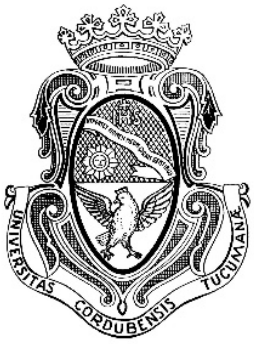
UNIVERSIDAD NACIONAL DE CÓRDOBA



FACULTAD DE MATEMÁTICA, ASTRONOMÍA, FÍSICA Y COMPUTACIÓN

Tesis de Licenciatura en Ciencias de la computación

Aceleración de un modelo numérico de predicción del tiempo para aplicaciones en alerta temprana a emergencias ambientales

---

Autor:
Luis Miguel Vargas Calderon

Directores:
Dr. Nicolás Wolovick
Mg. Rafael Andrés Lighezzolo

Córdoba, Argentina
Diciembre de 2018

Resumen

Esta tesis de licenciatura aborda el problema de implementar el modelo numérico de predicción del tiempo *Weather Research and Forecasting* (WRF) en uno de los clústeres dedicados a computo intensivo de la UNC, Mendieta, con el propósito de poder escalar el rendimiento para dicho modelo, usando tecnologías de computación de alto desempeño (HPC) tales como *Open Multi-Processing* (OpenMP) y *Message Processing Interface* (MPI).

El trabajo tiene como doble propósito generar valor en el campo de la ciencia aplicada, ya que el modelo WRF implementado en Mendieta será utilizado luego por el equipo *de Consultoría de Aplicaciones Espaciales de Alerta y Respuesta Temprana a Emergencias* (CAEARTE) perteneciente a *la Comisión Nacional de Actividades Espaciales* (CONAE), el cual se dedica al estudio de las emergencias ambientales, y por otro lado también tiene como propósito comprender como es el funcionamiento de un clúster dedicado a cómputo intensivo.

Ambos enfoques se complementan, ya que el modelo numérico WRF puede ser aprovechable al máximo por el equipo de CAEARTE al ejecutar grandes volúmenes de cómputo en paralelo, y de esa manera poder implementar un sistema de alerta temprana que proporcione pronósticos en tiempos más inmediatos. Por otro lado, el afrontar problemas como la implementación de WRF en Mendieta posibilita iniciar el entendimiento de la arquitectura y del uso de un clúster dedicado a cómputo como Mendieta, lo cual constituye una herramienta útil para futuros trabajos.

Abstract

This thesis addresses the problem of implementing the numerical weather prediction model *Weather Research and Forecasting* (WRF) in one of the dedicated to intensive computing clusters in the UNC, Mendieta, with the purpose of scaling up the performance for said model, using high performance computing (HPC) technologies such as *Open Multi-Processing* (OpenMP) and *Message Processing Interface* (MPI).

The work has as a double purpose to generate value in the field of applied science, since the WRF model implemented in Mendieta will be used later by the *Consulting Spatial Applications of Early Warning and Response to Emergencies* (CAEARTE) team belonging to the *National Commission of Space Activities* (CONAE) which is dedicated to the study of environmental emergencies, and on the other hand it also has a purpose understand how is performed the operation of a cluster dedicated to intensive computation.

Both approaches complement each other, since the numerical model WRF can be used to the maximum by the CAEARTE team when executing large volumes of computation in parallel, and in this way to be able to implement an early warning system that provides forecasts in more immediate times. On the other hand, facing problems such as the implementation of WRF in Mendieta makes it possible to begin the understanding of the architecture and the use of a cluster dedicated to computing such as Mendieta, which is a useful tool for future works.

Agradecimientos

Deseo expresar un profundo agradecimiento a la Universidad Nacional de Córdoba y en especial a la Facultad de Matemática, física, Astronomía y Computación por la formación de calidad que me brindó, deseo dar gracias también al sistema educativo público y gratuito que me apoyó para que pudiera continuar con mis estudios, al grupo de trabajo del Instituto Gulich: CONAE/CAEARTE, en especial a Andrés Lighezzolo y quien siempre me acompañó y brindó su apoyo en este trabajo como así también a Nicolás Wolovick que es un ejemplo y una guía a seguir en la profesión.

Y por último un especial agradecimiento a toda mi familia, en especial a mi hermana Gaby y a Clara, mi compañera de estos años por todo el aguante.

Índice

Contenido Página

Índice de Figuras

Figura Página

Lista de tablas

Tabla Página

Lista de abreviaturas

Abreviatura	Descripción
API	Application Programming Interface
ARW	Advanced Research WRF
CAEARTE	Consultoría de Aplicaciones Espaciales de Alerta y Respuesta Temprana a Emergencias
CCAD	Centro de Computación y Alto Desempeño de la UNC
CDF	Common Data Format
CLI	Command Line Interface
CFL	Courant, Friedrichs y Lewy
CONAE	Comisión Nacional de Actividades Espaciales
DLP	Data Level Parallelism
EM	Eurelian Mass
EPS	Ensamble Prediction System
FaMAF	Facultad de Matemática, Astronomía, Física y Computación
FLOPS	Floating-point operations per second
GDAS	Global Data Assimilation System
GFS	Global Forecast System
GPU	Graphics Processor Unit
GPGPU	General-Purpose Computing on Graphics Processing Units
GrADS	Grid Analysis and Display System
GRIB	General Regularly-distributed Information in Binary form
GSI	Gridpoint Statistical Interpolation
HDF5	Hierarchical Data Format (versión 5)
HPC	High-performance computing
ILP	Instruction Level Parallelism
LAM	Limited Area Model
MPI	Message Passing Interface
NAS	Network Attached Storage

NCAR	National Center for Atmospheric Research
NCEP	National Centers for Environmental Prediction
NeTCDF	Network Common Data Form
NMM	Nonhydrostatic Mesoscale Model
NUMA	Non Uniform Memory Access
NWP	Numerical Weather Prediction
OMP	OpenMP
SIMD	Simple Instruction Multiple Data
SLURM	Simple Linux Utility for Resource Management
TLP	Thread Level Parallelism
UMA	Uniform Memory Access
UNC	Universidad Nacional de Córdoba
WPS	WRF Preprocessing System
WRF	Weather Research and Forecasting

1. Introducción

1.1. Motivación

El Panel Intergubernamental para el cambio climático (Mastrandrea et. Al. 2010) [1], (Change I. C. 2014) [2] afirma que el calentamiento en el sistema climático es inequívoco y, desde la década de 1950, muchos de los cambios observados no han tenido precedentes en los últimos decenios a milenios. La atmósfera y el océano se han calentado, los volúmenes de nieve y hielo han disminuido, el nivel del mar se ha elevado y las concentraciones de gases de efecto invernadero han aumentado. Las emisiones continuas de gases de efecto invernadero causarán un mayor calentamiento y nuevos cambios en todos los componentes del sistema climático. A medida que el cambio climático debido al forzamiento antropógeno continúe, es probable que, cambios en los climas regionales y la ocurrencia de fenómenos meteorológicos extremos se vuelvan más comunes, aumentando aún más la necesidad de monitoreo continuo y de sistemas de alerta temprana. Sin sistemas de alerta temprana, la sociedad y los sectores productivos están en constante estado de vulnerabilidad.

