.

2.2.1. MIMD

Esta categoría, como su nombre lo indica, incluye arquitecturas capaces de procesar múltiples instrucciones sobre múltiples juegos de datos de forma asincrónica y semi-independiente entre ejecuciones. Para cumplir con este objetivo, la arquitectura debe poseer varias unidades de procesamiento que permitan la comunicación y sincronización entre ellas. Esta categoría se puede subdividir en dos tipos, de memoria compartida y de memoria distribuida, ambos términos asociados a la forma en que se maneja la memoria y cómo se relaciona ésta con las unidades de cómputo.

En una arquitectura MIMD con memoria compartida los procesadores coexisten en un único nodo o en un sistema altamente acoplado, donde la memoria es vista como una única unidad a nivel "global". Estos sistemas son altamente eficientes y fáciles de programar, pero no son escalables, ya que el acceso a la memoria se vuelve un cuello de botella. Este tipo de arquitecturas suele ser explotada utilizando hilos, donde cada uno de ellos ejecuta de forma concurrente en el sistema y accede a distintas porciones de memoria, de forma sincronizada de ser necesario.

Para el caso de las arquitecturas MIMD con memoria distribuida, los nodos son independientes entre sí, donde cada uno posee una memoria local y se interconecta con el resto de los nodos a través de una red de alta velocidad. Así, estas arquitecturas son altamente escalables, aunque es crítico para su eficaz funcionamiento una comunicación eficiente. Debido al creciente uso de clústers en las últimas décadas, esta arquitectura se ha tornado de interés para casi todas las áreas de la computación contemporánea. De esta forma, las herramientas de software orientadas a explotar estas características se han fortalecido, como por ejemplo el estándar Message Passing Interface (o simplemente MPI) [15] o las técnicas basadas en *map-reduce* [16].

2.2.2. SIMD

Los equipos incluidos en esta categoría de la taxonomía de Flynn presentan unidades de procesamiento que ejecutan una única instrucción a la vez sobre

múltiples conjuntos de datos, consiguiendo un paralelismo a nivel de los datos a procesar. Para ello, todas las entidades de cómputo deben estar conectadas a un mismo controlador de instrucciones, el cual dicta cada instrucción a ejecutar y las mantiene a todas sincronizadas.

Esta forma de programación paralela fue una de las primeras en surgir en el ámbito de las supercomputadoras y fue una de las más eficientes en los años 80', teniendo como referente la Cray X-MP (el cual fue el computador más potente entre 1983 – 1985 [17]). Hoy en día, su principal foco está volcado al área de procesamiento de imágenes y simulaciones, donde las GPUs o tarjetas gráficas, si bien presentan sutiles diferencias con la arquitectura SIMD canónica, son el hardware de referencia.

Debido a la restricción de sincronización entre las entidades de procesamiento, la utilización de estas arquitecturas queda limitada a ciertos problemas, en particular aquellos que implican realizar las mismas operaciones sobre grandes volúmenes de datos, como aplicaciones multimedia, aplicaciones numéricas sobre estructuras de datos y, como se mencionó antes, procesamiento de imágenes.

2.3. CPU multicore

Un ejemplo claro de una arquitectura de cómputo del tipo MIMD es una CPU multicore. Para lograr obtener estos múltiples núcleos, los fabricantes de procesadores, como *Intel* o AMD, procedieron a duplicar componentes de una CPU tradicional en un mismo chip, por ejemplo, agregando más ALUs, FPUs y manejadores de instrucciones [18]. Teniendo múltiples conjuntos de estos componentes, se puede administrar de forma simultánea varios hilos de ejecución, donde cada uno de estos tiene de forma dedicada un core con sus correspondientes unidades. En esta sección se verán algunas de las características más relevantes de los procesadores multicore y qué problemas intentan cubrir. Un compendio completo de todos los modelos de procesadores *Intel* y AMD se puede encontrar en Wikipedia, ordenados tanto por familia como por generación [19, 20].

Con la aparición de los núcleos múltiples, surgieron problemas a resolver que previamente no existían, como la comunicación entre estos procesadores, la utilización de recursos compartidos y la consistencia de los datos, entre otras. Uno de los primeros inconvenientes que surgen de utilizar varios cores

de forma simultánea es el manejo de los recursos compartidos, en particular, el acceso a la memoria principal. Este problema se introdujo a principios de la década del 80, cuando el crecimiento del desempeño de los procesadores de la época comenzó a aumentar de manera dispar al crecimiento del desempeño de las memorias RAM, generando una brecha que creció año a año hasta la actualidad [21]. Esta situación se agravó al agregar más cores, ya que potencialmente se podrían sumar los desempeños de estos núcleos funcionando en paralelo pero se mantiene un único *bus* de comunicación con la memoria. Un ejemplo de esta amplia diferencia se puede ver en los procesadores *Intel* core i7, los cuales pueden alcanzar un ancho de banda pico de 409 GB/s, mientras que una memoria RAM tipo alcanza solamente un ancho de banda pico de 25 GB/s.

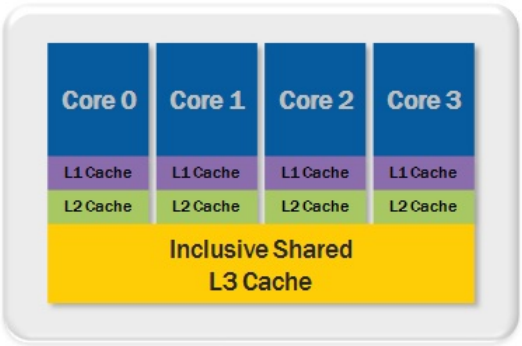


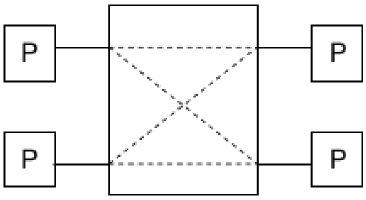
Figura 2.1: Ejemplo de Jerarquía de memorias caché utilizadas en las CPUs contemporáneas. Extraído de [22].

Con el objetivo de disminuir esta diferencia, se introducen a los procesadores multicore las memorias caché. Este tipo de memorias son colocadas entre la memoria principal y la CPU, de forma de funcionar como un buffer, explotando los principios de localidad espacial y temporal de los accesos a memoria. Si bien este tipo de almacenamiento tiene una velocidad de acceso mayor que la RAM, el costo de las mismas hace que se utilicen caché de tamaños reducidos y por lo general de más de un tipo. En la actualidad, y como se muestra en la Figura 2.1, las CPU multicore presentan varios niveles de memorias caché, teniendo por lo general dos (o uno, dependiendo de la arquitectura) niveles de

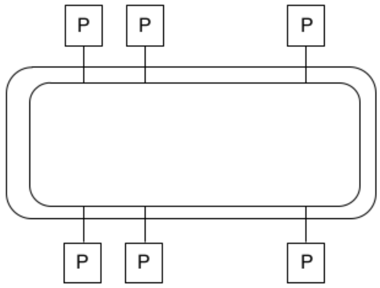
caché local a cada core y un nivel extra de esta memoria compartida entre todos los núcleos. De esta forma, se genera una jerarquía de memorias que varían en su tamaño y velocidad, permitiendo en su conjunto aumentar la velocidad del acceso a los datos requeridos. Por ejemplo, el procesador *Intel* core i7 de primera generación poseía 64KB de caché L1, 32KB para instrucciones y 32KB para datos, 256KB de caché L2 y 8MB de caché L3 [11].

Otro inconveniente a resolver cuando se utilizan varios núcleos es el medio de comunicación por el cual se conectan estas unidades [23]. En un principio, los fabricantes de procesadores utilizaban un *bus* común dentro del chip como medio de comunicación. Dado que esta implementación no resulta eficiente en latencia ni en ancho de banda, otras soluciones han sido propuestas. Como se puede ver en la Figura 2.2, existen actualmente cuatro topologías de uso común adoptadas por los fabricantes de procesadores: crossbar, anillos, mallas y toroides. La primera de ellas consiste en colocar tantos medios de comunicación como pares de cores se tiene, mientras que en la topología de anillo se conecta mediante un mismo medio de forma circular todos los núcleos presentes. En el caso de la estructura de malla, la idea es colocar cada core en una malla, proporcionándole a cada uno comunicación con sus vecinos, tanto horizontales como verticales. Finalmente, el toroide es una mejora de la malla, donde se agrega además un enlace de comunicación entre el primero y el último core de cada fila y de cada columna. Existen varias CPU comerciales que utilizan algunas de estas topologías, por ejemplo, la primera generación de la familia *core ix* de *Intel* utilizaba la distribución crossbar, mientras que arquitecturas más avanzadas de esa familia, como *Sandy Bridge*, utilizan un anillo de comunicación. En el caso de la malla, esta topología es utilizada en los procesadores *TILE* del fabricante Mellanox Technologies, en particular en la arquitectura TILE64 [24].

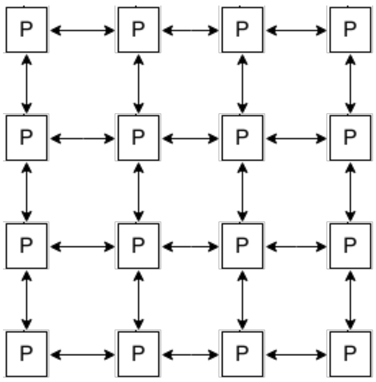
Además de los retos mencionados anteriormente aparece un tercero, que es cómo administrar los hilos que trabajan dentro o entre procesadores multicore, de forma de utilizar de manera eficiente el acceso a las distintas memorias. De este problema surge el concepto de afinidad, lo cual permite asociar a un hilo o conjunto de hilos a un cierto recurso de cómputo, buscando explotar la disponibilidad de acceso a memoria de estos recursos o de la localidad espacial de los datos [25]. Un ejemplo de configuraciones para este tipo de propiedad se puede ver en la API OpenMP [26], dónde se presentan tres tipos distintos de configuraciones. Por un lado, se tiene la afinidad *spread*, la cual distribuye los



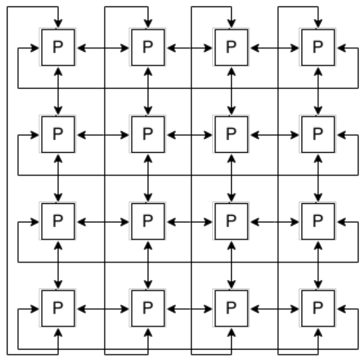
(a) Topología Crossbar



(b) Topología de anillo



(c) Topología de malla



(d) Topología toroidal

Figura 2.2: Ilustración de las cuatro interconexiones más utilizadas para CPUs.

hilos lo más lejano posible entre el procesador o los procesadores, de forma de aprovechar por ejemplo distintos buses de acceso a memoria principal de distintos procesadores. Por otro lado, la configuración *compact* coloca a los hilos a ejecutar lo más próximo posible entre sí, explotando de esta forma la localidad espacial de los datos a utilizar a través de memorias caché compartidas entre los cores asignados. Finalmente, existe una configuración manual que permite distribuir los hilos entre los cores disponibles, de forma que el usuario adapte de la mejor forma la afinidad a la aplicación que está utilizando.

2.4. GPUs

Las unidades de procesamiento gráfico, o por su sigla en inglés GPUs (Graphics Processing Units), son un coprocesador diseñado originalmente para el procesamiento de imágenes que está principalmente orientado a liberar la carga de cómputo de la CPU en aplicaciones con una alta demanda de este tipo de operaciones, como pueden ser videojuegos o simulaciones con un alto con-

tenido 3D, entre otras. Debido a su arquitectura, similar a SIMD, estos chips presentan un gran número de núcleos, de menor velocidad que los que posee una CPU, pero que en su conjunto resultan una herramienta potente para cálculos sistemáticos e independientes entre sí.

En la última década, se han desarrollado distintos lenguajes de programación general orientados a estas arquitecturas que permiten no sólo trabajar con procesamiento gráfico, sino que también con problemas de propósito general como simulaciones y análisis que necesiten un gran poder de cómputo y sean masivamente paralelizables.

2.4.1. GPGPU

GPGPU, del inglés General Purpose Computing on Graphics Processing Units, es el término que se reserva para las técnicas que aprovechan la naturaleza altamente paralelizable que ofrecen las tarjetas gráficas para codificar programas de propósito general [27].

En la década del 90, y hasta que fuera posible programar sobre estas tarjetas a fines de la primera década del siglo XXI, las GPUs estaban pura y exclusivamente dedicadas al procesamiento gráfico, ya que estas unidades eran preprogramadas de fábrica para realizar cómputos asociados al pipeline gráfico [28]. A pesar de ello, algunos investigadores notaron el gran poder de cálculo que este tipo de hardware ofrecía y decidieron explotar dicha virtud para otro tipo de problemas. Para ello utilizaron conocimientos específicos del pipeline gráfico y de las estructuras de datos que manejaban las GPUs de forma de poder empaquetar en dichas estructuras los valores necesarios para computar las funciones que deseaban calcular. Un ejemplo de esta situación fue el trabajo realizado en [29], en el cual se lleva un problema matemático (conocido como la Transformada Rápida de Fourier o FTT, del inglés Fast Fourier Transformation) a funciones de rasterización, logrando aprovechar el poder de cómputo de la GPUs. Los resultados alentadores obtenidos en este y otros trabajos de la época, sumado al constante crecimiento del poder de cómputo de las GPUs, llevó a la necesidad de generar lenguajes de programación de propósito general para esta plataforma de hardware.

A la fecha existen dos grandes lenguajes de programación de propósito general sobre GPUs: CUDA [30] y OpenCL [31]. OpenCL es un estándar creado por el Grupo Khronos, formado entre otros por Apple, AMD, IBM, *Intel* y

Nvidia, en el año 2008 cuyo objetivo es permitir al usuario construir programas que ejecuten tanto sobre CPUs multicore como GPUs. De esta forma, se busca crear aplicaciones que exploten arquitecturas paralelas y tengan un bajo costo de implementación, pudiendo crear funciones que se puedan compilar y utilizar en distintas plataformas. En cambio, CUDA es desarrollado por la empresa *Nvidia* desde el año 2007 y solamente puede ser ejecutado en las arquitecturas diseñadas por esta compañía. Junto con este lenguaje, *Nvidia* introdujo un modelo de programación paralela, similar a SIMD, llamado *Single Instruction Multiple Thread* (SIMT). Este modelo conserva alguna de las propiedades de su referente en la taxonomía de Flynn (SIMD), pero agrega tres características destacables que le ofrecen flexibilidad al momento de ejecutar instrucciones [32]. La primera propiedad que presenta este modelo es el manejo sencillo y eficiente de registros por parte de los hilos. Mientras que en SIMD es necesario particionar los datos en vectores de tamaños pequeños para hacer el procesamiento más eficiente, en SIMT cada hilo tiene sus propios registros donde almacenar sus variables, haciendo transparente el manejo de los datos para cada unidad de procesamiento. Otra característica que presenta esta arquitectura es la posibilidad de acceso a memoria no contigua de forma más eficiente y menos compleja al momento de programar que en SIMD. Por último, su singularidad más relevante frente a SIMD es la posibilidad de ejecutar distintos flujos de código de manera transparente al programador, lo que permite manejar divergencias en la ejecución de los hilos en los programas desarrollados (este concepto se explicará en más detalle en la próxima sección).

Considerando que muchos trabajos en GPU están desarrollados sobre CUDA (como se verá en la revisión del estado del arte en el capítulo siguiente), en este trabajo se toma dicho lenguaje como referencia, lo que lleva a explorar en este documento un poco más en profundidad las arquitecturas de hardware de GPUs del fabricante *Nvidia*.

2.4.2. GPUs de *Nvidia*

Existen a la fecha varias arquitecturas de tarjetas grafica *Nvidia* capaces de ejecutar código de propósito general. La primera arquitectura que soportó este tipo de programación fue llamada Tesla, y comprendía tanto tarjetas especializadas para juegos como las primeras orientadas al cómputo científico. Luego a esta arquitectura la siguieron Fermi, Kepler, Maxwell, Pascal y Volta [33], me-

jorando distintos aspectos tanto en poder de cómputo como velocidad y tamaño de la memoria disponible, entre otras cosas. En esta sección se verán algunas de las características que todas estas arquitecturas comparten y permiten el correcto funcionamiento de programas de propósito general, sin profundizar en ninguna arquitectura en particular.

Las tarjetas gráficas de *Nvidia* son coprocesadores compuestos por un arreglo escalable de multiprocesadores y un conjunto de memorias estructuradas en una jerarquía. Cada multiprocesador cuenta con varios núcleos, llamados *CUDA cores*, los cuales son capaces de ejecutar simultáneamente múltiples hilos, tanto para realizar operaciones enteras como de punto flotante. El código de una función que es procesada por la tarjeta es llamado *kernel*, siendo éste ejecutado por cada uno de los hilos. Para poder aprovechar de forma correcta los recursos de cómputo, y como se puede observar en la Figura 2.4, el *kernel* se organiza en una estructura bidimensional (aunque también puede ser tri- o unidimensional), llamada grilla. Esta estructura está compuesta por bloques, los cuales son asignados a cada uno de los multiprocesadores. Además, cada uno de estos bloques está compuesto por hilos, que serán ejecutados en cada uno de los *CUDA cores* presentes en cada multiprocesador. Para lograr que la ejecución de esta estructura de datos se realice de forma adecuada, la GPU posee un planificador de bloques, el cual decide qué conjunto de éstos se va a ejecutar en cada multiprocesador.

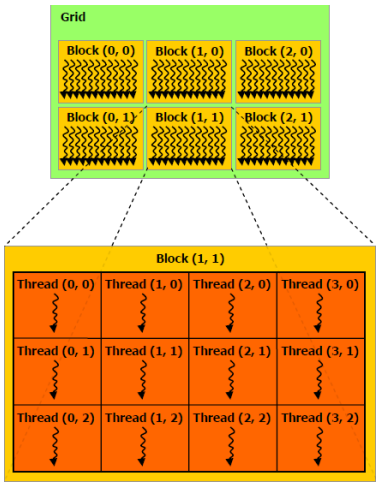


Figura 2.3: Jerarquía de hilos en una GPU *Nvidia*. Extraído de [30].

Dentro de cada multiprocesador, los bloques son separados en estructuras de 32 hilos, llamados *warps*, los cuales serán ejecutados en un mismo instante de tiempo en el multiprocesador. Al separar los bloques en warps, el manejador dentro del multiprocesador administra los *CUDA cores* de la forma más eficiente posible, por ejemplo cambiando de warp si el que está ejecutando debe realizar un acceso a memoria. Por esta razón se recomienda tener bloques con una cantidad de hilos mayor al número de cores disponibles en el multiprocesador, lo que permite un mejor aprovechamiento de los recursos de cómputo de la GPU.

Una característica de gran relevancia en la arquitectura de *Nvidia* es el uso de una jerarquía de memorias, con el objetivo acelerar el acceso a los datos a procesar y dotar a la tarjeta de un gran poder de almacenamiento. Dentro de cada microprocesador se encuentran los registros, los cuales funcionan como memoria privada a cada hilo. Los registros poseen un acceso de mayor velocidad si se los compara con el resto de las memorias presentes incluso dentro del multiprocesador, pero su capacidad es extremadamente limitada. Cuando un hilo supera la cantidad de registros asignados, la GPU comienza a utilizar la memoria local (residente fuera del multiprocesador) de forma transparente al usuario, deteriorando el desempeño del programa.

El siguiente nivel en la jerarquía es la memoria compartida, la cual también se encuentra dentro del multiprocesador. Esta memoria se comparte entre los hilos de un bloque y es útil para que puedan cooperar en sus operaciones. Es importante destacar que este espacio de almacenamiento es programable por el usuario, por lo que su uso debe realizarse de forma explícita. Al mismo nivel que la memoria compartida, y también dentro del multiprocesador, se encuentra una caché L1, la cual es transparente para el usuario. Además de estas memorias, dentro del microprocesador existen dos memorias más, la memoria constante y la memoria de texturas. Ambas memorias son programables por el usuario (previo a la ejecución del kernel) pero la memoria de textura solo se puede utilizar para procesamiento gráfico mientras que la constante es de propósito general. En las últimas arquitecturas, la caché L1 y la memoria de texturas están unificadas y funcionan como una memoria caché extendida [30]. Luego, fuera del multiprocesador, se encuentra un segundo nivel de memoria caché y la memoria global. Esta última memoria es compartida por todos los

bloques del kernel en ejecución y es la más abundante en la arquitectura, pero también es la de acceso más lento. Por último, este espacio de memoria también tiene como función interactuar con la memoria de la CPU. En la Figura 2.4 se presenta un ejemplo de la jerarquía descripta previamente para la arquitectura de tarjetas *Nvidia* Kepler.

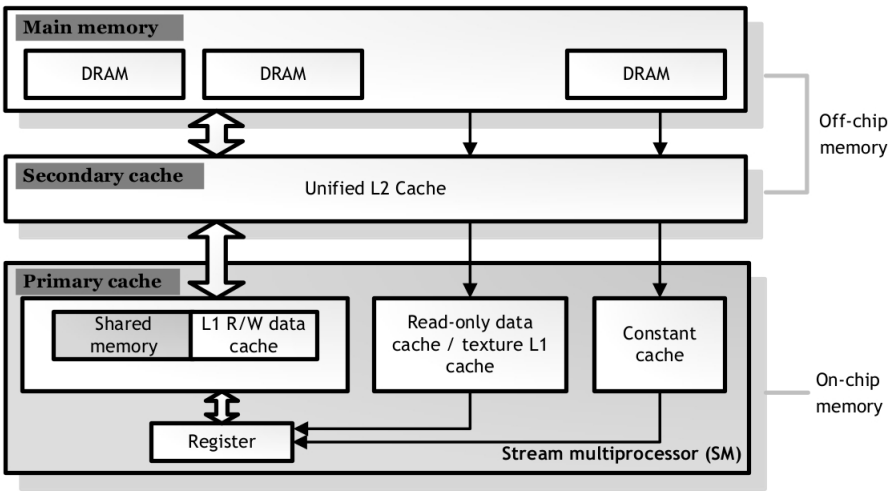


Figura 2.4: Jerarquía de memorias para la GPU GTX780 (Kepler). Extraído de [34].

Con respecto al uso de memoria es importante mencionar que las GPUs de *Nvidia* utilizan una técnica para disminuir los accesos a memoria global, llamada *acceso coalesced*. Esta práctica consiste en intentar agrupar los accesos a memoria de los hilos de un *warp*, o *half-warp* dependiendo de la arquitectura, de forma de realizar menos peticiones a memoria y obtener la misma cantidad de datos necesarios para ejecutar los hilos. En las primeras implementaciones para lograr un *acceso coalesced* era necesario no sólo que los hilos accedieran a lugares dentro del mismo segmento de memoria sino que también requería que estos accedieran en orden (i-ésimo hilo accede a la i-ésima palabra). La técnica fue evolucionando conforme avanzaban las arquitecturas hasta llegar a la implementación actual, donde para obtener un *acceso coalesced* se permite el acceso en desorden dentro de un mismo segmento y, de no cumplirse la condición, los hilos pueden dividirse en subgrupos de forma de obtener la menor cantidad de acceso para ese *warp*.

Una peculiaridad a tener en cuenta al momento de programar un *kernel* es la *divergencia de hilos* dentro de un *warp*. Esta situación ocurre cuando hilos de un mismo *warp* toman distintos flujos de ejecución dentro del *kernel*. Si bien este comportamiento no sería posible en una arquitectura SIMD tradicional, las GPUs de *Nvidia* lo permiten, aunque no es recomendable que suceda ya que esto degrada el desempeño de la tarjeta. Esta pérdida de performance surge debido a que los hilos del warp que ejecutan un flujo alternativo deben esperar para poder ejecutar por el resto de los hilos y viceversa.

Finalmente, una capacidad interesante, diseñada desde el lanzamiento de CUDA, pero realmente soportada completamente en forma más reciente en las tarjetas gráficas de *Nvidia*, es el uso de unidades lógicas que permiten separar los distintos kernels a ejecutar en diferentes flujos de ejecución. Con el uso de esta estructura, llamada *Stream*, se busca que kernels en distintos *Streams* ejecuten, si es posible, de forma simultánea en la tarjeta. Además, junto con el uso de la copia asincrónica a memoria, se permite solapar los costos de envío de datos entre la memoria de la CPU y de la GPU, los cuales usualmente impactan fuertemente en la ejecución total de una aplicación que utiliza esta arquitectura.

2.5. Xeon-Phi

Los coprocesadores Xeon-Phi comenzaron a comercializarse a fines del año 2012 por el fabricante *Intel* [35], con el objetivo de conseguir un gran poder de cómputo paralelo a un costo energético moderado. Adicionalmente, se busca con esta arquitectura brindar dicho poder de cómputo con un bajo esfuerzo de codificación de las aplicaciones. Considerando este último objetivo, los coprocesadores Xeon-Phi soportan C y Fortran como lenguajes de programación y bibliotecas de técnicas paralelas ampliamente utilizadas como son las APIs OpenMP y MPI. El uso de estos lenguajes y herramientas permite a desarrolladores sacar provecho fácilmente de esta arquitectura, tanto para el desarrollo de nuevas implementaciones, así como para adaptar programas ya codificados. Adicionalmente a estas facilidades, los coprocesadores presentan tres modos distintos de ejecución para su utilización, nativo, simétrico y *offload*, que van desde un uso exclusivo del coprocesador hasta un modo en el que solamente ciertas funciones son ejecutadas en él [36]. En particular, el modo nativo consiste en compilar la aplicación en su totalidad para ejecutar en Xeon-Phi,

copiar el ejecutable al coprocesador y finalmente ejecutarlo en el mismo. Por otro lado, el modo simétrico consiste en compilar la aplicación deseada tanto para Xeon-Phi como para el procesador y luego ejecutarla utilizando MPI, buscando que los procesos de la CPU como del coprocesador ejecuten equitativamente. Finalmente, el modo *offload* consiste en localizar en el código aquellas porciones que pueden aprovechar el tipo de cómputo del coprocesador y modificarlo, mediante directivas similares a OpenMP, para que solamente esas secciones ejecuten en el procesador Xeon-Phi.

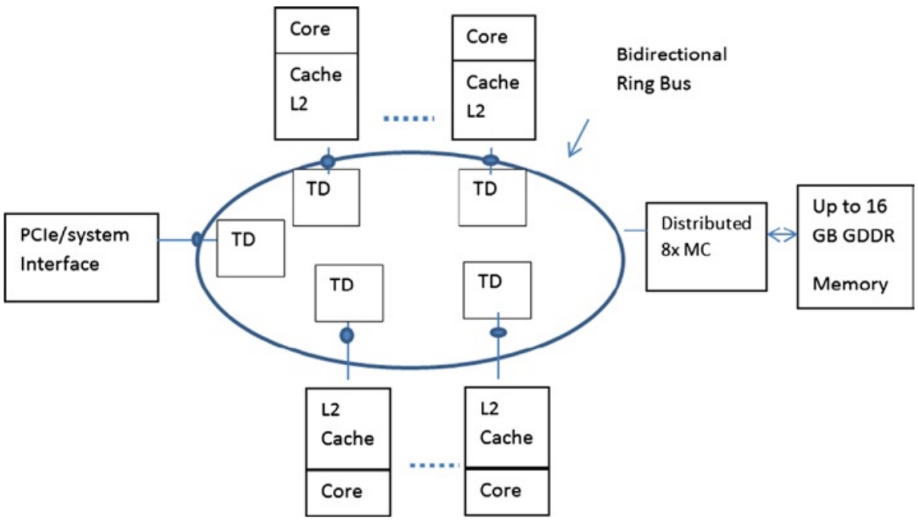


Figura 2.5: Diagrama de la arquitectura del Xeon-Phi. Extraído de [37].

La arquitectura del Xeon-Phi (Knights Corner) está compuesta por decenas de núcleos interconectados a la memoria global del coprocesador por un anillo bidireccional de comunicación, distribuidos como se puede ver en la Figura 2.5 [37]. Los núcleos presentes en este coprocesador son procesadores de la familia Pentium modificados para soportar instrucciones de 64 bits y programación simultánea de cuatro vías. Adicionalmente, estos procesadores poseen una nueva *vector process unit* (VPU) capaz de realizar 16 operaciones de punto flotante de precisión simple en simultáneo (u ocho operaciones de precisión doble) y cuentan con una jerarquía de memorias caché, compuesta por cachés L1 y L2. Gracias a estas modificaciones, cada core es capaz de soportar la ejecución de hasta cuatro hilos simultáneamente, pudiendo entonces ejecutar todo el coprocesador cientos de hilos de forma paralela. Una característica interesante que

diferencia a estos núcleos de procesamiento de los de una GPU es que, al ser procesadores de propósito general, cada hilo puede realizar distintas tareas sin tener que esperar ni bloquear la ejecución de ningún otro hilo.

Una de las claves para el funcionamiento eficiente de esta arquitectura es el anillo de comunicación, el cual permite a los procesadores acceder a los datos necesarios. Este anillo presenta un canal de comunicación hacia cada dirección, compuesto cada canal por tres capas, una de datos, otra de direcciones y una de control o *acknowledgement*. La capa de datos es la encargada de enviar la información a procesar a los núcleos mientras que la capa de direcciones sirve como canal para enviar las direcciones de memoria que se desean leer o escribir. Finalmente, la capa de control es la encargada de transportar los datos de control de flujo y de coherencia de memoria.

Una propiedad interesante que agrega este anillo de comunicación a cada core, es la posibilidad de acceder no sólo a la memoria global sino también a las memorias caché L2 del resto de los procesadores. Para lograr este objetivo, cada core presenta adicionalmente un *tag directory* (TD), de forma que entre todos constituyen una tabla con todas las porciones de memoria repartida entre las distintas memorias caché L2.

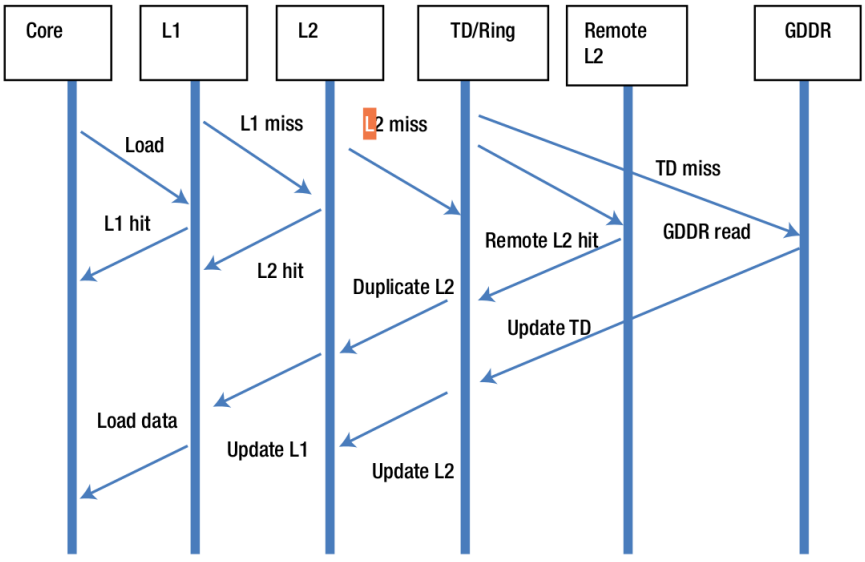


Figura 2.6: Protocolo de obtención de datos de un núcleo de Xeon-Phi. Extraído de [37].

Como se puede ver en la Figura 2.6, el proceso de obtención de un dato para un core consiste, en un principio, en preguntar a su propia jerarquía de memorias caché. En caso de no encontrar el dato en su propio entorno, se envía una petición al TD para buscarlo fuera del procesador. En ese momento el TD envía al resto de estas estructuras una petición por ese dato y en caso de que esté almacenado en alguno de los cores, otro TD le envía de la caché L2 de su procesador el dato requerido. Finalmente, el TD original le envía el dato al procesador y éste lo utiliza, previamente almacenándolo en su caché L2. En caso de que no exista el dato en la estructura conformada por todas las caché L2 de los procesadores, el TD gestiona la petición de éste a la memoria global y cuando sea obtenido lo retorna al procesador.

Utilizando esta estructura de hardware y protocolos de comunicación cada core es capaz de obtener los datos necesarios para su ejecución, minimizando potencialmente el acceso a la memoria global. Esta cualidad impacta positivamente en el desempeño de las aplicaciones, ya que potencialmente disminuye fuertemente la latencia de acceso a datos, lo cual es un cuello de botella en toda arquitectura many-core.

Para finalizar, es importante mencionar que la arquitectura descrita previamente no es la versión más reciente de los procesadores Xeon-Phi [38]. En la actualidad la arquitectura cuenta con decenas de unidades de procesamiento llamadas *tiles*, que incluyen dos procesadores cada uno, donde cada procesador presenta un core y dos VPUs. Además, el anillo de comunicación fue remplazado por una grilla 2D de *tiles*, donde cada *tile* puede comunicarse con sus vecinos tanto en las direcciones verticales como horizontales. Asimismo, el procesador Xeon-Phi viene no solo en versión coprocesador, sino que en la actualidad puede ser utilizado como procesador principal, en lugar de una CPU clásica. A pesar de que esta arquitectura es sustancialmente distinta a la descripta en esta sección, en este trabajo se utilizó hardware basado en la arquitectura original y, por esta razón, es la que se describe con mayor detalle.

2.6. Medidas de desempeño

A la hora de evaluar un programa que posee una sección paralela, es necesario seleccionar métricas que nos permitan ponderar las mejoras introducidas al explotar el paralelismo presente. Adicionalmente, es importante intentar conocer cuál es la máxima mejora alcanzable por el programa.

Una de las métricas más utilizadas para evaluar la mejora que se obtiene de ejecutar un programa utilizando más de un recurso de cómputo es la conocida como `speedup`. Esta medida calcula el cociente de los tiempos de ejecución entre la versión del programa ejecutada utilizando un único recurso de cómputo (conocida como versión secuencial) y la versión del programa que utiliza más de uno de estos recursos. La ecuación utilizada para calcular esta métrica es la siguiente:

$$speedup = \frac{tiempo\_versión\_secuencial}{tiempo\_versión\_multihilo}. \tag{2.1}$$

Adicionalmente a esta medida, se suele computar en el estudio de los tiempos de ejecución de un programa paralelo la *eficiencia* de dicho programa. El objetivo de esta métrica es evaluar que tanto provecho obtiene el programa ejecutado en paralelo respecto a una implementación serial al agregar unidades de cómputo. El valor de *eficiencia* se calcula de la siguiente forma:

$$eficiencia = \frac{speedup}{cantidad\_de\_recursos}. \tag{2.2}$$

Finalmente, cuando se desea evaluar una implementación paralela de un programa, resulta atractivo calcular cual será la cota superior del desempeño de esta implementación. En otras palabras, computar de forma teórico-práctica cual es la mayor disminución en el tiempo de ejecución del programa que puede ser alcanzada con la versión propuesta. Para calcular esta máxima mejora, se utiliza el concepto introducido por Gene Amdahl, conocido como la Ley de Amdahl [39]. En esta ley, Amdahl plantea que si se considera una región paralelizable f dentro de un programa con otras secciones secuenciales, el `speedup` alcanzable tiene la siguiente forma:

$$speedup(f, N) = \frac{1}{(1 - T(f)) + \frac{T(f)}{N}} \tag{2.3}$$

donde N es la cantidad de núcleos utilizados para ejecutar la sección paralelizable f y $T(f)$ es el tiempo que toma ejecutar la región f. Se puede notar entonces que, si se dispone de una cantidad infinita de recursos para ejecutar la sección f el máximo `speedup` alcanzable es de $\frac{1}{(1-T(f))}$, por lo que, teóricamente, la mayor reducción es la alcanzada cuando se considera que toda la sección f es quitada de la ejecución del programa.