Inundaciones, incendios, sequías, heladas, granizo, descargas eléctricas, olas de calor o salud pública, son ejemplos de amenazas que dependen de manera directa o indirecta de una o más variables meteorológicas y que no son ajenas a nuestra región.

Los recientes modelos numéricos de predicción del tiempo (NWP) permiten obtener un conjunto de variables meteorológicas pronosticadas con anticipación de varios días y actualizado diariamente. Por esta razón estos modelos numéricos son el primer eslabón de un sistema de alerta temprana. Los avances de los modelos numéricos de predicción del tiempo en las últimas décadas y el rango de sus aplicaciones son cada vez más amplios (Subbiah et al 2008) [3]. En la mayoría de los centros operativos de predicción meteorológica se utiliza una combinación de modelos globales y locales. Los modelos globales son generalmente usados en pronósticos de rango medio (2 días o más) y debido a que ellos tienen un dominio horizontal global, que incluye toda la tierra, estos modelos no pueden correr con una alta resolución. Para un pronóstico más detallado es necesario aumentar la resolución, es aquí donde se utilizan los Modelos de Área Limitada (LAMs) en la región de interés. Gracias a su alta resolución los LAMs tienen la ventaja de ser más precisos en la reproducción de fenómenos a pequeña escala. (Steppeler et al 2003 [4]; Kain et al 2006 [5]; Narita y Ohmori 2007 [6]; Lean et al 2008 [7]; Kain

et al 2008 [8]; Weisman et al 2008 [9]). Sin embargo, no hay un consenso respecto al tamaño de retícula más conveniente, ya que algunos autores priorizan un bajo costo computacional, mientras que otros priorizan la información detallada que se obtiene con las resoluciones más elevadas.

Paralelamente, otra de las tendencias utilizadas en los centros operativos es la de pronósticos probabilísticos (llamados ensambles) (Xue et al 2010 [10]). En cuanto a esta técnica, (Grimit y Mass 2002 [11]) señalan que el pronóstico por ensambles proporciona una forma práctica de considerar la variabilidad en las condiciones iniciales, las incertidumbres de la física del modelo y las propias de la predicción de la situación atmosférica, logrando así una estimación más precisa de los estados futuros de la atmósfera a través de, por ejemplo, el valor medio de una variable del ensamble. A su vez, la variación entre los miembros del ensamble provee una medida de la incertidumbre en la predicción.

La alta resolución en los dominios de cálculo y la técnica por ensambles son factores que tienden a la mejora significativa de la predicción, sin embargo, ambos factores tienen como contraposición el alto costo computacional.

En este contexto se propone el desarrollo de la presente tesis de carácter netamente interdisciplinar, que toma como base la experiencia adquirida en la puesta operativa del modelo numérico de predicción del tiempo Weather Research and Forecasting (WRF) en la Unidad de Consultoría en Aplicaciones Espaciales de Alerta y Respuesta Temprana a Emergencias (CAEARTE) de la Comisión Nacional de Actividades Espaciales (CONAE) y toda la experiencia e infraestructura del grupo de HPC y GPGPU Computing de la FaMAF, que tiene el objetivo general descrito a continuación.

1.2. Objetivo general

El principal objetivo de esta tesis se orienta a lograr las bases de operatividad de una predicción numérica meteorológica por ensambles en alta resolución horizontal de utilidad en sistemas de alerta temprana a emergencias. Para ello se focalizará en la aceleración de los procesos de cálculo, aprovechando los recursos de cómputo de CPUs y/o GPUs modernas, cálculo paralelo y procesamiento en entornos de clúster.

1.3. Objetivos específicos

- Instalar, configurar y poner en funcionamiento el modelo WRF en el clúster Mendieta.
- Aplicar el modelo a alguna region del territorio argentino.
- Evaluar eficiencia y escalabilidad del modelo en el clúster.
- Generar archivos de salida con información meteorológicamente relevante.
- Sentar las bases para la transferencia de conocimiento de la implementacion de WRF en un clúster de alto desempeño computacional.

2. El pronóstico y los modelos numéricos

2.1. Marco Teórico

Los modelos numéricos de predicción del clima (NWP) utilizan modelos matemáticos de la atmósfera para hacer predicciones de estas basándose en las condiciones actuales de la misma.

Más específicamente, estos modelos resuelven un conjunto de ecuaciones matemáticas basadas en las leyes físicas que gobiernan el comportamiento de la atmósfera y generan una predicción de su evolución a partir de un estado inicial, proveyendo una sucesión de representaciones tridimensionales de los estados futuros de la atmósfera.

Un sistema moderno operativo de pronóstico del tiempo consiste básicamente en cinco componentes:

- Recopilación de datos
- Asimilación de datos
- Predicción numérica del tiempo
- Post procesamiento de modelos de salida
- Presentación del pronóstico al usuario final

Cotidianamente distintos tipos de observaciones meteorológicas se realizan y recopilan a lo largo del planeta. Las fuentes de datos más relevantes para la realización de un pronóstico son las siguientes:

- Observaciones en superficies
- Radiosondeos
- Reportes de aeronaves
- Observaciones de boyas
- Observaciones de radar
- Observaciones de satélites

Existen diferencias de resolución espacial, temporal, y de exactitud, entre las observaciones. La calidad de las observaciones varía según la plataforma de observación, la hora del día y las condiciones meteorológicas. La asimilación de los datos es el proceso mediante el cual las observaciones se introducen en los ciclos de pronóstico de un modelo numérico de predicción del tiempo, brindando además una medida de protección contra el crecimiento de los errores del modelo y contribuyendo a establecer las condiciones iniciales para el próximo ciclo de ejecución del modelo.

La asimilación de datos es un campo de la predicción meteorológica que se halla en fase de crecimiento, ya que siguen aumentando tanto el volumen como la variedad de los datos que se incorporan en los modelos de pronóstico.