Capítulo 3

Predicción de la generación de energías renovables

Las energías renovables son aquellas obtenidas del medio ambiente, las cuales son naturalmente de flujo persistente y repetitivo [40]. También llamadas energías sustentables, éstas pueden ser explotadas sin comprometer en el futuro su obtención, a diferencia de por ejemplo el petróleo o el gas natural. Dentro de ellas se encuentran la energía solar, eólica, mareomotriz e hidroeléctrica, entre otras. Un inconveniente intrínseco a este tipo de energías es que, si bien son de carácter duradero, es difícil asegurar su regularidad en el tiempo.

En este capítulo se resume el pasado y presente de la explotación de las energías renovables en Uruguay, haciendo especial foco en el fenómeno de la energía solar. Adicionalmente, se presentan las principales características de una herramienta desarrollada en la Facultad de Ingeniería capaz de predecir las distintas propiedades de la atmósfera basado en un modelo numérico y se resumen distintos trabajos realizados sobre este modelo para mejorar su desempeño computacional.

3.1. Energías renovables y pronóstico de energía en Uruguay

Uruguay posee vastos recursos naturales para el desarrollo y la explotación de las energías renovables. Históricamente, Uruguay ha utilizado ampliamente la energía de origen hidráulica, contando con cuatro represas hidroeléctricas (Gabriel Terra, Baygorria y Constitución ubicadas sobre el río Negro y Salto

Grande localizada en el norte del río Uruguay) con un total de 1500 MW de potencia instalada. Como este tipo de energía depende fuertemente del caudal de los ríos y expone al país a la falta de energía en caso de sequía, en los últimos años se ha estado trabajando firmemente en la diversificación de la matriz energética. El objetivo de este esfuerzo es estabilizar el poder alcanzable por fuentes renovables y así disminuir o eliminar el consumo de energías obtenidas del petróleo o gas natural, las que tienen costos muy elevados tanto económicos como ambientales. En particular, en la última década se ha invertido fuertemente en el desarrollo de la energía eólica, contándose en la actualidad con 35 parques en funcionamiento (con una potencia conjunta de 1270 MW, dónde sobresalen los parques Pampa con una potencia de 141 MW y Peralta I y II con una potencia nominal de 117 MW) y otros cuatro en etapa de construcción [1].

Relacionado a la energía solar, Uruguay está ubicado en un rango de latitud geográfica que registra una irradiación diaria promedio sobre el plano horizontal equivalente a unos 4,70 kWh/m^2 (aproximadamente la mitad de la energía eléctrica consumida por día en una familia tipo en Uruguay). Este valor promedio se puede ver afectado tanto por la variación estacional, o sea por la diferencia en la intensidad de la radiación según la época del año, así como por la variación territorial, que refiere a la uniformidad geográfica de nuestro territorio. Para la variación estacional los cambios suelen ser significativos, con valores medios que oscilan entre 2,20 kWh/m^2 en invierno y 7,00 kWh/m^2 en verano, mientras que la variación territorial suele ser menor, teniendo medias que van de un mínimo de irradiación anual de 4,35 kWh/m^2 en Rocha y un máximo de 4,80 kWh/m^2 en la zona de Artigas [41]. Estos datos pueden ser observados en el mapa solar del país, ver la Figura 3.1, realizado por la Dirección Nacional de Energías en conjunto con investigadores de la Facultad de Ingeniería - UdelaR [41].

En nuestro país, la energía solar es una temática de gran interés en la actualidad. En el año 1992, UTE[1] inició la primera experiencia con paneles fotovoltaicos, colocando una baja cantidad en escuelas públicas, destacamentos policiales y policlínicas rurales, alejadas de la red de distribución eléctrica. Como este ensayo fue satisfactorio, se inició un plan entre el año 2004 y 2010 para instalar 1000 equipos fotovoltaicos en distintos lugares del país que cumplieran

[1] Administración Nacional de Usinas y Trasmisiones Eléctricas (UTE): empresa estatal encargada de la generación (en principal medida), distribución y comercialización de la energía en el país.

con las mismas características que los de la primera experiencia. Posteriormente, en el marco de un plan piloto y en colaboración con capitales japoneses, se instaló en 2013 la primera planta fotovoltaica del país. En la actualidad se encuentran en funcionamiento tres parques adicionales, sumando una potencia total de 78 *MWh*, y se encuentran en etapa de habilitación y construcción otros 14 parques de estas características [42].

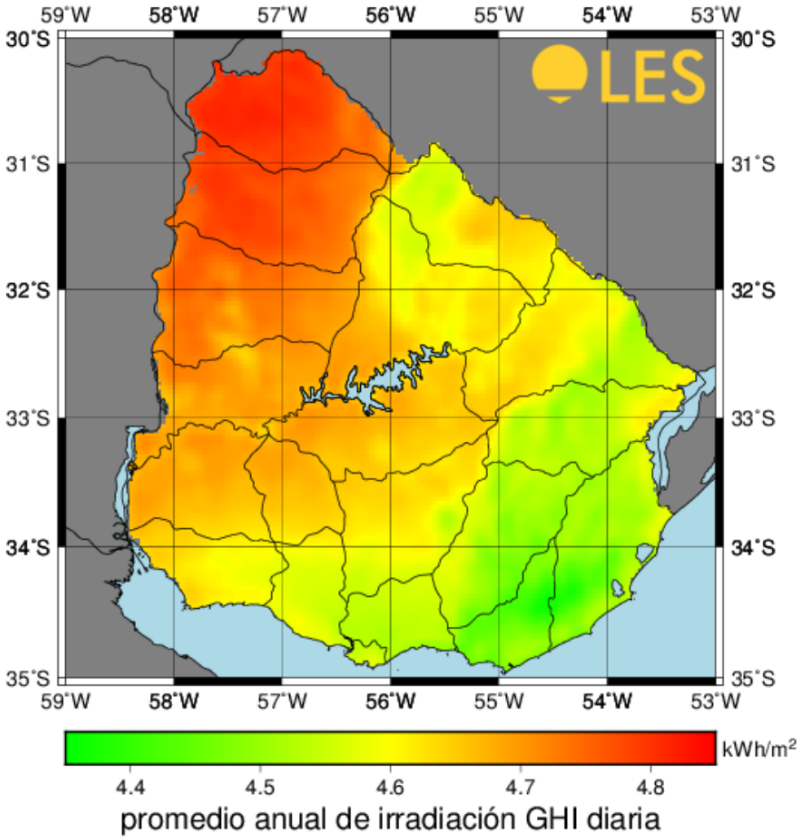


Figura 3.1: Mapa solar de Uruguay – Irradiación diaria promedio en (kWh/m^2). Extraído de [41].

Es oportuno señalar que existe una relación entre los movimientos de masas de aire y las diferentes temperaturas y presiones en la atmósfera provocadas por la absorción de la radiación solar, lo que resulta en que la energía solar y la energía eólica se comporten de forma complementaria [43]. De esta forma, cuando una de ellas se encuentra en un momento de generación baja, la otra generalmente se encuentra en un momento de producción alto. Así, el

uso conjunto de este tipo de energías renovables se vuelve de interés para la conformación de la matriz energética de cualquier país, y en particular, para el Uruguay.

Una vez decididas las fuentes de energía que van a proveer a la red de un país, surge el desafío de administrar estos recursos de forma eficiente, tarea conocida como despacho de carga. Esta tarea debe considerar cuál es la mejor configuración para mantener el suministro de energía necesario para satisfacer a la matriz energética en cierto espacio temporal, considerando costos tanto ambientales como económicos. En esta tarea es de suma importancia igualar la producción de energía con el consumo en todo instante de tiempo, de forma de evitar un desabastecimiento energético por un lado y estados inestables por otro. Este equilibrio suele no ser sencillo de alcanzar, resultando por lo general en ciertos excedentes de producción.

Este excedente energético presenta un desafío ya que, al no ser consumido por la red, y al no existir métodos eficientes de almacenamiento energético, tiene que o bien transmitirse a otra matriz energética, a través de la venta de esta energía a otro país, o bien debe ser eliminado de la red. En caso de no existir un remanente de energía producida, la tarea de despacho de carga debe decidir qué energías serán las utilizadas para abastecer a la red, teniendo en cuenta por ejemplo el impacto económico de utilizar o no una fuente de energía. Un ejemplo de este procedimiento en nuestro país es decidir si se necesita incorporar la energía obtenida por petróleo, la cual resulta en nuestro contexto altamente costosa. Otro aspecto a tener en cuenta es el impacto ambiental que trae utilizar cierta fuente de energía, por ejemplo, sería deseable abastecer la red utilizando el mayor porcentaje de fuentes renovables. Además, sería razonable priorizar aquellas que no pueden almacenarse, como la energía solar o la eólica, dejando el uso de la energía hidroeléctrica postergado, ya que ésta puede acumularse mediante embalses (hasta cierta cota) para un futuro.

Por último, esta tarea requiere una minuciosa planificación que permita prever en una ventana temporal aceptable tanto el consumo como el abastecimiento ya que ciertas decisiones no pueden ser instantáneas. Por ejemplo, tener una planificación a futuro permite anticiparse a un excedente y contactarse con otros despachos de carga para intentar que ellos lo absorban o comenzar el proceso de extracción de energía que presenta un retardo en la generación, como es el caso de las centrales que utilizan petróleo. Vistos todos

los inconvenientes que conlleva la difícil tarea del despacho de carga, se ve destacada la importancia de disponer de herramientas para la predicción de la potencia suministrada por las distintas fuentes de energía a la red energética nacional.

3.2. Energía solar

La energía solar, como su nombre lo indica, es aquella energía obtenida a partir de la explotación de la radiación electromagnética proveniente del Sol. La energía solar a gran escala puede ser aprovechada principalmente de dos formas, mediante la conversión térmica de alta temperatura y la conversión fotovoltaica [44].

La primera forma de uso consiste en transformar la energía solar en energía térmica, a través del almacenamiento en un fluido, el que usualmente es aire, agua o aceite. Para cumplir con este fin, se utilizan los dispositivos denominados colectores, los cuales se clasifican en dos familias, los estacionarios y los de concentración [45]. Por un lado, los estacionarios son colectores que se encuentran siempre en una posición fija y utilizan toda su superficie para acumular y transmitir el calor al fluido. Por otro lado, los de concentración se basan en utilizar superficies cóncavas reflectivas, que pueden moverse conforme se mueve el sol, de forma de concentrar todo el flujo solar en una superficie pequeña, donde se encuentra el fluido a calentar [45].

La segunda forma de usufructuar esta energía se basa en transformar de forma directa la energía solar en energía eléctrica, utilizando placas formadas por celdas fotovoltaicas [46]. Estas células fotovoltaicas son capaces de generar energía a partir de los fotones presentes en la radiación solar a través del efecto fotovoltaico. Para lograr dicho efecto, las celdas están compuestas por dos semiconductores unidos, uno con carga positiva y otro con carga negativa, formando lo que se conoce como la unión n-p (n representa al semiconductor negativo y p al positivo). Esta unión genera un campo eléctrico en la zona de contacto de las mismas. Cuando se aplica radiación solar a esta celda, la energía absorbida permite que los electrones escapen de su posición en los átomos y formen parte de la corriente eléctrica, pudiendo utilizar esta corriente para abastecer a algún circuito conectado a dicha celda [46, 47].

Resulta de interés entonces describir el fenómeno de radiación solar (previo a entrar a la atmósfera y hasta su contacto con la superficie terrestre) con el objetivo de comprender qué porcentaje de esta energía puede ser potencialmente aprovechada. Como se puede ver en la Figura 3.2, el Sol emite hacia la Tierra lo que se llama radiación de onda corta, es decir radiación en el espectro electromagnético que está principalmente entre los 0,23 y 2,26 μm. Se estima que la radiación de onda corta en la zona extraterrestre (zona previa a la atmósfera terrestre) es de 1367 W/m^2, conocida como constante solar. Una vez que la radiación alcanza la atmósfera, aproximadamente el 30 % de ésta es reflejada al espacio nuevamente, ya sea por reflexión de la atmósfera, las nubes o la propia superficie terrestre. Se estima además que el 16 % es absorbido por vapor de agua, polvo u ozono presente en la atmósfera y el 4 % es absorbido por las nubes, dejando llegar entonces a la superficie terrestre alrededor de la mitad de la radiación que alcanza a la zona extraterrestre. Esta porción de radiación que alcanza la superficie terrestre puede provenir de dos vías, o bien es parte del flujo de radiación que no fue afectado, llamado rayo solar, o es radiación de onda corta proveniente del fenómeno de dispersión de la atmósfera (llamada radiación difusa). Una vez que la radiación de onda corta es absorbida por la Tierra, nubes o la propia atmósfera, esta energía es emitida en forma de radiación de onda larga, cuya onda es de más de 3 μm. Finalmente, esta onda es atrapada en su mayoría en la atmósfera terrestre gracias a los gases de efecto invernadero [47].

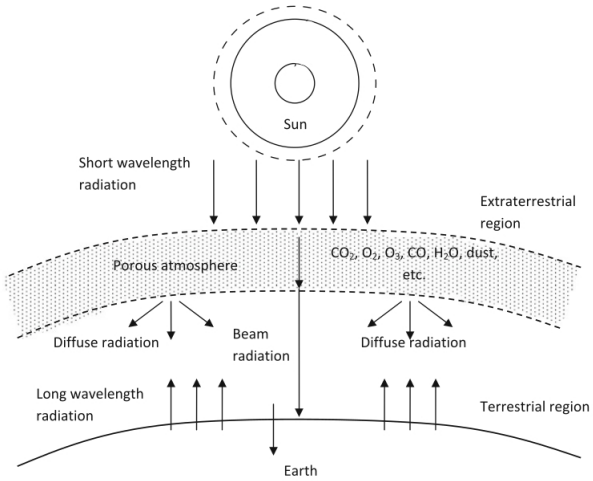


Figura 3.2: Interacción de la radiación solar con la Tierra y sus distintas etapas. Extraído de [47].

Se estima que el consumo de energía eléctrica mundial anual durante 2010 fue de aproximadamente 500 *EJ*, mientras que se calcula que la energía total que llega a la atmósfera terrestre en forma de radiación solar equivale a 5,460,000 *EJ* al año. Si bien no toda la energía que llega a la atmósfera alcanza la superficie terrestre, aun suponiendo que solamente un 10 % de la energía total emitida la alcanza, bastaría con aprovechar un 0,1 % de ésta para abastecer al mundo de energía [48]. Teniendo en cuenta estos números, se hace evidente el gran potencial que presenta esta forma de generación de energía.

3.3. Herramienta para la predicción de la generación de energía solar

Un inconveniente al momento de utilizar la energía solar fotovoltaica para abastecer la red eléctrica de una región es la intermitencia de potencia que presenta. Esta característica hace que este tipo de energía sea difícil de incorporar, dado que vuelve compleja la tarea de coordinar el uso de las distintas fuentes de energía para mantener la estabilidad de la red. Por esta razón, resulta de interés poder pronosticar su comportamiento y así aprovechar al máximo todos los recursos presentes en la matriz energética, pudiendo incluso diferir el uso de aquellas fuentes que se puedan almacenar de manera eficiente.

3.3.1. Predicción de energía solar en Uruguay

En los últimos años, se ha comenzado a estudiar en el país las posibilidades de desarrollo que ofrece la energía solar fotovoltaica, principalmente como complemento a la energía eólica. Por este motivo, en el Instituto de Mecánica de los Fluidos e Ingeniería Ambiental (IMFIA-FING-UdelaR) se ha comenzado a desarrollar una herramienta de predicción de energía solar fotovoltaica sobre el territorio del país, similar a la existente para la predicción de la generación de la energía eólica [49]. Específicamente, para la energía eólica se realizó un modelo de predicción de la generación que consiste en tres etapas. Concretamente, estas etapas se pueden resumir en: obtener los datos del Global Forecast System (GFS), ejecutar un modelo regional de la atmósfera con

dichos datos utilizando el modelo WRF y finalmente realizar ciertas correcciones de la salidas considerando la dirección de la velocidad del viento, así como la interferencia entre los aerogeneradores. Teniendo en cuenta este esfuerzo, se construyó para la energía solar un modelo compuesto por cinco etapas, que van desde la obtención de datos iniciales para realizar la simulación de la atmósfera hasta la estimación de energía eléctrica que va a producir la planta fotovoltaica.

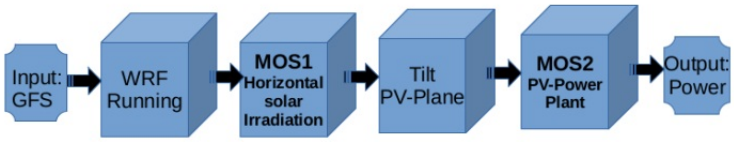


Figura 3.3: Arquitectura de la herramienta de predicción de energía fotovoltaica. Extraído de [4].

Como se puede observar en la Figura 3.3 el proceso comienza con la extracción de datos del GFS, los cuales son utilizados como condiciones iniciales y de frontera para el modelo numérico predictor de clima. Una vez obtenidos los datos, se procede a ejecutar un modelo regional, en este caso el WRF, en la ventana de tiempo que se quiere predecir. Al finalizar la ejecución del modelo, se obtienen los datos de predicción de la atmósfera, de los cuales se extraen las variables relacionadas con la radiación solar. Posteriormente, se procesan estas salidas utilizando un modelo estadístico, llamado Model Output Statistics (MOS). En particular, en esta etapa se realiza una regresión lineal utilizando datos medidos por piranómetros y los datos de salida del WRF, para así obtener la radiación solar horizontal corregida. Luego de esta etapa, se ajustan los valores de radiación en función de la inclinación de los paneles solares, obteniendo así valores aproximados de la energía potencialmente producida por la planta que se desea predecir. Finalmente, se aplica un segundo MOS utilizando los valores de energía generados por la planta fotovoltaica en períodos anteriores, obteniendo así la predicción de cantidad de energía a generar por dicha planta en la ventana de tiempo simulada [4].

Dentro de este complejo proceso de predicción y corrección de datos, una de las etapas que más recursos computacionales necesita es el cómputo del

modelo de predicción de clima, el WRF. Esto se debe a que, si bien para el análisis de la generación de energía fotovoltaica sólo se necesitan los datos de radiación, es necesario simular todos los fenómenos (procesos físicos) presentes en la atmósfera (o al menos los más significativos) para obtener datos correctos relacionados a esta característica. Por esta razón, es importante analizar el comportamiento de este modelo tanto desde un punto de vista conceptual como computacional.

3.3.2. El modelo WRF

El programa *Weather Research and Forecasting*, o WRF, es un modelo numérico mesoescala que permite predecir la dinamica de la atmosfera, orientado tanto al ámbito de la investigación atmosférica como al pronóstico meteorológico diario. Este modelo fue creado a principios de la década de los 90 por distintos centros de estudio relacionados a la investigación de los fenómenos de la atmósfera, como el National Center for Atmospheric Research (NCAR), el National Oceanic and Atmospheric Administration (NOAA) y la Universidad de Oklahoma, entre otros. Existen dos variantes de este modelo, el WRF-ARW (*The Advanced Research WRF*) y el WRF-NMM (*Nonhydrostatic Mesoscale Model*). El primero es mantenido por la comunidad del NCAR Mesoscale and Microscale Meteorology Division y el segundo por la comunidad de Developmental Testbed Center (DTC). Ambas versiones son utilizadas y siguen vigentes hasta la fecha [50]. El WRF es utilizado por una gran cantidad de usuarios en casi 150 países y, dado su amplio propósito, es utilizado en diversos contextos que van desde universidades y centros de investigación, pasando por centros nacionales de predicción como el National Observatory of Athens o el INstituto Uruguayo de METeorología (INUMET), hasta aplicaciones comerciales orientadas a usuarios finales, como WindGuru [51].

El WRF es un modelo no hidrostático, euleriano y compresible [52]. Se entiende por modelo no hidrostático aquel que no supone un equilibrio en el eje vertical entre la presión y la fuerza gravitatoria. El modelo se dice euleriano porque utiliza un modelo de coordenadas fijo, en este caso con respecto a la Tierra. Finalmente, el modelo es compresible porque contempla los cambios en las densidades de los fluidos involucrados en las simulaciones. El modelo admite simulaciones tanto sobre datos reales, lo cual es útil por ejemplo para pronósticos meteorológicos, así como casos ideales (por ejemplo, para estudiar

fenómenos atmosféricos como huracanes), variando las zonas sobre las cuales se ejecuta desde algunos metros hasta cientos de kilómetros.

Debido a la complejidad y gran tamaño que presenta el código de la aplicación, sumado a que su desarrollo es fundamentalmente comunitario, la arquitectura del software WRF se separa en capas, de forma de hacer el proyecto mantenible y escalable. Como se presenta en la Figura 3.4, el WRF tiene un módulo superior que se encarga del control del programa, lo que incluye: crear las estructuras de datos necesarias para la simulación, cargar los datos iniciales en estas estructuras, tomar decisiones sobre el nivel de paralelismo que va a tener la aplicación, entre otras. En la segunda capa, es donde se encuentra el solver de atmósfera, el cual puede ser el ARW o el NMM. Estos solvers son los encargados de la integración espacial y temporal de las ecuaciones de Euler, calcular algunas microfísicas presentes en el modelo y mantener algunas variables de la simulación. En particular, para el caso del ARW, el modelo utiliza un método de Runge–Kutta de tercer orden modificado para la integración temporal (es de orden cúbico en la precisión para las ecuaciones lineales y de orden cuadrático para las ecuaciones no lineales), mientras que para la discretización espacial utiliza una grilla del tipo Arakawa Staggered C-Grid [53]. Finalmente, se tienen dos capas, una que actúa de interfaz para comunicarse con los módulos que se encargan del cálculo de fenómenos físicos y otra que contiene dichos módulos. Dentro de los módulos para calcular los fenómenos físicos de la atmósfera se encuentran distintas variantes para computar cada uno de éstos, y en especial se cuenta con varias implementaciones para calcular la radiación solar.

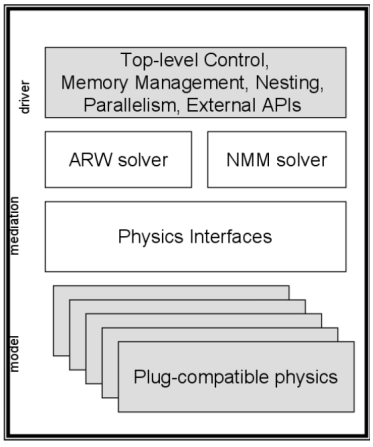


Figura 3.4: Arquitectura del programa WRF. Extraído de [54].

El WRF está codificado en lenguaje Fortran, originalmente utilizando la versión Fortran 77, aunque en la actualidad la mayoría del código es traducido automáticamente a Fortran 90 (o Fortran 95) para lograr así sacar provecho de las facilidades en programación modular y programación de arreglos que este lenguaje provee.

Además, el WRF presenta una implementación capaz de aprovechar el poder de cómputo de plataformas de hardware paralelas, orientada al paralelismo de datos tanto distribuido como centralizado. Específicamente, el modelo puede separar la grilla sobre la cual va a trabajar en subgrillas, las que serán enviadas a ejecutar a cada uno de los nodos (en caso de trabajar en plataformas distribuidas). Luego, esas subgrillas pueden ser separadas en `tiles`, los cuales serán ejecutados cada uno en un hilo dentro de ese nodo, aprovechando así arqutecturas de hardware multinúcleo.

Módulo de Radiación

El módulo de radiación es el encargado de administrar la simulación del fenómeno de la radiación solar, el cual involucra todo el proceso descripto en la Sección 3.2. En términos generales, se separa el cálculo en dos secciones, una que calcula la radiación de onda corta (SW -del inglés Shortwave-) y otro la radiación de onda larga (LW -del inglés Longwave-). Este módulo forma parte del driver de física del modelo WRF y es el encargado de ejecutar todos los módulos relacionados a distintas propiedades físicas de la atmósfera, como por ejemplo la microfísica, la configuración de nubes cumulus, la capa de superficie, etc. En concreto, la salida del módulo de la radiación sirve de entrada para diversos módulos de física presentes en el modelo. Por esta razón, y dado que este esfuerzo se basa en la predicción de energía fotovoltaica, resulta de interés calcular la radiación solar la mayor cantidad de veces posible dentro del modelo, con el objetivo de simular con la mayor fidelidad y precisión esta propiedad. Sin embargo, debido al alto costo computacional que involucra calcular la radiación solar, llevar a cabo esta idea impacta directamente en la ejecución del modelo WRF, aumentando considerablemente el tiempo de ejecución total de la herramienta.

Una forma de disminuir el impacto del cálculo de esta propiedad es utilizar diferentes modelos de simulación para la radiación de forma de mantener la mejor precisión posible en los resultados pero buscando reducir al máximo el tiempo de ejecución. El modelo WRF permite, como se mencionó antes, incorporar el uso de distintos módulos de radiación tanto para la radiación LW como para la SW, gracias al diseño del driver de radiación. Las estrategias disponibles para el cómputo de la radiación (extraídas de [55]) para la versión 3.5.1 de WRF (la utilizada en los modelos empleados en la FING) son las siguientes:

- **Radiación de onda larga (LW):**
  - Modelo RRTM (Rapid Radiative Transfer Model).
  - Modelo Improved RRTM (RRTMG).
  - Modelo GFDL (Geophysical Fluid Dynamics Laboratory).
  - Modelo CAM (NCAR Community Atmospheric Model).
  - Modelo New Goddard.
  - Modelo Held-Suarez.
- **Radiación de onda corta (SW):**
  - Modelo Dudhia.
  - Modelo Goddard.
  - Modelo New Goddard.
  - Modelo RRTMG.
  - Modelo GFDL.
  - Modelo CAM.

Otra posibilidad que brinda el WRF es ajustar el período de ejecución del módulo de radiación. Esto permite al usuario buscar un compromiso entre el tiempo de ejecución y la precisión del modelo, aunque esta tarea no resulta sencilla, ya que encontrar la frecuencia de ejecución depende de la resolución de la grilla y la forma en la que se calculan otras propiedades de la atmósfera. Para cumplir con esta tarea, el WRF calcula la radiación solar, tanto de onda larga como de onda corta, en un paso de tiempo y utiliza este valor para un conjunto de pasos de simulación posteriores (m en la Figura 3.5). Esta cantidad de pasos debe ser fijada por el usuario al principio de la ejecución del modelo y está expresada como un intervalo en minutos de simulación. Si bien esta

estrategia permite aumentar la precisión de los resultados de la radiación [56], el aumento en la frecuencia de invocación de dicho módulo hace crecer el tiempo de ejecución del modelo. Esto se debe a que, como se puede ver en la Figura 3.5, el cálculo de la radiación se hace de forma sincrónica al resto de las propiedades atmosféricas computadas, por lo que aumentar la frecuencia hace que el modelo se detenga más veces a refrescar (computar) los datos de la radiación.

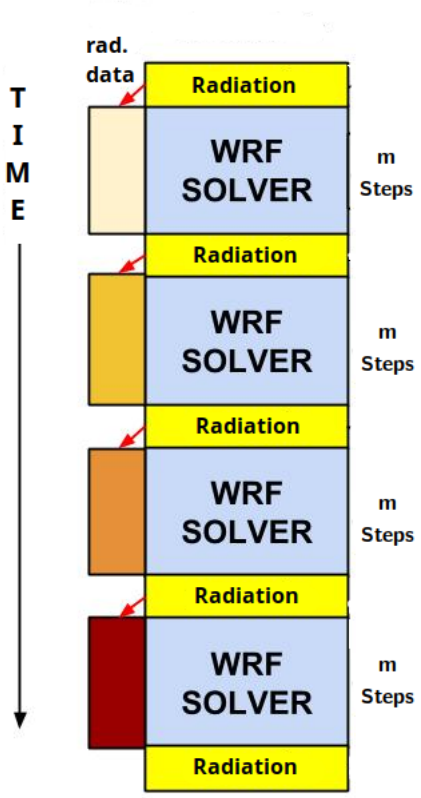


Figura 3.5: Ejecución del modelo WRF, discriminando entre el cómputo de radiación solar y el resto del modelo.

A partir de esta situación, resulta atractivo estudiar experimentalmente el impacto que el cálculo de la radiación causa en el modelo WRF, utilizando una frecuencia que permita obtener resultados de la radiación solar con la mayor precisión posible.

Antes de finalizar esta sección, es de interés mencionar que la precisión del cálculo de la radiación se ve fuertemente afectada por el módulo descripto, pero no es la única fuente de incertidumbre. Por ejemplo, el cálculo de las nubes o los contenidos de aerosoles en la atmosfera en la zona a estudiar pueden afectar

la precisión de los valores de radiación, pero el estudio del cómputo de estas y otras propiedades se consideran fuera del alcance de esta tesis.

3.3.3. Evaluación de la herramienta

En esta sección se evalúa el efecto que tiene variar la frecuencia de cómputo de la radiación en el desempeño total del modelo WRF. Además, se estudia el tiempo de ejecución y la escalabilidad cuando se varía la cantidad de hilos de cómputo, tanto del modelo WRF en su totalidad como del módulo de radiación solar en particular.

Ambiente de pruebas

Para realizar este experimento se confeccionó un caso real que involucra doce potenciales plantas solares sobre el territorio uruguayo. Esta instancia usa una grilla de 1500×1830 Km en el plano horizontal (como se puede ver en la Figura 3.6), discretizada en 50×61 puntos y 30 puntos en el plano vertical, y con un paso de simulación del modelo (tiempo que transcurre por cada paso discreto de simulación) de tres minutos. Además, este caso de prueba simula tres días, desde el 15 de mayo al 17 de mayo de 2016, y utiliza los módulos `Dudhia` y `RRTM` para la radiación de onda corta y onda larga, respectivamente. Se emplean estos módulos para el cómputo de la radiación contemplando un buen balance entre tiempo de cómputo y precisión de los resultados.

Para esta simulación se utilizó la versión 3.5.1 del modelo WRF, compilado usando el compilador de Fortran de *Intel*, *ifort*, en su versión 17.0.4 con la opción de *openMP* activada.

La infraestructura utilizada en esta evaluación consiste en un servidor con cuarto procesadores AMD Opteron 6128, con 12 cores a 2.0 Ghz cada uno y una caché L3 de 12 MB. Adicionalmente, la plataforma cuenta con 16 GB de RAM y el sistema operativo CentOS Linux 6.2.

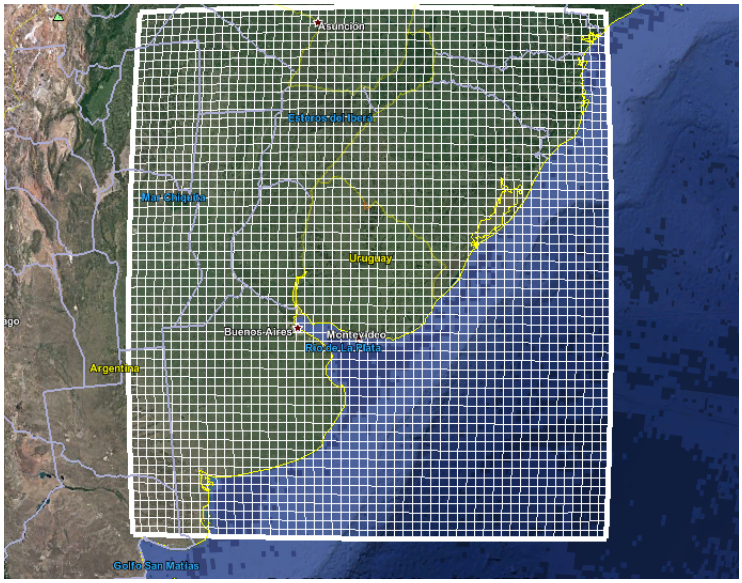


Figura 3.6: Grilla utilizada como caso de estudio.

Resultados experimentales

Para comenzar con la evaluación experimental de la herramienta WRF, se estudió el impacto en el tiempo de ejecución del modelo cuando se varía la cantidad de hilos, midiendo también el tiempo que consume el cálculo de la radiación. Adicionalmente, se computó la *escalabilidad* y el *speed-up* para cada conficuración. Para este experimento, se mantuvo para todos los casos la frecuencia del cómputo de la radiación en seis minutos de simulación, o sea, se calcula una radiación cada dos pasos del resto del modelo. Es imporante mencionar que los tiempos de ejecución presentados en este estudio son el promedio de 10 ejecuciones independientes, siendo cada simulación ejecutanda por separado y en un ambiente de uso exclusivo. Considerando esto, y teniendo en cuenta que el comportamiento del WRF es determinista, se optó por no agregar ningún otro estudio númerico más que el mencionado.

Como se puede ver en la Tabla 3.1, el WRF obtiene mejoras en los tiempos de ejecución hasta los 12 hilos, donde se logra una aceleración de 3,97 veces con respecto a la versión secuencial. Sin embargo, para el caso de 24 hilos, el modelo comienza a aumentar su tiempo de ejecución. Además, si bien hasta 12 hilos el tiempo de ejecución del modelo disminuye, ya para el caso de seis hilos la escalabilidad es de solo 0,53, lo cual muestra que la herramienta

no está explotando correctamente las unidades de cómputo disponibles. Este comportamiento puede deberse a que el tamaño del problema no es lo suficientemente grande para explotar el paralelismo de datos implementado en el WRF.

Tabla 3.1: Tiempo de ejecución (en segundos) del modelo WRF en su versión original variando la cantidad de hilos.

# hilos	*Tiempo Radiación*	*Tiempo Total*	*Eficiencia*	*Speedup*
1	573,43	1548,73	–	-
4	158,89	567,83	0,68	2,73×
6	111,18	490,69	0,53	3,16×
8	82,88	425,07	0,46	3,64×
12	63,07	390,49	0,33	3,97×
24	42,89	395,77	0,16	3,91×

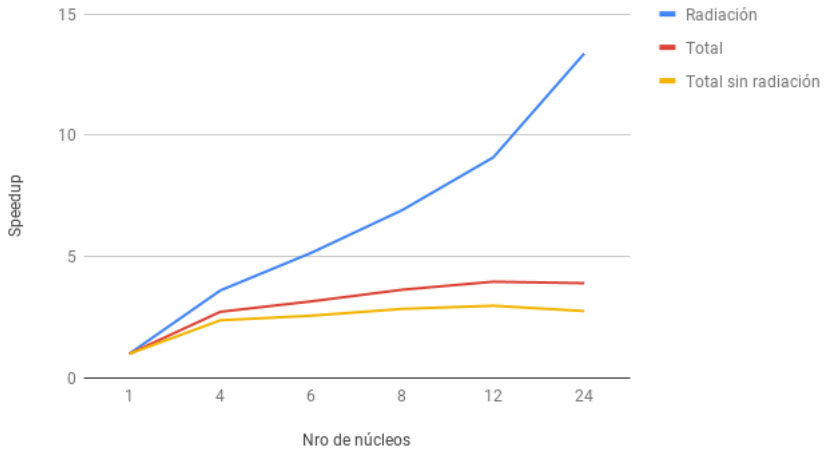


Figura 3.7: Grafica de speedups para el tiempo total de ejecución del modelo, el tiempo de ejecución de la radiación y el tiempo del resto de los cálculos realizados por el modelo.

Si se realiza un estudio de speedup más detallado sobre los datos obtenidos (ver Figura 3.7), se puede ver claramente que el módulo de la radiación solar obtiene mayores beneficios al utilizar más unidades de cómputo que el resto del modelo, debido a que el modelo sin la radiación presenta mejoras claramente más modestas, lo que limita la ejecución del modelo en su totalidad.

Adicionalmente, es importante notar que la radiación solar seleccionada en este caso escala de mejor forma que el modelo en general, ya que pasa de representar alrededor de 1/3 del tiempo de ejecución total del modelo en la versión secuencial a solamente el 16 % del mismo para la configuración de 12 hilos.