El objetivo de la asimilación de datos consiste en optimizar la exactitud del pronóstico fusionando las observaciones con un pronóstico previo a corto plazo. Esta fusión es la condición inicial que utilizan los modelos numéricos para realizar la predicción.

Los modelos numéricos de predicción del tiempo resuelven un conjunto de ecuaciones matemáticas basadas en leyes físicas que gobiernan el comportamiento de la atmósfera, prediciendo su evolución a partir de un estado inicial.

Vilhelm Bjerknes, en 1904 (Lynch 2008) [12], fue el primero en reconocer que la predicción numérica del tiempo era posible y propuso que esencialmente el pronóstico del tiempo se puede considerar como un problema matemático de condiciones iniciales.

Una representación matemática simple de esta clase de problema es la siguiente [13]:

$$\frac{\partial A}{\partial t} = F(A), \quad A(0) = A_0, \qquad (1)$$

Donde A es alguna variable meteorológica que se modifica en el tiempo debido a la acción de F.

F se conoce como el forzado, combinando todos los tipos de acciones de cambio sobre A.

La solución de ecuación (1) tiene la forma siguiente:

$$A_{Pronosticado} = A_{Inicial} + \int F(A)dt. \qquad (2)$$

Este proceso se sigue para configurar las ecuaciones para los NWP. Las ecuaciones básicas que gobiernan los procesos en la atmósfera se presentan a continuación.

2.2. Las ecuaciones primitivas

Vilhelm Bjerknes fue el primero en dar un conjunto completo de siete ecuaciones con siete incógnitas que gobiernan la evolución de la atmósfera. Estas son:

La conservación del momento o segunda Ley de Newton (Para las tres componentes de la velocidad)

$$\frac{d\mathbf{v}}{dt} = -\alpha \nabla p - \nabla \phi + \mathbf{F} - 2\boldsymbol{\Omega} \times \mathbf{v}$$

La ecuación de continuidad o conservación de la masa.

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v})$$

La ecuación de estado para los gases ideales

$$p\alpha = RT$$

La ecuación de conservación de la energía o primera Ley de la termodinámica.

$$Q = C_p \frac{dT}{dt} - \alpha \frac{dp}{dt}$$

La ecuación de conservación para la masa de agua.

$$\frac{\partial \rho q}{\partial t} = -\nabla \cdot (\rho \mathbf{v} q) + \rho(E - C)$$

Las siete incógnitas son las tres componentes de la velocidad *V=(u,v,w)*, la temperatura T, la presión *p*, $\alpha = 1/\rho$ es el volumen específico y *q* es la razón de mezcla del vapor de agua. Estas ecuaciones son conocidas con el nombre de ecuaciones primitivas. Para mayores detalles sobre estas ecuaciones ver Kalnay (2003) [14].

Debido a su complejidad las ecuaciones primitivas deben ser resueltas numéricamente utilizando aproximaciones algebraicas.

2.3. Discretización

El proceso por medio del cual se obtiene la solución del sistema de ecuaciones algebraicas está constituido básicamente por dos etapas. La primera etapa, llamada discretización, consiste en transformar el dominio continuo en una malla de nodos.

Un modelo NWP divide la atmósfera en varias capas discretas horizontales y cada una de ellas a su vez es dividida en un número de celdas, donde las variables son evaluadas en el centro de cada celda. Ver figura 2.1.

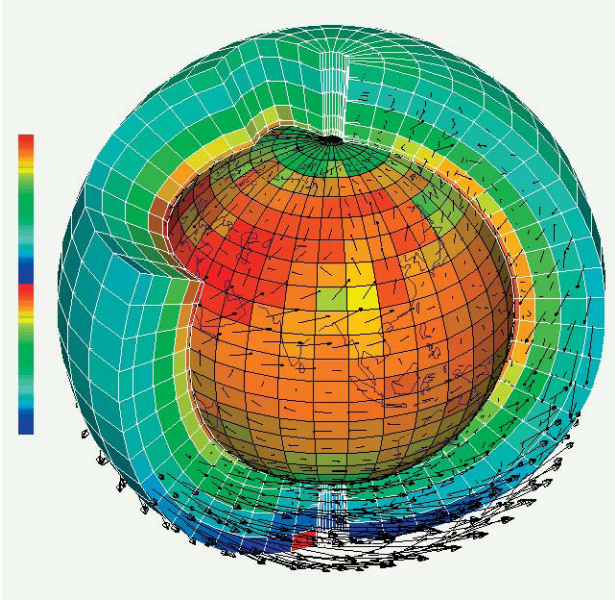


Figura 2.1: Discretización del dominio a modelar

En una atmósfera real la temperatura, la presión, el viento y la humedad varían de manera suave y continua de un lugar a otro, mientras que en una representación en grilla cada celda tiene asignado un valor que es el promedio de los valores contenidos en esta última.

Un ejemplo de esta representación puede verse en la siguiente figura:

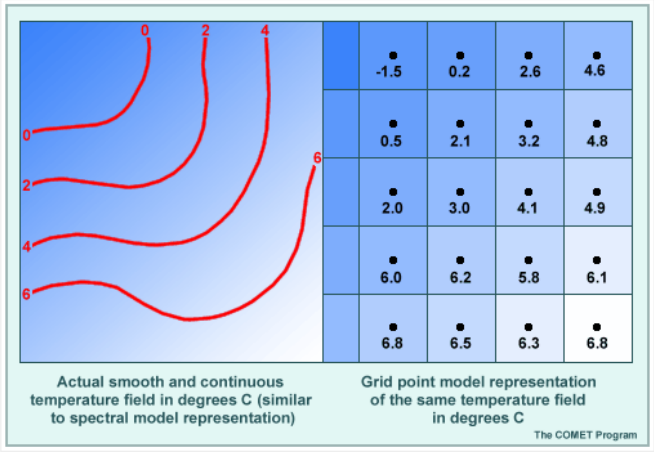


Figura 2.2: Asignación de valores en la grilla

La segunda etapa del proceso requiere un método numérico adecuado para obtener la solución del sistema de ecuaciones algebraicas planteado.

Existe una gran variedad de métodos numéricos para resolver estas ecuaciones, el método más común es el denominado de “diferencias finitas”. En esta técnica los valores de las variables

continuas son representados por un conjunto de datos de valores finitos y sus derivadas son aproximadas por diferencias los valores de los puntos adyacentes.

De manera similar el tiempo es llevado a un número finito de pasos discretos, siendo la evolución continua de las variables aproximada a pasos discretos. Este método fue ampliamente aplicado en la resolución de ecuaciones diferenciales por su utilidad y simpleza conceptual.