Una vez obtenida la mejor configuración de hilos para este caso de prueba, se procede a estudiar el impacto que tiene la frecuencia del cómputo de la radiación solar sobre el tiempo total de ejecución del modelo de esta variante. Los resultados de evaluar diferentes frecuencias para esta instancia del WRF se resumen en la Tabla 3.2

Como se puede ver en los valores presentados, aumentar la frecuencia de cálculo de la radiación hace que crezca el tiempo de ejecución del modelo. Específicamente, cuando la radiación se calcula cada 30 minutos de simulación, ésta consume menos del 4 % del tiempo de ejecución total. Sin embargo, a medida que la frecuencia crece, el porcentaje aumenta hasta un 28 % del total cuando se computa cada tres minutos de simulación, o sea, cuando se realiza para todos los pasos de simulación del modelo.

Tabla 3.2: Tiempo de ejecución (en segundos) del modelo WRF, discriminando el tiempo de ejecución de la radiación solar y el porcentaje que ésta consume, al cambiar la frecuencia del cálculo de la misma.

Paso radiación	***Tiempo radiación***	***Tiempo total***	***% radiación***
3	127,43	458,82	27,77
6	63,07	390,49	16,11
12	31,64	362,23	8,73
30	12,68	339,80	3,73

Considerando estos resultados, es claro que el enfoque disponible en el modelo WRF en cuanto a paralelismo no obtiene los resultados deseados para el escenario considerado, perjudicando el desempeño de la herramienta de predicción de la generación de energía fotovoltaica desarrollada. Además, se puede notar que la técnica de paralelismo disponible en el modelo subutiliza los recursos de cómputo presentes en la plataforma de hardware para el experimento realizado. Teniendo en cuenta que el objetivo del presente trabajo es, por un lado, mejorar el desempeño de la herramienta de predicción de generación de energía fotovoltaica y, por otro, que los cómputos se adapten mejor a las plata-

formas masivamente paralelas disponibles actualmente, es necesario entonces introducir una mejora a la implementación original del modelo. Por esta razón, resulta interesante relevar qué tipo de mejoras han sido propuestas por otros autores sobre el WRF y analizar si alguna de ellas se adapta a las características particulares de las necesidades planteadas. El relevamiento está centrado principalmente en el estudio de nuevos enfoques de paralelismo sobre el WRF así como las posibilidades de utilizar hardware de cómputo adicional.

3.4. Estudio del estado del arte

En los últimos años surgieron diversos esfuerzos para mejorar el desempeño computacional del modelo WRF. Por un lado, en trabajos como [57] y [58] se estudian las técnicas de paralelismo actualmente presentes en el WRF, evaluando en particular el desempeño de la versión híbrida basada en el uso de MPI+OpenMP y la relación de esta propiedad con la descomposición de los dominios de simulación en `tiles`. En ambos trabajos se concluye que, para dominios de suficiente tamaño, el enfoque híbrido obtiene mejores tiempos frente a la versión que sólo utiliza MPI. Por otro lado, una cantidad considerable de estudios se centran en incorporar el uso de hardware masivamente paralelo, como son las GPUs y los procesadores Xeon-Phi, al procesamiento de ciertos módulos o rutinas del WRF.

En [59] se resumen algunos trabajos que aplican GPU al modelo WRF. En [5], Michalakes y Vachharajani estudian la aceleración en GPU del módulo Single Moment 5 Cloud Microphysics (WSM5) utilizando CUDA C para portar esta rutina a GPU. En dicho trabajo se subdividió la grilla de simulación de forma de explotar la independencia de datos presente entre las subgrillas, asignando un hilo de ejecución a cada una. En este caso se obtiene una mejora de 17× contra la versión con un sólo hilo en CPU, incluyendo los tiempos de transferencia, proyectándose una aceleración de 1,25× para el modelo WRF en su totalidad. En Wang et al. [60] se porta el módulo Double Moment-5 (WDM5) a GPU siguiendo técnicas similares a las usadas por Michalakes y Vachharajani pero sumándole técnicas de transferencia asincrónica de datos a la tarjeta. Este esfuerzo logra alcanzar mejoras de 206× comparado con la versión de un hilo en CPU sin contar los tiempos de transferencia y una mejora de 147× si se consideran estos tiempos. En el caso de Vanderbauwhede

y Takemi [61], se porta una rutina relacionada a la propiedad de advección utilizando OpenCL y se estudian las mejoras obtenidas variando el fabricante de procesador, la GPU utilizada y el compilador. Esta propuesta alcanza mejoras de hasta 7× contra la versión secuencial en CPU sin contar los tiempos de transferencia y de 2× si se consideran los mismos.

Cuando se considera la plataforma Xeon-Phi, se puede abordar su utilización de distintas formas, dependiendo del modo que se utilice (ver Sección 2.6). En el trabajo de Saini et al. [62] se estudia el desempeño del WRF ejecutado de forma nativa en el coprocesador y utilizando una versión simétrica. Como se puede observar en este trabajo, la implementación nativa del WRF presenta una fuerte desventaja frente a su contraparte en CPU (siendo la versión en CPU 2,31× mejor que la ejecutada en el Xeon-Phi) pero en el caso del modelo de ejecución simétrica se alcanzan mejoras de hasta 33 % utilizando dos procesadores Xeon-Phi y la CPU. En [63] se presenta el portado y optimización del módulo Purdue-Lin a procesadores tipo Xeon-Phi, utilizando el método de *offload* para ejecutarlo en el coprocesador. En particular, se reordena el código de forma de aprovechar de mejor forma la localidad de datos y se procesa cada columna en forma paralela, debido a la independencia de datos entre ellas. Estas modificaciones permiten utilizar adecuadamente el procesamiento vectorial del coprocesador, alcanzando mejoras de la rutina de 2,4× frente a la versión sin mejoras ejecutada en esta unidad de cómputo. Complementariamente, en [64] se realiza un esfuerzo similar al trabajo anterior, pero sobre el módulo *Goddard* de microfísica, estudiando el impacto de las modificaciones introducidas al código tanto en CPU como en el Xeon-Phi. Concretamente, es posible lograr mejoras de 2,8 veces en la versión ejecutada en CPU y 4,8 veces en la versión ejecutada en Xeon-Phi. Finalmente, en [65] se porta el módulo Thompson de microfísica para nubes a Xeon-Phi, siguiendo las técnicas ya mencionadas, alineando además ciertos vectores críticos en la memoria. La combinación conjunta de estas técnicas permite alcanzar mejoras de 1,8 veces tanto en la versión de CPU como la del coprocesador.

Si bien los esfuerzos mencionados logran de alguna forma mejorar el desempeño de diferentes módulos del modelo WRF, ya sea buscando una configuración óptima para la ejecución en CPU o agregando hardware adicional, en todos ellos el enfoque de paralelismo está fuertemente ligado a la división de los datos, que en la mayoria de los casos no impacta fuertemente en el desempeño del modelo en general. Dada esta característica, resulta de interés en este

trabajo evaluar la introducción del paralelismo funcional para el módulo de radiación. Una vez alcanzado este nivel de paralelismo, es interesante además estudiar si esta separación lógica puede resultar beneficiosa a la hora de agregar hardware masivamente paralelo. En esta línea, parece provechoso estudiar si esta técnica ayuda a ocultar los tiempos asociados a la transferencia de datos, una característica determinante en los tiempos de ejecución en algunos de los trabajos resumidos en esta revisión.

Capítulo 4

Arquitectura asincrónica para el cálculo de la radiación en el WRF

En este capítulo se describe el diseño e implementación de una propuesta para modificar la estrategia de cálculo de la radiación dentro del modelo WRF. En particular, se presenta una arquitectura asincrónica entre el cálculo de la radiación y el resto del modelo WRF, alcanzando así una variante que explota el paralelismo funcional. Considerando la complejidad que requiere introducir dicha modificación al modelo, se busca además que la solución diseñada impacte de la menor manera posible en la implementación del WRF, por lo que se utiliza el mismo lenguaje de programación y herramientas de paralelismo ya presentes en la implementación del modelo estudiada. En la primera sección se describen las posibles opciones para realizar la arquitectura asincrónica, y una vez seleccionada una estrategia, se explica en detalle la misma. En la siguiente sección se realiza una evaluación de la correctitud del diseño de la arquitectura planteada, haciendo foco en el correcto funcionamiento del protocolo de comunicación introducido entre el módulo de la radiación y el resto del modelo. Luego, se describen los detalles que surgen a la hora de implementar el diseño propuesto, discutiendo distintas variantes. Finalmente, se toman las implementaciones descriptas y se evalúan experimentalmente, considerando tanto su tiempo de ejecución como su precisión numérica.

4.1. Diseño de una nueva arquitectura paralela para el WRF

Debido a los elevados tiempos de ejecución requeridos por la herramienta de predicción de energía solar fotovoltaica, y en particular por el modelo WRF, el principal objetivo de este trabajo es disminuir los tiempos de ejecución del módulo de radiación manteniendo la precisión numérica de los cálculos realizados para esta propiedad. Visto de otra forma, la idea es incrementar la frecuencia con la que se calcula la radiación, aumentando así la precisión numérica para este fenómeno, sin afectar drásticamente el tiempo de ejecución del modelo. Como se mencionó previamente (ver Sección 3.3.2), el tiempo de ejecución del WRF se encuentra linealmente relacionado con la frecuencia de cómputo de la radiación, debido al sincronismo presente entre estas dos secciones de cómputo. Por esta razón, esta propuesta se basa fuertemente en diseñar una nueva arquitectura que permite desacoplar el cómputo del módulo de la radiación del resto de los cálculos del modelo, explotando así un modelo de paralelismo funcional.

Actualmente, la relación entre el modelo global (como se denominará de ahora en adelante al modelo WRF sin el cálculo de la radiación) y el módulo de radiación se basa en que, cuando el modelo global necesita actualizar los datos de la radiación, invoca al módulo de radiación, espera que se ejecute y posteriormente obtiene los datos deseados. La arquitectura propuesta se caracteriza por eliminar la espera de los resultados del módulo de la radiación en el modelo global, desacoplando la llamada a éste y utilizando un patrón de paralelismo de tipo *pipeline*. Para poder llevar a cabo este enfoque, se separa el proceso de comunicación entre el modelo global y la radiación en dos etapas, una que se encarga de enviar los datos necesarios al módulo de la radiación para que pueda computar nuevos valores y otra etapa que obtiene dichos valores, una vez que el módulo de la radiación finalizó sus cálculos. Como complemento a esta estrategia, se crean en el módulo de radiación otras dos rutinas de comunicación, una de recepción de datos de entrada y otra de envío de los datos actualizados de la radiación. El comportamiento provocado por esta arquitectura se resume en la Figura 4.1.

Dado este mecanismo, surgen tres posibles alternativas al momento de administrar la comunicación. Por un lado, se tiene un enfoque basado en usar una frecuencia preestablecida para el envío de datos de la radiación y otra

para la recepción de los resultados. Concretamente, se configura el protocolo de comunicación para que se envíen los datos al módulo de la radiación y se adquieran los datos de éste cada cierta cantidad de pasos de simulación, esperando en caso de no poder cumplirse alguna de estas tareas en la frecuencia seleccionada.

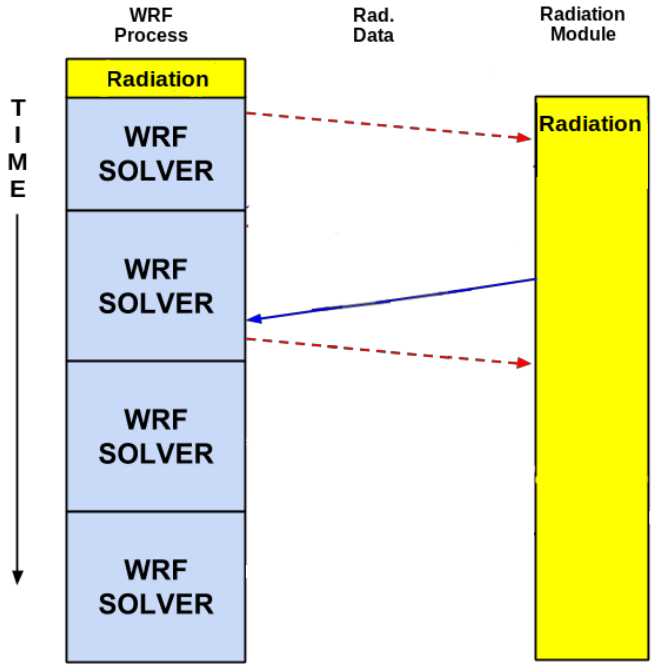


Figura 4.1: Interacción, en la propuesta, entre el modelo global y el módulo de la radiación.

Esta alternativa agrega la necesidad de obtener de forma experimental los valores de frecuencia que logran el mejor desempeño de la simulación para la plataforma particular sobre la que se está ejecutando, de forma de evitar combinaciones que introduzcan posibles tiempos ociosos.

Por otro lado, se puede utilizar una política *best-effort*, que consiste en realizar cada etapa cuando los datos estén disponibles, independientemente del paso de simulación sobre el que se esté calculando. En otras palabras, el modelo manda los datos a la radiación en cualquier paso de simulación en que esté ocioso y a su vez recibe los datos del módulo cada vez que haya nuevos resultados disponibles. Uno de los inconvenientes que presenta esta técnica es que, si bien minimiza el impacto del cálculo de la radiación en el tiempo del modelo WRF, no asegura ni cuándo se va a actualizar el dato de la radiación

ni cuantas veces se va a calcular la radiación en un período de tiempo, lo que puede comprometer la precisión de los resultados obtenidos.

Finalmente, otra posible implementación consiste en realizar una opción híbrida de las dos alternativas descritas anteriormente, donde se sigue una estrategia *best-effort* pero con alguna restricción sobre la cantidad de radiaciones calculadas en un período.

Debido a que la alternativa *best-effort* ya fue estudiada en un trabajo previo [66] y considerando las restricciones relativas a la precisión numérica de los datos de salida para la radiación, en esta propuesta se seguirá una estrategia que utiliza únicamente una frecuencia fija. Además, una vez que las primeras dos alternativas hayan sido estudiadas, el pasaje a la tercera opción híbrida es directa.

Para explicar con mayor profundidad la aplicación del paradigma de paralelismo pipeline al problema abordado se analiza una instancia particular de la nueva arquitectura, la cual consiste en calcular de forma concurrente la radiación solar con dos pasos de simulación del modelo global. Es importante remarcar que, si bien a efectos prácticos se discute esta implementación específica, la idea planteada es genérica y puede implementarse con distintos valores de frecuencia.

La nueva propuesta consiste en ejecutar de forma solapada el cálculo de la radiación y el modelo global, utilizando diferentes unidades de cómputo, o líneas del pipeline, para cada una de las partes. En particular, un conjunto de unidades de cálculo se destina al modelo global mientras que otro conjunto (disjunto al primero, al menos desde el punto de vista lógico) es utilizado para la radiación. Luego de una ejecución secuencial inicial de ambas partes, necesaria para inicializar la simulación del modelo, el comportamiento de la nueva arquitectura, dada la configuración seleccionada, se desarrolla de la siguiente forma:

1. Si el módulo de la radiación está ocioso y el paso de simulación en el modelo global es múltiplo de la frecuencia seleccionada, el modelo global transfiere los datos de entrada a la radiación.
2. En paralelo:
   a) El modelo global computa el paso k y $k+1$, utilizando los datos de la última radiación calculada.
   b) El módulo de radiación calcula la nueva radiación, utilizando los datos calculados por el modelo global en el paso $k-1$.
3. Si el paso de simulación corresponde a la frecuencia establecida, el modelo global lee los nuevos valores de radiación.

Es importante destacar que para el paso 1 y 3, si el paso de simulación es el decidido para realizar el envío o recepción de datos con el módulo de radiación pero este no se encuentra disponible, el modelo global deberá esperar para sincronizar los datos con su contraparte.

Dado este modelo de concurrencia y suponiendo a priori que tanto la radiación como el modelo global no comparten estructuras de datos dentro de la arquitectura, surge la necesidad de contar con un método de comunicación que permita mantener consistentes los datos que habilitan la interacción entre ellos además de controlar el acceso a dichos datos. Por esta razón, se diseñaron dos algoritmos de comunicación-sincronización que habilitan a ambas partes a comunicarse de forma adecuada. En el Algoritmo 1 se muestra el comportamiento del módulo de radiación en este proceso de comunicación-sincronización. En detalle, al comienzo del bucle el módulo espera a que haya información disponible para calcular la radiación, la cual es cargada en un buffer por el modelo global. Una vez recibida la información, el módulo procede a calcular las variables de radiación para ese paso de simulación. Al finalizar los cálculos, la rutina copia las variables de radiación a un buffer y notifica al modelo global que esta información está lista para ser consumida. Este proceso de cómputo de la radiación se ejecuta tantas veces como sea necesario hasta que la simulación total de la atmósfera concluya. Es importante mencionar que ambos procesos de notificación, el que indica a la radiación que comience a ejecutar y el que avisa al modelo global que consuma los nuevos datos, se realizan a través de dos variables booleanas, llamadas *send_available* y *receive_available*, accesibles a ambas componentes.

En el Algoritmo 2 se presenta la otra parte del proceso de comunicación,

```
1 Procedure Radiation_processor()
2 |  while TRUE do
3 |  |  wait(send_available)
4 |  |  receive_data(data_to_process)
5 |  |  process_data(data_to_process, result)
6 |  |  send_result(result)
7 |  |  send_available = FALSE
8 |  |  receive_available = TRUE
9 |  end
```

Algoritmo 1: Procedimiento de comunicación en el módulo de radiación para la arquitectura propuesta.

es decir el comportamiento del modelo global. Como se puede observar, lo primero que realiza el modelo global es constatar si la radiación está computando o se encuentra esperando por nuevos datos a procesar. En caso de estar en espera, el modelo global envía la información necesaria al módulo de radiación y le notifica mediante la variable *send_available*. Luego de estos pasos, el modelo evalúa si existen datos nuevos de la radiación para actualizar el modelo observando la variable *receive_available* y, en caso de que ésta lo habilite, copia los datos actualizados por el módulo de radiación hacia el modelo WRF. Finalmente, de ocurrir una actualización de los datos de radiación, modifica la variable *receive_available*.

```
1  Initialization
2  |  send_available = FALSE
3  |  receive_available = FALSE
4  Procedure Radiation_handler(data_to_process, result)
5  |  if frequency_achieved then
6  |  |  wait(NOT send_available)
7  |  |  send_data(data_to_process)
8  |  |  send_available = TRUE
9  |  |  wait(receive_available)
10 |  |  result = receive_result()
11 |  |  receive_available = FALSE
12 |  end
```

Algoritmo 2: Procedimiento de comunicación en el modelo global en la arquitectura propuesta.

Es importante notar que, dado el modelo de comunicación y sincronización descrito anteriormente, el costo de computar una radiación para el modelo global disminuye considerablemente. Específicamente, se reduce este valor al

tiempo que el modelo global requiere para enviar los datos al módulo de radiación y el tiempo necesario para asimilar los nuevos datos obtenidos de éste, siempre que no exista tiempo de sincronización al momento de recibir estos nuevos datos. Además, este mecanismo de comunicación permite al modelo global calcular el resto de las propiedades de la atmósfera para los pasos k y $k+1$ mientras el módulo de radiación calcula de forma simultánea las variables de esta propiedad.

4.2. Correctitud del diseño

Como se menciona en la sección anterior, para el funcionamiento correcto de la comunicación entre el módulo de la radiación y el modelo global, es necesario utilizar dos buffers, los que se acceden a través de un algoritmo de sincronización. Dentro de la computación tradicional, esta estrategia de cómputo presenta un problema de concurrencia para el acceso a estos buffers, donde la lectura y/o escritura de los mismos debe pertenecer a una sección crítica, ya que estas tareas se deben realizar de forma secuencial entre todos los procesos involucrados.

Cuando se diseña un programa que debe ejecutar de forma concurrente pero incluye el acceso a una sección crítica, se deben tener presentes tres propiedades. Por un lado, es necesario asegurar que la sección crítica sea accedida por un sólo proceso o hilo de forma simultánea, lo que se conoce como mutuo-exclusión. Por otro lado, es importante verificar que el diseño de la aplicación no genera *deadlocks*, o sea, que no existe ningún orden de ejecución entre los procesos que resulta en que todos queden a la espera de acceder a la sección crítica, lo que genera un bloqueo total del programa. Finalmente, es importante considerar que todos los hilos de ejecución eventualmente accedan a la sección crítica cuando lo requieran, lo que se conoce como evitar la inanición de los procesos.

Dadas estas tres propiedades, resulta de interés encontrar un método que permita verificarlas sobre el modelo de comunicación diseñado, previo a su etapa de implementación. En [67] se plantea una metodología práctica para probar estas características, basada en realizar una máquina de estados. Concretamente, la idea es identificar claramente la sección crítica y los protocolos de sincronización utilizados, para crear la máquina de estados que contiene a todos los posibles órdenes de ejecución que enmarcan estas secciones de códi-

go. Una vez realizado este grafo, se puede evaluar la correctitud del algoritmo examinando la presencia o ausencia de ciertos estados o transiciones.

Siguiendo esta estrategia, lo primero a considerar es cómo se van a conformar los estados de dicha máquina. Considerando que lo que se desea estudiar con el diagrama es el flujo del programa, una de las características de los estados tiene que ser los punteros de instrucción tanto del modelo global como del módulo de radiación, los cuales indican que línea de código está ejecutando cada proceso. Adicionalmente a estos dos atributos, es necesario controlar los valores de las variables que involucran el proceso de control en la comunicación entre la radiación y el modelo global, las cuales son *send_avalaible*, *receive_avalaible* y *frequency_achieved*. Teniendo en cuenta estas variables, los estados de la máquina a construir tienen la siguiente forma:

$$(GM_i, R_i, EF, SA, RA).$$

Donde GM_i es la línea en la que se encuentra el cursor del programa sobre el modelo global y R_i es la línea en la que se encuentra el cursor del programa sobre el módulo de la radiación. Adicionalmente se utilizan las tres variables de control, EF, SA y RA que son los estados de las variables *frequency_achieved*, *send_avalaible()* y *receive_avalaible()*, respectivamente.

Considerando estas cinco características, la máquina a construir puede tener tantos estados como combinaciones de los valores de estos atributos. O sea, la máquina de estados puede tener $V(GM_i) \times V(R_i) \times V(EF) \times V(SA) \times V(RA)$, donde la función $V(a)$ indica el número de posibles valores que puede tomar la variable a. En este caso, la cantidad de estados es $10 \times 8 \times 2 \times 2 \times 2$, resultando en un total de 640 posibles combinaciones. Dada la gran cantidad de estados que resultan de este análisis, es necesario simplificar los Algoritmos 1 y 2 con el objetivo minimizar la máquina de estados y así poder abordar el estudio de la correctitud del algoritmo a través de esta técnica.

Por un lado, si se observa el condicional IF que involucra a la variable *frequency_achieved*, se puede notar que, de no cumplirse esta condición, no se accede al protocolo que controla la comunicación ni a la sección crítica. Adicionalmente, esta condición se cumple eventualmente, ya que en algún momento se está en la frecuencia seleccionada para realizar la comunicación, por lo que eventualmente se ejecutará el protocolo de control. Las dos características mencionadas anteriormente permiten, a efectos de este análisis, eliminar este condicional. Por otro lado, a efectos de disminuir la cantidad de líneas,

se puede eliminar el *while* presente en el código de la radiación y considerar únicamente el comportamiento de esta estructura de control, o sea, retornar a la primera línea de código luego de haber ejecutado la última.

Del estudio descrito previamente, se puede reescribir el código para esta sección, como se muestra a continuación:

```
1 wait(send_available)
2 receive_data(data_to_process)
3 process_data(data_to_process, result)
4 send_result(result)
5 send_available = FALSE
6 receive_available = TRUE
```

```
1 wait(NOT send_available)
2 send_data(data_to_process)
3 send_available = TRUE
4 wait(receive_available)
5 result = receive_result()
6 receive_available = FALSE
```

Una segunda simplificación se puede aplicar a estas porciones de código, donde las zonas críticas son eliminadas, ya que no es de interés estudiar qué es lo que sucede dentro de la zona crítica, sino evaluar el correcto acceso a ella por parte de ambas secciones. Considerando esto, el código final tiene la siguiente forma:

```
1 wait(send_available)
2 send_available = FALSE
3 receive_available = TRUE
```

```
1 wait(NOT send_available)
2 send_available = TRUE
3 wait(receive_available)
4 receive_available = FALSE
```

Considerando los cambios realizados en el código a estudiar, se modifican los estados de la máquina, los cuales quedan de la siguiente forma:

$$(GM_i, R_i, SA, RA).$$

Una vez determinado los fragmentos de código a estudiar y la representación que permite construir la máquina de estado, se procede a realizar la misma, como se presenta en la Figura 4.2.

La primera propiedad a verificar sobre este grafo es la de mutuoexclusión. Específicamente, verificar en la máquina que los estados $(2, 2, *, *)$ y $(3, 4, *, *)$ no se encuentran, lo que implicaría que ambos flujos del programas o bien accedieron simultáneamente a los datos a enviar a la radiación o a los nuevos datos de radiación, respectivamente. Como se puede observar, estos estados no están presentes, lo que significa que no hay posibilidad que ambas partes se encuentren accediendo a los arreglos de datos de forma simultánea.

La siguiente característica a revisar es la no existencia de *deadlocks*. Para verificar esta propiedad, es necesario ver si existe un camino en el grafo que permita eventualmente acceder a la sección crítica de ambas partes. En otras palabras, es necesario verificar que cuando una de las partes abandone su sección crítica, la máquina se encuentre en un estado que permita a otros procesos acceder a su sección en algún momento. Considerando que el grafo resultante de este análisis presenta claramente un ciclo en el flujo de ejecución, se puede afirmar entonces que no existe ningún camino posible por el cual el programa se bloqueé por completo.

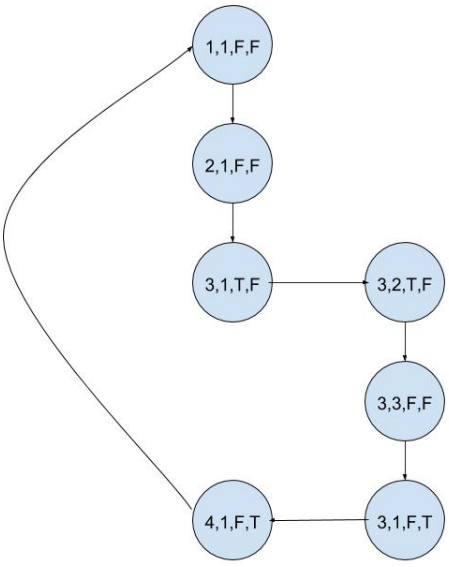


Figura 4.2: Máquina de estados que sintetiza el comportamiento del protocolo de comunicación-sincronizacion

Finalmente, la última propiedad a cumplir es la de no inanición. Específicamente, esta propiedad se muestra verificando que no existen *deadlocks* y que no existe otra porción de código, además de las secciones críticas estudiadas, que bloqueé el flujo de los procedimientos. Para este caso, se sabe que no existe otra rutina que impida el flujo de los procedimientos estudiados, por lo que se alcanzarán eventualmente ambas secciones críticas, y ya se demostró que no existen *deadlocks*, por lo que se puede afirmar que se cumple la característica de no inanición.

Una vez probadas estas tres propiedades, se puede afirmar que el mecanismo de comunicación propuesto para el intercambio de datos entre el módulo de radiación y el modelo global es correcto desde el punto de vista del acceso a los datos y no perjudicará la ejecución del modelo ni la integridad de sus datos. Esta conclusión permite proceder a la etapa de implementación del algoritmo propuesto, que se describe en la siguiente sección.

4.3. Detalles de implementación

En este apartado se discuten aspectos relacionados con la implementación de la nueva arquitectura asincrónica propuesta y verificada en las secciones anteriores. En particular, se ven características de la implementación del protocolo de comunicación y pequeñas variantes que surgen al considerar la validez y precisión de los datos. Para comenzar, se describen los aspectos que todas las implementaciones comparten, explicando posteriormente las diferencias entre las variantes en distintos apartados.

Una de las primeras decisiones a considerar es sobre qué tipo de ambiente debe ejecutar la nueva arquitectura. En este aspecto, se puede trabajar en un ambiente de memoria compartida, el cual involucra "una sola" computadora, o se puede utilizar un ambiente de tipo distribuido. Dadas estas dos opciones, existen para ambos paradigmas herramientas de desarrollo ampliamente utilizadas. Para el paradigma de memoria compartida se utiliza la API OpenMP o la biblioteca Pthreads [68], las cuales permiten utilizar hilos de ejecución dentro de CPUs en una computadora para dividir el cómputo entre distintos núcleos. Para la otra alternativa se puede utilizar el estándar MPI, que permite enviar mensajes entre distintas computadoras y así coordinar esfuerzo entre

ellas.

En este trabajo, y a efectos de desarrollar un prototipo de la arquitectura agilmente que permita probar sus cualidades, se implementa la solución siguiendo un enfoque centralizado, utilizando hilos a través tanto de la API OpenMP como la biblioteca Pthreads. Específicamente, al inicio de la simulación se crean dos hilos utilizando Pthreads, uno para el modelo global y otro para el módulo de radiación. Luego de ciertas primitivas de inicialización, cada sección crea cierta cantidad de hilos extra para procesar las propiedades de la atmósfera que cada uno computa, utilizando OpenMP. De esta forma, se genera una jerarquía de hilos, como se muestra en la Figura 4.3, capaz de asignar distintas cantidades de éstos tanto al modelo global como al módulo de radiación, y así explotar de manera eficiente el poder de cómputo de la CPU multicore.

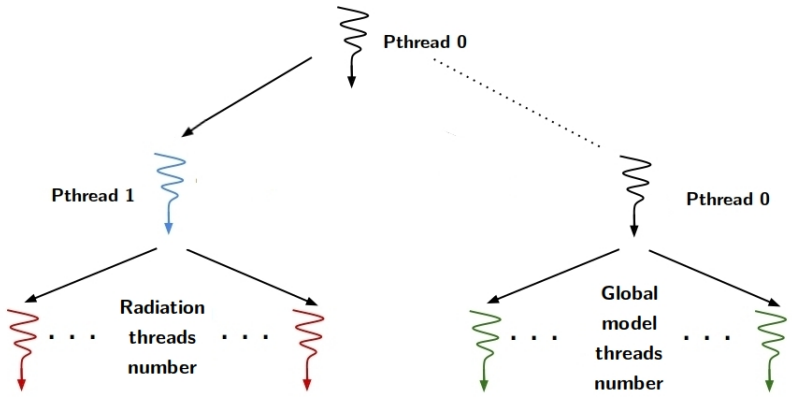


Figura 4.3: Jerarquía de hilos utilizada en la nueva arquitectura.

A partir de la decisión de utilizar un ambiente centralizado, otro aspecto a considerar es cómo se van a almacenar los buffers de recepción y envío de datos al igual que la ubicación de las variables de control que limitan el acceso a éstos. En particular, para esta implementación se utilizan dos estructuras de datos de tipo *structure* para representar los buffers de comunicación, una que contiene los datos a enviar al módulo de radiación para que calcule un nuevo paso de esta propiedad y otra que mantiene los datos a recibir por el modelo global para actualizar la radiación. Como se mencionó previamente, existen

diversos procedimientos para calcular tanto la radiación de onda larga como la de onda corta, lo que conlleva al uso de distintas variables relacionadas a propiedades atmosféricas, dependiendo la estrategia que se utilice. Como se porta el módulo de radiación por completo, o sea el driver que se encarga de realizar las llamadas a las distintas implementaciones del cálculo de radiación, es necesario considerar la gama completa de posibles variables que se comunican entre el modelo global y la radiación.

En la implementación original del WRF, el modelo contempla este abanico de variables utilizando punteros (*pointer*) para almacenar las variables o marcando las mismas como opcionales (*optional*). En la implementación asincrónica, para contemplar las posibles combinaciones de variables, se realiza un chequeo previo sobre éstas, tanto en carácter de entrada como de salida del módulo de radiación, y se almacena en un arreglo booleano cuales de ellas son utilizadas en esa simulación. Con esta estructura de chequeo, se limita la comunicación únicamente a las variables necesarias para el cálculo de la radiación para esa simulación, lo que disminuye no sólo el almacenamiento a utilizar sino también los tiempos de transferencia. Finalmente, para cumplir con el diseño de la arquitectura asincrónica verificado en la sección anterior, y dado que se tomó la decisión de utilizar un ambiente de memoria compartida, se usan dos variables booleanas, las cuales se encargan de controlar el acceso a los datos de comunicación entre ambas partes. Concretamente, se utiliza una variable booleana para avisar al módulo de radiación que tiene nuevos datos para calcular esta propiedad y otra variable para notificar al modelo global que existe un nuevo cómputo de la radiación para asimilar.

A partir de las opciones de diseño tomadas previamente, surgen dos interrogantes relacionadas al cómputo de la radiación. Por un lado, es importante considerar para que instante de tiempo de la simulación se debe calcular cada radiación y, por otro lado, si es necesario permitir al modelo incorporar cada uno de estos datos previo a enviar nuevas variables de entrada al módulo de radiación. Ambos planteos pueden afectar la precisión numérica de la radiación ya que puede existir dependencias entre los datos de la radiación y los del resto del modelo WRF. Un enfoque para mitigar esta diferencia numérica podría involucrar el estudio de cada variable de entrada y salida al modelo de la radiación, sin embargo, esta tarea tiene una dificultad muy alta, ya que la llamada al modelo involucra el uso de poco más de 300 variables, producto de que existen diversas implementaciones de este módulo. Considerando estas particu-

laridades, se implementan cuatro versiones de la arquitectura asincrónica con distintos enfoques. A continuación, se describen las características principales de cada una de estas variantes.

Variante directa $Asyn_{Str}$

En esta versión, se implementa directamente la arquitectura descrita, sin considerar ningún otro aspecto más que el asincronismo. Es decir, el módulo de radiación calcula esta propiedad para cierto paso de tiempo n y el modelo adquiere dichos datos para el paso n+2. Dado que el diseño propuesto no realiza los cálculos de la radiación en el mismo orden que la arquitectura sincrónica original del WRF, se espera que se introduzca cierta desviación en la precisión de los datos simulados para la radiación solar. Esta desviación puede suceder tanto al momento de comparar las salidas de este módulo contra las obtenidas utilizando la versión original del WRF, así como para las otras propiedades físicas que se alimentan de estos valores.

Variante modificando el paso temporal de la radiación $Asyn_{AdjT}$

Con el objetivo de mitigar las posibles diferencias numéricas entre el modelo WRF original y la novel propuesta, se considera una segunda implementación, donde se ajusta el paso temporal de la simulación para el que se calcula la radiación. La idea es ajustar las variables que dependan del tiempo, en el cálculo de la radiación, para que se computen considerando un paso futuro y no para el paso para el que se suministraron los datos de entrada. En particular, se busca ajustar estas variables para que sean computadas para el paso donde van a ser (presumiblemente) incorporadas por el modelo, similar al comportamiento que el modelo realiza cuando selecciona el paso en el que se computa la radiación. Para lograr este objetivo, se ajusta la variable de entrada *xtime* (variable que indica la cantidad de minutos simulados desde que se inicio la ejecución del modelo) al tiempo donde la radiación va a ser incorporada por el modelo global.

Variante con asimilación del paso de la radiación $Asyn_{UpdV}$

Otro problema que surge de la implementación directa de la arquitectura asincrónica es el impacto que puede tener el uso de las variables calculadas en el módulo de radiación sobre el resto del modelo. En concreto, la interrogante

es qué cambio puede generar una desviación de los datos de la radiación sobre el resto de las variables del modelo, y en particular si este cambio puede propagarse hasta las propias variables de entrada del módulo de la radiación, afectando la precisión de los cálculos a futuro. Teniendo en cuenta que en la variante $Asyn_{Str}$ se mandan los datos para calcular la siguiente radiación previo a asimilar los datos de la última radiación calculada, en esta versión se actualizan primero los datos de la radiación y, luego de un paso de simulación para asimilar esta característica, se envían nuevos datos para calcular la siguiente radiación. Es importante mencionar que, si bien esta variante modifica el comportamiento del algoritmo asincrónico estudiado en la Sección 4.2, el cambio introducido no afecta al algoritmo de forma crítica, por lo que los resultados obtenidos en cuanto a correctitud desde el punto de vista de la concurrencia del mismo siguen siendo válidos.