Para las ecuaciones diferenciales que gobiernan la dinámica de la atmósfera existen criterios de estabilidad que evitan tener inestabilidades numéricas que conduzcan a una solución irreal del problema. La precisión depende de los tamaños de los pasos de tiempo y de espacio, Δt y Δx respectivamente. Es razonable esperar que los errores se vuelvan mas pequeños cuanto más refinada es la grilla espacio-temporal.

Courant, Friedrichs y Lewy publicaron un criterio (conocido como criterio CFL) para determinar los tamaños relativos de las grillas de espacio y tiempo.

Δx	200km	100km	20km	10km
Δt	500s	250s	50s	25s

Tabla 2.1: Estimación de pasos de tiempo y espacio según el criterio CFL.

2.4. Resolución horizontal

La resolución horizontal está relacionada con el espaciamiento entre los puntos de grilla y meteorológicamente relacionada con el fenómeno particular que se quiera simular o predecir. A mayor resolución el modelo puede describir fenómenos de menor escala.

Típicamente se requiere al menos cinco puntos de grilla para describir un fenómeno del tamaño de un punto de grilla. Para el caso de un modelo con una resolución horizontal de 20 km no puede predecir fenómenos con tamaños menores a 100 km.

Un ejemplo de la importancia de la resolución horizontal puede verse en la figura siguiente.

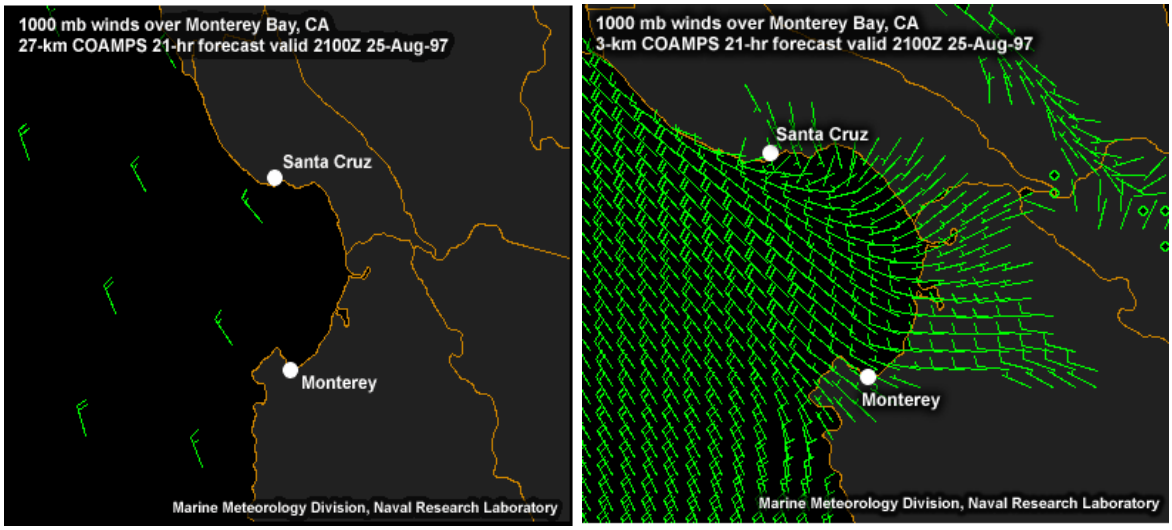


Figura 2.3: Ejemplo de alta resolución horizontal

Lo que muestra la figura es una predicción del viento para 1000 mb[1] de presión a la misma hora sobre la bahía de Monterrey, Méjico. La imagen de la izquierda muestra el resultado de la simulación con una resolución espacial de 27 km, mientras que la imagen de la derecha tiene una resolución espacial de 3 km. En general se observa en un viento fluyendo hacia la costa, pero el modelo a 3 km puede representar además una brisa marina fluyendo sobre la bahía e incluyendo una circulación hacia el sur cerca de Santa Cruz. Es decir, la simulación de 3 km es capaz de captar la brisa marina mientras que la de 27 km la ignora.

El incremento en la resolución demanda el incremento de los recursos computacionales ya que el modelo necesita calcular una mayor cantidad de puntos de grilla. Por ejemplo, si se quiere reducir la resolución horizontal de 27 km en un tercio, el número de cálculos para la misma área aumenta nueve veces. La figura 2.4 ilustra lo dicho.

[1] milibar

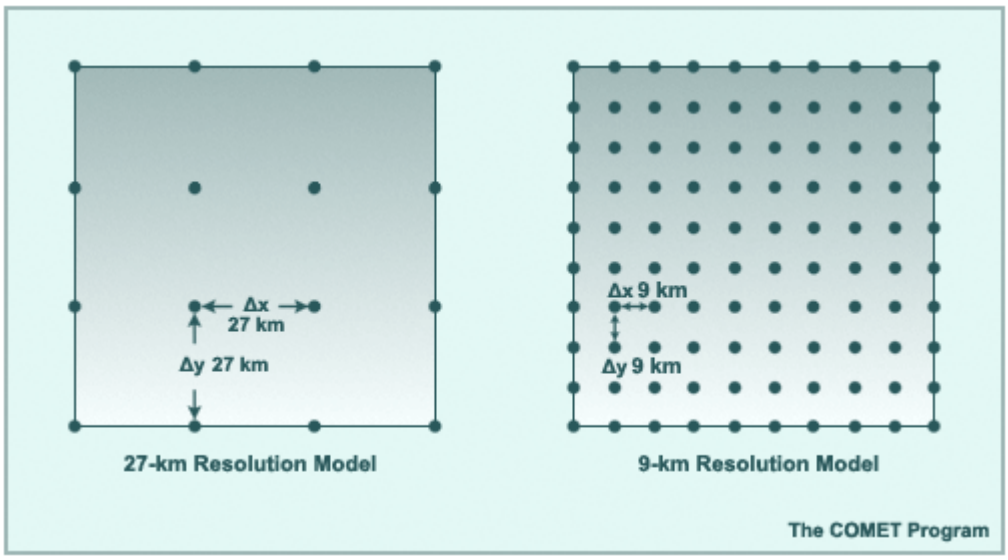


Figura 2.4: Incremento de resolución en la grilla

Además, cuando se aumenta la resolución espacial decrece el paso de tiempo de cálculo entre dos estados sucesivos. Es decir, aumenta la resolución temporal del cálculo. Sin embargo, la alta resolución mejora notablemente las capacidades del modelo para representar el terreno, que, a la vez, influye de manera directa sobre la meteorología.

2.5. Parametrizaciones

Los modelos de clima tienen típicamente una resolución horizontal del orden de varios cientos de kilómetros, los modelos de pronósticos globales tienen resoluciones de 13-100 km, y los modelos regionales de mesoescala[2] tienen resoluciones de 1–30 km. En la dirección vertical, la resolución y la extensión vertical ha sido incrementada sustancialmente en los modelos actuales teniendo entre 10 y 50 niveles verticales, extendiendo desde la superficie hasta la estratósfera e inclusive la mesósfera. Así mientras el poder de las computadoras siga aumentando, la resolución de los modelos también lo hará. Pero a pesar del continuo incremento de la resolución, hay muchos procesos importantes y escalas de movimientos en la atmósfera que no pueden ser explícitamente resueltos con los modelos actuales.

[2] La mesoescala en Meteorología es el estudio de sistemas del tiempo atmosférico más pequeños que la escala sinóptica meteorológica, pero más grandes que la microescala y la escala de tormenta de los sistemas de nubes cúmulos. Sus dimensiones horizontales generalmente oscilan de cerca de 9 km a varios centenares de km.

Esto incluye desde movimientos turbulentos con escalas de centímetros hasta la escala de la grilla del modelo, así como también procesos que ocurren a nivel de escala molecular, como condensación, evaporación, fricción y radiación. Se suele referir a todos los procesos que no pueden resolverse explícitamente como "procesos a escala de sub-grilla".

Claramente no es realista esperar que los modelos numéricos resuelvan los procesos a escalas tan pequeñas sin importar que tan alta sea la resolución. Por ejemplo, las fuerzas de fricción provocada por un edificio en particular. Sin embargo, se puede cuantificar el efecto total de todos los obstáculos sobre un flujo de aire con un valor que represente la fricción dentro de la celda considerada.

Este método de cuantificar tales efectos, sin calcularlos directamente, se denomina parametrización.

Stensrud (2009) [15] resume a los esquemas de parametrizaciones en distintas categorías: superficie terrestre, interacción atmosférica, interacción agua-atmósfera, capa límite planetaria y turbulencia, convección, microfísica y radiación. La figura siguiente muestra algunos de estos procesos.

Estos efectos son parametrizados básicamente por tres razones:

1. Los recursos computacionales no son lo suficientemente potentes para calcular directamente el fenómeno ya sea porque son de escalas muy pequeñas o muy complejas en la resolución numérica.

2. Los procesos no están bien entendidos como para ser representados en una ecuación.

3. Los efectos impactan profundamente en el modelo y son cruciales para generar un pronóstico realista.

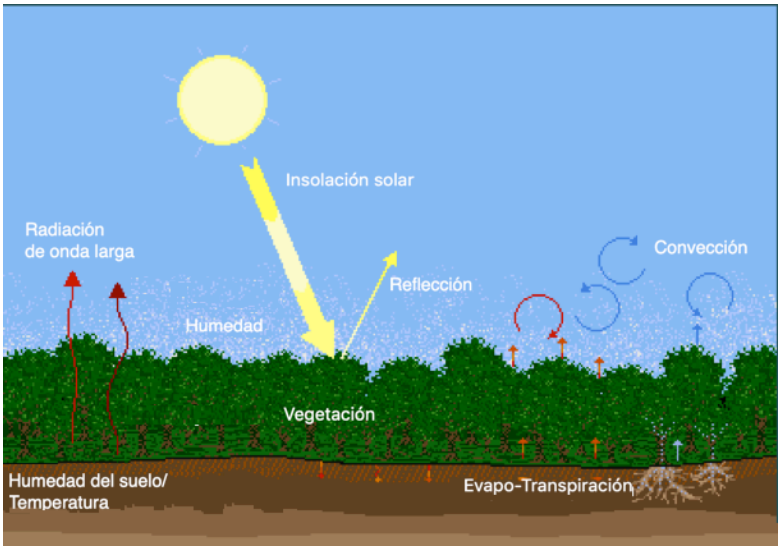


Figura 2.5: Fenómenos naturales a parametrizar

Los detalles de las parametrizaciones tienen profundos efectos en los modelos de pronóstico, principalmente a escalas de tiempos grandes, y son el motivo de muchas y continuas investigaciones [32].

2.6. Alta resolución

Muchos de los centros operacionales de pronóstico meteorológico están direccionando sus esfuerzos hacia la implementación de modelos de alta resolución para pronósticos a corto plazo. Como ejemplo de centros se pueden nombrar el Met Office UK, Japan Meteorological Agency y Germany's National Meteorological Service.

Resoluciones horizontales en rangos de entre 2-4 km han sido implementados y probablemente resoluciones de 1 km serán comunes en los próximos años Lean et al 2008 [7].

Existen varias razones del porqué los modelos de alta resolución podrían mejorar un pronóstico. Es esperable que el incremento en la resolución permita al modelo representar características de mesoescala que de otra manera no puede ser resuelta, por ejemplo, representar una convección explícita más que por una parametrización de la convección. Estudios como los de (Weisman et al. 1997 [16]; Romero et al. 2001 [17]; Speer and Leslie 2002 [18]; Done et al. 2004 [19]) demuestran que existen mejoras en la representación de tormentas severas y frentes convectivos cuanto menor es la longitud de la grilla.

Por otro lado, los modelos de alta resolución son capaces de utilizar datos de entrada de alta resolución. Esto puede hacerse mediante el uso de datos de alta resolución como por ejemplo la orografía y uso del suelo, o mediante la asimilación de datos de radar o satélites.

2.7. Ensambles

Para la realización de un pronóstico del tiempo preciso es esencial una detallada y certera representación de la condición inicial de la atmósfera.

Como se dijo anteriormente las observaciones están siendo continuamente coleccionadas y combinadas con pronósticos a corto plazo, mediante las técnicas de asimilación de datos, para estimar la mejor descripción del estado actual de la atmósfera.

A pesar de la enorme cantidad de observaciones, nunca son suficientes para dar una exacta y completa descripción tridimensional de la atmósfera global. Por lo tanto, hay siempre alguna incerteza en las condiciones iniciales. Esto es un problema para los NWP porque la atmósfera es un sistema caótico donde los pequeños errores iniciales pueden crecer en el tiempo hasta distorsionar profundamente el resultado del pronóstico (este es el conocido “efecto mariposa”) (Richarson 2011 [20]).

Un ejemplo de la sensibilidad de un NWP a pequeñas incertezas en las condiciones iniciales se refleja en la siguiente figura.