Variante combinada $Asyn_{AdjT+UpdV}$

En esta versión se utiliza una combinación de las implementaciones antes descriptas, $Asyn_{AdjT}$ y $Asyn_{UpdV}$. El objetivo de esta variante es analizar si esta combinación resulta en una mejora significativa de la precisión numérica, incluso mejor que la ofrecida por las otras versiones cuando se consideran por separado.

4.4. Evaluación experimental

En esta sección se comparan las distintas variantes de implementación de la arquitectura asincrónica propuesta contra la versión original sincrónica del modelo WRF. De esta forma, se espera medir y analizar los posibles beneficios que se pueden obtener al modificar el paradigma de cómputo en el modelo. En una primera etapa de evaluación, se comparan todas las variantes implementadas contra el modelo original WRF desde un punto de vista numérico, analizando la posible desviación numérica que introduce cada una de las propuestas. En una segunda etapa, se toman las variantes con un desempeño numérico aceptable (comparables en calidad con la variante original) y se las compara nuevamente con el modelo original, pero en este caso considerando su desempeño computacional.

4.4.1. Ambiente de evaluación

Para evaluar experimentalmente las implementaciones propuestas, se crearon dos casos de prueba reales sobre 12 plantas solares en el territorio uruguayo. Ambos casos utilizan una grilla con $50 \times 61 \times 30$ puntos de discretización, un paso de simulación del modelo de tres minutos y una frecuencia de radiación de seis minutos. El primer caso, utilizado para validar la nueva arquitectura desde una perspectiva numérica, ejecuta un mes completo de simulación que abarca del 15 de mayo al 15 de junio de 2016. La segunda prueba es una simulación de tres días, análoga a la descrita en la Sección 3.3.3, que se utiliza principalmente para realizar el análisis de las implementaciones considerando el desempeño computacional de las diferentes variantes.

En cuanto al proceso de compilación y la plataforma sobre la cual se llevaron a cabo estas pruebas, ambas características son iguales a las descritas para evaluar el modelo WRF original (ver Sección 3.3.3).

4.4.2. Validación numérica

Considerando que la arquitectura propuesta para el cálculo de la radiación modifica el patrón original de cómputo de esta propiedad, resulta de interés estudiar el impacto que esta modificación genera sobre los valores de salida de la herramienta de simulación. Para evaluar correctamente este aspecto es importante definir, previo a realizar las ejecuciones, al menos dos cosas:

- cuáles serán los valores contra los que compararse (conocidos como línea base).
- cómo se va a medir la diferencia o distancia con dichos valores de referencia.

Por un lado, se consideran los valores obtenidos del flujo de radiación solar hacia abajo en superficie (del inglés downward short wave flux at ground surface - almacenado en la variable SWDOWN-) por el modelo WRF original, calculando la radiación en cada paso de simulación, como la línea base contra la cuál comparar los resultados arrojados por los experimentos. Es importante mencionar que estos valores de salida del modelo WRF, y en general de toda la herramienta de predicción de la generación de la energía solar, fueron validados previamente contra datos medidos sobre un período de un año [3, 4].

Por otro lado, se selecciona como medida a utilizar para este experimento el Error Porcentual Absoluto Medio (-MAPE- del inglés Mean Absolute Percentage Error). Se emplea esta medida teniendo en cuenta que los pronósticos están orientados al uso en la gestión del sistema eléctrico y es la medida que se usa habitualmente por los técnicos del sistema. Concretamente, se utiliza esta medida de la siguiente forma:

$$\frac{100}{n}\sum_{t=1}^{n}\left|\frac{BS_t - Rad_t}{BS_t}\right|, \tag{4.1}$$

donde n es la cantidad de muestras almacenadas en el tiempo de simulación para la variable SWDOWN, BS_t es un valor de la línea base en el instante t y Rad_t es un valor de la versión a evaluar en el paso de tiempo t.

A partir de estas decisiones, se procede a realizar el primer experimento para evaluar el desempeño numérico de la novel arquitectura. Para ello, se ejecutan las cuatro implementaciones descriptas en la sección anterior, evaluando el promedio del MAPE mensual de las 12 plantas seleccionados. De manera adicional a estas mediciones de error, se ejecutan y cotejan con la línea base otras tres ejecuciones del modelo WRF original variando la frecuencia del paso de la radiación. Estas tres configuraciones, WRF_{rad6}, WRF_{rad12} y WRF_{rad30}, tienen una frecuencia de cálculo de la radiación de 6, 12 y 30 minutos, respectivamente. El propósito de estas pruebas es evaluar los errores de las variantes asincrónicas no sólo desde un punto de vista porcentual, sino además si ese porcentaje es aceptable en términos comparativos con pequeños cambios en el modelo original. Para este estudio se evalúa una ejecución para cada configuración dado que el WRF es determinista y el ambiente está dedicado a las ejecuciones de este experimento, por lo que el resultado no varía más allá de diferencias introducidas por el uso de aritmética de punto flotante.

Tabla 4.1: MAPE para las diferentes configuraciones del WRF original y para las distintas variantes asincrónicas implementadas.

Versión	***MAPE(%)***
WRF_{rad6}	6,6
WRF_{rad12}	23,1
WRF_{rad30}	31,5
$Asyn_{Str}$	47,2
$Asyn_{AdjT}$	7,4
$Asyn_{UpdV}$	22,7
$Asyn_{AdjT+UpdV}$	7,0

En la Tabla 4.1 se muestran los errores porcentuales obtenidos de comparar las distintas implementaciones propuestas y las variantes de la versión original del WRF contra la línea base. Como se puede observar, las versiones $Asyn_{Str}$ y $Asyn_{UpdV}$ presentan porcentajes elevados, teniendo valores de MAPE de 45 % y 20 %, respectivamente, mientras que las versiones $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ presentan un error menor al 10 %. Además, en el caso de las de las variantes $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ los errores arrojados son similares a los obtenidos para WRF_{rad6} y se encuentra considerablemente por debajo de los alcanzados por el caso WRF_{rad12}, lo cual representa una cota aceptable.

Dados los resultados obtenidos, se realiza un segundo estudio sobre las variantes $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ con el objetivo de estudiar si aumentando la granularidad temporal del análisis del error se conservan las cotas vistas, o por el contrario existe una gran variación entre los valores puntuales. Para esta nueva evaluación se consideró cada una de las 12 plantas solares por separado y se calculó el MAPE diario para todo el mes de simulación.

En la Figura 4.4 se muestra de forma gráfica el error diario para cada una de las 12 plantas solares, donde WRF_{rad6}, WRF_{rad12}, $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ están representados por los colores azul, rojo, amarillo y verde, respectivamente. Observando este conjunto de gráficas, se puede corroborar que se mantienen las cotas observadas en el estudio anterior, ya que para ningún día los valores de $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ son distantes a los medidos en WRF_{rad6} y se mantiene siempre por debajo de los de WRF_{rad12}.

Dados los resultados obtenidos en estas dos evaluaciones, se concluye que tanto $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ presentan resultados numéricos comparables frente a los resultados del WRF original. Por esta razón, se evaluará el desempeño computacional solamente de estas dos variantes en futuros análisis.

4.4.3. Evaluación del desempeño

Una vez estudiada la calidad numérica de las versiones implementadas, el siguiente paso es evaluar el desempeño computacional de las versiones que presentaron una mejor precisión. Visto que la variante del WRF original WRF_{rad6} se asemeja mucho en precisión numérica con las versiones $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$ de la arquitectura asincrónica implementada, se utiliza esta configuración del modelo original como línea base para estudiar el desempeño computacional de la nueva arquitectura. Teniendo en cuenta esta selección, se

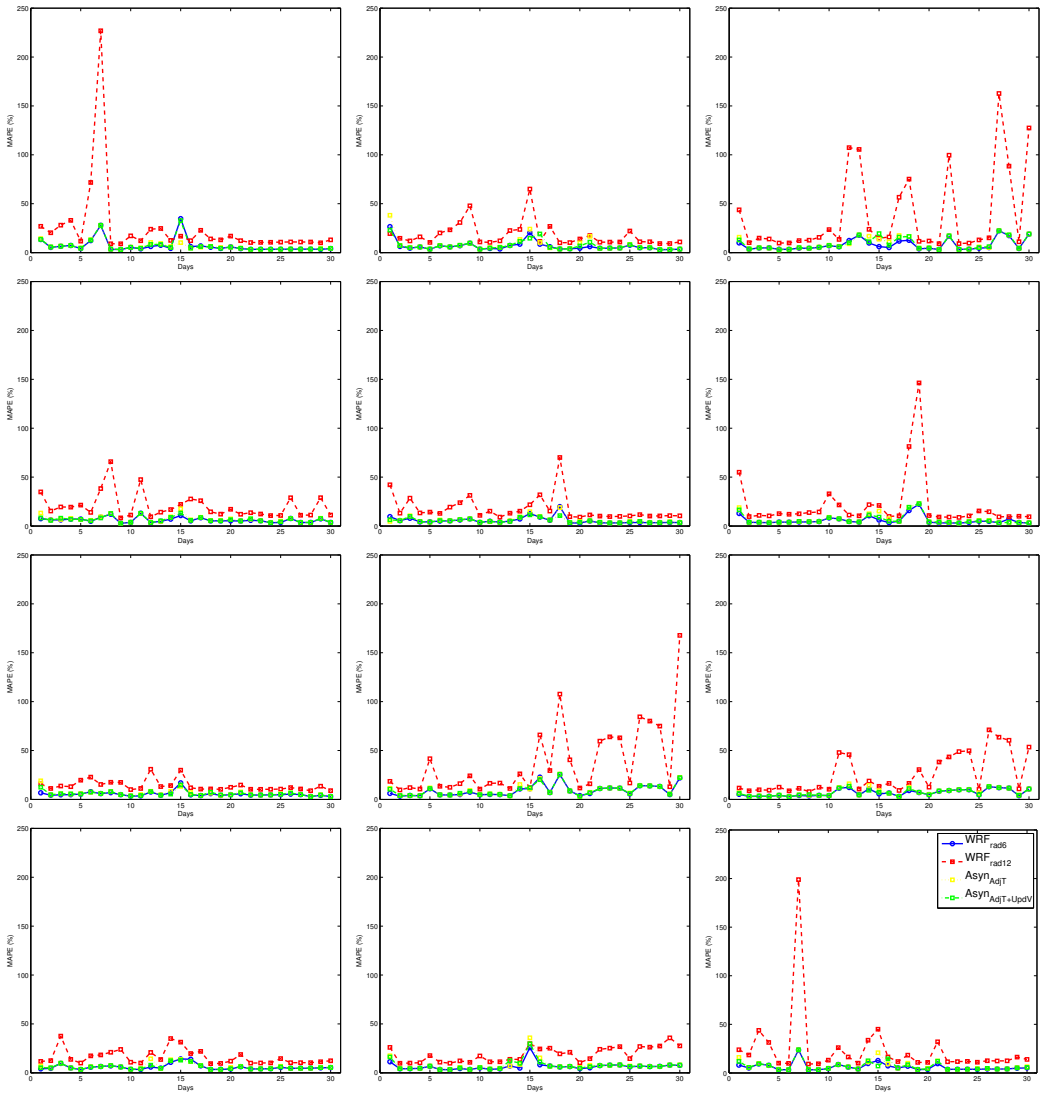


Figura 4.4: Análisis gráfico del MAPE diario para las versiones WRF_{rad6}, WRF_{rad12}, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ sobre los puntos de la grilla de interés.

toma además la mejor configuración de hilos para esta línea base, que, como se vio en la Sección 3.3.3, es la que utiliza 12 hilos, resultando en un tiempo de ejecución total de 390,49 segundos. Previo a realizar la evaluación experimental, es importante mencionar que existe un límite teórico alcanzable para cualquiera de las soluciones planteadas, el cual se desprende de la Ley de Amdahl. En esta evaluación experimental en particular, la máxima disminución teórica alcanzable por cualquiera de las variantes asincrónicas es de 63,07 segundos (debido a que es el tiempo de ejecución de la radiación para WRF_{rad6} utilizando 12 hilos), lo que equivale a una reducción total del tiempo del modelo en un 16,1 %.

Gracias a las decisiones de diseño tomadas (ver Sección 4.3), el modelo asincrónico implementado puede ser ejecutado con una amplia variedad de configuraciones de hilos, de forma de adaptarse mejor a la arquitectura de hardware que se vaya a utilizar. Para comenzar el análisis de esta arquitectura, primero se evalúa una configuración simétrica de hilos entre las dos secciones de código (el modelo global y la radiación). En particular, se estudian cuatro configuraciones de hilos, 6-6, 8-8, 12-12, y 24-24, donde el primer valor corresponde a la cantidad de hilos asignada al modelo global mientras que el segundo valor es la cantidad de hilos destinados al módulo de radiación solar. Los resultados obtenidos de estas ejecuciones se resumen en la Tabla 4.2, discriminando los tiempos en cuatro secciones de interés, *Trans*, *Sync*, *Rad* y *GM*. *Trans* representa el tiempo total requerido para transferir los datos entre el modelo global y la radiación, *Sync* es el tiempo que, de ser necesario, el modelo debe esperar por un dato nuevo de radiación, *Rad* es el tiempo que consume el cálculo de radiación y *GM* el tiempo que toma ejecutar el modelo global. Es importante destacar que la suma de las partes discriminadas no debería coincidir con el tiempo total, debido a que existen solapamientos entre algunas de las etapas.

Tabla 4.2: Tiempo de ejecución (en segundos) de las variantes propuestas utilizando el mismo número de hilos para el modelo global y el módulo de radiación.

# hilos	$Asyn_{AdjT}$					$Asyn_{AdjT+UpdV}$				
	Trans	***Sync***	***Rad***	***GM***	***Total***	***Trans***	***Sync***	***Rad***	***GM***	***Total***
6 - 6	6,59	0,001	126,21	361,40	370,14	6,86	0,260	134,29	366,89	376,05
8 - 8	6,85	0,001	100,74	344,90	353,87	6,52	0,001	110,11	358,63	367,37
12 - 12	6,52	0,001	78,50	339,95	347,68	6,52	0,001	96,57	357,60	366,09
24 - 24	6,90	0,001	67,47	371,81	381,11	6,55	0,001	96,54	393,96	401,73

Como se puede ver, la mejor configuración para $Asyn_{AdjT}$ es la que utiliza 12 hilos en cada componente, alcanzando una reducción del tiempo total del modelo de 42,81 segundos. Proporcionalmente, esta mejora equivale a un 11,0 % del tiempo total de ejecución, frente al 16,1 % del máximo teórico. Profundizando en el análisis de los resultados obtenidos, esta versión involucra tiempos de transferencia entre las dos partes, de casi siete segundos, que no pueden ser solapados totalmente, lo que impacta negativamente en el tiempo total de ejecución del modelo. Como consecuencia, la máxima reducción teórica alcanzable se reduce a un 14,5 % (56,55s), lo que resulta en que la versión $Asyn_{AdjT}$ consigue una mejora del 75,70 % de este límite teórico-práctico.

En el caso de la versión $Asyn_{AdjT+UpdV}$ la mejor configuración de hilos fue también la que asigna 12 hilos a cada parte. Sin embargo, las mejoras obtenidas por esta versión son más modestas, logrando reducir solamente 24 segundos con respecto al tiempo de la mejor implementación del modelo WRF. Esta reducción se traduce porcentualmente en un 6,2 % de mejora, lo cual está lejos del 16,1 % teórico (y tambien del 14,5 %). Si se observa con detenimiento estos resultados se puede notar que esta mejora tenue no se debe a que existan tiempos elevados de sincronización, ya que para este caso los valores de *Sync* están por debajo de un segundo. De este análisis surge la hipótesis de que las pobres reducciones alcanzadas son consecuencia de la saturación del ancho de banda del bus de memoria de la CPU. En otras palabras, ambas partes se encuentran compitiendo por el acceso a memoria, lo que impacta negativamente en la ejecución del modelo total, llevándolo de 327,42 segundos en la versión WRF_{rad6}, si se considera el tiempo de la radiación totalmente solapado (390,49 - 63,07), a 357,60 segundos para esta variante asincrónica con 12 hilos.

Considerando los resultados anteriores, se realiza un segundo experimento con el objetivo de corroborar la hipótesis planteada. Para esta evaluación se realizan ejecuciones con configuraciones asimétricas de la cantidad de hilos, es decir, distinto número de hilos para el modelo global y el módulo de radiación. Es importante mencionar que, como se busca obtener el mejor tiempo de ejecución posible para el modelo WRF, se tomaron para todos los casos de prueba 12 hilos para el modelo global, dado que esta configuración es la que presenta los menores tiempos. En concreto se consideran tres configuraciones 12-4, 12-6 y 12-8, buscando evaluar el impacto que genera el uso de distinta cantidad de hilos en el módulo de radiación sobre los tiempos de ejecución totales del modelo WRF.

Tabla 4.3: Tiempo de ejecución (en segundos) de las variantes propuestas utilizando distinto número de hilos para el modelo global y el módulo de radiación.

# threads	$Asyn_{AdjT}$					$Asyn_{AdjT+UpdV}$				
	Trans	***Sync***	***Rad***	***GM***	***Total***	***Trans***	***Sync***	***Rad***	***GM***	***Total***
12 - 4	6,40	0,09	209,05	347,84	355,86	6,88	29,56	156,27	341,36	380,26
12 - 6	6,34	0,02	143,79	340,87	349,20	6,51	0,22	149,59	344,75	352,92
12 - 8	6,49	0,001	137,02	345,36	354,16	6,556	0,09	152,48	349,48	358,24

Como se puede notar en la Tabla 4.3, la mejor configuración para ambas versiones en este caso de prueba es la que utiliza seis hilos para la radiación. Por un lado, en el caso de $Asyn_{AdjT}$ no se nota un cambio sustancial en el tiempo de ejecución total de la herramienta comparado con la versión de 12-12 hilos, alcanzando un tiempo de 349,20 segundos. Sin embargo, para la versión $Asyn_{AdjT+UpdV}$ se puede ver una mejora interesante en los tiempos de ejecución total, pasando de 366,09 segundos, para la mejor versión asincrónica hasta el momento, a 352,92 segundos con esta nueva configuración. Este resultado prueba la validez de la hipótesis planteada previamente, lo que indica que al introducir esta nueva forma de computar la radiación se transforma el problema de uno principalmente CPU-bound a uno memory-bound [69]. Notar que, agregar más unidades de cómputo no mejora el tiempo de ejecución.

Para concluir con este estudio, se realiza una tercera evaluación experimental, en este caso limitando la cantidad total de hilos en las variantes asincrónicas de la arquitectura a los utilizados para la mejor versión original del modelo WRF, la cual es 12 hilos. Como se puede observar en la Tabla 4.4, se presenta el desempeño tanto de la versión $Asyn_{AdjT}$ como $Asyn_{AdjT+UpdV}$ utilizando dos configuraciones, una simétrica 6-6 hilos y otra asimétrica utilizando 8-4 hilos. Para esta prueba, ambas versiones logran superar a la mejor implementación del modelo WRF original, obteniendo mejoras de aproximadamente 7,0 % para la variante $Asyn_{AdjT}$ y mejoras del orden de 3,7 % para la implementación $Asyn_{AdjT+UpdV}$.

Tabla 4.4: Tiempo de ejecución (en segundos) de las variantes propuestas utilizando una cantidad fija de 12 hilos para todo el modelo WRF.

# threads	$Asyn_{AdjT}$					$Asyn_{AdjT+UpdV}$				
	Trans	***Sync***	***Rad***	***GM***	***Total***	***Trans***	***Sync***	***Rad***	***GM***	***Total***
6 - 6	6,59	0,001	126,21	361,40	369,53	6,86	0,26	134,29	366,89	376,12
8 - 4	6,39	0,071	198,46	354,87	363,22	7,46	20,04	190,97	352,40	380,86

A modo de compendio de los resultados analizados se agregan a continuación las Figuras 4.5 y 4.6, las cuales resumen los tiempos de ejecución de todas

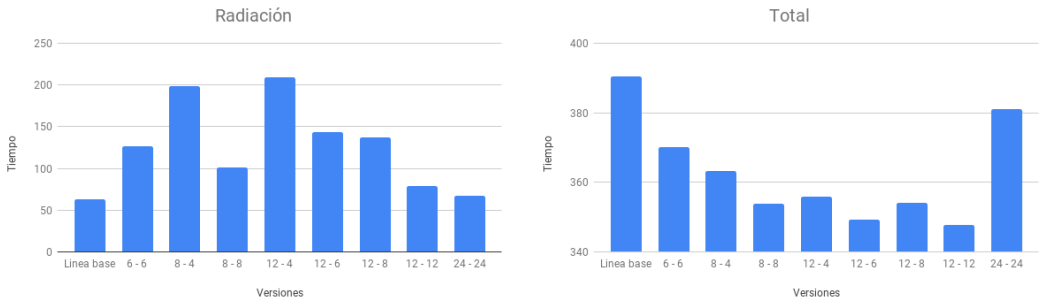


Figura 4.5: Resumen de los resutados obtenidos para la version $Asyn_{AdjT}$ y la línea base, separados por el tiempo de la radiación y el tiempo total del modelo WRF

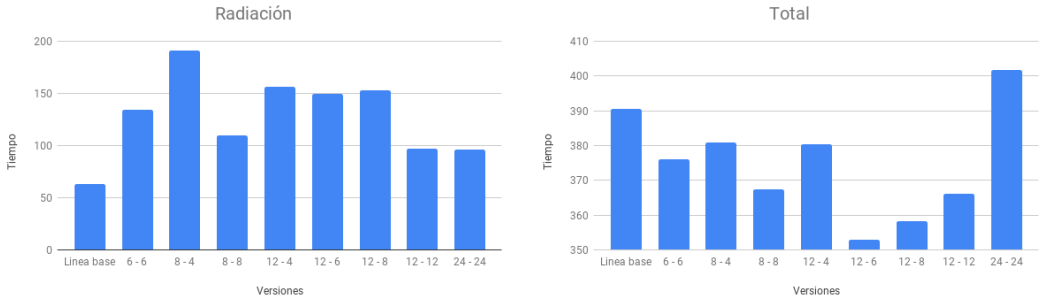


Figura 4.6: Resumen de los resutados obtenidos para la version $Asyn_{AdjT+UpdV}$ y la línea base, separados por el tiempo de la radiación y el tiempo total del modelo WRF

las configuraciones de hilos ejecutadas y la línea base, discriminadas por la versión del modelo asincrónico utilizado y partida según el tiempo de la radiación y el tiempo total del modelo.

Los resultados obtenidos en estos tres experimentos muestran que la versión asincrónica propuesta en este trabajo resulta ser una alternativa interesante para el cálculo de la radiación, ya que admite incorporar más recursos de cómputo para el procesamiento del modelo WRF, lo que permite aprovechar plataformas de cómputo paralelo más potentes. Adicionalmente, esta novel arquitectura logra reducir los tiempos de ejecución aun cuando se suministran los mismos recursos de cómputos que para su contraparte original, siendo entonces una alternativa más eficiente para el caso de estudio relacionado a la herramienta de predicción de la generación de energía fotovoltaica desarrollada.

Capítulo 5

Utilización de arquitecturas masivamente paralelas

A partir de los resultados alentadores arrojados por los experimentos realizados sobre la arquitectura asincrónica para el cálculo de la radiación, resulta interesante evaluar la posibilidad de portar parte del cómputo de esta propiedad fuera de la CPU, dado que dicho cálculo en la novel arquitectura se encuentra desacoplado de la ejecución del modelo. En particular, es interesante evaluar la posibilidad de ejecutar la radiación, o al menos una porción de ésta, en un coprocesador masivamente paralelo, como puede ser una GPU o procesador un Xeon-Phi.

En este capítulo se describe, en primera instancia, una variante de la arquitectura mencionada en la sección anterior y se evalúa el tiempo de ejecución de las rutinas presentes en el módulo de radiación, a efectos de encontrar potenciales secciones de código a portar a dispositivos secundarios de cómputo. Posteriormente a la identificación de dichas porciones de código, se describe su portado tanto a GPU como a un procesador Xeon-Phi, mencionando las distintas estrategias seguidas para ejecutar la sección de forma eficiente en cada uno de los coprocesadores. Finalmente, se evalúan experimentalmente ambas implementaciones y se discuten conclusiones específicas de cada implementación, así como deducciones más generales en relación a la arquitectura asincrónica con cualquiera de los dos coprocesadores.

5.1. Diseño de la solución

En esta sección se discuten los cambios necesarios para introducir el uso de hardware many-core a la arquitectura asincrónica previamente descripta en esta tesis. Concretamente, el objetivo de esta propuesta es portar parcial o totalmente el módulo de la radiación a un coprocesador, de forma de liberar a la CPU de este cálculo. Esta migración de cómputo permite, potencialmente, utilizar los recursos dedicados en un principio a la radiación, ya sea tanto en lo que refiere al poder de cómputo como al ancho de banda de la memoria principal, al resto del cómputo del modelo WRF y mejorar así su tiempo de ejecución. Siguiendo esta premisa, se debe extender el diseño de la arquitectura asincrónica, como se muestra en la Figura 5.1.

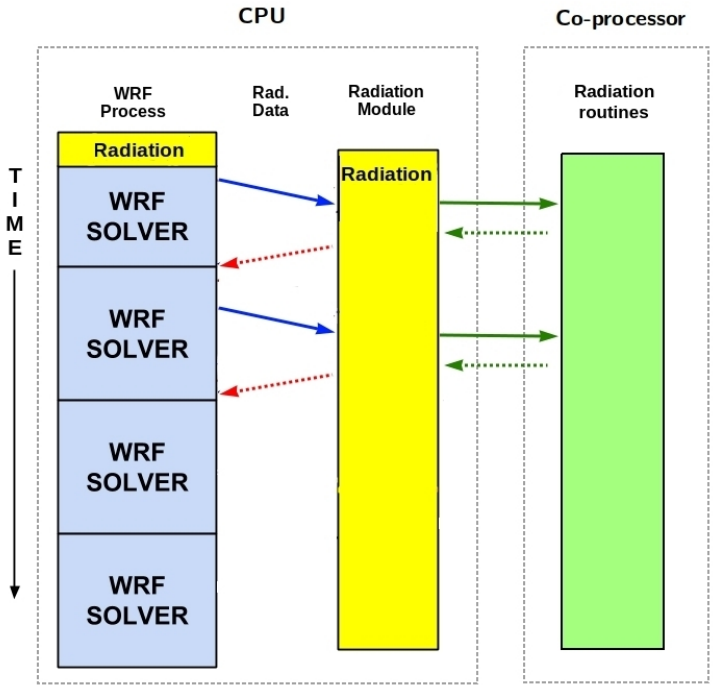


Figura 5.1: Diagrama de la nueva arquitectura asincrónica, portando el cálculo de la radiación a un coprocesador.

Como se puede observar en este diagrama, la idea de esta modificación se basa en que el módulo de la radiación es el encargado de enviar el cálculo de esta propiedad a un coprocesador y recibir los resultados. Por este comportamiento, es posible liberar a la CPU del cómputo de la radiación, al menos

parcialmente, y "esconder" los tiempos de transferencia entre el modelo WRF y el coprocesador, ya que el módulo de la radiación se encuentra funcionando de forma asincrónica al resto del modelo.

A partir de la reestructura del diseño de la solución, se implementa una nueva variante de la arquitectura asincrónica, pero esta vez utilizando únicamente la API OpenMP. Para ello se utiliza la primitiva `OMP_SECTION`, asignando un hilo al modelo global y un hilo al módulo de la radiación, separando así el flujo de ejecución de ambas partes. Adicionalmente, se crean grupos de hilo para ambos flujos, anidados a los creados inicialmente, permitiendo así a cada parte ejecutar sus tareas de forma paralela. Por último, y con el fin de facilitar la configuración de ambas partes con distinta cantidad de hilos en su ejecución, se debe agregar en el segundo nivel de anidamiento de la radiación un hilo auxiliar, dejando en un tercer nivel de anidamiento los hilos que computan efectivamente las distintas partes de la radiación. Si bien este último nivel es útil principalmente cuando se considera solamente la arquitectura asincrónica, puede no ser necesario utilizar más de un hilo cuando se agregue el uso de coprocesadores a la misma, dependiendo del porcentaje de cómputo de la radiación que sea portado a esta unidad de procesamiento. Tomadas todas estas consideraciones, se resume en la Figura 5.2 la jerarquía de hilos resultante.

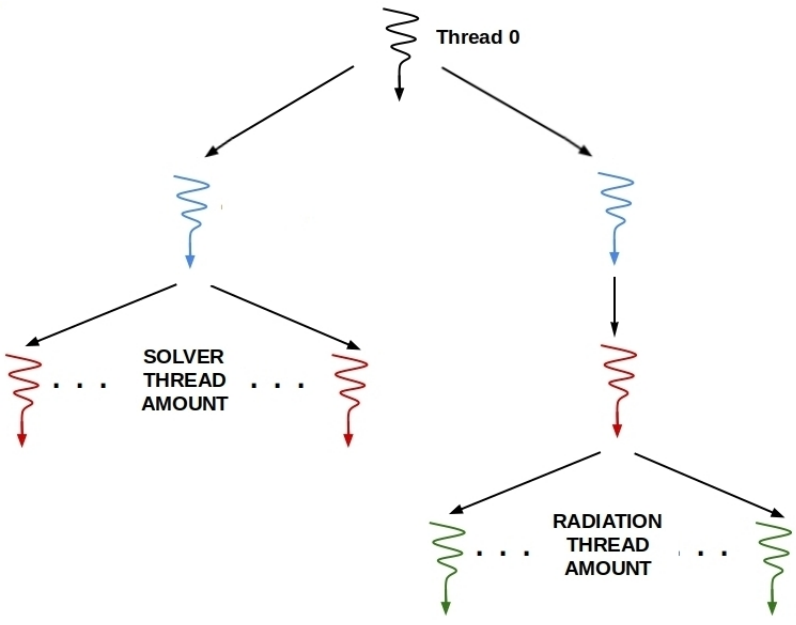


Figura 5.2: Jerarquía de hilos de la implementación de la arquitectura asincrónica utilizando OpenMP.

En lo que refiere al método de comunicación y sincronización de la arquitectura, se utilizan las mismas estructuras de datos y variables booleanas vistas en las variantes anteriores, tanto para la comunicación entre las partes como para controlar el acceso a dichas estructuras.

Una vez reformulada la implementación de la arquitectura asincrónica, se evalúa el tiempo de ejecución del módulo de radiación, desglosando éste entre las distintas funciones que lo componen. En particular, se estudian los tiempos de ejecución de los esquemas Dudhia y RRTM, para la propiedad de onda corta y larga, respectivamente, utilizando la misma plataforma en la que se llevaron a cabo los experimentos de la Sección 4.4.3. El objetivo de este análisis es encontrar secciones de código que presenten un alto consumo de tiempo de cómputo (originalmente de la CPU), detectando así posibles porciones candidatas a ser portadas para sacar provecho del poder de cómputo de los coprocesadores masivamente paralelos.

A partir de esta evaluación, se concluye que dentro de las funciones involucradas en el cálculo de la radiación se destacan las rutinas: `rtrn`, `taugb3` y `taugb5`. Específicamente, la suma de los tiempos de ejecución de estas tres funciones implica el 12 % del tiempo total de ejecución del modelo WRF, mientras que ninguna de las restantes rutinas del módulo de la radiación sobrepasa el 1 % de este tiempo de ejecución. Analizando el código al que pertenecen estos tres procedimientos, se puede ver que todas forman parte del módulo de radiación de onda larga RRTM, por lo que se estudia este módulo con mayor profundidad. Una propiedad interesante que se observa en la ejecución de este módulo es que, si bien consume una porción relativamente importante del tiempo de ejecución, no necesita un gran volumen de datos para su ejecución.

Otra característica interesante que presenta este módulo está relacionada con la forma en que se calcula la radiación en los distintos puntos de la grilla tridimensional. Concretamente, el módulo, mediante bucles anidados sobre los ejes de coordenadas `x` e `y`, extrae cada columna vertical del eje `z` y las procesa cada una por separado. Esta técnica para procesar la grilla, que como ya se vio en otros trabajos (ver Sección 3.4) está presente en distintos módulos del modelo WRF, indica que no existe dependencia entre los datos en el plano `x-y`, lo que permite fácilmente paralelizar el procesamiento de las columnas del eje `z`.

A partir de las consideraciones anteriores, se decide portar el módulo RRTM en su totalidad a un coprocesador y así estudiar los beneficios que la nueva arquitectura puede brindar.

En esta tesis en particular, la migración del módulo se realiza a dos tipos de coprocesadores, GPU y Xeon-Phi, de forma de poder evaluar la nueva versión del WRF con distintas arquitecturas masivamente paralelas. A continuación, se describen las principales características del portado del módulo RRTM a ambas arquitecturas de hardware.

5.1.1. Cómputo de la radiación en GPU

Para el estudio de la arquitectura asincrónica calculando el módulo RRTM en GPU, se utiliza integramente el código de dicha rutina del trabajo de Ruetsch et al. [70]. En dicha publicación, el autor realiza un portado del módulo de onda larga a GPU utilizando CUDA Fortran [71], ajustando el código de forma de aprovechar de la mejor forma posible los beneficios que brindan las GPUs de *Nvidia*. Concretamente, en el mencionado trabajo se crearon dos funciones, `rrtminit_cuda` y `rrtmlwrad_cuda`, que se encargan de ejecutar el módulo RRTM en GPU, actuando como interfaz a ser utilizadas por el modelo WRF. La primera función, `rrtminit_cuda`, es la responsable de inicializar valores que no dependen del paso de tiempo de la simulación, como tablas de consulta y constantes de la grilla, tanto en CPU como en GPU. Por otro lado, la función `rrtmlwrad_cuda` realiza la transferencia a la tarjeta de los datos necesarios para el cálculo de la radiación de onda larga que dependen del paso temporal, ejecuta el código para computar esta propiedad en la GPU y obtiene los valores de salida del módulo nuevamente a la CPU.

De forma de aprovechar el poder de cómputo de la GPU, el cálculo de la radiación es separado en varios kernels, con el objetivo de explotar la independencia de datos en los ejes `x` e `y`, sacando provecho además de algunos datos que presentan independencia en el eje `z`. Utilizando esta estrategia de cómputo, no sólo se utiliza intensamente el poder de cómputo de la GPU, sino que se esconden los tiempos de transferencia, ya que cuando se está ejecutando alguno de los kernels se pueden transferir los datos necesarios para computar el siguiente kernel. Adicionalmente, esta separación de código permite liberar presión sobre los registros asignados a cada hilo, ya que cada kernel maneja un conjunto reducido de todas las variables del módulo de radiación, por lo que

permite tener más variables en los registros de los multiprocesadores.

Además de las características ya descriptas, en el trabajo presentado por Ruetsch et al. se plantea la transposición de algunas matrices cuando se envían a la GPU, de forma de beneficiarse del acceso coalesed al momento de acceder a éstas. Finalmente, se reordenaron ciertos condicionales (instrucciones *IF*) en el código del módulo, de forma de minimizar la divergencia de hilos dentro de un mismo warp.