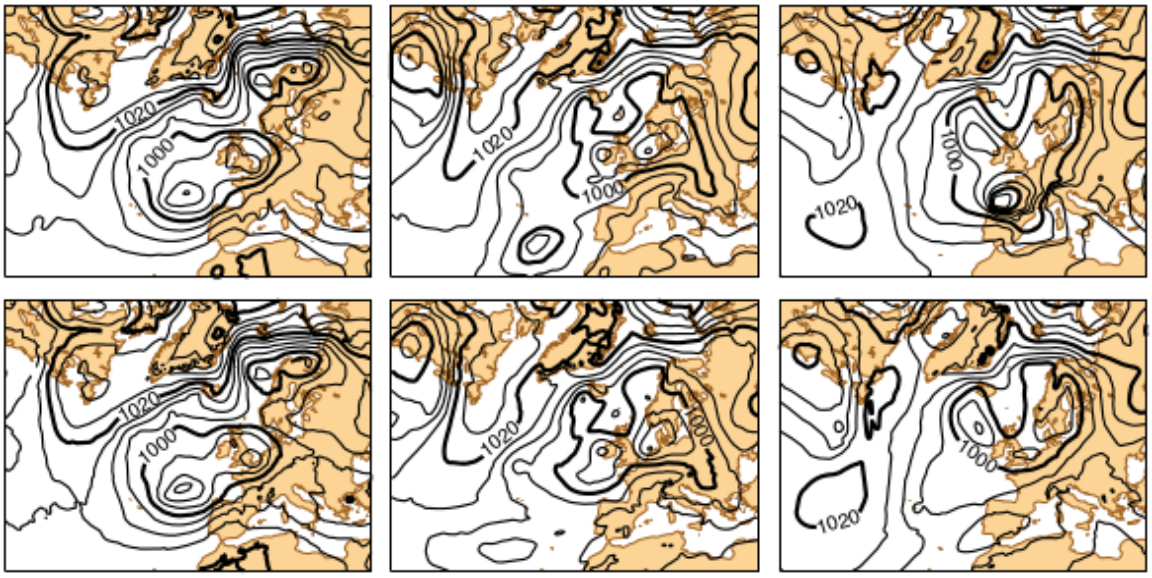


Figura 2.6: Ensambles

En la parte superior del gráfico se puede observar la presión a nivel de suelo para tres diferentes tiempos del pronóstico: la condición inicial (izquierda), el segundo día pronosticado (centro) y el tercer día de pronóstico (derecha). En el mismo se observa un sistema de baja presión se desarrolla al sur de Azores que rápidamente se profundiza produciendo tormentas severas afectan a una amplia zona del Oeste de Europa.

En la parte inferior del gráfico se observa la situación inicial con una leve perturbación. Las diferencias son casi imperceptibles al comienzo, pero el cambio es suficiente como para que el pronóstico no de indicios de tormentas como se observó en la parte superior.

Esta sensibilidad representa un claro dilema para los pronosticadores, ya que da lugar a preguntas como: ¿Cuál es el correcto? ¿Es potencialmente peligrosa esta tormenta si ocurre?

De la figura anterior los pronosticadores no tienen forma de saber qué pronóstico es el correcto ya que ambos son consistentes con las condiciones iniciales observadas.

Con la idea de resolver este problema fueron desarrollados los sistemas de pronósticos por ensambles (EPS, Ensemble Prediction System).

Epstein (1969) [21] introdujo la idea de pronóstico estocástico-dinámico que consiste en intentar determinar la función de distribución de probabilidad de los posibles estados futuros de la atmósfera a partir de un estado inicial. La forma propuesta para determinar esta distribución de

probabilidades fue la de realizar múltiples simulaciones de la atmósfera utilizando condiciones iniciales o modelos ligeramente diferentes y de esta manera determinar algunas propiedades (varianza, valor medio) de dicha distribución que no podía ser conocida en detalle dado el gran número de grados de libertad que presenta el problema. Una de las hipótesis fundamentales del pronóstico por ensambles es que, si la forma de obtener los diferentes pronósticos es adecuada, las propiedades estadísticas del conjunto se asemejan a las propiedades estadísticas de la verdadera función de los estados futuros de la atmósfera.

Recientemente, esta metodología que permite reducir la incertidumbre y que a la vez provee información sobre dicha incertidumbre a lo largo del plazo del pronóstico, ha sido aplicada en modelos globales a mediano plazo y en pronósticos regionales a corto plazo con resultados satisfactorios Ruiz (2008) [22].

Es importante destacar que existen diversas fuentes de errores vinculados con los modelos numéricos que, como se mencionó anteriormente, tienen que ver con los métodos de discretización espacial y temporal de las ecuaciones y errores al representar fenómenos de menor escala que la resolución horizontal utilizada. Una manera de considerar estos errores en la formulación de los modelos es construir ensambles donde cada miembro es un modelo distinto o dentro de un mismo modelo se utilizan diferentes parametrizaciones. Esta última técnica es ampliamente utilizada en actuales centros de pronósticos operativos.

Los EPS proveen una estimación más precisa de los estados futuros de la atmósfera a través de la media del ensamble y además provee también una medida de incertidumbre en la previsión considerando la desviación estándar entre los miembros del mismo.

Este valor agregado en la información, impacta directamente sobre los productos o guías que intervienen en el proceso de toma de decisión de los diferentes usuarios del pronóstico.

2.8. Modelo GFS

El modelo Global Forecast System (GFS) es desarrollado por el National Centers for Environmental Prediction (NCEP) EEUU, es un sistema global de predicción numérica del tiempo que corre cuatro veces al día y produce pronósticos de hasta 16 días, cuyos datos de salida disponibles libremente en un dominio público en internet.
El modelo corre en dos partes. La primera de ellas tiene alta resolución y pronostica 192 horas (8 días), mientras que la segunda parte corre desde la hora 192 hasta la 384 (16 días) en baja

resolución. La resolución del modelo varía en cada parte del modelo: horizontalmente la resolución varía desde 35 a 70 km, verticalmente divide la atmósfera en 64 capas y temporalmente produce pronósticos cada 3 horas en las primeras 192 horas y cada 12 horas hasta las 384 horas. El GFS es un modelo acoplado compuesto por cuatro modelos separados, un modelo de atmósfera, un modelo de océano, un modelo de tierra/suelo y un modelo de hielo marino. Estos cuatro modelos trabajan juntos para proveer mayor precisión en la representación de las condiciones climáticas.

Para inicializar los pronósticos, el GFS utiliza información que proviene del Global Data Assimilation System (GDAS). El GDAS es el componente operativo del Gridpoint Statistical Interpolation (GSI) el cual es un sistema de asimilación de datos variacional tridimensional (3D_VAR).

El GSI puede asimilar los siguientes tipos de observaciones:

- Observaciones en superficies
- Radiosondeos
- Reportes de aeronaves
- Observaciones de boyas
- Observaciones de radar
- Observaciones de satélites

Este modelo global proporciona la entrada de datos para el sistema LAMs WRF.

2.9. Modelo WRF

El modelo regional (WRF) es de código abierto, desarrollado y respaldado de manera continua por varias instituciones de EEUU y el mundo. Es un NWP diseñado para utilizarse en la investigación tanto como para la realización de pronósticos operacionales. WRF cuenta con dos núcleos principales, un sistema de asimilación de datos y una arquitectura de software que permite la paralelización de procesos. El modelo permite un amplio rango de aplicaciones meteorológicas en escalas con rangos que van desde unos pocos kilómetros hasta miles de kilómetros. El desarrollo del WRF comenzó a principio de los años 90 y las principales entidades que colaboraron en este proyecto se enumeran seguidamente. La cantidad de estas instituciones y la calidad de las mismas evidencian la importancia y el interés en la utilidad del proyecto.

- National Center for Atmospheric Research (NCAR)
- National Centers for Environmental Prediction (NCEP)
- Forecast Systems Laboratory (FSL)
- Air Force Weather Agency (AFWA)
- Naval Research Laboratory (NLR)
- University of Oklahoma
- Federal Aviation Administration (FAA)

El modelo WRF permite a los investigadores producir simulaciones considerando datos reales o en condiciones de atmósfera idealizada, mientras que a la vez provee un pronóstico operacional flexible en una plataforma robusta basada en la contribución de los últimos avances en física, modelado numérico y técnicas de asimilación de datos desarrollados por los investigadores.

WRF se encuentra actualmente operacional en el NCEP y muchos otros centros de pronósticos operacionales. Cuenta con una creciente comunidad de usuarios (más de 23000 usuarios registrados de 150 países diferentes) y se realizan continuas actualizaciones, tutoriales y congresos. Este modelo ofrece dos diferentes módulos (solvers) para el cálculo de las ecuaciones que gobiernan la atmósfera. El primero de ellos, **utilizado en esta tesis**, es conocido como WRF-ARW (Advanced Research WRF) que es respaldado por la división de Meteorología de Mesosescala y Microescala del NCAR. El segundo se conoce como WRF-NMM Nonhydrostatic Mesoscale Model) que es una variante basada en el modelo *Eta* [40], y luego del NMM desarrollado por NCEP. El WRF-NMM es respaldado por el DTC (Developmental Testbed Center).

El WRF es de dominio público y libremente disponible para el uso de la comunidad. El modelo se compone de los siguientes módulos:

2.9.1. Pre procesamiento: WPS

La herramienta WRF Preprocessing System (WPS) es un conjunto de tres programas que trabajan colaborativamente con el objetivo de preparar los inputs para la etapa de procesamiento.
El programa que consume los outputs generados por WPS es *real.exe* (etapa de procesamiento) el cual a su vez produce inputs para *wrf.exe* para simulaciones con datos reales.
Estos programas son los siguientes:

- *geogrid.exe:* define el dominio de simulación del modelo e interpola datos geográficos estáticos (gribfiles).
- *ungrib.exe:* extrae campos meteorológicos desde los archivos gribfiles, provistos por GFS. La definición de los campos requeridos a extraer desde los gribfile se encuentra en una tabla denominada VTable (Variable Table).
- *metgrid.exe*: interpola horizontalmente los campos meteorológicos extraídos por ungrib para los dominios definidos por *geogrid.exe*. Genera archivos en formato NetCDF.

Por ejemplo, para un pronóstico de 24 h dado en la siguiente fecha:

- 2018 Enero 20 - 12:00 UTC hasta 2018 Enero 21 - 12:00 UTC

WPS genera los siguientes archivos:

- *met_em.d01.2018-01-20_12:00:00.nc*
- *met_em.d01.2018-01-20_18:00:00.nc*
- *met_em.d01.2018-01-21_00:00:00.nc*
- *met_em.d01.2018-01-21_06:00:00.nc*
- *met_em.d01.2018-01-21_12:00:00.nc*

La convención en el uso de *"met"* indica que los outputs generados por el componente metgrib.exe de WPS.
La parte "*d01*" en el nombre identifica el dominio de pertenencia. El modelo WRF permite trabajar con varios dominios anidados, sin embargo, en el presente trabajo no se usó esa característica.

El flujo de datos de WPS es el descrito en la figura 2.7.

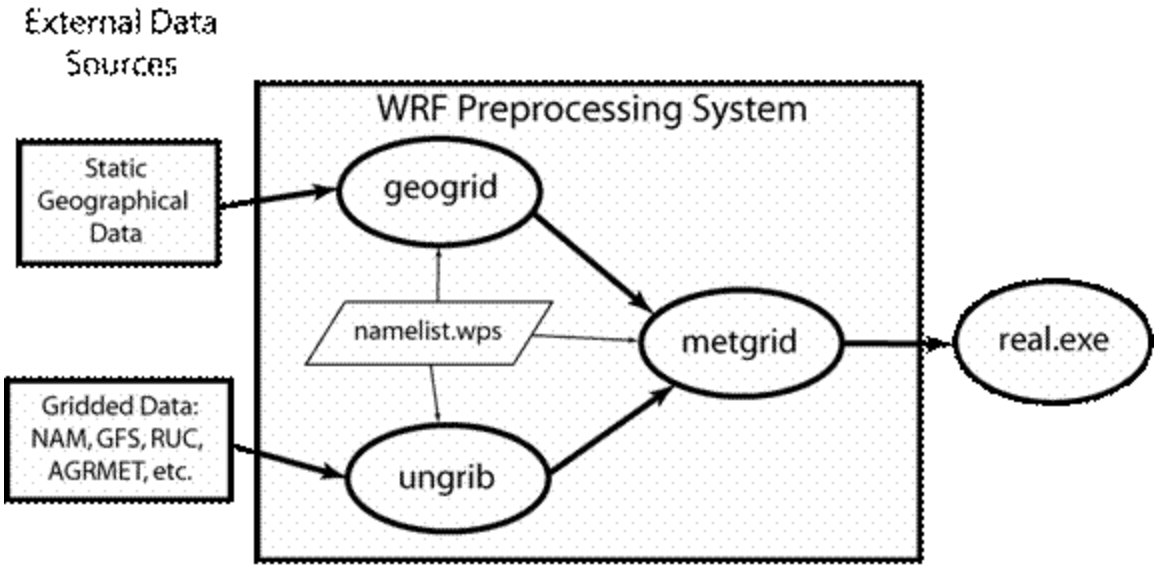


Figura 2.7: WRF Preprocessing System (WPS)