5.1.2. Cómputo de la radiación en Xeon-Phi

Previo a comenzar con la descripción de la implementación del código RRTM para el procesador Xeon-Phi, es importante destacar que no se encontró en la bibliografía del área ningún trabajo que realizara el portado de este módulo a Xeon-Phi, por lo que esta tarea se realiza como parte del trabajo de tesis. Adicionalmente, el centro de la contribución de este desarrollo es evaluar la novel arquitectura, propuesta en el capítulo anterior, en plataformas de hardware masivamente paralelas, por lo que el objetivo de este portado es obtener una implementación que utilice eficientemente el procesador Xeon-Phi. Se destaca entonces que no se ahondó en técnicas avanzadas de Xeon-Phi sobre la rutina a portar, teniendo en cuenta que aplicar estos métodos aumentan la dificultad de la codificación considerablemente, principalmente por ser un modelo numérico legado. Para comenzar con el portado, y considerando que el coprocesador soporta nativamente la API OpenMP y la implementación actual del módulo de onda larga utiliza esta herramienta, se toma como base para el código en Xeon-Phi la implementación del módulo RRTM presente en la versión original del modelo WRF. Adicionalmente, en esta implementación se hace foco en dos aspectos: minimizar la transferencia de datos entre la CPU y el coprocesador y reordenar la distribución de la grilla y ciertas porciones de código para que el módulo saque provecho de una mayor cantidad de hilos.

Para cumplir con estas tareas se diseñan dos funciones, `rrtminit_phi` y `rrtmlwrad_phi`, las cuales adoptan un comportamiento similar a las utilizadas en GPU. La función `rrtminit_phi` es la encargada de inicializar las estructuras de datos necesarias para el cálculo de la radiación, además de transferir datos constantes al coprocesador, es decir datos que no dependen del paso temporal de la simulación. Por otro lado, la función `rrtmlwrad_phi` es la responsable de administrar los datos transferidos hacia y desde el coprocesador, reordenar

el particionado de la grilla y realizar el cálculo de la radiación de onda larga en el procesador Xeon-Phi. Para poder llevar adelante esta tarea, se separa esta función en dos partes, una que ejecuta en CPU y otra en el Xeon-Phi. La porción de este procedimiento en la CPU se encarga de reordenar la distribución de la grilla, utilizando dos variables, `xtiles` y `ytiles`, las cuales indican de qué tamaño serán los bloques a procesar por cada hilo en el coprocesador. Luego de esta tarea, la función envía los datos a procesar y la nueva distribución de los *tiles* al Xeon-Phi, llamando finalmente a la función que ejecuta en ese hardware y esperando por sus resultados. Dentro del coprocesador, la función que calcula la radiación toma los datos enviados por la CPU, crea la cantidad de hilos correspondientes a la nueva cantidad de bloques creados, es decir `xtiles`×`ytiles`, y adjudica a cada hilo la porción de datos que le corresponde, computando así la radiación para ese instante de tiempo.

5.2. Evaluación experimental

En esta sección se reportan los resultados del análisis experimental realizado sobre la extensión de la nueva arquitectura, calculando la radiación en un coprocesador, concretamente en una GPU de *Nvidia* y un procesador Xeon-Phi. En primer lugar, se mencionan las plataformas utilizadas, así como el escenario seleccionado para llevar a cabo estas pruebas. Luego se procede a evaluar la nueva arquitectura, estudiando las implementaciones para cada uno de los coprocesadores por separado, resumiendo y discutiendo los resultados obtenidos.

5.2.1. Ambiente de evaluación

Dado que se utilizan para ejecutar la radiación dos coprocesadores de características muy distintas, para los experimentos se consideran dos plataformas de hardware diferentes, una para cada uno de estos coprocesadores.

Por un lado, para la ejecución basada en el procesador Xeon-Phi se utiliza una plataforma con dos procesadores *Intel* Xeon E5-2650, cada uno con 8 núcleos a 2.00GHz., y una memoria RAM de 64GB. Adicionalmente, esta máquina cuenta con un coprocesador *Intel* Xeon-Phi 31S1P, que tiene 57 cores (capaz de soportar cada uno un máximo de cuatro hilos) a 1.1 GHz. y 8 GB de memoria GDDR5. Esta plataforma utiliza un sistema operativo CentOS 6.5 y

cuenta con los compiladores de *Intel ifort* e *icc* en su versión 14.0.2.

Por otro lado, en el caso de la implementación empleando una GPU de *Nvidia*, se utiliza una máquina con un procesador *Intel* Core i7 6700 con cuatro núcleos a 3.40 GHz. y una memoria RAM de 64 GB. Además, se usa una GPU *Nvidia* Titan X (de la arquitectura Maxwell) que cuenta con 3070 CUDA cores a 1.0 GHz. y una memoria interna de 12 GB de tipo GDDR5. Esta plataforma cuenta asimismo con el sistema operativo CentOS 7.0 y el compilador de CUDA Fortran de la empresa PGI en su versión 17.4.

Para ambas plataformas se utiliza la misma versión de WRF y un escenario idéntico al empleado en los experimentos previos. Además, se usan los módulos de radiación, Duhia y RRTM, y un paso de simulación de 3 minutos. Adicionalmente, se utilizó para ambas plataformas la versión $Asyn_{UpdV}$ de la arquitectura asincrónica presentada en este trabajo.

Finalmente, se emplean dos frecuencias distintas para el cálculo de la radiación en la arquitectura asincrónica, debido al desbalance que presenta el desempeño relativo entre la CPU y el coprocesador utilizado, intentando no perjudicar a ninguna unidad de cómputo dentro de cada plataforma. Concretamente, en el caso de prueba que involucra al Xeon-Phi se utiliza una frecuencia de cálculo de una radiación cada dos pasos de tiempo del modelo WRF, mientras que para la arquitectura propuesta utilizando la GPU es de una radiación por paso de simulación.

5.2.2. Análisis experimental

En esta sección se presentan los resultados experimentales obtenidos al analizar el desempeño de las implementaciones asincrónicas del WRF basadas en el uso de un coprocesador. Adicionalmente a estas dos evaluaciones, se incluyen los tiempos de ejecución de otras tres variantes relacionadas, con el objetivo de comparar las nuevas versiones propuestas y resaltar los posibles beneficios que éstas presentan. En particular, estas implementaciones que sirven de referencia son, una versión original del modelo WRF, una versión híbrida de la arquitectura original del WRF que utiliza el coprocesador para computar la radiación solar y una versión en CPU de la arquitectura asincrónica descripta en este capítulo, calculando la radiación por completo en la CPU.

Con el objetivo de presentar los datos experimentales de forma ordenada, se divide la evaluación experimental en dos subsecciones, comenzando con el

estudio de la propuesta utilizando Xeon-Phi y, posteriormente, analizando los resultados de su contraparte en GPU.

Es importante destacar que, considerando que se utiliza una plataforma distinta para cada coprocesador, se realiza toda la batería de experimentos en cada una de las computadoras, con el objetivo de ser lo más justo posible a la hora de analizar los beneficios alcanzados por la implementación paralela utilizando cada unidad de cómputo secundaria. Además, cabe mencionar que el objetivo de los experimentos presentados en esta sección es evaluar la novel arquitectura asincrónica utilizando distintos coprocesadores y no así comparar el desempeño entre las dos implementaciones (coprocesadores) propuestas en este capítulo. Finalmente, para esta sección no se utilizará la métrica speedup ni eficiencia para comparar las variantes a estudiar ya que los procesadores presentes en una CPU y los que utiliza un coprocesador difieren ampliamente tanto desde un punto de vista conceptual como de implementación, por lo que utilizar dichas métricas no aporta información de utilidad para los experimentos a desarrollar.

Evaluación de la implementación en Xeon-Phi

Para comenzar con el análisis de los resultados de la implementación que utiliza el coprocesador Xeon-Phi, se realizan ejecuciones del WRF en su versión original, con el objetivo de ver el desempeño y escalabilidad tanto de éste como del módulo de radiación. En particular, se ejecuta el modelo con la arquitectura original sobre la plataforma empleada, variando la cantidad de hilos, midiendo tanto el tiempo de ejecución del modelo WRF en su totalidad, como el tiempo requerido por el módulo de radiación. En la Tabla 5.1 se resumen los resultados obtenidos en estas ejecuciones, variando entre 1, 2 y 4 la cantidad de hilos asignada a la ejecución de todo el modelo (y el módulo de radiación).

Tabla 5.1: Tiempo de ejecución (en segundos) del modelo original WRF y del módulo de la radiación.

# Hilos	***Tiempo radiación***	***Tiempo total***
1	303,68	744,42
2	156,09	393,97
4	82,59	217,13

Como se puede observar, tanto el modelo WRF como el módulo de radiación presentan una reducción en los tiempos de ejecución cuando se incrementa el

número de hilos para su cómputo. Específicamente, el mejor desempeño para ambas partes se obtiene cuando se utilizan cuatro hilos, superando en 3,43× y 3,68× al modelo WRF y el módulo de radiación, respectivamente. Además de estos beneficios, se puede observar que ambas partes evaluadas del modelo original presentan una buena eficiencia cuando se las ejecuta en la CPU de la plataforma, logrando valores de 85 % para el modelo y de casi 92 % para el módulo de la radiación cuando se utilizan cuatro hilos.

En un segundo experimento se evalúa la implementación híbrida CPU-Xeon-Phi, siguiendo un enfoque tradicional relativo a la inclusión del código en el coprocesador (como se puede observar en los trabajos descriptos en la Sección 3.4). En otras palabras, la idea es ejecutar el modelo original WRF en la CPU y, cuando llegue el momento de calcular la radiación de onda larga, se invoca sincrónicamente la rutina en el coprocesador, la cual ejecuta el cómputo de la radiación y retorna los valores de esta propiedad a la CPU. En particular, se estudia esta propuesta variando la cantidad de hilos en la CPU entre 1, 2 o 4 y utilizando el procesador Xeon-Phi en su máxima capacidad de cómputo, o sea con 228 hilos.

En la Tabla 5.2 se resumen los tiempos de ejecución de la mencionada implementación híbrida. Como se puede observar en estos valores, el tiempo de ejecución del módulo de radiación cuando se ejecuta parte de él en el Xeon-Phi es considerablemente menor que el tiempo de ejecución de dicho módulo cuando se utiliza sólo un hilo en CPU. Sin embargo, cuando se utilizan más hilos en la CPU, estos tiempos de ejecución se ven disminuidos en mayor medida en la variante original del módulo de radiación, la cual utiliza solamente la CPU. De estos resultados se puede concluir que, si bien usar la implementación híbrida puede ser una opción interesante cuando se tiene una cantidad acotada de hilos para la ejecución del modelo, en ambientes multicore con un número razonable de núcleos (cuatro en este caso) el modelo original en CPU presenta mejores tiempos de ejecución. Esta conclusión puede llevar a descartar el uso del coprocesador Xeon-Phi a la hora de acelerar distintos módulos del modelo WRF, y en particular al módulo de radiación.

Tabla 5.2: Tiempo de ejecución (en segundos) de la implementación híbrida CPU-Xeon-Phi, utilizando la arquitectura original del WRF.

# hilos	*Tiempo radiación*	*Tiempo total*
1	104,25	579,74
2	100,32	376,61
4	98,17	236,36

Para finalizar con la evaluación de este coprocesador, se analiza la arquitectura asincrónica del WRF y la extensión de ésta utilizando el Xeon-Phi para el cómputo del módulo RRTM. El objetivo de estas pruebas es estudiar la arquitectura asincrónica en la plataforma propuesta y analizar posteriormente el desempeño de la misma cuando se utiliza el procesador Xeon-Phi, variando la cantidad de hilos a utilizar tanto en el módulo de radiación como en el resto del modelo. Ambas implementaciones se evalúan utilizando dos configuraciones distintas de hilos en CPU, una que usa dos hilos para el módulo de la radiación y dos hilos para el resto del modelo y otra que utiliza un hilo para el módulo de radiación y cuatro para el resto del modelo. En el caso de la arquitectura asincrónica extendida, se reestructura la grilla de forma que el coprocesador Xeon-Phi es utilizado con su máxima capacidad de hilos, que son 228. Los resultados de evaluar estas configuraciones sobre ambas implementaciones se resumen en la Tabla 5.3. En este caso, y para comprender de mejor forma de donde provienen los tiempos de ejecución, se presenta dicha métrica discriminada en: el tiempo consumido por el módulo de la radiación (*Rad*), el tiempo de transferencia de datos entre el módulo de la radiación y el resto del modelo (*Trans*), el tiempo de sincronización que introduce la arquitectura asincrónica (*Sinc*) y el tiempo total de ejecución del modelo WRF (*Total*). Es importante mencionar que la suma de estos tiempos individuales es mayor al tiempo total de ejecución del modelo WRF, debido a que existen solapamientos entre las etapas.

Tabla 5.3: Tiempo de ejecución (en segundos) de las implementaciones asincrónicas del modelo WRF.

# Hilos	*Arquitectura asincrónica*				*Arquitectura asinc + Xeon-Phi*			
	Rad	*Trans*	*Sinc*	*Total*	*Rad*	*Trans*	*Sinc*	*Total*
2-2	156,99	7,23	0,09	243,38	101,12	7,65	0,56	247,31
4-1	306,51	7,38	175,39	322,51	100,02	7,70	0,71	154,38

Analizando los resultados para la implementación de la arquitectura asincrónica en OpenMP, se puede notar que la configuración que presenta mejor tiempo de ejecución es la que utiliza 2-2 hilos. Sin embargo, este tiempo de ejecución es ligeramente peor si se lo compara con los resultados obtenidos por el modelo original WRF cuando se utiliza la misma cantidad de recursos de cómputo.

En cambio, en el caso de la implementación de la nueva arquitectura utilizando Xeon-Phi se pueden observar una reducción considerable en los tiempos de ejecución del WRF. Específicamente, la configuración que utiliza 4-1 hilos disminuye el tiempo de ejecución total del modelo en 62,75 segundos, lo que equivale a una mejora del 28,9 %. Es importante mencionar que existe un límite teórico de 38,0 % sobre las mejoras que se pueden obtener en esta implementación, que corresponde al solapamiento total del tiempo de ejecución del cómputo de la radiación (82,59 segundos de un total de tiempo de ejecución de 217,13 para la versión original del WRF sobre esta plataforma). Adicionalmente, si se considera que existen tiempos de transferencia de datos entre el módulo de la radiación y el resto del modelo que no se pueden solapar con otras tareas, este límite disminuye a 34,5 %. Considerando estos valores, se nota claramente que el tiempo de ejecución requerido para calcular el módulo de radiación no restringe el desempeño de esta implementación, aunque esta condición es válida siempre que el tiempo necesario para calcular esta propiedad se mantenga por debajo del tiempo de ejecución del resto del modelo.

Para concluir con este análisis, es importante mencionar que la extensión de la arquitectura asincrónica que utiliza el coprocesador Xeon-Phi aprovecha mejor los recursos, tanto la CPU como el coprocesador, que su contraparte híbrida basada en la arquitectura original del WRF.

Evaluación de la implementación en GPU

En este apartado se realiza la evaluación experimental de la arquitectura extendida, utilizando una GPU de *Nvidia* como procesador de cómputo secundario. Para llevar a cabo este análisis, se realiza una batería de pruebas análogas a las descriptas en la subsección anterior.

En primer lugar, se estudia el desempeño del modelo WRF en su versión original sobre la nueva plataforma, variando la cantidad de hilos que se utilizan

para éste entre 1, 2 y 4 hilos.

Tabla 5.4: Tiempo de ejecución (en segundos) del modelo original WRF y del módulo de radiación.

# hilos	*Tiempo radiación*	*Tiempo total*
1	189,11	455,58
2	102,50	282,86
4	55,44	181,87

Como se puede ver en la Tabla 5.4, el mejor tiempo de ejecución para el modelo original se alcanza con la configuración de cuatro hilos. En este caso, el desempeño alcanzado para el modelo WRF supera a la versión secuencial del mismo en 2,50×, mientras que para el módulo de la radiación este beneficio es de 3,41× frente a su contraparte serial. En lo que refiere a la eficiencia de esta implementación, se tiene que el módulo de radiación obtiene valores del 85 % cuando se utilizan cuatro hilos, mientras que para el modelo WRF con la misma configuración logra un 63 %.

El segundo experimento consiste en ejecutar la versión híbrida utilizando la GPU sobre la arquitectura original del modelo WRF, al igual que se analizó en la subsección anterior.

Tabla 5.5: Tiempo de radiación en segundos de la arquitectura original del modelo WRF, ejecutando el módulo RRTM en GPU.

# hilos	*Tiempo radiación*	*Tiempo total*
1	73,57	347,71
2	69,63	247,97
4	61,31	191,19

En la Tabla 5.5 se presentan los tiempos relevados tras ejecutar esta variante híbrida, cambiando la cantidad de hilos de la CPU entre 1, 2 y 4 hilos. Como se puede notar en estos resultados, esta implementación híbrida presenta una interesante reducción en los tiempos de ejecución si se la compara contra la versión original del WRF en una configuración secuencial. Sin embargo, cuando se utilizan cuatro hilos, la implementación original del WRF es levemente superior a la propuesta sincrónica utilizando CPU y GPU, tanto en tiempos de ejecución del módulo de radiación como en el tiempo de ejecución total del modelo WRF. Este comportamiento puede deberse a que existen tiempos de transferencia entre la CPU y la GPU que no se pueden ocultar, dado que tanto

dicha etapa como la ejecución de la radiación en el coprocesador se realiza de forma sincrónica con el resto del modelo. Como consecuencia de esto, el uso de esta arquitectura masivamente paralela no introduce ninguna mejora para este escenario.

Para finalizar la evaluación experimental, se estudia la implementación de la arquitectura asincrónica que usa únicamente OpenMP y su variante extendida, que utiliza además la GPU para calcular la radiación de onda larga. Para ejecutar estas pruebas, se toman las mismas configuraciones de hilos descriptas anteriormente, una simétrica de 2-2 hilos y una asimétrica de 4-1 hilos. En la Tabla 5.6 se resumen los resultados obtenidos de estas ejecuciones, discriminando nuevamente los tiempos de ejecución.

Tabla 5.6: Tiempo de ejecución (en segundos) de las implementaciónes asincronicas del modelo WRF.

# Hilos	***Arquitectura asincónica***				***Arquitectura asinc + GPU***			
	Rad	***Trans***	***Sinc***	***Total***	***Rad***	***Trans***	***Sinc***	***Total***
2-2	108,93	4,48	0,01	187,95	64,09	4,41	0,01	186,13
4-1	209,53	4,50	58,92	216,17	74,94	4,36	0,01	141,07

Considerando los resultados obtenidos, se nota claramente que la mejor configuración de las evaluadas para la arquitectura asincrónica pura es la que utiliza 2-2 hilos. Sin embargo, estos resultados son levemente peores cuando se comparan con los del modelo original con cuatro hilos, a pesar de utilizar la misma cantidad de recursos de procesamiento. Si se observan los valores de la arquitectura asincrónica extendida, se puede ver que la mejor configuración es la que utiliza 4-1 hilos (141,07 segundos). Adicionalmente, esta configuración presenta interesantes mejoras cuando se la compara frente a la mejor versión del modelo original WRF, disminuyendo los tiempos de ejecución de éste en un 22,4 %. La mejora obtenida se encuentra además cerca del límite teórico alcanzable, el cual equivale a que el tiempo total del modelo WRF se reduzca a 126,43 segundos (181,87 - 55,44).

De los resultados presentados anteriormente se puede confirmar que el uso de la arquitectura asincrónica utilizando la GPU para el cómputo del módulo RRTM explota de forma eficiente todos los recursos de hardware involucrados, mejorando el desempeño de ambas estrategias (el uso de la arquitectura asincrónica y la migración a GPU del módulo de radiación de onda larga) por separado.

5.2.3. Resumen

Para finalizar, en este apartado se resumen alguno de los resultados más destacables alcanzados al incorporar coprocesadores masivamente paralelos al cómputo del WRF.

La primera conclusión de interés que se desprende de la evaluación en ambos casos es que, si se utiliza el coprocesador de forma tradicional para acelerar únicamente el módulo RRTM en la arquitectura sincrónica del WRF, este cambio no introduce ningun beneficio frente a la implementación paralela ya provista en el modelo. Este resultado puede deberse a que los tiempos extra de transferencia entre la CPU y el coprocesador contrarrestan la posible mejora obtenida al explotar el uso de dispositivos de cómputo secundario de naturaleza masivamente paralela, generando poco o ningún impacto en los tiempos de ejecución del modelo en su totalidad.

Por otro lado, es importante notar que la arquitectura asincrónica por sí sola no logra generar beneficios sobre ninguna de las dos plataformas consideradas, obteniendo peores resultados que el modelo original cuando se utiliza la misma cantidad de recursos de cómputo. Esto se debe a la limitada cantidad de hilos utilizados en la CPU (cuatro en ambas arquitecturas estudiadas), lo que permite al modelo de paralelismo presente en el WRF escalar correctamente. A esta condición se suma que la arquitectura asincrónica incurre en tiempos de ejecución asociados a tareas que son difíciles de solapar y que en las plataformas utilizadas resultan demasiado elevados (en particular las transferencias internas).

Otro resultado obtenido del análisis realizado es que el uso combinado de la arquitectura asincrónica y un coprocesador para el cómputo parcial de la radiación generan interesantes mejoras en los tiempos de ejecución del modelo, siendo en ambos casos cercanos a los límites teóricos alcanzables por la arquitectura propuesta. De este análisis se desprende además que esta combinación permite liberar correctamente a la CPU del trabajo de computar el módulo RRTM, lo que habilita a redistribuir los recursos de cómputo en la CPU y así disminuir el tiempo global de ejecución del WRF.

Para finalizar, es importante destacar que no se llevaron a cabo experimentos de precisión numérica en esta etapa ya que utilizar los coprocesadores no introduce mayores diferencias en los resultados numéricos, a no ser por aquellas asociadas al uso de aritmética de punto flotante y cálculos en paralelo. Adicio-

nalmente, es importante reiterar que si bien ambos conjuntos de experimentos permiten realizar un análisis adecuado de lo propuesto y extraer conclusiones útiles, estos no pueden ser comparados entre sí directamente. Esto se debe a que, si bien la naturaleza de las pruebas es similar, las arquitecturas de hardware utilizadas difieren considerablemente y la configuración del cómputo de la radiación es distinto en ambos casos, por lo que una comparación directa del desempeño de las plataformas, por ejemplo, para identificar que coprocesador es mejor, puede resultar en conclusiones equivocadas.

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo se resumen los distintos esfuerzos realizados en el presente trabajo, así como los principales resultados y contribuciones. Adicionalmente, se mencionan posibles líneas de trabajo a futuro que surgen como consecuencia de dicho análisis.

6.1. Conclusiones

En el presente trabajo se abordó el estudio de una herramienta de predicción de la generación de la energía solar fotovoltaica desde una perspectiva computacional. Específicamente, su tiempo de ejecución resulta de carácter crítico para realizar predicciones precisas que ayuden a administrar la matriz energética del país. En un esfuerzo previo se detectó que, de las distintas etapas que componen esta herramienta, la que exige más tiempo y recursos computacionales es el modelo de predicción de la atmósfera WRF. Por esta razón, se procedió a estudiar los tiempos de ejecución de este modelo y cómo el cálculo de la radiación afecta a éstos, considerando que se desean predicciones con una buena precisión. De este estudio se concluyó que, cuanto más precisos son los valores de radiación solar, mayor es el tiempo de ejecución que lleva el cómputo de esta propiedad, lo cual impacta fuertemente en los tiempos de ejecución total del modelo. Una característica interesante que se comprobó de este análisis es que la relación entre la cantidad de cálculos de la radiación y los pasos de simulación del modelo es directamente proporcional al tiempo de ejecución del mismo. En otras palabras, aumentar la frecuencia del cómputo de la radiación incrementa el tiempo ejecución total del modelo. Se observó entonces que este

comportamiento surge como consecuencia de que dicha propiedad atmosférica se computa de forma sincrónica al resto de las otras propiedades físicas del modelo.

Teniendo en cuenta lo expresado anteriormente, en este trabajo se propuso el diseño e implementación de una arquitectura asincrónica que habilite a efectuar el cálculo de la radiación de forma asincrónica al resto del modelo. Con esta estrategia se busca aumentar la precisión del cálculo de la radiación sin degradar notoriamente el tiempo de ejecución del modelo. Concretamente se desacopló el módulo de la radiación siguiendo una estrategia de paralelismo de tipo pipeline en un ambiente multiprocesador de memoria compartida, desarrollando un mecanismo de comunicación y sincronización de datos entre este módulo y el resto del modelo. Para la arquitecura propuesta se desarrollaron cuatro versiones diferentes, $Asyn_{Str}$, $Asyn_{AdjT}$, $Asyn_{UpdV}$ y $Asyn_{AdjT+UpdV}$, que varían en el instante para el cual se computa la próxima radiación y/o el paso en el cual se envían los nuevos datos de entrada para procesar la próxima radiación.

La arquitectura diseñada se evaluó de forma experimental en sus cuatro variantes, teniendo en cuenta tanto su desempeño en tiempos de ejecución como en la precisión numérica de los resultados. Del análisis de la precisión numérica se seleccionaron las versiones $Asyn_{AdjT}$ y $Asyn_{AdjT+UpdV}$, las cuales presentaron un nivel de error MAPE aceptable. Considerando que ambas versiones alcanzaron resultados similares, la evaluación de desempeño se centró únicamente en estas implementaciones. En los tiempos de ejecución se obtuvieron reducciones de 11 % para la versión $Asyn_{AdjT}$ y casi 10 % para la versión $Asyn_{AdjT+UpdV}$, utilizando la mejor configuración de hilos para ambos casos.

A partir de estos resultados alentadores, se desarrolló una segunda versión de la arquitectura asincrónica que incorpora el uso de coprocesadores masivamente paralelos para el cómputo del módulo de radiación. En una primera instancia se reimplementó la arquitectura asincrónica utilizando la API OpenMP y se estudió experimentalmente el tiempo de ejecución de las rutinas presentes en el módulo de radiación. Analizando los resultados de dicho estudio, se decidió que la candidata a ser portada era la rutina RRTM. En particular, se portó dicho módulo a dos coprocesadores distintos, un procesador Xeon-Phi y una GPU de *Nvidia*, con el objetivo de analizar el uso de estas arquitecturas. Para evaluar estas dos variantes, se utilizaron dos plataformas distintas, cada

una con un coprocesador de las arquitecturas de hardware estudiadas. De esta evaluación se obtuvieron mejoras en el tiempo de ejecución total del modelo WRF de casi 29 % en el caso del Xeon-Phi y de 22 % utilizando la GPU, para sus mejores configuraciones.

Considerando todos los resultados obtenidos, se puede afirmar que el cambio de paradigma relativo al cómputo de la radiación presenta claros beneficios sobre el tiempo de ejecución del modelo, siendo una alternativa ventajosa cuando la solución paralela nativa del modelo no es capaz de aprovechar al máximo el poder de cómputo de la CPU. Adicionalmente, esta arquitectura permite obtener interesantes mejoras al utilizar coprocesadores para el cálculo de la radiación, permitiendo mitigar los costos de transferencia que presenta el uso de estas arquitecturas con enfoques tradicionales.

6.2. Difusión de resultados

Como fruto del trabajo realizado en esta tesis se participó de distintas instancias de difusión. En primer lugar, se participó en el año 2016 del taller de incorporación de pronósticos de generación eólica y solar a la operación del sistema eléctrico (PRONOS), donde se presentaron los trabajos:

- Bayá R., Dufrechou E., Porrini C., Gutierrez A., Cazes G., Pedemonte M. & Ezzatti P. "Introducción de técnicas de HPC para la predicción de la generación de energía solar".
- Bayá R., Porrini C., Pedemonte M., & Ezzatti P. "Evaluación de un paradigma asincrónico de cómputo para el WRF".

En el primero de ellos se presentó el análisis de los métodos de paralelismo disponibles en el WRF, así como el uso de la GPU para el cómputo de ciertas propiedades. En el segundo se mostró conceptualmente la arquitectura asincrónica propuesta para el cálculo de la radiación y su estudio en una máquina de moderado poder de cómputo.

En cuanto a difusión en ámbitos científicos referenciados, a continuación se enumeran los trabajos realizados, categorizados por el tipo de publicación.

Congresos internacionales:

- Bayá, R., Porrini C., Pedemonte M. & Ezzatti P. "Task parallelism in the WRF model through computation offloading to many-core devices". Parallel, Distributed and Network-Based Processing (PDP) 2018, Cambridge, Reino Unido.

Congresos Regionales:

- Bayá, R., Porrini C., Pedemonte M. & Ezzatti P. "Paralelismo funcional en el WRF". Congreso sobre Métodos Numéricos y sus Aplicaciones (ENIEF) 2016, Córdoba, Argentina.

Revistas indexadas:

- Bayá, R., Pedemonte M., Gutierrez A. & Ezzatti P. "Enhancing the performance of the WRF model with an asynchronous computation architecture". Journal of Parallel and Distributed Computing. (*En proceso de revisión.*)

6.3. Trabajos futuros

Considerando los resultados obtenidos y resumidos en el presente manuscrito, es importante mencionar algunas líneas de trabajo que quedaron por fuera del alcance pero que merecen ser estudiadas en un futuro.

Por un lado, dado los tiempos de ejecución obtenidos por la arquitectura asincrónica, resulta de interés estudiar más en profundidad el balance entre precisión numérica de los datos de salida de la radiación y el tiempo de ejecución que esto representa. Concretamente, probar el asincronismo incorporado con otros módulos de radiación, tanto de onda corta como de onda larga, que involucren cómputos más complejos de esta propiedad y generen predicciones más exactas.

Adicionalmente, puede ser provechoso verificar formalmente el modelo de paralelismo presentado utilizando alguna herramienta automática, realizando así un analisis complementario y exhaustivo al propuesto en el Capitulo 4.2.

Otra línea a evaluar es el desarrollo de la presente arquitectura con otras variantes de hardware, destacándose la evaluación de ésta en un ambiente distribuido (por ejemplo un cluster) o utilizando coprocesadores de última

generación, como puede ser las nuevas GPUs de la arquitectura Volta o la familia x200 de Xeon-Phi.

Otro posible trabajo a futuro consiste en revaluar el modelo WRF, detectando otros módulos que consuman un alto tiempo de ejecución y puedan ser desacoplados utilizando la arquitectura asincrónica vista, de forma de reducir aún más los tiempos totales del modelo. Adicionalmente, es interesante analizar, como se propuso en la Sección 4.1, un versión híbrida que combine la técnica asincrónica de paso fijo implementada junto con un enfoque *best-effort*, analizando si el uso combinado de estas dos estrategias permiten alcanzar mejores tiempos de ejecución.

Finalmente, resulta de interés estudiar el uso de técnicas de pronóstico más complejas, como son las ejecuciones ensambladas o enjambre de ejecuciones (conjunto de ejecuciones), las que se pueden beneficiar de las reducciones de tiempo obtenidas.

Referencias bibliográficas

[1] Dirección Nacional de Energía - Ministerio de Industria, Energía y Minería, "Programa de Energía Eólica en Uruguay (in Spanish)." http://www.energiaeolica.gub.uy/. [Online; accessed 14-September-2017].

[2] "Dirección Nacional de Energía - Ministerio de Industria, Energía y Minería (in Spanish)." http://www.dne.gub.uy/. [Online; accessed 14-September-2017].

[3] C. Porrini, A. Gutiérrez, G. Cazes, E. Dufrechou, M. Pedemonte, and P. Ezzatti, "Leveraging HPC Techniques to Develop a Prediction Tool for Photovoltaic Solar Energy in Uruguay," in *2nd Frontiers in Computational Physics Conference: Energy Sciences, 3-5 June 2015, Zurich, Switzerland*, 2015.

[4] C. Porrini, A. Gutiérrez, G. Cazes Boezio, G. Hermida, D. Oroño, and M. Puppo, "Development of a model output statistic and implementation of an operational solar photovoltaic energy forecast model based in wrf," in *IEEE PES Conference on Innovative SMART GRID Technologies (ISGT-LA 2015), Montevideo, Uruguay, 5-7 oct*, pp. 248–253, IEEE, 2015.

[5] J. Michalakes and M. Vachharajani, "GPU acceleration of numerical weather prediction," *Parallel Processing Letters*, vol. 18, no. 04, pp. 531–548, 2008.

[6] J. P. Silva, J. Hagopian, M. Burdiat, E. Dufrechou, M. Pedemonte, A. Gutiérrez, G. Cazes, and P. Ezzatti, "Another step to the full GPU implementation of the weather research and forecasting model," *The Journal of Supercomputing*, vol. 70, no. 2, pp. 746–755, 2014.

[7] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*. Addison-Wesley Professional, first ed., 2004.

[8] S. Benkner, E. Bajrovic, E. Marth, M. Sandrieser, R. Namyst, and S. Thibault, "High-level support for pipeline parallelism on many-core architectures," in *Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings* (C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, eds.), pp. 614–625, 2012.

[9] A. J. Anderson, *Foundations of computer technology*. CRC Press, 1994.

[10] R. M. Russell, "The cray-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, 1978.

[11] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *IEEE Signal Processing Magazine*, vol. 26, no. 6, 2009.

[12] "Sitio Web Top 500." https://www.top500.org/lists/2017/11/. [Online; accessed 14-September-2017].

[13] J. Fusco, *The Linux programmer's toolbox*. Pearson Education, 2007.

[14] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.

[15] M. Snir, *MPI–the Complete Reference: the MPI core*, vol. 1. MIT press, 1998.

[16] J. Dean and S. Ghemawat, "Map-reduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] M. C. August, G. M. Brost, C. C. Hsiung, and A. J. Schiffleger, "Cray x-mp: The birth of a supercomputer," *Computer*, vol. 22, no. 1, pp. 45–52, 1989.

[18] A. V. P. Tahera Tabassum, Arokia Priya Charles, "Multicore versus multiprocessor: A review," in *High-Performance Computing in Remote Sensing V*, vol. 9646, p. 964605, International Society for Optics and Photonics, 2015.

[19] "Lista de procesadores Intel." https://en.wikipedia.org/wiki/List_of_Intel_microprocessors. [Online; accessed 14-September-2017].

[20] "Lista de procesadores AMD." https://en.wikipedia.org/wiki/List_of_AMD_microprocessors. [Online; accessed 14-September-2017].

[21] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[22] "Top Eight Features of the Intel Core i7 Processors for Test, Measurement, and Control," *National Intruments*, 2011.

[23] S. W. Keckler, H. P. Hofstee, and K. Olukotun, *Multicore processors and systems*. Springer, 2009.

[24] S. K. Shukla, C. Murthy, and P. Chande, "A survey of approaches used in parallel architectures and multi-core processors, for performance improvement," in *Progress in Systems Engineering*, pp. 537–545, Springer, 2015.

[25] A. Eichenberger, C. Terboven, M. Wong, and D. an Mey, "The design of openmp thread affinity," *OpenMP in a Heterogeneous World*, pp. 15–28, 2012.

[26] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[27] D. B. Kirk and W.-M. W. Hwu, *Programming massively parallel processors: a hands-on approach,* 2nd edition. Morgan kaufmann, 2016.

[28] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-time rendering*. Crc Press, 2008.

[29] K. Moreland and E. Angel, "The fft on a gpu," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 112–119, Eurographics Association, 2003.

[30] "Nvidia CUDA C programming guide." https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. [Online; accessed 14-September-2017].

[31] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara, S. Miki, and S. Tagawa, "The opencl programming book," *Fixstars Corporation*, 2010.

[32] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, 03 2008.

[33] "Modelos de tarjetas gráficas Nvidia que soportan CUDA." https://developer.nvidia.com/cuda-gpus. [Online; accessed 14-September-2017].

[34] X. Mei and X. Chu, "Dissecting gpu memory hierarchy through microbenchmarking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 72–86, 2017.

[35] "History of Many-Core Leading to Intel Xeon Phi." http://download.intel.com/newsroom/kits/xeon/phi/pdfs/Many-Core-History_Backrounder.pdf. [Online; accessed 14-September-2017].

[36] A. More, "Intel xeon phi coprocessor high performance programming," *Journal of Computer Science & Technology*, vol. 13, 2013.

[37] R. Rahman, *Intel® Xeon Phi™ Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress, 2013.

[38] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights landing: Second-generation intel xeon phi product," *Ieee micro*, vol. 36, no. 2, pp. 34–46, 2016.

[39] H. Shen and F. Pétrot, "Using amdahl's law for performance analysis of many-core soc architectures based on functionally asymmetric processors," in *International Conference on Architecture of Computing Systems*, pp. 38–49, Springer, 2011.

[40] J. Twidell and T. Weir, *Renewable energy resources*. Routledge, 2015.

[41] R. Alonso Suárez, *Estimación del recurso solar en Uruguay mediante imágenes satelitales*. PhD thesis, UdelaR, 2017.

[42] Dirección Nacional de Energía - Ministerio de Industria, Energía y Minería, "Programa de Energía Solar en Uruguay (in Spanish)." http://www.energiasolar.gub.uy/. [Online; accessed 14-September-2017].

[43] R. G. Barry and R. J. Chorley, *Atmosphere, weather and climate*. Routledge, 2009.

[44] B. Sørensen, *Renewable Energy. Physics, Engineering, Environmental Impacts, Economics and Planning*. Academic Press, 5th edition ed., 2017.

[45] S. A. Kalogirou, "Solar thermal collectors and applications," *Progress in Energy and Combustion Science*, vol. 30, 2004.

[46] A. Khaligh and O. C. Onar, *Energy harvesting: solar, wind, and ocean energy conversion systems*. CRC press, 2009.

[47] G. N. Tiwari, A. Tiwari, and Shyam, *Handbook of Solar Energy: Theory, Analysis and Applications*. Springer, 2016.

[48] C. J. Chen, *Physics of solar energy*. John Wiley & Sons, 2011.

[49] A. Gutiérrez, P. Santoro, V. Nunes, and J. Cataldo, "Despacho de parques eólicos: primeros avances sobre predicción de corta duración," *presentado en el 7o Encuentro de Energía, Potencia, Instrumentación y Medidas de IEEE*, 2008.

[50] "Pagina del modelo WRF." https://www.mmm.ucar.edu/weather-research-and-forecasting-model. [Online; accessed 14-September-2017].

[51] "Sitio Web de WindGuru." https://www.windguru.cz/. [Online; accessed 14-September-2017].

[52] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers, "A description of the advanced research wrf version 2," tech. rep., DTIC Document, 2005.

[53] R. A. Lighezzolo, *Integración de modelos numéricos de predicción meteorológica en sistemas de alerta temprana a emergencias*. PhD thesis, CONAE, 2014.

[54] "Tutorial Notes: WRF Software 2.1." https://dtcenter.org/events/wrf-nmm_tutorial06_summer/Presentations/WRF.Software.Arch.Dave1.pdf. [Online; accessed 14-September-2017].

[55] J. Dudhia, "WRF physics options," *NCAR WRF basic tutorial*, pp. 26–30, 2010.

[56] A. Gutiérrez, G. Cazes, and S. De Mello, "Analysis of the optimal grid resolution for the forecasting of wind energy in different wind farms," in *Conference Record - IEEE Instrumentation and Measurement Technology Conference*, pp. 1512–1516, 2014.

[57] C. G. Kruse and D. Del Vento, "Optimizing performance of the weather research and forecasting model at large core counts: A comparison between pure

mpi and hybrid parallelism and an investigation into domain decomposition [poster]," 2014.

[58] "Performance hints for WRF on Intel architecture." https://software.intel.com/en-us/articles/performance-hints-for-wrf-on-intel-architecture. [Online; accessed 14-September-2017].

[59] Y. Sharma and D. B. Kulkarni, "Gpu based acceleration of wrf model: A review," in *International Journal of Science and Research*, 2015.

[60] J. Wang, B. Huang, A. Huang, and M. D. Goldberg, "Parallel computation of the weather research and forecast (wrf) wdm5 cloud microphysics on a many-core gpu," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pp. 1032–1037, IEEE, 2011.

[61] W. Vanderbauwhede and T. Takemi, "An analysis of the feasibility and benefits of gpu/multicore acceleration of the weather research and forecasting model," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 7, pp. 2052–2072, 2016.

[62] S. Saini, H. Jin, D. Jespersen, S. Cheung, J. Djomehri, J. Chang, and R. Hood, "Early multi-node performance evaluation of a knights corner (knc) based nasa supercomputer," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pp. 57–67, IEEE, 2015.

[63] J. Mielikainen, B. Huang, and A. H.-L. Huang, "Optimizing zonal advection of the advanced research wrf (arw) dynamics for intel mic," in *High-Performance Computing in Remote Sensing IV*, vol. 9247, p. 92470M, International Society for Optics and Photonics, 2014.

[64] J. Mielikainen, B. Huang, and A.-L. Huang, "Intel xeon phi accelerated weather research and forecasting (wrf) goddard microphysics scheme," *Geoscientific Model Development Discussions*, no. 6, pp. 8941–8973, 2014.

[65] J. Mielikainen, B. Huang, and A. Huang, "Revisiting intel xeon phi optimization of thompson cloud microphysics scheme in weather research and forecasting (wrf) model," in *High-Performance Computing in Remote Sensing V*, vol. 9646, p. 964605, International Society for Optics and Photonics, 2015.

[66] J. P. Aguerre and R. Bayá, "Aceleración de una herramienta para la predicción de energía eléctrica de origen solar mediante arquitectura de hardware híbridas (in spanish)," tech. rep., INCO, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, 2015.

[67] M. Ben-Ari, *Principles of concurrent and distributed programming.* Pearson Education, 2006.

[68] B. Nichols, D. Buttlar, and J. Farrell, *Pthreads programming: A POSIX standard for better multiprocessing.* O'Reilly Media, Inc., 1996.

[69] A. Silberschatz, P. B. Galvin, and J. L. Peterson, *Operating system concepts.* Addison-Wesley, 1991.

[70] G. Ruetsch, E. Phillips, and M. Fatica, "GPU acceleration of the long-wave rapid radiative transfer model in WRF using CUDA Fortran," in *Many-Core and Reconfigurable Supercomputing Conference*, 2010.

[71] G. Ruetsch and M. Fatica, *CUDA Fortran for scientists and engineers: best practices for efficient CUDA Fortran programming.* Elsevier, 2013.

APÉNDICES

Apéndice 1

Paralelismo funcional en el WRF

PARALELISMO FUNCIONAL EN EL WRF

Rodrigo Bayá[a], Claudio Porrini[b], Martín Pedemonte[a] y Pablo Ezzatti[a]

[a]*Instituto de la Computación,Universidad de la República, Montevideo, Uruguay.*

[b]*Instituto de Mecánica de los Fluidos e Ingenieria Ambiental,Universidad de la República, Montevideo, Uruguay.*

Palabras Clave: WRF, HPC, Paralelismo funcional.

Resumen. En la última década, Uruguay ha comenzado a migrar su matriz energética hacia energías renovables de origen eólico y solar. Debido a que es difícil prever el comportamiento de los fenómenos atmosféricos asociados a dichas fuentes de generacion de energía, desde la Facultad de Ingeniería se están desarrollando herramientas que permitan predecir la generación de la energía asociada en una ventana de corto plazo. Sin embargo, estas herramientas presentan tiempos de ejecución elevados, específicamente en lo que respecta al cálculo del módelo regional de tiempo y clima WRF. En este trabajo se propone una arquitectura asincrónica del modelo WRF, buscando disminuir los tiempos de cálculo de la radiación solar. Esta novel propuesta alcanza una aceleración de 1.2 veces cuando se compara con la mejor configuración del WRF original, utilizando los mismos recursos de hardware para ambos casos.

1. INTRODUCCIÓN

Uruguay se encuentra en un proceso vertiginoso de modificación de su matriz de generación de energía eléctrica. En particular, en la última década se ha dado un importante impulso a la generación de energía eólica y de origen solar. Alineado con esta política, investigadores del Instituto de Mecánica de los Fluidos e Ingeniería Ambiental (IMFIA) de la Facultad de Ingeniería (FING) de la Universidad de la República (UDELAR) desarrollaron una herramienta para la predicción de la generación de energía eléctrica de origen eólico en el territorio Uruguayo. La herramienta se basa en la ejecución de un modelo numérico de circulación regional de la atmósfera y su post-procesamiento estadístico, así como en la asimilación en tiempo real de la potencia generada por los parques eólicos en la última hora y produce, como salida, un estimativo de la potencia generada por los mismos. Para modelo de pronóstico del tiempo, la etapa más costosa en recursos de cómputo, se usa el Weather Research and Forecasting (WRF) (Skamarock et al., 2001), uno de los modelos numéricos más populares a nivel mundial para dicho fin. Desde el año 2014, los planes de desarrollo se han focalizado en la generación de energía de origen solar fotovoltaica. En concordancia con el esfuerzo anterior, se ha comenzado a desarrollar una herramienta de predicción de la generación de energía solar fotovoltaica sobre el territorio del país. El proyecto reúne investigadores del IMFIA, del Insituto de Computación (INCO) y del Instituto de Ingeniería Electrica (IIE) de la FING. Esta herramienta también se basa en el WRF, pero buscando potenciar los cálculos referidos a la radiación solar para mejorar la precisión de dicha estimación. En resumen, ambas herramientas demandan importantes volúmenes de cálculos, lo que motiva estudiar la aceleración del WRF para disminuir su tiempo de ejecución (y/o mejorar la precisión numérica de los resultados).

Por otro lado, en los últimos años el área de computación de alto desempeño (HPC, del inglés High Performance Computing), ha cambiado radicalmente. Además de las grandes plataformas de hardware reservadas para centros de cómputo con acceso a grandes fuentes de financiamiento, ha cobrado impulso el uso de plataformas de hardware de bajo costo. En especial, aquellas que incluyen procesadores multi-core y aceleradores de hardware (como las GPUs y los procesadores Intel Xeon-Phi). Este tipo de hardware ofrece capacidades de cómputo importantes con costos económicos y niveles de consumo energéticos razonables (Padoin et al., 2013). Pero, como contrapartida, estas plataformas que son especialmente aptas para el paralelismo de datos, exigen enfoques donde se puedan abatir, o al menos acotar de manera importante, las dependencias entre datos. Si bien el WRF incluye el uso de técnicas de paralelismo mediante la aplicación de la API OpenMP, evaluaciones preliminares realizada sobre los casos de estudio de interés mostraron que la herramienta ofrece una escalabilidad pobre en este tipo de escenarios. Esta situación limita fuertemente el aprovechamiento de equipos con números elevados de cores y/o arquitecturas masivamente paralelas.

Considerando lo expuesto en los párrafos anteriores, nuestra propuesta se centra en el desarrollo de una variante del WRF capaz de ejecutar módulos de forma concurrente. Logrando así, sacar partido del paralelismo de datos y de tareas al mismo tiempo. En particular, se desarrolló un prototipo que permite desacoplar los cálculos de radiación del modelo siguiendo un paradigma de *pipeline*, dónde en un paso de tiempo se están calculando los cómputos del modelo en general y al mismo tiempo el módulo de radiación, cuya salida será entrada en pasos futuros del modelo general. Los resultados preliminares alcanzados muestran reducciones en los tiempos de ejecución de alrededor de un 20 % utilizando los mismos recursos de hardware.

El resto del documento se estructura de la siguiente forma. En la Sección 2 se describe someramente la herramienta para predecir la generación de energía solar fotovoltaica, prestando

especial atención en el modelo WRF. Luego, en la Sección 3 se presenta la propuesta, seguida de la evaluación experimental de la misma en la Sección 4. Finalmente, en la Sección 5, se ofrece un resumen de las principales conclusiones arribadas durante el trabajo y posibles líneas de trabajo futuro.

2. PREDICCIÓN DE GENERACIÓN DE ENERGÍA SOLAR FOTOVOLTAICA

La herramienta de predicción de la generación de energía de origen solar fotovoltaico desarrollada incluye diferentes componentes, como se resumen en la Figura 1. En la primera etapa, se toman los datos del modelo global de circulación de la atmósfera, Global Forecast System (GFS) (con una resolución de $0.5\,^{o} \times 0.5\,^{o}$) y estos son utilizado para establecer las condiciones de borde e iniciales del modelo regional WRF (Numerical Weather Prediction, NWP). Luego, basado en estos datos, se ejecuta el WRF utilizando diferentes esquemas de parametrización. En la tercera etapa, se aplica un Model Output Statistics –MOS– (MOS1 en la Figura 1) para corregir la radiación solar horizontal. Posteriormente, la radiación es proyectada según la inclinación de los paneles fotovoltaicos, utilizando la información de plano de inclinación definida en cada planta. En la última etapa, se aplica un segundo MOS (MOS2 en la Figura 1), que consiste en aplicar una regresión lineal utilizando datos específicos de cada planta solar (como por ejemplo, índice de claridad, ajuste territorial, etc.) de forma de estimar finalmente la producción de energía de la planta fotovoltaica.

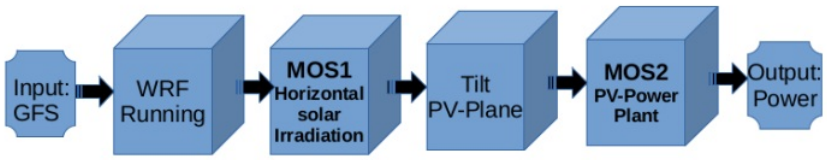


Figura 1: Módulos de la herramienta de predicción de la generación de energía de origen solar fotovoltaico propuesta.

Si se analiza esta herramienta desde el punto de vista del costo computacional, la mayor parte del tiempo de ejecución necesario para realizar la estimación de la planta es consumido por el modelo numérico de pronóstico del tiempo, el WRF. En particular, como se está estudiando el fenómeno de radiación, se utilizan módulos que permitan calcular esta propiedad de forma precisa. Cuando se analiza el funcionamiento del WRF en lo que respecta al cálculo de la radiación (y como se verá en profundidad más adelante), el modelo computa la radiación y utiliza estos valores por un lapso de tiempo posterior, configurable según una variable inicial. Si bien esta característica busca espaciar el cálculo de la radiación y así disminuir el tiempo de cálculo asociado, al momento de actualizar estos datos, el modelo debe detenerse y realizar los cálculos del pronóstico de la radiación nuevamente para el siguiente lapso temporal. La necesidad de una buena frecuencia de cálculo para este caso y el uso de un módulo de alta precisión para el cálculo de la radiación, inciden en que el WRF sea la etapa más costosa de la herramienta.

Considerando lo expresado anteriormente, nuestro esfuerzo se centra en modificar el paradigma de cómputo del WRF, por esta razón en el siguiente apartado se profundiza en la descripción del modelo.

2.1. El WRF

El programa WRF Michalakes et al. (2001) fue creado a principios de la década de los 90 por distintos centros de estudio relacionados a la investigación de los fenómenos de la atmósfera, principalmente ubicados en América del Norte. El WRF es un modelo numérico mesoescala, no hidrostático, euleriano y compresible, que permite predecir el clima orientado tanto al ámbito de la investigación atmosférica como al pronóstico del tiempo diario. Esta herramienta, permite pronosticar diversas propiedades físicas y químicas de la atmósfera, como el movimiento de masas de aire, humedad, formación de nubes y radiación emitida hacia y desde la superficie terrestre, entre otras.

Desde el punto de vista computacional, el WRF es un programa estructurado en capas y módulos, donde cada módulo presenta un solver de alguna propiedad química o física de la atmósfera. Para cada una de estas propiedades existen distintos módulos, donde varía la precisión del cálculo y en consecuencia el tiempo de ejecución del mismo. Para lograr la correcta simulación de la atmósfera, estos módulos son llamados por capas superiores del programa, siguiendo un orden en particular, debido a que la información de ciertas propiedades atmosféricas son utilizadas para los cálculos de otras. En particular, la radiación es una propiedad importante en el modelo y es utilizada por diversos módulos.

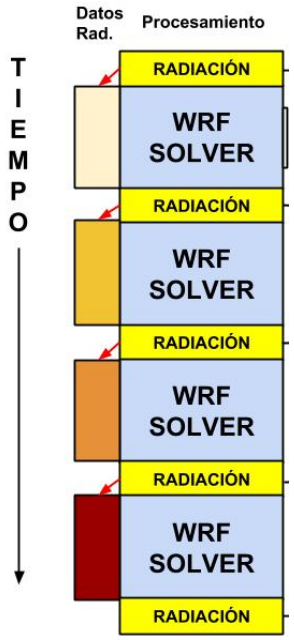


Figura 2: Funcionamiento de la arquitectura sincrónica del modelo WRF, cuando se lo divide según la porción de código que ejecuta la radiación y el resto de los módulos.

En la Figura 2, se muestra el comportamiento presente entre el módulo de la radiación y el resto de los cálculos del modelo, observándose dos características interesantes. La primera es, como se dijo anteriormente, que la radiación no se ejecuta en todos los pasos de simulación, sino que se calcula según una frecuencia determinada por una variable inicial. Esta variable le indica al código cada cuanto se ejecuta este módulo y a su vez cuanto tiempo se reutiliza el dato calculado. Este parámetro de configuración le permite al modelo definir la precisión de las variables relacionadas a la radiación independientemente del modelo que se ejecutó para su cálculo. Lo segundo a observar, es que para actualizar los datos de la radiación, el modelo

"se detiene" y ejecuta únicamente el módulo asociado a este fenómeno. Esto implica que los tiempos del modelo se ven fuertemente afectados por el tiempo que implican dichos cálculos, y más aún si se quiere que el modelo realice un cálculo preciso de esta propiedad.

En busca de disminuir los tiempos de ejecución, el modelo incluye estrategias de computación de alto desempeño, en particular permite emplear el paradigma Single Program Multiple Data (SPMD), es decir que el mismo programa computa en paralelo diferentes secciones de los datos (Foster, 1995). Sobre esta técnica, el modelo ofrece implementaciones con herramientas de memoria distribuida utilizando el estándar MPI (Snir, 1998; Gabriel et al., 2004) y memoria compartida utilizando la API OpenMP (Quinn, 2003). Específicamente, el WRF procede primero dividiendo los datos de entrada (el dominio de simulación) en sub-dominios (tantos como unidades de cómputo se especifiquen), para luego, ejecutar los cómputos de predicción sobre cada uno de los subdominios por separado. A pesar de este esfuerzo, cuando se requiere ejecutar la simulación sobre un dominio de dimensiones reducidas, como los dominios utilizados para las ejecuciones dentro del territorio Uruguayo, el paradigma se encuentra rápidamente limitado y resulta en que la herramienta no alcance altos niveles de eficiencia (Silva et al., 2014; Michalakes y Vachharajani, 2008; Skamarock et al., 2005). En la Sección 4 se estudia experimentalmente este comportamiento.

Otra herramienta que ofrece el WRF para reducir los tiempos de ejecución del modelo es el uso de dominios encajados. Esto permite realizar simulaciones de forma concurrente de varias grillas, que se contienen una dentro de la otra. De esta forma, el modelo permite aplicar los cálculos de una grilla como condición de frontera de otra, de forma de mejorar los cálculos de aquellas que son más interiores y poseen mayor precisión. En este esfuerzo no se considera el uso de dominios encajados por lo cual no se profundiza en la técnica.

3. PROPUESTA

Debido al alto costo computacional que el WRF presenta en las configuraciones necesarias para obtener un buen pronóstico de la radiación solar, resulta de interés analizar la herramienta e intentar introducir alguna técnica que permita disminuir los costos asociados. En particular, debido al pronunciado crecimiento de las arquitecturas multi-core y many-core en los últimos años, es útil introducir al modelo alguna técnica de computación de alto desempeño que permita aprovechar el poder de cómputo que estas arquitecturas ofrecen.

Como se vio en la Sección 2, el WRF presenta un sincronismo entre el cálculo de la radiación y el resto de los cálculos asociados al modelo general. Además, tiene implementado un paralelismo de datos utilizando OpenMP, que para los casos utilizados para Uruguay, no ofrece buenos niveles de escalabilidad. Por esta razón, parece interesante abordar en este contexto una propuesta basada en cambiar la arquitectura de cómputo del modelo WRF por un paradigma asincrónico, que no sólo permita realizar el cálculo en paralelo, sino que además no incurra en los problemas que surgen del paralelismo de datos ya presente en la herramienta. Específicamente, como se muestra en la Figura 3, resulta interesante implementar una técnica de paralelismo funcional que permita desacoplar los cálculos de la radiación solar y el resto del modelo, para poder ejecutar estos cómputos en forma solapada.

Este cambio en el modelo WRF permitirá hacer un uso eficiente de plataformas de hardware multi-core y many-core, aún con dominios de dimensiones modestas (como las discretizaciones usadas para representar nuestro país en los modelos numéricos implicados). En especial, el nuevo paradigma propuesto en este trabajo permite que el gestor de radiación se compute al mismo tiempo que otros módulos, disminuyendo e incluso evitando las esperas por la actualización de

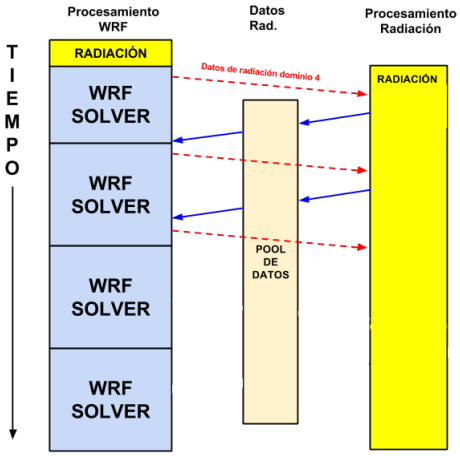


Figura 3: Funcionamiento de la arquitectura asincrónica del WRF.

los datos de este fenómeno.

Lo primero que se hizo fue desacoplar el código del cálculo de radiación del resto del modelo, que debido a la estructura en capas del modelo, no fue una tarea de alta dificultad. Luego, se estudiaron las variables involucradas en el cálculo de la radiación y se las clasificó de acuerdo a si eran de entrada al módulo o de salida de éste (o ambas). Posteriormente a esta etapa, se pasó a desarrollar el módulo de comunicación, una de las partes críticas para el funcionamiento correcto y eficiente de esta propuesta. Para ello, este módulo presenta dos *buffers*, uno dónde se almacenan temporalmente los datos a ser procesados por el módulo de radiación y otro, dónde se almacenan los datos de la última radiación calculada. Estos buffers son de interés para la implementación, ya que permiten reducir al máximo los tiempos de sincronización, los cuales se presentan solamente al momento de actualizar u obtener los datos. Por otro lado, este módulo presenta cuatro funciones, dos que le permiten a las partes actualizar los datos en los buffers correspondientes y dos que le permiten a estas partes obtener los datos que necesitan para continuar su ejecución. Además, el módulo maneja variables de mutuo exclusión, que sirven para avisar a ambas partes si pueden actualizar u obtener los datos pertinentes.

Una vez cubierta la separación de las secciones de código y la comunicación entre dichas partes, se pasó a diseñar el uso de paralelismo mediante *pipeline* que es la técnica elegida para explotar el asincronismo. Como se discutió anteriormente, el modelo necesita los datos de la radiación actualizados cada cierto período de tiempo, por esta razón, el prototipo presentado respeta este comportamiento, de forma de preservar la coherencia de datos original. Es decir, la propuesta no modifica el paso fijo seleccionado en la variable inicial, ya que el estudio del comportamiento de los valores de la radiación en este caso, requiere indagar más profundamente el efecto en el solver de usar un paradigma completamente asincrónico. Para cumplir con esta restricción, se utilizaron dos variables de mutuo-exclusión, que permiten, en caso de que no esté el dato necesario, retrasar la ejecución del modelo, de forma de no comprometer la precisión y coherencia de los datos, pero asegurando el comportamiento asincrónico deseado. En otras palabras, las variables en cuestión no permiten que los datos de la radiación se desactualicen más allá del intervalo que se indica como aceptable en las variables iniciales del modelo.

Es importante destacar, que la técnica introducida no afecta el paralelismo de datos presente en la herramienta, por lo que tanto el módulo de radiación desacoplado como el resto del modelo mantienen el uso de dicha estrategia.

4. EVALUACIÓN EXPERIMENTAL

En esta sección se resume la evaluación experimental realizada sobre la propuesta. En este sentido, se describen primero los casos de prueba considerados, luego la plataforma de hardware empleada y, por último, los resultados experimentales obtenidos propiamente dichos.

4.1. Casos de prueba

Para la evaluación experimental se usó como caso de estudio un dominio que comprende la zona sur y sur-este de Uruguay, discretizado en 243.756 puntos (74 en el eje este-oeste, 61 en el norte-sur y 54 en el vertical). Se utilizan como datos de entrada los registros ofrecidos por la National Oceanic and Atmospheric Administration (NOAA) para el día 15/07/2012, haciendo una simulación total de 3 horas a partir de las 9 de la mañana de esa fecha, con un paso temporal de 15 segundos. El caso usado es representativo, en cuanto a exigencias computacionales, del tipo de simulación realizada para la predicción de la generación de energía solar con la herramienta desarrollada para el terriotorio nacional.

4.2. Plataforma de hardware

Las ejecuciones se realizaron sobre un equipo con un procesador AMD Opteron 6272 de 64 núcleos a 2.09GHz y con 48GB de RAM.

El sistema operativo es CentOS 6 y el compilador GNU C/FORTRAN v4.4.7. Como banderas de optimización se utilizó "-O3" y las banderas de empaquetado de estructuras en memoria "-frepack-arrays -fpack-derived".

4.3. Resultados experimentales

En primera instancia se estudió el desempeño computacional de la variante original del WRF sobre el caso de estudio definido. El objetivo del estudio se centró en identificar la mejor configuración para la variante original y, al mismo tiempo, validar la premisa de la pobre escalabilidad del modelo. En este sentido la Tabla 1 presenta los tiempos de ejecución, medidos en segundos, que implica el WRF original al variar la cantidad de hilos utilizados, ademas de los valores de eficiencia ($\frac{Tiempo_{paralelo} \times \#hilos}{Tiempo_{sequencial}}$) conseguida. La Figura 4 presenta la misma información previa en forma gráfica. De los datos de la tabla[1] se puede deducir que la configuración que requiere menor tiempo de ejecución es utilizando 32 hilos. Además, como se puede ver en las gráficas, el paralelismo de datos disponible en la herramienta (para el caso abordado) no ofrece una escalabilidad buena. Notar que no sólo el menor tiempo de ejecución es con 32 hilos sino que la eficiencia decrese fuertemente al aumentar la cantidad de hilos.

[1]Notar que si bien, por cuestión de espacio, en la tabla se resumen los tiempos de ejecución para cantidad de hilos múltiplo de 8, la evaluación se realizó para todas las cantidades múltiplo de 2 (como se presenta en la Figura 4) y los tiempos al utilizar 30 y 34 hilos fue superior a utilizar 32.

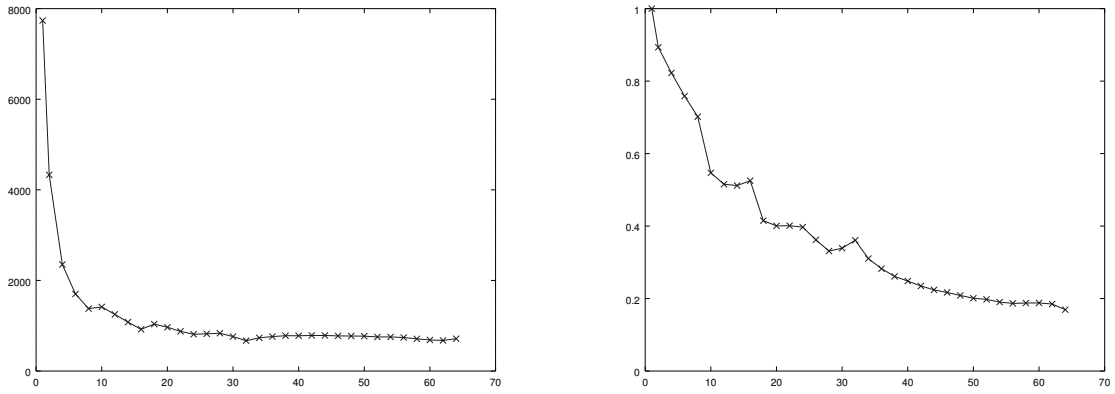


Figura 4: A la izquierda, progresión de los tiempos de ejecución (en segundos). A la derecha progresión de la eficiencia del modelo WRF en su versión original al cambiar la cantidad de hilos.

# Hilos	***Tiempo (s)***	***Eficiencia***
1	7738	-
2	4331	0.89
4	2352	0.82
8	1378	0.70
16	921	0.53
24	812	0.40
32	670	0.36
40	779	0.25
48	773	0.21
56	740	0.19
64	714	0.17

Tabla 1: Tiempos de ejecución (en segundos) y eficiencia del modelo WRF en su versión original.

Los resultados obtenidos, en lo que respecta al desempeño computacional, por la variante propuesta se resumen en la Tabla 2. La tabla incluye la configuración de la propuesta, es decir la cantidad de hilos para el modelo en general y del módulo de radiación, que implicó menores tiempos de ejecución. Además es importante destacar que se configuró la propuesta para que realice una cantidad de cálculos de radiación comparable con la cantidad que realiza la variante orginal.

Versión	***# Hilos***	***Tiempo (s)***	***Aceleración***
Original	32	670	-
Propuesta	30+30	564	1.19x

Tabla 2: Tiempos de ejecución (en segundos) de la mejor configuración del modelo original y de la propuesta asincrónica.

Si se considera la mejor configuración para cada versión, o sea la que minimiza el tiempo de ejecución del modelo, que en este caso son la de 32 hilos para el caso original (670s) y para la variante propuesta 30 hilos para el núcleo del WRF y 30 hilos para el cálculo de la radiación

(564s), los resultados muestran una mejora del orden de 1.2×. Estos resultados, conseguidos utilizando la misma plataforma de hardware, permiten afirmar que el cálculo asincrónico de la radiación logra aprovechar de mejor manera los recursos de cálculo de la plataforma.

Finalmente, es importante mencionar que para todas las ejecuciones de la nueva arquitectura, los resultados numéricos obtenidos se asemejan a los que arroja el modelo original. En particular, los resultados obtenidos en esta propuesta no presentan variaciones significativas a su par en el modelo original cuando se realizan la misma cantidad de pasos de radiación.

5. CONCLUSIONES Y TRABAJO FUTURO

En el trabajo se presenta una primera aproximación al desarrollo de una variante asincrónica del WRF. Este esfuerzo se centró en modificar el paradigma de cálculo de los módulos de radiación, permitiendo obtener un prototipo que computa de manera solapada los módulos de radiación con los módulos de cómputo del modelo general. La propuesta realizada permite utilizar paralelismo de datos en cada una de las secciones del modelo, módulos de radiación y modelo general, y a su vez paralelismo funcional entre las secciones al explotar la técnica de pipeline.

Los resultados obtenidos, si bien son preliminares, permiten vislumbrar que esta estrategia permitirá explotar de manera más eficiente las plataformas de hardware masivamente paralelas. Es conveniente señalar que se alcanzaron mejoras del entorno de un 20 % al utilizar la variante propuesta sobre una misma plataforma de hardware.

El desarrollo del trabajo no permitió dar una respuesta completa a varios desafíos y, además, se detectaron varias líneas potenciales para extender la propuesta. Algunos de estos desafíos están siendo abordados actualmente y otros se espera estudiarlos en trabajos futuros. Las líneas principales de trabajo son:

- Modificar la restricción de paso de tiempo fijo, esto permitiría mejorar los tiempos de ejecución al solapar en forma menos restringida.
- Abordar la aplicación de técnicas de dominios encajados.
- Evaluar el uso de plataformas de hardware híbridas, por ejemplo equipos que incluyan tarjetas gráficas (GPUs).
- Estudiar el impacto de la variente propuesta en el consumo eléctrico de la herramienta.

AGRADECIMIENTOS

R. Bayá agradece el financiamiento parcial del centro ICT4V (Information and Communication Technologies for Verticals) mediante la beca POS_ICT4V_2016_1 02. Además, los autores quieren agradecer al PEDECIBA (Programa de Desarrollo de las Ciencias Básicas) y a la ANII (Agencia Nacional de Investigación e Innovación), por la financiación recibida a través del proyecto FSE_2013_10975.

REFERENCIAS

Foster I. Designing and building parallel programs. 1995.

Gabriel E., Fagg G.E., Bosilca G., Angskun T., Dongarra J.J., Squyres J.M., Sahay V., Kambadur P., Barrett B., Lumsdaine A., et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. En *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, páginas 97–104. Springer, 2004.

Michalakes J., Chen S., Dudhia J., Hart L., Klemp J., Middlecoff J., y Skamarock W. Development of a next generation regional weather research and forecast model. En *Developments in Teracomputing: Proceedings of the Ninth ECMWF Workshop on the use of high performance computing in meteorology*, volumen 1, páginas 269–276. World Scientific, 2001.

Michalakes J. y Vachharajani M. Gpu acceleration of numerical weather prediction. *Parallel Processing Letters*, 18(04):531–548, 2008.

Padoin E.L., Pilla L.L., Boito F.Z., Kassick R.V., Velho P., y Navaux P.O. Evaluating application performance and energy consumption on hybrid cpu+ gpu architecture. *Cluster Computing*, 16(3):511–525, 2013.

Quinn M.J. *Parallel Programming*, volumen 526. TMH CSE, 2003.

Silva J.P., Hagopian J., Burdiat M., Dufrechou E., Pedemonte M., Gutiérrez A., Cazes G., y Ezzatti P. Another step to the full gpu implementation of the weather research and forecasting model. *The Journal of Supercomputing*, 70(2):746–755, 2014.

Skamarock W.C., Klemp J.B., y Dudhia J. Prototypes for the wrf (weather research and forecasting) model. En *Preprints, Ninth Conf. Mesoscale Processes, J11–J15, Amer. Meteorol. Soc., Fort Lauderdale, FL.* 2001.

Skamarock W.C., Klemp J.B., Dudhia J., Gill D.O., Barker D.M., Wang W., y Powers J.G. A description of the advanced research wrf version 2. Informe Técnico, DTIC Document, 2005.

Snir M. *MPI–the Complete Reference: The MPI core*, volumen 1. MIT press, 1998.

Apéndice 2

Enhancing the performance of the WRF model with an asynchronous computation architecture

Enhancing the performance of the WRF model with an asynchronous computation architecture

Rodrigo Bayá[b], Martín Pedemonte[b,*], Alejandro Gutiérrez[a], Pablo Ezzatti[b]

[a] *Instituto de Mecánica de los Fluidos e Ingeniería Ambiental, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, 11300, Montevideo, Uruguay.*
[b] *Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, 11300, Montevideo, Uruguay.*

Abstract

The Weather Research and Forecasting (WRF) is a numerical model for weather prediction and atmospheric simulation that is widely used by the climate and weather scientific communities. We have developed a software tool for forecasting the photovoltaic energy fed into the electrical grid in Uruguay that uses this model. The execution of the WRF is the most computationally expensive stage of the tool. In our context, the WRF have not offered an adequate scalability when large computational resources are available. This is caused because the calculation of the solar radiation, which is critical for the accuracy of the tool, is fully synchronous, i.e. the model is paused until the calculation of the radiation is completed. In this article, we propose a change of paradigm in the architecture of the WRF model that allows the asynchronous calculation of the radiation. The experimental evaluation conducted on real world data from twelve solar power plants shows that the new proposed architecture is able to make a better use of the

*Corresponding author
Email address: mpedemon@fing.edu.uy (Martín Pedemonte)

Preprint submitted to Journal of Parallel and Distributed Computing September 15, 2017

hardware resources available on the computing platform, reducing the total runtime of WRF while maintaining the numerical accuracy.
Keywords: Weather forecasting, Solar energy, Multicore hardware, Pipeline parallelism

1. Introduction

The use of renewable energy in Uruguay has changed dramatically over the last ten years, as a result of government programs to encourage the construction of large-scale wind and photovoltaic cell farms. Wind power has evolved from being marginally used in domestic scale to producing more than 35% of the total electric power generated nowadays [1]. The inclusion of solar photovoltaic energy in the electric power generation matrix is more recent and currently represents a mere 2%. However, the opening of several farms is expected for the next year, doubling the production and increasing the share of this energy source up to 5% (229 MW out of 4220 MW) [2, 3].

The growth in the use of photovoltaic energy and the nature of this type of energy have motivated the development of a software tool to forecast the electrical power injected into the grid from this source [3, 4], assisting in the decision making process associated with the management of the electrical grid. The different stages of this tool involve the acquisition of public global meteorological forecasts, the execution of the Weather Research and Forecasting (WRF) model to generate high resolution forecasts of the solar radiation on the surface of the region of interest, the adjustment of the forecasts using Model Output Statistics (MOS), and finally, the prediction of the photovoltaic energy fed into the electrical grid. The execution of the WRF

represents the most computationally expensive of the whole forecasting tool. Additionally, we are currently evaluating the use of WRF ensembles, i.e., the execution of different model instances with some perturbations on the initial conditions, in order to produce a more accurate forecast and estimate the uncertainty of the predicted values.

The calculation of the solar radiation, especially on the surface of some region of interest, plays a critical role for producing accurate forecasts of the photovoltaic energy produced. This computation is fully synchronous due to the software architecture of the WRF, i.e., the rest of the model is paused until the calculation of the radiation is completed. As a consequence, the computational cost of the radiation routine has a direct impact in the total runtime of the whole model. Since the routine is computationally expensive, the WRF is usually executed limiting the frequency of the computation of the radiation at the cost of losing accuracy in the forecast of the energy produced.

Considering the aspects described above, a reasonable option is to use High Performance Computing (HPC) techniques in order to accelerate the computation of the radiation. Computer servers in modern HPC infrastructures consist of general-purpose multicore processors (CPUs) with several cores, usually between 12 and 64, and accelerators like Graphics Processing Units (GPUs) and Intel Xeon Phi processors. However, the straightforward application of HPC techniques to the WRF simulation in our context does not show linear scalability with the number of processing units employed and the use of accelerators is limited by data transfer overheads [5, 6].

In this article, we follow a different approach that consists in modify-

ing the computational paradigm used for the calculation of the radiation in the WRF model. In particular, we design, implement and evaluate an asynchronous processing strategy for running the radiation module following the pipeline parallel paradigm [7, 8]. The architecture proposed decouples the radiation module from the main model, which allows overlapping the computations, exploiting task level parallelism. The experimental evaluation conducted shows that the proposal is able to make a better use of the computational power offered by modern multicore hardware platforms while keeping the numerical accuracy at the same time.

The rest of the paper is structured as follows. In Section 2, we briefly review the WRF model and how it is used for solar energy forecasting. Section 3 introduces the design and four different implementations of the newly proposed architecture for the WRF model. Next, we experimentally evaluate our proposal in Section 4. Finally, we outline the conclusions of this work and discuss future research directions in Section 5.

2. Weather Research and Forecasting (WRF) Model

The tool for solar photovoltaic energy forecasting has a four-stage workflow [9]. First, it obtains public global meteorological forecasts generated by the NOAA/National Weather Service [10] using the general circulation model of the atmosphere known as Global Forecast System. These forecasts have a grid resolution of $1^\circ \times 1^\circ$, approximately 100km×100km. Then, the WRF model executes, forecasting the solar radiation in the whole country, using the data from the first stage and some initial atmosphere conditions as input variables. In this work, we use a 30km×30km grid that comprises the

whole Uruguayan soil and some border territories of Argentina and Brazil. This grid was previously used in [11]. After that, in the third stage, the results obtained are adjusted using the MOS technique in the region of interest. Finally, the forecast of the photovoltaic energy produced in the solar power plant is calculated using a numerical model of the panels of the plant employed to harvest this energy. More details on the last two stages can be found in [4].

The values forecasted by the tool have been verified with real measured data for a period of one year, confirming that the tool is able to produce highly accurate results [3]. However, the runtime required for executing the tool is an important issue. In particular, the critical stage regarding the runtime is the execution of the WRF model simulation. For this reason, this stage is the target of this article.

2.1. The WRF

The WRF is a numerical model for weather prediction and atmospheric simulation, used for both research and operational applications [12]. It is Eulerian (it uses a fixed coordinate system with respect to the earth), non-hydrostatic (it includes explicit equations to calculate the pressure and the gravitational force on the vertical axis) and compressible (it considers density variations suffered by the various fluids involved). From the two simulation cores available in the WRF, the forecasting tool uses the Advanced Research WRF (ARW) core. The ARW core was primarily designed for research purposes but it is also used for day-to-day weather prediction. The code is written in Fortran and C, and it currently has about half a million lines of code.

The WRF can be executed in parallel using both shared memory and distributed memory approaches, partitioning the domain into rectangular patches in order to exploit the available data parallelism, for example, by assigning each domain to different processors using MPI [13]. This patches are also subdivided into tiles that can be processed by different threads using the OpenMP API [14, 15]. Since tiles usually need data from neighboring tiles, in the border of the patches it is necessary to propagate changes between them.

2.2. Radiation Computation in WRF

The radiation module is responsible of the numerical calculations associated with the transmission of radiation in and out of the atmosphere. Short wave and long wave radiation are computed separately as a result of the physical behavior of the radiation, which is shown in Figure 1.

Radiation is the driver of atmospheric circulations; it passes through the atmosphere and reaches the Earth's surface in amounts that are unequally distributed in space and time. This unequal energy distribution, due in part to the Earth's spherical shape, produces horizontal temperature gradients that produce atmospheric motions.

The most important process responsible for energy transfer in the atmosphere is electromagnetic radiation, which travels in waves at the speed of light [17]. The sun emits short wave radiation that is absorbed or reflected by the Earth and clouds. The absorption of solar energy produces heat, which makes the Earth and clouds emit long wave radiation as infrared light.

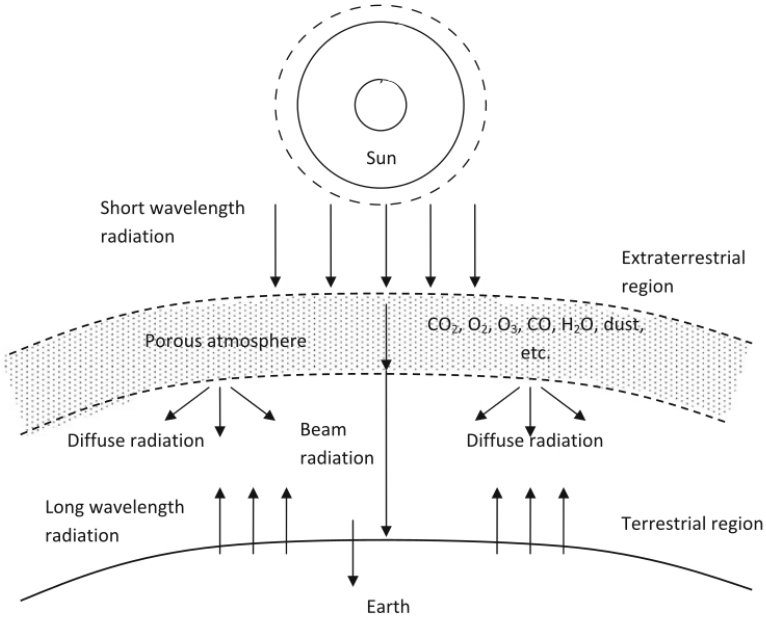


Figure 1: Behavior of the short and long wave radiation over the atmosphere. Taken from [16].

Clouds are extremely relevant in numerical simulations of the atmosphere [18], as the presence of clouds is associated with unstable conditions and factors that produce such conditions. Radiation parameterizations provide a fast and accurate method of determining the total radiative flux at any given location.

Several physics parameterizations model can be used to calculate the short and long wave radiations regarding different requirements [19]. In particular, the WRF model includes the following long wave radiation schemes: Rapid Radiative Transfer Model (RRTM) [20], Improved RRTM (RRTMG), Geophysical Fluid Dynamics Laboratory (GFDL) [21] [22], NCAR Commu-

nity Atmospheric Model (CAM) [23], New Goddard, and Held-Suarez. It also includes the following short wave radiation schemes: Dudhia, Goddard, New Goddard, RRTMG [24] [25], GFDL [26], and CAM [23]. The forecasting tool has to be executed in an affordable runtime which imposes a tight schedule for the WRF execution. For this reason, the Dudhia short wave scheme and the RRTM long wave scheme are used, since they are the less computationally expensive routines.

The radiation module is part of the physics driver, which contains every module associated with physics computation, such as microphysics, cumulus cloud parametrization, surface layer, etc. The outputs of the radiation module are inputs of the global model and other physics modules. In the ideal case, the radiation is calculated on each step of the WRF solver with the aim of providing the most accurate data possible. However, and due to the high computational cost implied by each radiation computation, the WRF implements the strategy that is described next to reach adequate results in acceptable execution times.

At the beginning of the execution of the WRF model, the first radiation step is calculated, storing the results in the associated grid variables that are used in the iterative procedure. Then, each solver step uses the last calculated radiation value for the rest of the computation. The number of solver steps that are computed between two radiation calculations can be configured by the user (m in Figure 2). This interval is expressed in minutes of simulation, and it usually set to one minute per square kilometer of the grid. As stated previously, a reduction in the interval allows reaching more accurate results [11], but leads to longer execution times. Figure 2 presents

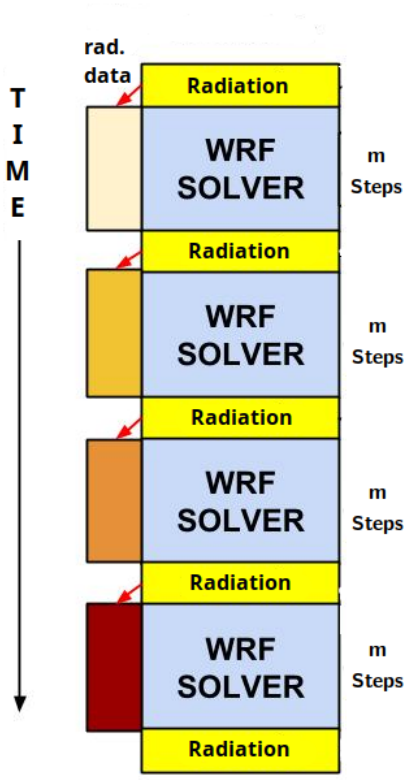


Figure 2: Execution of the radiation module and the remaining modules.

the behavior of the original WRF for computing the radiation values (in the figure the radiation module is executed four times). This approach is only acceptable when using up-to-date radiation values is not critical. As a consequence, an alternative computation strategy is required when more accurate radiation values on the surface are needed and the time budget for the execution of the WRF is short.

2.3. Evaluation of the computational performance of the radiation module

In this section, we evaluate the effect of using different invocation frequencies of the radiation module on the performance of the WRF model. Additionally, we study the runtime and the scalability of the whole model and the radiation module when they are executed using different number of threads.

2.3.1. Experimental settings

The test instance used in this section is a real world scenario from twelve solar power plants located on the Uruguayan territory. The instance uses a grid with $50 \times 61 \times 30$ discretization points and a simulation step of three minutes. It is a three-day simulation from May 15th to May 17th and WRF is configured to use the Dudhia short wave scheme and the RRTM long wave scheme.

The source codes of the WRF (version 3.5.1) were compiled using the Intel *ifort* 17.0.4 compiler with the openmp option. The execution platform is a computer server with four AMD Opteron 6128 (each one with 12 cores at 2.0 GHz.) with 16 GB RAM using the CentOS Linux 6.2 operating system.

2.3.2. Experimental results

Table 1 presents the execution time of the radiation module and the whole WRF simulation, as well as the scalability of the whole model when considering an increasing number of threads. The WRF uses a radiation step of six minutes, or equivalently a frequency of $\frac{1}{2}$.

The results obtained show that runtime of the whole WRF model decreases when a larger number of threads is considered, reaching the best

Table 1: Runtime (in seconds) of the original WRF model with different number of threads.

# threads	*Radiation Time*	*Total Time*	*Scalability*
1	573.43	1548.73	–
4	158.89	567.83	0.68
6	111.18	490.69	0.53
8	82.88	425.07	0.46
12	63.07	390.49	0.33
24	42.89	395.77	0.16

Table 2: Runtime (in seconds) and percentage of the total model runtime consumed by the radiation module for different radiation frequencies.

Radiation step	*Radiation Time*	*Total Time*	*Radiation %*
3	127.43	458.82	27.77
6	63.07	390.49	16.11
12	31.64	362.23	8.73
30	12.68	339.80	3.73

results at twelve threads. However, the speedup value reaches only 2.7× when the WRF executes with four threads. The results also show that the WRF does not offer adequate scalability values for the shared memory parallelization, e.g., the scalability is only 0.68 for four threads.

Table 2 shows the execution time (in seconds) of the whole WRF model and the radiation module using OpenMP with twelve threads (the configuration with the best execution time in the previous experiment) for several configurations of the radiation time step (in minutes).

The obtained results show that the longer the radiation time step the shorter the proportion of runtime required by the radiation. As a matter of fact, when the radiation time step is 30 minutes, the percentage of the total

runtime of the WRF that is consumed by the radiation routine is only almost 4%. However, when a higher frequency is used, the percentage rises up to almost 28%.

3. Our Proposal

This section describes the design and implementation of our new proposal, aiming to provide an asynchronous software architecture for the calculation of the radiation in the WRF model.

3.1. A novel software architecture for the WRF

Due to the requirements of the forecasting tool, our goal is to increase the radiation module execution frequency without affecting drastically the execution time of the whole model or, conversely, to reduce the total runtime of the WRF performing the same number of radiation calculations. As it was previously stated, the WRF runtime increases almost linearly with the radiation frequency in the current WRF architecture because of the synchronism between the radiation module and the rest of the WRF. For this reason, we propose a new architecture that decouples the radiation computation from the rest of the model, exploiting task parallelism through the asynchronous execution of the radiation module. In this way, when a large multicore platform or an additional computational unit (e.g., hardware accelerators) is available, the WRF execution can benefit from a better use of these extra hardware resources.

In the original WRF architecture, the execution of the solver and the radiation routine is sequential and synchronous. In certain steps of the model

execution (according to the number of solver steps between radiation calculations described in Section 2.2), the model invokes the radiation module and waits until the module finishes its execution to obtain the results. The outcome returned by the module is used by the model for several iterations until another step that has to invoke the radiation module is executed.

The proposed architecture is based in the concurrent execution of the radiation module and the WRF solver following a pipeline parallelism pattern. The interaction between the global model (as we will call the WRF excluding the radiation module from now on) and the radiation module is separated into two different stages. In the first stage, the model launches the radiation routine, including the passage of data necessary to perform the calculations from the global model to the routine. In the second stage, after the radiation is calculated, the global model reads the results from the radiation module.

The interaction between the global model and the radiation module can be implemented following two different alternatives. The first option is to use a fixed frequency for the radiation module invocation and the reading of the results. In this approach, both frequencies have to be experimentally determined in order to decide in which iterations the global model must send data to the radiation module and in which ones it must read the radiation results. Since the invocation frequencies have to be manually adjusted, this approach has the disadvantage of requiring a readjustment for each different execution platform. The second alternative consist in following a *best-effort* computing strategy through the use of application-specific strategies, in which the computations are not guaranteed to be executed. This idea has already been explored in [27]. The main disadvantage of this approach is the impossibility

to define a maximum value for the numerical error (which is related to the frequency of the radiation computation). For this reason, we adopted the first approach in this work. In the future, an hybrid alternative has to be evaluated, in which the radiation module uses a *best-effort* strategy while a certain level of accuracy is guaranteed (e.g., setting a limit for the number of the global model iterations between two radiation runs).

In order to better explain the proposed architecture, we focus in one particular instantiation of our idea that consists in computing the radiation calculations concurrently with two global model steps. However, we want to remark that the proposed idea is more general and it can be used with other configurations.

As we stated previously, the new proposal conceptually works as a pipeline in which the main model and the radiation computation are overlapped using different computation units (computing lines). One computing line of the pipeline is used for the global model and an additional computing line is devoted to calculate the radiation. Thus, after an initial stage where one radiation and one global model steps are computed, the procedure, depicted in Figure 3, is the following:

1. The global model transfers data to the radiation module.
2. (a) The general model computes the steps k and $k+1$, using the last available computed radiation values.
   (b) The radiation module computes a future radiation step, with the global model data from the step $k-1$.
3. The global model reads the computed values from the radiation module.

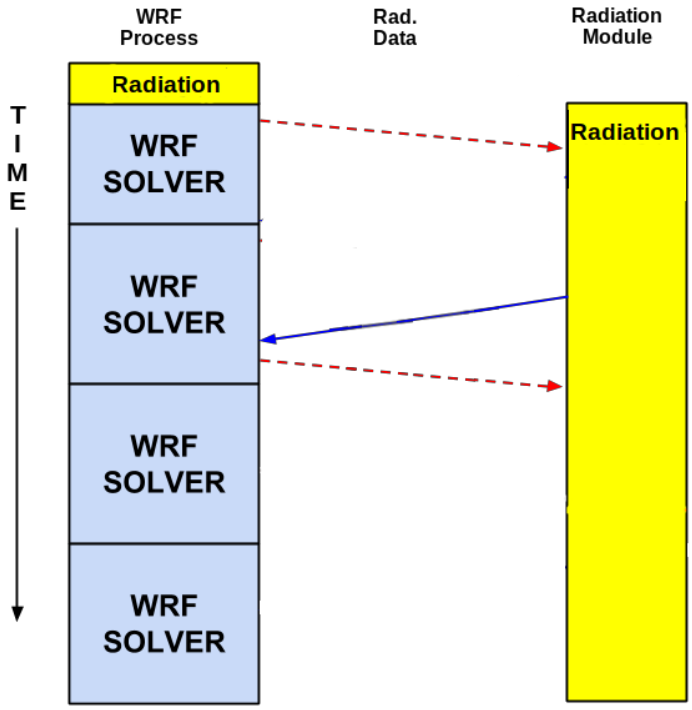


Figure 3: Interaction between the global model and the radiation module

Since the global model and the radiation module have disjoint storage, it is critical to keep data consistent and control its modification through a communication–synchronization procedure. Algorithm 1 presents the pseudocode of the communication procedure used by the radiation module. As it can be observed, the first step of the loop (in Step 3) waits until there is information available to compute the radiation module. After receiving the data, the module calculates the radiation for the current step. After this computation is completed, the routine copies the results to a buffer and notifies the global model that the radiation has already been calculated (in Step 7), i.e., that the module is available to compute a new radiation calcu-

lation. This acknowledgement procedure is implemented using two boolean variables, *send_available* and *receive_available*, that are shared between both processes. The whole process is repeated in a loop as many times as needed, ending when the whole atmosphere simulation is over.

```
1 Procedure Radiation_processor()
2     while TRUE do
3         wait(send_available)
4         receive_data(data_to_process)
5         process_data(data_to_process, result)
6         send_result(result)
7         send_available = FALSE
8         receive_available = TRUE
9     end
```

Algorithm 1: Communication procedure in the radiation module.

Algorithm 2 presents the pseudocode of the communication procedure used by the global model side. The global model checks whether the radiation is idle or not in Step 4 and, in the first case, sends the necessary data and notifies the radiation module through the *send_available* variable. Afterwards, the global model checks if there are new radiation values available by querying the *receive_available* variable, updating the radiation results and setting the *receive_available* variable in that case.

Following the communication procedure described before, the time to compute the radiation for the global model is reduced to the time associated with sending the data to the radiation module and reading the updated

```
1  Initialization
2      send_available = FALSE
3  Procedure Radiation_handler(data_to_process, result)
4      if NOT send_available then
5          send_data(data_to_process)
6          send_available = TRUE
7      end
8      if receive_available then
9          result = receive_result()
10         receive_available = FALSE
11     end
```

Algorithm 2: Communication procedure in the global model.

radiation values from it. Furthermore, while the radiation module is computing the next step, the global model is executing the rest of the atmosphere simulation steps.

3.2. Implementation details

In the current work, four different variants of the proposed architecture were implemented. All variants are based on the WRF shared memory version, and could be easily extended to other versions. Next, we describe the aspects that are common to all variants.

The input and output data of the radiation model heavily depends on the radiation model used. In the WRF, the main radiation function (i.e., the function that invokes the radiation model configured) has parameters for every possible model, making some variables to be null or not allocated

in each call. The original code handles this problem using *optional* and *pointer* variables in Fortran, while our new implementation handles an array of booleans indicating which variable is used (i.e, sent and received during the execution) by the modules in the new architecture.

The implementation of the newly proposed architecture is based on the shared memory parallelism model. In particular, we use the *pthread* API that allows to divide a process load into several threads of execution that communicate through a shared memory address space. The API is used to create one main thread for the global model (which uses the original number of threads configured by OpenMP) and another one for the radiation module (which also uses OpenMP with a number of threads that can be configured with a new environment variable). In other words, a two-level nested thread parallelism is created, as it is shown in Figure 4.

The communication between the modules is achieved using the shared memory address space and mutual-exclusion. It uses two boolean variables to indicate when new radiation values can be read by the global model and when new data is available for the radiation module, following the procedure previously described.

Next, we detail the particularities of each one variant studied in this work.

3.2.1. Straightforward version, $Asyn_{Str}$

The first variant designed is a straightforward implementation of the procedure previously described. When the radiation module is idle, the global

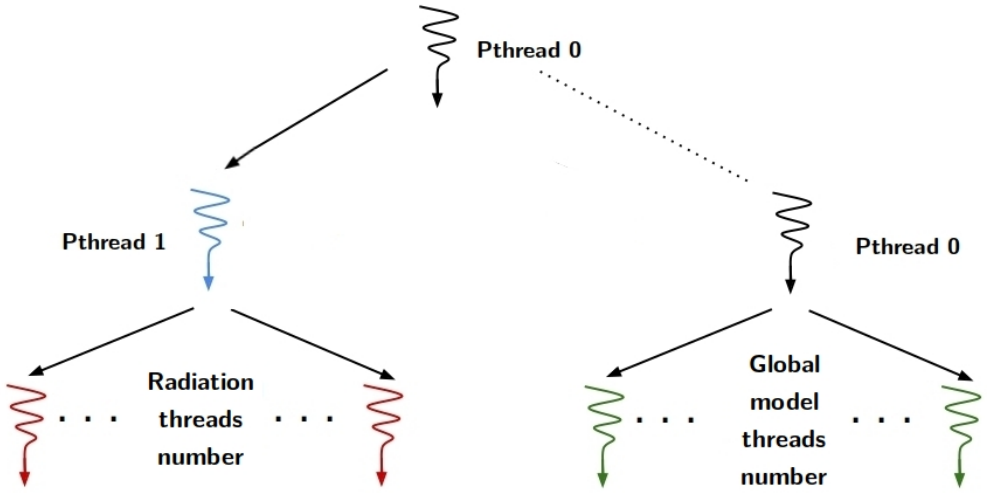


Figure 4: Threads hierarchy in the novel proposal.

model transfers the input data for the radiation. The radiation module performs the calculations for the current time step, while the output generated is used by the global model in a later time step of the simulation. Since the radiation values are not updated in the same simulation step as in the synchronous WRF, the forecasting values obtained with this implementation should deviate from the results obtained with the original WRF.

3.2.2. Adjust time version, $Asyn_{AdjT}$

In order to reduce the numerical difference between the results of the original WRF model and our proposal, we consider a second variant that adjusts the time-dependant variables. In this version, the idea is to compute the radiation values for a future time window, in which they will be used by the global model. The radiation module is able to compute the radiation

values for a future time changing the input parameter *xtime* (including the additional advance of time).

3.2.3. Update radiation version, $Asyn_{UpdV}$

Another issue of $Asyn_{Str}$ version is the impact that the rest of the model has over the variables of the radiation module. Since the global model sends data to the radiation module before it uses the new radiation values, this strongly impacts in the accuracy of the forecasted values. Considering this situation, we let the global model perform a simulation step to use the radiation values into the other methods and then, in the following step, the data needed for computing the next radiation step is sent.

3.2.4. Combined version $Asyn_{AdjT+UpdV}$

In the final version, we apply both improvements described in $Asyn_{AdjT}$ and in $Asyn_{UpdV}$ simultaneously. The goal of this version is to asses if the combination of both ideas can help to reduce even further the numerical gap between the results of the original WRF and our novel proposal.

4. Experimental Evaluation

The main objective of this section is to compare the different variants of our asynchronous scheme with the original version of WRF, in order to examine the benefits that can be obtained by changing the computing paradigm. As a first step we evaluate the numerical accuracy of the new proposed architecture, particularly, we compare the radiation results of the different variants with the values obtained by the original WRF implementation. Then, we study the execution time of our proposals and the original model, varying

the number of threads, with the goal of obtaining the best variant for each implementation and compare them. Finally, considering the results obtained, some possible extensions to this work are discussed.

4.1. Experimental settings

In the experimental evaluation, we use two different real world test instances over the same twelve solar power plants on the Uruguayan territory. Both test instances use a grid with $50 \times 61 \times 30$ discretization points and a simulation step of 3 minutes. The first instance is a full month simulation from May 15th to June 15th of 2016 that is used for assessing the numerical accuracy. The second instance is used for analyzing the computational performance and it is the the same three-day simulation already used in Section 2.3.1. Both instances use the same grid and the same modules than Section 2.3.1. The compilation process and the execution platform is exactly the same as the one described in Section 2.3.1.

4.2. Numerical validation

Since the novel architecture modifies the computation pattern of the radiation, we consider the original WRF model calculating the radiation in each step of the simulation, as a base line implementation for assessing the numerical accuracy of our proposal. It should be stated that the original version of the tool, which includes the original WRF model, has been previously validated with real values of the downward short wave flux at ground surface (SWDOWN) for a one-year period [3, 4].

To compare the radiation outputs of the different implementations and the base line implementation we use the Mean Absolute Percentage Error

Table 3: MAPE of the different configurations and implementations studied.

Version	***MAPE(%)***
WRF_{rad6}	6.6
WRF_{rad12}	23.1
WRF_{rad30}	31.5
$Asyn_{Str}$	47.2
$Asyn_{AdjT}$	7.4
$Asyn_{UpdV}$	22.7
$Asyn_{AdjT+UpdV}$	7.0

(MAPE) of the SWDOWN on twelve points of the grid (where the solar power plants are located). MAPE is computed as:

$$\frac{100}{n}\sum_{t=1}^{n}\left|\frac{BS_t - Rad_t}{BS_t}\right|, \tag{1}$$

where n is the number of temporal points evaluated, BS_t is a temporal point t of the base line result and Rad_t is a temporal point t of the evaluated version.

In order to understand if the measured error represents an important deviation from the radiation results, we consider the original version of the WRF with three different radiation steps. In particular, we study three variants WRF_{rad6}, WRF_{rad12} and WRF_{rad30} which use a radiation step of 6, 12 and 30 minutes, respectively.

Table 3 shows the error (measured with the MAPE metric) of the three versions of the WRF with different radiation step and the four implementations of the proposed architecture with respect to the baseline implementation. As it can be seen, $Asyn_{Str}$ and $Asyn_{UpdV}$ versions have rather high percentages of MAPE, greater than 45% and 20% respectively, while MAPE is less than 10% for $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ versions. Furthermore, the

percentage error of $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ are similar to the error of WRF$_{rad6}$ and far below the MAPE of WRF$_{rad12}$.

Figure 5 graphically shows the daily MAPE value for each one of the twelve locations of the solar power plants. WRF_{rad6}, WRF_{rad12}, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ are indicated in blue, red, yellow, and green, respectively. The daily MAPE of the monthly simulation corroborates that both variants, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ are able to compute simulations with a numerical accuracy comparable to the original WRF model. In particular, the error values of both versions are close to the error values of the WRF_{rad6} variant and they are inferior than the daily MAPE of the WRF_{rad12} variant in all the days of the simulation.

From these results, we conclude that $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ are numerically competitive with the original WRF. For this reason, we only use this two variants for further analysis.

4.3. Computational Performance Evaluation

For the computational performance evaluation of our proposal, we consider the WRF_{rad6} variant as a base line, since it is the variant of the original architecture that has numerical accuracy as similar as $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$Às it was shown in Section 2.3.2, the total runtime of WRF_{rad6} with the best thread configuration (12 threads) is 390.49 seconds, while the execution time involved in the radiation computation is 63.07 seconds. As a consequence, the theoretical maximum reduction in the execution time that can be obtained using the new architecture proposed is 16.1%.

To begin the study of the computational performance of our proposal, we evaluate the two new implementations $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ with

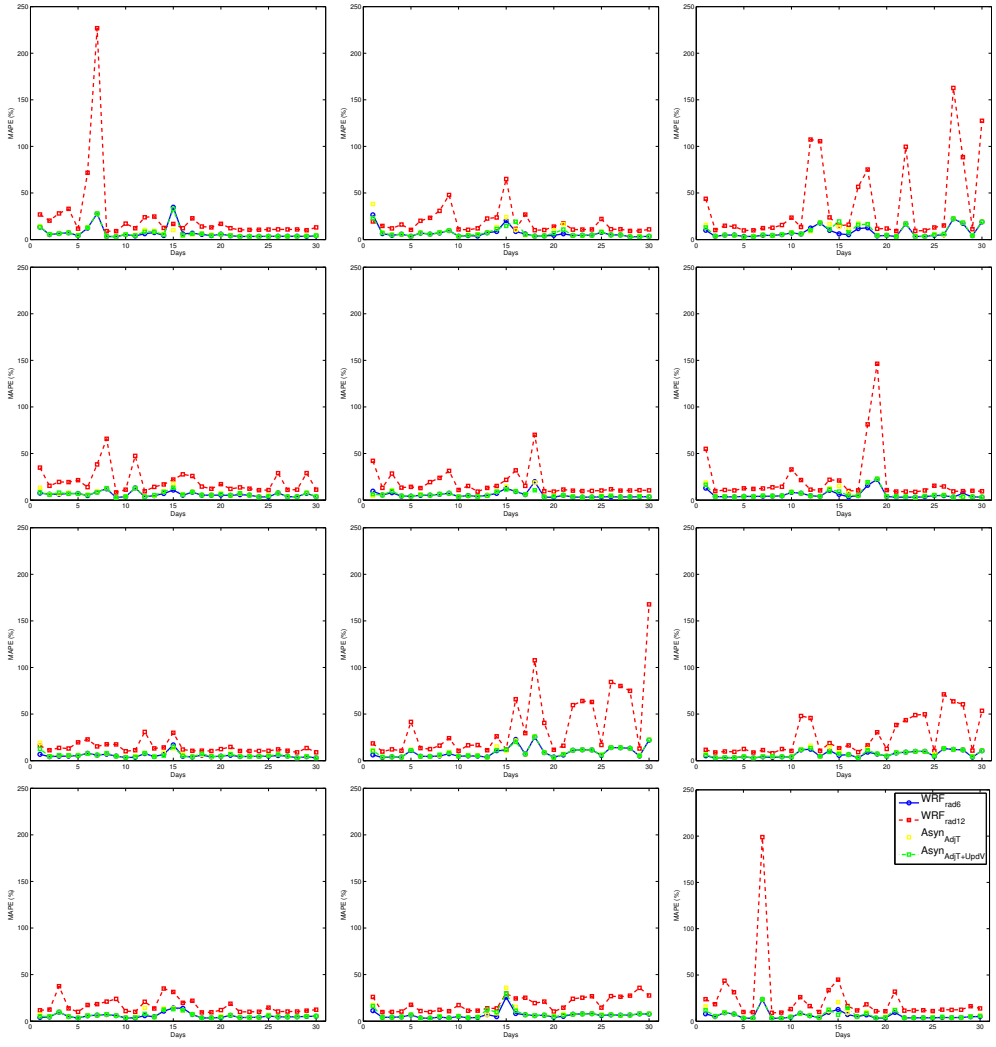


Figure 5: Graphical analysis of daily MAPE values for versions WRF_{rad6}, WRF_{rad12}, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ on each point of interest.

Table 4: Execution time (in seconds) of the studied versions using the same number of threads on the radiation module and the global model.

# threads	$Asyn_{AdjT}$					$Asyn_{AdjT+UpdV}$				
	Trans	***Sync***	***Rad***	***GM***	***Total***	***Trans***	***Sync***	***Rad***	***GM***	***Total***
6 - 6	6.59	0.001	126.21	361.40	370.14	6.86	0.26	134.29	366.89	376.05
8 - 8	6.85	0.001	100.74	344.90	353.87	6.52	0.001	110.11	358.63	367.37
12 - 12	6.52	0.001	78.50	339.95	347.68	6.52	0.001	96.57	357.60	366.09
24 - 24	6.90	0.001	67.47	371.81	381.11	6.55	0.001	96.54	393.96	401.73

different thread configurations. We consider four different configurations 6-6, 8-8, 12-12, and 24-24, where the first and the second number indicates the number of threads for the execution of the global model and the radiation module respectively. The obtained runtime are summarized in Table 4. The total runtime is discriminated in the time consumed by data transfer (*Trans*), synchronization (*Sync*), the radiation module execution (*Rad*) and the global model execution (*GM*). It should be noted that, since the radiation calculation and the global model are executing in parallel, the sum of the execution times of the individual stages discriminated is larger than the total runtime.

The best performing thread configuration for the $Asyn_{AdjT}$ version is twelve threads for each component, which is able to reduce in 42.81s the total runtime of the WRF. The reduction represents an improvement of 11.0% out of a maximum of 16.1%. The overhead associated with the transference of data between the radiation module and the global model is an inherent feature of this version, and it is practically impossible to overlap the data transference with the rest of the stages of the WRF, so that it does not impact in the total execution time. As a consequence, the real maximum achievable reduction is 14.5% (56.55s). Thus, the implementation of the $Asyn_{AdjT}$ version obtains 75.7% of the achievable improvement.

Table 5: Execution time (in seconds) of the studied versions using different number of threads on the radiation module and the global model.

# threads	$Asyn_{AdjT}$					$Asyn_{AdjT+UpdV}$				
	Trans	*Sync*	*Rad*	*GM*	*Total*	*Trans*	*Sync*	*Rad*	*GM*	*Total*
12 - 4	6.40	0.09	209.05	347.84	355.86	6.88	29.56	156.27	341.36	380.26
12 - 6	6.34	0.02	143.79	340.87	349.20	6.51	0.22	149.59	344.75	352.92
12 - 8	6.49	0.001	137.02	345.36	354.16	6.556	0.09	152.48	349.48	358.24

On the other hand, the best performing thread configuration for the $Asyn_{AdjT+UpdV}$ version is with twelve threads for each component. Unlike the previous case, $Asyn_{AdjT+UpdV}$ is only able to reach moderates improvements (only up to 24s) over the original version. These values are really far from the theoretical improvement bound, since the runtime reduction of this variant is a mere 6.2%. Since the synchronization time is negligible, a reasonable hypothesis is that this situation is produced by a saturation of the memory bandwidth due to the increase in the computational workload. In other words, the radiation module and the global model in $Asyn_{AdjT+UpdV}$ are competing for the access to this resource, which affects the runtime of the global model, e.g. the global model requires 327.42s of execution in WRF_{rad6} with twelve threads but this execution time raises to 357.60s in the case of $Asyn_{AdjT+UpdV}$ also with twelve threads.

To corroborate our hypothesis, we study how the number of threads used for the radiation calculation affects the runtime of the global model (with a fixed number of twelve threads for the global model). Table 5 shows the runtime of $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ with the following thread configurations: 12-4, 12-6, and 12-8.

The best performing thread configuration for both versions is 12-6. While the execution time is almost the same that in the previous configuration

Table 6: Execution time (in seconds) of the studied versions using a fixed number of 12 thread for the whole model.

# threads	$Asyn_{AdjT}$					$Asyn_{AdjT+UpdV}$				
	Trans	***Sync***	***Rad***	***GM***	***Total***	***Trans***	***Sync***	***Rad***	***GM***	***Total***
6 - 6	6.59	0.001	126.21	361.40	369.53	6.86	0.26	134.29	366.89	376.12
8 - 4	6.39	0.071	198.46	354.87	363.22	7.46	20.04	190.97	352.40	380.86

evaluated (12-12) in the case of $Asyn_{AdjT}$ (349.20s vs 347.68s), the use of a smaller number of threads yields an important reduction on the runtime of $Asyn_{AdjT+UpdV}$ (from 366.09s to 352.92s). These results show that our proposal shifts the implementation from compute bounded to memory bounded.

Finally, taking into account that in the previous studies we have used the best thread configuration for each of the implementations, we analyze the performance of the two implementations of the new architecture limiting the computational horsepower to the best thread configuration used in the original WRF. Table 6 shows the runtime of $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ with the following thread configurations: 6-6 and 8-4 (totalling twelve threads). Even under these circumstances, the proposed architecture is able to reduce the total execution time of WRF. Specifically, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ can reduce the total runtime up to 7% and 4%, respectively.

The results obtained in the study of the computational performance show that the new architecture proposed for the WRF is able to make a better use of the computing platform than the original architecture. This makes the idea proposed in this article an interesting alternative, specially when the whole model does not present an adequate scalability.

Table 7: Execution time (in seconds) of different radiation modules and the whole WRF.

Long wave	***Short wave***	***Rad***	***Total***
RRTM	*Dudhia*	63.07	390.49
RRTMG	*Dudhia*	209.24	541.92
New Goddard	*Dudhia*	75.12	406.86
Held-Suarez	*Dudhia*	3.58	328.87
GFDL	*Dudhia*	42.96	365.98
RRTM	*Goddard*	157.51	483.81
RRTM	*RRTMG*	237.80	565.52
RRTM	*New Goddard*	153.57	485.11
RRTM	*GFDL*	101.86	426.57

4.4. Future extensions

The forecasting tool uses the Dudhia short wave scheme and the RRTM long wave scheme for the radiation calculation. However, this choice has been mostly based on the reduced computational cost associated with these modules. There are other radiation modules that can yield better quality results but may involve execution times that could compromise the overall performance of the tool. The new software architecture allows us to widen the set of radiation modules available in the forecasting tool. For this reason, we evaluate the runtime of other radiation modules and how they affect the overall execution time of the WRF, these results are presented in Table 7. The table includes the runtimes of the current configuration in the first row as a base line. The rest of the rows include configurations that only change one of the radiation modules.

The results presented in Table 7 show that the radiation modules used by the forecasting tool are among the computationally cheapest alternatives. In particular, all the other short wave radiation modules and two out of the four

long wave radiation modules considered involve larger execution times for the radiation and for the whole model. As a consequence, the idea proposed in this article could offer larger benefits when other radiation modules are used in the WRF.

Another possible application of the idea proposed in this article is to unload computations to other computing platforms. Since the new software architecture offers coarse grain parallelism by uncoupling large sections of the computations, it is possible to exploit the use of distributed hardware platforms (more than one computing node), as well as the computational power of accelerators (like GPUs or Intel Xeon Phi). Several previous efforts [5, 6, 28, 29, 30] have shown that a considerable acceleration can be obtained when specific modules of the WRF are executed on massively parallel computing platforms. However, the effect on the runtime of the whole model is in general negligible due to the time required by data transfers between the CPUs and the devices, and the synchronism between the execution of the global model and the modules ported to the accelerators. In the novel architecture, the asynchronous execution allows to overlap the two computations, while the time involved in data transfers can be partially hidden.

5. Concluding Remarks and Future Work

In this work, we have addressed the acceleration of the WRF model. In particular, we have studied the modules related to the radiation calculation, and we have proposed a novel asynchronous computation software architecture for the WRF model, in order to reduce its execution time. Four variants that follow this computing paradigm and different data management ap-

proaches have been designed, implemented and evaluated in two real world scenarios.

From the results of the experimental evaluation, we can conclude that two implementations of the newly proposed architecture, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ have an acceptable numerical accuracy. In particular, these implementations have around 7% of MAPE when compared with the original WRF model computing the radiation on each simulation step. Additionally, this value is similar to the MAPE obtained by the original WRF model with a radiation step of 6 minutes.

Regarding the computational performance of our proposal, $Asyn_{AdjT}$ is able to reduce the runtime in almost 11% (43 out of 390 seconds) over the best thread configuration of the original WRF, when the same number of threads is used for both, the global model and the radiation module. If a different number of threads is allowed for each computation, $Asyn_{AdjT}$ does not improve the computational performance but $Asyn_{AdjT+UpdV}$ achieves a reduction of almost 38 seconds over the original version of the WRF. In summary, the best thread configuration for $Asyn_{AdjT}$ is 12-12 threads and for $Asyn_{AdjT+UpdV}$ is 12-6 threads, which obtain a 75.7% and a 66.4% of the theoretical achievable improvement, respectively. Finally, and in order to make the comparison as fair as possible, we have also evaluated the computational performance of the different versions using a fixed number of threads with the best thread configuration of the original WRF. In this scenario, $Asyn_{AdjT}$ and $Asyn_{AdjT+UpdV}$ outperform the original WRF, reducing the runtime 7% and 4%, respectively. From these results, we can conclude that our proposal exploits the hardware platform in a better way than the original WRF model.

Along this work, we have identified three main areas that deserve further study. The first issue is to extend our work by evaluating the use of other radiation modules considering the balance between the quality of the results and the computational performance. The second line of interest consist in evaluating and adjusting our implementations in order to exploit hardware platforms that include either distributed nodes or accelerations devices. Finally, and considering that the results obtained are promising, we plan to address the ensemble of models, which involves several executions of the WRF model with different initial conditions, with the aim of estimating the uncertainty of the forecasted values.

Acknowledgment

We would like to thank Ernesto Dufrechou for his helpful comments and suggestions. We also would like to thank Jose Aguerre, who contributed on an early stage of this work.

The work presented in this paper has been partially carried out at ICT4V (Information and Communication Technologies for Verticals - www.ict4v.org) and has been funded by ICT4V through the grant POS-ICT4V-2016-1-02. The authors acknowledge support from PEDECIBA Informática, ANII, and SNI.

[1] Dirección Nacional de Energía - Ministerio de Industria, Energía y Minería, Programa de Energía Eólica en Uruguay (in Spanish), `http://www.energiaeolica.gub.uy/`, [Online; accessed 14-September-2017].

[2] Dirección Nacional de Energía - Ministerio de Industria, Energía y Minería (in Spanish), http://www.dne.gub.uy/, [Online; accessed 14-September-2017].

[3] C. Porrini, A. Gutiérrez, G. Cazes, E. Dufrechou, M. Pedemonte, P. Ezzatti, Leveraging HPC Techniques to Develop a Prediction Tool for Photovoltaic Solar Energy in Uruguay, in: 2nd Frontiers in Computational Physics Conference: Energy Sciences, 3-5 June 2015, Zurich, Switzerland, 2015.

[4] C. Porrini, A. Gutiérrez, G. Cazes, G. Hermida, D. Oroño, M. Puppo, Development of a Model Output Statistic and implementation of an operational solar photovoltaic energy forecast model based in WRF, in: 2015 IEEE PES Innovative Smart Grid Technologies Latin America (ISGT LATAM), 2015, pp. 248–253.

[5] J. Michalakes, M. Vachharajani, GPU acceleration of numerical weather prediction, Parallel Processing Letters 18 (04) (2008) 531–548.

[6] J. P. Silva, J. Hagopian, M. Burdiat, E. Dufrechou, M. Pedemonte, A. Gutiérrez, G. Cazes, P. Ezzatti, Another step to the full GPU implementation of the weather research and forecasting model, The Journal of Supercomputing 70 (2) (2014) 746–755.

[7] T. Mattson, B. Sanders, B. Massingill, Patterns for Parallel Programming, 1st Edition, Addison-Wesley Professional, 2004.

[8] S. Benkner, E. Bajrovic, E. Marth, M. Sandrieser, R. Namyst, S. Thibault, High-level support for pipeline parallelism on many-core ar-

chitectures, in: C. Kaklamanis, T. Papatheodorou, P. G. Spirakis (Eds.), Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings, 2012, pp. 614–625.

[9] E. Cornalino, A. Gutiérrez, G. Cazes, M. Draper, R. Chaer, A valorization of the short-term forecasting of wind power, in: Proceedings of the 2012 6th IEEE/PES Transmission and Distribution: Latin America Conference and Exposition, T and D-LA 2012, 2012, pp. 1–5.

[10] NOAA/National Weather Service, `http://www.nco.ncep.noaa.gov/`, [Online; accessed 14-September-2017].

[11] A. Gutiérrez, G. Cazes, S. De Mello, Analysis of the optimal grid resolution for the forecasting of wind energy in different wind farms, in: Conference Record - IEEE Instrumentation and Measurement Technology Conference, 2014, pp. 1512–1516.

[12] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, W. Wang, The Weather Research and Forecast Model: Software architecture and performance, in: 11th ECMWF Workshop on HPC in Meteorology, 2004, pp. 156–168.

[13] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, 3rd edition, MIT Press, 2014.

[14] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon,

Parallel Programming in OpenMP, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[15] B. Chapman, G. Jost, R. van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), The MIT Press, 2007.

[16] G. N. Tiwari, A. Tiwari, Shyam, Handbook of Solar Energy: Theory, Analysis and Applications, Springer, 2016.

[17] K. N. Liou, An Introduction to Atmospheric Radiation, Academic Press, 2002.

[18] W. R. Cotton, Storm and Cloud Dynamics, Academic Press, 1989.

[19] J. Dudhia, WRF physics options, NCAR WRF basic tutorial (2010) 26–30.

[20] E. J. Mlawer, S. J. Taubman, P. D. Brown, M. J. Iacono, S. A. Clough, Radiative transfer for inhomogeneous atmosphere: RRTM, a validated correlated-k model for the long-wave, J. Geophys. Res. 102 (D14) (1997) 16663–16682.

[21] S. B. Fels, M. D. Schwarzkopf, The simplified exchange approximation: A new method for radiative transfer calculations, J. Atmos. Sci. 32 (1975) 1475–1488.

[22] M. D. Schwarzkopf, S. B. Fels, The simplified exchange method revisited an accurate, rapid method for computation of infrared cooling rates and fluxes., J. Geophys. Res. 96 (D5) (1991) 9075–9096.

[23] W. Collins, et al., Description of the NCAR Community Atmosphere Model (CAM 3.0), Tech. Rep. TN-464+STR, NCAR (2004).

[24] A. A. Lacis, J. E. Hansen, A parameterization for the absorption of solar radiation in the earth's atmosphere, Journal of the Atmospheric Sciences 31 (1) (1974) 118–133.

[25] G. L. Stephens, Radiation profiles in extended water clouds. Part II: Parameterization schemes, J. Atmos. Sci. 35 (1978) 2123–2132.

[26] T. Sasamori, J. London, D. V. Hoyt, Radiation budget of the southern hemisphere., Meteor. Monogr. 13 (35) (1972) 9–23.

[27] J. P. Aguerre, R. Bayá, Aceleración de una herramienta para la predicción de energía eléctrica de origen solar mediante arquitectura de hardware híbridas (in spanish), Tech. rep., INCO, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay (2015).

[28] J. Mielikainen, B. Huang, J. Wang, H. L. A. Huang, M. D. Goldberg, Compute unified device architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme, Computers & Geosciences 52 (2013) 292–299.

[29] J. Mielikainen, B. Huang, H. L. A. Huang, M. D. Goldberg, GPU Acceleration of the Updated Goddard Shortwave Radiation Scheme in the Weather Research and Forecasting (WRF) Model, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 5 (2) (2012) 555–562.

[30] J. Mielikainen, B. Huang, H.-L. Huang, M. Goldberg, A. Mehta, Speeding up the computation of WRF double-moment 6-class microphysics scheme with GPU, Journal of Atmospheric and Oceanic Technology 30 (12) (2013) 2896–2906.

Apéndice 3

Task parallelism in the WRF model through computation offloading to many-core devices

Task parallelism in the WRF model through computation offloading to many-core devices

Rodrigo Bayá, Claudio Porrini, Martín Pedemonte and Pablo Ezzatti
Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay
Emails: {rbaya,cporrini,mpedemon,pezzatti}@fing.edu.uy

***Abstract*—In the last decade the use of hybrid hardware (e.g., multicore processors + coprocessors) has been growing on the HPC field. However, this evolution in the HPC hardware has not been fully exploited by the WRF model since it shows limitations in the scalability when a large number of computing units are used. In a previous work, we proposed an asynchronous architecture for the WRF that overlaps the radiation computation with the execution of the rest of the model. In this work, we extend this idea with the aim of exploiting the computational power offered by hybrid hardware platforms. Specifically, we implement an OpenMP version of the asynchronous architecture and include the use of two types of coprocessors, a Xeon Phi and a GPU. The experimental evaluation performed shows that our proposal is able to adequately exploit these secondary computation devices, reaching interesting runtime reductions when solving tests cases from real scenarios.**

***Index Terms*—Solar radiation computation, GPUs, Xeon Phi.**

I. Introduction

In the last decade the use of hybrid hardware architectures, machines that involves multicore processors with a large number of cores (e.g. 16 and 32) and accelerators (e.g. Xeon Phi and GPUs), has been growing on the high performance computing (HPC). However, this evolution in the HPC hardware has not been fully exploited by software tools that show strong limitations in the scalability with the number of computing units. An example of an application that suffers from a poor value of scalability, when small domains are used, is the WRF model [1]. This deficiency on the use of parallel computational resources is a major drawback when high precision simulation values are required. In our context, where accurate solar radiation values are needed maintaining a relatively low runtime, this situation impacts negatively.

In [2] we have introduced a novel software architecture for the WRF following an asynchronous paradigm. More in detail, we have proposed a strategy where the radiation calculation is computed concurrently with the rest of the model, thus taking advantage of the computational power offered by multicore platforms. The new architecture was implemented using the *pthread* API and it was validated on a multicore processor with up to 48 cores. Several authors have proposed the use of GPUs for the WRF [3], but these efforts are focused on accelerating a single routine instead of the whole model.

In this paper, we extend and enhance our previous developments including the offloading of some radiation routines to many-core accelerators. In detail, the major contributions of the present work are: i) Presenting a new implementation of our software architecture for the WRF model based on the OpenMP API. ii) Developing Xeon Phi and GPU implementations for computing the radiation and analyzing its impact on the performance of the novel architecture of the WRF model. The experimental analysis performed shows that the combined use of the new asynchronous architecture with many-core hardware offers interesting benefits. Specifically, the implementations that offload the radiation calculations to secondary computing devices outperform the best configuration of the original WRF in 25–30%. Additionally, the implementations that are based on the asynchronous architecture are able to make a better profit of the accelerators than similar implementations that are based on the original WRF architecture.

II. Weather Research and Forecasting (WRF)

The WRF is a numerical model for weather prediction and atmospheric simulation [4]. The WRF can be executed in parallel using both shared and distributed memory approaches by partitioning the simulation domain, i.e. data parallelism. For example, the grid could be divided into patches to exploit a distributed environment by assigning each part to different processors using MPI. These could be also subdivided into tiles that can be processed by different threads using OpenMP.

In this work, we focus on the acceleration of the routines involved in the solar radiation calculation. The radiation is the driver of atmospheric circulations; it passes through the atmosphere and reaches the Earth's surface in amounts that are unequally distributed in space and time. This distribution, due in part to the Earth's spherical shape, produces horizontal temperature gradients that cause atmospheric motions. The sun emits short wave radiation that is absorbed or reflected by the Earth and clouds. The absorption of solar energy produces heat, which makes the Earth and clouds emit long wave radiation as infrared light. In the WRF model, the radiation module is responsible of the calculations associated with the transmission of radiation in and out of the atmosphere, computing short and long wave radiations separately. The model presents different schemes, with different precision and execution time. Since we have to execute the WRF in an affordable runtime, we consider the Dudhia short wave scheme [5] and the Rapid Radiative Transfer Model (RRTM) long wave scheme [6], which are two of the less computationally expensive routines.

In the ideal case, the radiation is calculated on every step of the WRF for obtaining the most accurate results. However, the WRF computes the radiation on a fixed simulation step and

uses these values for several subsequent steps to keep acceptable runtime. The steps that can be computed without updating the radiation values is expressed in minutes of simulation, and it is usually set to one minute per square kilometer of the grid. As it was shown in [7], reducing the interval between radiation computations makes possible to obtain more accurate results, but this approach is only feasible when the runtime of the whole WRF model is not critical. Consequently, a different approach has been devised for obtaining up-to-date radiation values when the execution time budget is limited [2].

A. An asynchronous architecture for the WRF

The runtime of the WRF model increases linearly with the frequency of the radiation computation due to the synchronism between the radiation module and the rest of the WRF. For this reason, an asynchronous architecture has been proposed in [2]. The new architecture decouples the radiation computation from the rest of the model, exploiting task parallelism. The idea follows a pipeline parallelism pattern [8], controlling the interaction between the radiation module and the rest of the WRF model. In particular, the frequency of this interaction is fixed by the user at the beginning of the simulation.

In order to give a better picture of the interaction between the radiation module and the WRF model, we describe one instantiation that concurrently computes the radiation with two WRF solver steps. In this particular scenario, the interaction between the two parts of the WRF has the following behavior:

1) The WRF solver transfers data to the radiation module.
2) a) The WRF solver computes the steps k and $k+1$, using the last available radiation values.
   b) The radiation module concurrently computes a future step, with data from the step $k-1$ of the WRF solver.
3) The solver reads the new values from the radiation module.

As the WRF and the radiation module have disjoint storage, it is critical to keep data consistent, controlling its modification using a communication–synchronization procedure. For this reason, two buffer arrays are used for storing the data to be transferred between the two components. Additionally, two boolean variables are used by the processes for controlling the access to the buffers. Algorithm 1 and 2 present the pseudocode of the communication procedure used by the radiation module and the rest of the WRF, respectively.

```
1  Procedure Radiation_calculation()
2      while TRUE do
3          wait(send_available)
4          data_to_process = receive_data()
5          result = process_data(data_to_process)
6          send_result(result)
7          send_available = FALSE
8          receive_available = TRUE
9      end
```

Algorithm 1: Communication in the radiation module.

As a consequence of these procedures, the computation time of the radiation in the WRF solver is just the time required for sending data to the radiation module and reading the updated values from that module. Moreover, while the radiation module is computing the next step, the rest of the WRF model is executing the remaining atmosphere simulation steps.

```
1  send_available = FALSE
2  Procedure Radiation_handler(data_to_process, result)
3      if Fixed_frequency then
4          wait(NOT send_available)
5          send_data(data_to_process)
6          send_available = TRUE
7      end
8      if Fixed_frequency then
9          wait(receive_available)
10         result = receive_result()
11         receive_available = FALSE
12     end
```

Algorithm 2: Communication in the WRF solver.

A property of the asynchronous architecture is that not only makes a better use of the resources available in multi-core environments, but also allows to offload the radiation computation to secondary many-core devices. Besides, this architecture helps to partially hide the time required for sending data to the coprocessor for computation due to the decoupling of the radiation module and the asynchronous execution. In this work, we extend this architecture to exploit the use of accelerators for computing the radiation module.

III. Our Proposal

In this section, we discuss the implementation devised to introduce a secondary many-core computing hardware to the asynchronous architecture. The goal of this proposal is to offload radiation computations from the CPU to the coprocessor, freeing up computing resources as well as memory bandwidth on the CPU that can potentially be used by the rest of the WRF. In particular, as it can be seen in Figure 1, we have instantiated a new implementation of the asynchronous architecture of the WRF and ported some critical procedures of the radiation module to GPU and Xeon Phi platforms.

We first implemented a variant of the asynchronous architecture using OpenMP. In particular, we use the `OMP SECTION` primitive to assign one thread to the radiation module and another thread to the rest of the WRF. Additionally, each thread uses OpenMP to execute its work, creating a nested thread hierarchy. In favor of executing these two regions with different number of threads, an additional dummy level of `OMP PARALLEL` was added to the radiation module section. The thread hierarchy can be seen in Figure 1. Regarding to the communication process, the implementation uses a shared memory address space to maintain the communication buffers and the two boolean variables to control the access to them.

In order to determine which routines port to the coprocessor, we identified the most computationally expensive procedures of the modules chosen. In this study, we have concluded that `rtrn`, `taugb3` and `taugb5` functions (all from RRTM) take 12% of the total WRF runtime. Moreover, the RRTM module not only takes a considerable portion of the total runtime but also requires a relatively low data volume for its execution.

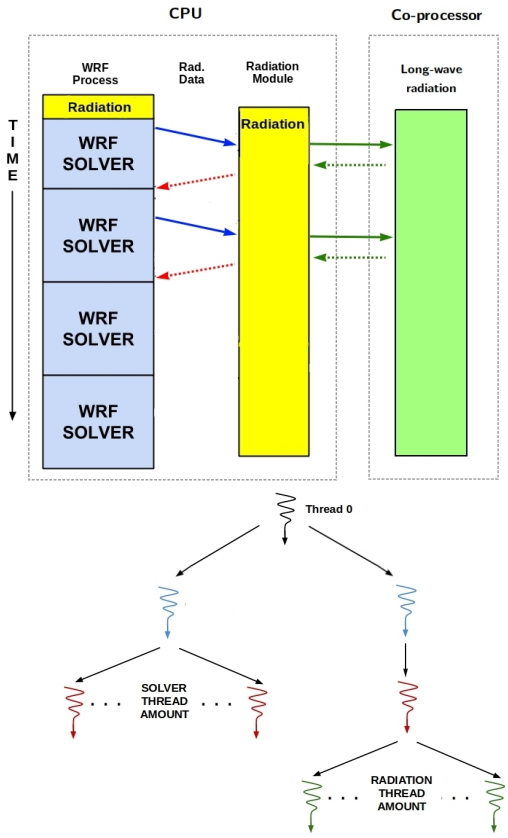


Figure 1: (Up) Interaction between modules on the extension of the asynchronous architecture. (Down) Thread hierarchy in the implementation of the asynchronous architecture.

An interesting characteristic of the RRTM module is how the radiation is calculated through the three-dimensional grid. In detail, the module extracts and scales every vertical column of the z-axis using a nested loop over the x-axis and the y-axis, processing each column independently. As a consequence, there might be data dependencies within each column computation but each column has no data dependencies with the rest of them, making this module easy to parallelize.

From this analysis, we have decided to port the whole RRTM module to both GPU and Xeon Phi devices, with the aim of taking profit from the massively parallel power available on this coprocessors. In the next two subsections, we describe the important aspects of both implementations.

A. Radiation calculation on a Xeon Phi coprocessor

Up to our knowledge the RRTM module has not been ported to Xeon Phi. Since Xeon Phi coprocessors natively support OpenMP and the WRF has a shared memory implementation that uses this API, we have taken the implementation of RRTM available on the WRF as a starting point for developing our own implementation. There are two aspects that have to be specially considered in the adjustment of the original code to Xeon Phi platforms: the data transference between the CPU and the coprocessor, and the modification of the grid distribution in order to work with a higher number of threads.

The Xeon Phi version is based on `rrtminit_phi` and `rrtmlwrad_phi` functions. The first initializes the RRTM module on the coprocessor, sending all the constant needed for the radiation calculation. The `rrtmlwrad_phi` routine is separated in two parts, one that executes on the CPU and one on the Xeon Phi. The CPU part redefines the tiles using two environment variables, `xtiles` and `ytiles`, which specify the size of the new tiles on the x- and y-axis, respectively. This routine also sends the data to be computed to the coprocessor and calls the routine `rrtmlwrad` on the device. The Xeon Phi side of the routine initializes `xtiles*ytiles` threads and sets the indexes for every thread, specifying which part of the grid has to be processed by each thread.

B. Radiation calculation on a GPU

The RRTM module on GPU used was taken from [9], where authors create two functions, `rrtminit_cuda` and `rrtmlwrad_cuda`, to manage the memory transference and launch the GPU kernels. The first function initializes lookup tables and constants values on the GPU. The other function transfers time dependant data and executes the long wave computation. In order to calculate the radiation values, the function is subdivided in several kernels, trying to exploit the x-and y-axis data independence, also taking advantage of z-axis one when the code allows it. The implementation not only uses the GPU power but also exploits the partially overlapping of data transference CPU – GPU with the GPU computation and the reduction in register pressure due to the use of multiple kernels. In addition, the authors transpose some matrices when they are transferred to the GPU to benefit from coalesced access as well as rearrange some fragments of the code in favor of reducing the thread divergence.

IV. Experimental Evaluation

This section reports on the experimental analysis conducted to evaluate the performance of the radiation calculation offloading to secondary computation devices.

A. Experimental settings

In our experiments we include two different hardware platforms: I) The Xeon Phi platform consists of 2 Intel Xeon E5-2650 with 8 cores at 2.00 Ghz., 64 GB of RAM and a Xeon Phi 31S1P coprocessor with 57 cores (each one supporting up to 4 threads) at 1.1 Ghz. and 8GB of internal memory. II) The GPU platform has an Intel Core i7 6700 with 4 cores at 3.40 Ghz., 64 GB of RAM and a Nvidia Titan X (Maxwell) with 3072 CUDA cores at 1.0 Ghz. and 12 GB of RAM.

On both experimental platforms, the version 3.5.1 of the WRF (ARW variant) was employed. The WRF was configured using the Dudhia and the RRTM schemes for the radiation calculation. As a simulation scenario, we use one real world case from 12 solar power plants located on Uruguay. The test scenario consists of a three-day simulation from May 15th to May 17th 2016. It uses a grid with $50\times61\times30$ discretization

points and a simulation step of three minutes. Two different configurations of the frequency for computing the radiation values were set, one for the Xeon Phi platform that compute the radiation every two simulation steps and another with one radiation on each simulation step for the GPU platform.

B. Experimental analysis

In this subsection we include three other implementations to set an actual comparison basis. These implementations are the original WRF model, a hybrid CPU-coprocessor version using the original WRF architecture but computing the radiation values in the coprocessor, and the CPU implementation of the asynchronous architecture using OpenMP.

1) Experimental results of the Xeon Phi-based implementation: In our first experiment we have executed the original synchronous WRF model to gain insight on the whole model and the radiation module, considering the runtime and the scalability when they are executed using different number of threads. Table I summarizes the execution time on CPU of the radiation module and the original WRF model using 1, 2 and 4 threads.

Table I: Runtime (in seconds) of the radiation module and the original WRF model (in CPU) over the Xeon Phi platform.

# Threads	*Radiation time*	*Total time*
1	303.68	744.42
2	156.09	393.97
4	82.59	217.13

The results show that there is a reduction on runtime of the radiation module and the whole WRF model when an increasing number of threads is considered. The best performance for the radiation routines and the model is attained when four threads are used. The results also show that the WRF and the radiation module are able to scale properly in this CPU platform. The scalability is up to 85% for the whole model and almost 92% for the routines with four threads.

Later, we evaluate a traditional hybrid CPU-Xeon Phi variant using the original WRF architecture and porting the radiation routines to the Xeon Phi device. Table II shows the execution time of this hybrid implementation when 1, 2 and 4 threads are used on the CPU. The results show that the runtime of the radiation routines is much shorter in the Xeon Phi than in the CPU variant with one thread. However, when a large number of threads is used in the CPU, the execution of the routines in the Xeon Phi is outperformed by the pure CPU implementation. For this reason, while it is possible to obtain reductions in the runtime of the whole model with the hybrid CPU-Xeon Phi implementation when a small number threads are used in the CPU, the original implementation of the WRF model has a shorter execution time than the hybrid implementation when a large number of threads are used (four threads in this case). In conclusion, the straightforward hybrid implementation is a worse alternative than the original shared memory variant, possibly leading to rule out the use of Xeon Phi coprocessors for accelerating radiation routines.

Our final experiment consists in evaluating the asynchronous architecture of the WRF. We have executed our

Table II: Runtime (in seconds) of original WRF model with the radiation module ported to the Xeon Phi.

# Threads	*Radiation time*	*Total time*
1	104.25	579.74
2	100.32	376.61
4	98.17	236.36

Table III: Runtime (in seconds) of the implementation of the asynchronous architecture of the WRF model.

# Threads	*Radiation*	*Transference*	*Blocking*	*Total*
Asynchronous architecture				
2-2	156.99	7.23	0.09	243.38
4-1	306.51	7.38	175.39	322.51
Asynchronous architecture + Xeon Phi				
2-2	101.12	7.65	0.56	247.31
4-1	100.02	7.70	0.71	154.38

implementation of the asynchronous architecture of the WRF model with and without the long wave radiation module ported on the Xeon Phi coprocessor. This experiment allows us to study the computational performance reached by the asynchronous architecture and the offloading of the radiation module to the coprocessor with different number of threads, considering both the radiation routines and the whole WRF model. Both implementations were evaluated using two different thread configurations on the CPU, one with two threads for the radiation driver and two threads for the WRF and another configuration with one thread for the radiation module and four threads for the rest of the model. The results of this experiment are shown on Table III. The table discriminates the time consumed by the radiation module, data transference, synchronization and blocking, and the total runtime of the whole execution. It should be noted that the sum of the individual stages is larger than the total runtime due to overlapping.

As it can be seen, the 2-2 configuration is the best performing alternative for the pure OpenMP implementation of the new architecture. However, the original WRF executed with four threads outperforms this implementation. On the other hand, the implementation of the new architecture with the radiation offloading to the Xeon Phi is able to produce an interesting reduction on the runtime of the WRF. In particular, the implementation with the 4-1 thread configuration outperforms the original WRF in a 28.9% (154.38 vs 217.13 seconds). It should be highlighted that the maximum theoretical reduction that this implementation can produce is 38.0% (82.59 of the total 217.13 seconds), and it would be obtained only if the radiation runtime is completely eliminated (hidden). Additionally, the asynchronous architecture involves data transference that cannot be avoided, limiting the possible reduction to just 34.5%.

From these results, it is notorious that offloading the radiation computations to the Xeon Phi in the novel architecture makes a better use of the accelerator than the hybrid CPU-Xeon Phi implementation based on the original WRF.

2) Experimental results of the GPU-based implementation: For the experimental evaluation of the GPU implementation, we have carried out similar experiments to the ones performed for the Xeon Phi architecture. We present the results obtained

in the same order than in the previous subsection.

Table IV reports the runtime of the radiation module and the WRF model with the synchronous architecture when the number of threads varies between 1, 2 and 4. The performance results obtained are slightly worse than the ones in the previous platform, showing a reasonable scalability of almost 63% for the whole simulation on the four thread case, and even a better value of up to 85% for the radiation module.

Table IV: Runtime (in seconds) of the radiation module and the original WRF model (in CPU) over the GPU platform.

# Threads	*Radiation time*	*Total time*
1	189.11	455.58
2	102.50	282.86
4	55.44	181.87

Later, we consider a hybrid CPU-GPU version using the original WRF architecture but with the radiation module executing in the GPU. Table V presents the runtimes of this hybrid implementation for 1, 2 and 4 threads on the CPU. As in the Xeon Phi case, the straightforward hybrid implementation reduces the runtime when a small number of threads are used. However, when four threads are provided, the execution of the routine in the GPU and the execution of the whole WRF is in both cases outperformed by the pure CPU implementation of the WRF. It should be noted that in this implementation, the data transfer from the CPU to the GPU and viceversa cannot be hidden. As a consequence, the use of a GPU to accelerate the runtime in this scenario does not produce any improvement.

Table V: Runtime (in seconds) of original WRF model with the radiation module ported to a GPU.

# Threads	*Radiation time*	*Total time*
1	73.57	347.71
2	69.63	247.97
4	61.31	191.19

In the last experiment, we evaluate the asynchronous architecture of the WRF with and without offloading the radiation computation to the GPU, using the same thread configurations on the CPU as in the the Xeon Phi experiment. Table VI presents the runtime of these executions. The 2-2 configuration is also the best performing alternative for the pure OpenMP implementation of the new architecture but it is also slower than the original WRF with four threads. However, the implementation of the new architecture offloading the radiation to the GPU produces a strong reduction on the execution time. Specifically, the 4-1 thread configuration takes 141.07 seconds for its execution, which represents an improvement of 22.4% over the original WRF runtime and comes near the theorical limit.

Table VI: Runtime (in seconds) of the implementation of the asynchronous architecture of the WRF model.

# Threads	*Radiation*	*Transference*	*Blocking*	*Total*
Asynchronous architecture				
2-2	108.93	4.48	0.01	187.95
4-1	209.53	4.50	58.92	216.17
Asynchronous architecture + GPU				
2-2	64.09	4.41	0.01	186.13
4-1	74.94	4.36	0.01	141.07

These results also confirm that the asynchronous architecture is able to make a better profit of the hardware accelerator than the original WRF architecture.

V. Concluding Remarks and Future Work

In this work, we have addressed the acceleration of the WRF model using an asynchronous architecture and hybrid hardware architectures. In particular, we have assessed the runtime of the radiation module and concluded that the RRTM routines are responsible of a large portion of the total runtime. From this analysis, we have proposed an extension of the asynchronous architecture of the WRF that offloads the computation of this module to a Xeon Phi coprocessor or a GPU.

We can conclude that the OpenMP implementation of the asynchronous architecture of WRF does not offer any benefit in a context with a small number of threads (cores). However, the combined use of the new software architecture with accelerators improves considerably the performance of the whole WRF model. Specifically, this option is able to outperform the best configuration of the original model in values that are close to the theoretical execution time of the proposal. Additionally, it also allows to make a better use of the hardware accelerators than in the original WRF architecture.

As future work it seems interesting to study other radiation modules and other hardware platforms. It is also interesting to evaluate other atmospheric properties that are naturally prone to be implemented following the asynchronous paradigm.

Acknowledgment

The work in this paper has been partially funded by ICT4V (www.ict4v.org) through the grant POS-ICT4V-2016-1-02

References

[1] J. P. Silva, J. Hagopian, M. Burdiat, E. Dufrechou, M. Pedemonte, A. Gutiérrez, G. Cazes, and P. Ezzatti, "Another step to the full GPU implementation of the weather research and forecasting model," *The Journal of Supercomputing*, vol. 70, no. 2, pp. 746–755, 2014.

[2] R. Bayá, M. Pedemonte, A. Gutiérrez, and P. Ezzatti, "Enhancing the performance of the WRF model with an asynchronous computation architecture," *Journal of Parallel and Distributed Computing (JPDCE) (under review)*.

[3] Y. Sharma and D. B. Kulkarni, "GPU based acceleration of WRF model: A review," *International Journal of Science and Research*, 2015.

[4] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, "The Weather Research and Forecast Model: Software architecture and performance," in *11th ECMWF Workshop on HPC in Meteorology*, 2004, pp. 156–168.

[5] J. Dudhia, "A multi-layer soil temperature model for mm5," in *Preprints, The Sixth PSU/NCAR mesoscale model users workshop*, 1996, pp. 22–24.

[6] E. J. Mlawer, S. J. Taubman, P. D. Brown, M. J. Iacono, and S. A. Clough, "Radiative transfer for inhomogeneous atmosphere: RRTM, a validated correlated-k model for the long-wave," *J. Geophys. Res.*, vol. 102, no. D14, pp. 16 663–16 682, 1997.

[7] A. Gutiérrez, G. Cazes, and S. De Mello, "Analysis of the optimal grid resolution for the forecasting of wind energy in different wind farms," in *IEEE I2MTC*, 2014, pp. 1512–1516.

[8] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, 1st ed. Addison-Wesley Professional, 2004.

[9] G. Ruetsch, E. Phillips, and M. Fatica, "GPU acceleration of the long-wave rapid radiative transfer model in WRF using CUDA Fortran," in *Many-Core and Reconfigurable Supercomputing Conference*, 2010.