2.9.2. Procesamiento: ARW solver

Este es el componente clave del sistema de modelado, está compuesto de varios programas de iniciación para simulaciones de casos ideales y reales, y por programas de integración. Para la versión de WRF 3.8 se compone de cuatro programas.

```
[lvargas@mendieta wrf_mendieta]ls -l  WRF3.8/WRFV3/main/*.exe
-rwxrwxr-x 1 lvargas lvargas 30919176 Mar  6 00:21 WRF3.8/WRFV3/main/ndown.exe
-rwxrwxr-x 1 lvargas lvargas 30800296 Mar  6 00:22 WRF3.8/WRFV3/main/real.exe
-rwxrwxr-x 1 lvargas lvargas 30469232 Mar  6 00:22 WRF3.8/WRFV3/main/tc.exe
-rwxrwxr-x 1 lvargas lvargas 35527904 Mar  6 00:21 WRF3.8/WRFV3/main/wrf.exe
```

Para las simulaciones realizadas en el presente trabajo solo se utilizan los siguientes:
real.exe: Este programa, desarrollado en Fortran, es el que toma como inputs datos de observaciones reales previamente pre-procesados y formateados a formato NetCDF por parte de WPS.
El programa *real.exe* genera 2 outputs, los archivos *"wrfinput_d01"* y *"wrfbdy_d01"* los cuales son inputs para *wrf.exe*.
wrf.exe: es el núcleo de toda la simulación, es un programa altamente paralelizable. Tambien esta desarrollado en Fortran. Un cómputo de este programa puede durar muchas días si no se

explota su capacidad de ejecutar en paralelo, para lo cual es crucial aprovechar al máximo las herramientas de paralelismo que dicho programa soporta. Para el ejemplo anteriormente visto generaría un único output:
El archivo *wrfout_d01:2000-01-24_12:00:00*, el cual contiene un pronóstico de 24 h. con un tiempo de intervalo de 3 h.

Algunas de las características principales de WRF se nombran a continuación:

- Ecuaciones: eulerianas, no hidrostáticas con opciones hidrostáticas en tiempo de ejecución.
- Variables pronosticadas: componentes de velocidad horizontal *u* y *v* en coordenadas cartesianas, velocidad vertical *w*, perturbación potencial de la temperatura, perturbación geopotencial y perturbación de la presión del aire seco en la superficie. Opcionalmente, energía cinética turbulenta y escalares como razón de mezcla de vapor de agua, razón de mezcla de lluvia/nieve, razón de mezcla de agua/hielo en la nube y distintas especies químicas.
- Coordenadas verticales: coordenadas que siguen el terreno, coordenadas de presión vertical hidrostática en la parte superior del modelo.
- Grilla horizontal: de clase Arakawa_C.[3]
- Integración temporal: Runge-Kutta de orden 2 y 3 con pequeños pasos de tiempo para los modos de ondas de gravedad y acústicas. Capacidad de variar el paso del tiempo de integración.
- Discretización espacial: opciones de advección de orden 2 a 6 en la horizontal y en la vertical.
- Mezclado turbulento y modelos de filtros: formulación de la turbulencia a escala de subgrilla en ambas coordenadas y espacio físico. Amortiguación de la divergencia, modo externo de filtrado. Opciones explícitas de filtrado.
- Condiciones iniciales: tridimensionales para datos reales y una, dos o tres dimensiones para datos idealizados. Capacidad de iniciación con filtrado digital (DFI) para casos reales.

[3] El sistema de cuadrícula de Arakawa describe diferentes formas de representar y calcular cantidades físicas ortogonales (especialmente cantidades relacionadas con la velocidad y la masa) en cuadrículas rectangulares utilizadas para los modelos del sistema terrestre para meteorología y oceanografía. Por ejemplo, el Modelo de pronóstico e investigación del tiempo utiliza la cuadrícula C escalonada de Arakawa en sus cálculos atmosféricos cuando usa el núcleo ARW. Las cinco cuadrículas de Arakawa (A-E) se introdujeron por primera vez en Arakawa y Lamb 1977.

- Condiciones de contorno laterales: periódicas, abiertas, simétricas y con opciones especificadas.
- Condiciones de contorno superior: absorción de ondas de gravedad (difusión, amortiguación Rayleigh, o amortiguación implícita de Rayleigh para las velocidades verticales). Niveles de presión constante en el tope.
- Condiciones de contorno inferior: físicas o libres.
- Rotación terrestre: términos de Coriolis completamente incluidos.
- Anidado: interactivo en una dirección, interactivo en dos direcciones y anidados móviles.
- Nudging (método de asimilación de datos): capacidad de asimilar con esta técnica.

Las opciones físicas del modelo WRF están divididas en 5 categorías, las cuales pueden ser elegidas y modificadas. Las categorías se describen brevemente a continuación:

- Microfísica: esquemas que van desde modelos simples de la microfísica de nubes, válidos para casos ideales, en los que se incluyen pocos parámetros termohidrodinámicos hasta los más complejos con una microfísica sofisticada (con cambios de fase, mezclas, crecimiento de partículas, etc.) adecuados para NWP y estudios más realistas de las nubes.
- Parametrizaciones de cúmulos: esquemas ajustados y de flujo masa al modelado de mesoescala.
- Física de la superficie: modelos de superficie multicapa que van desde un simple modelo termal a un completo modelo de vegetación y humedad del suelo, incluyendo cobertura de nieve y hielo marino.
- Física de la capa límite planetaria: predicción de energía cinética turbulenta o esquemas *K* no locales.
- Física de la radiación atmosférica: esquemas de ondas cortas y largas con bandas multiespectrales, un esquema simple de onda corta adecuado para aplicaciones meteorológicas. Los efectos de nubes y flujos de la superficie son también incluidos.

Para representar datos físicos como la velocidad o la masa se utilizan grillas o matrices del tipo Arakawa. Existen 5 tipos. WRF utiliza la clase C. Por ejemplo, para modelar cantidades vectoriales como la velocidad en dirección norte-sur (variable v) y la velocidad en dirección este-oeste (variable u) y la masa (variable h) se evalúan las componentes en el centro de los

lados izquierdo y derecho de la celda, mientras que para las componentes de *v* la evaluación podría ser en el centro del lado superior e inferior de la celda.

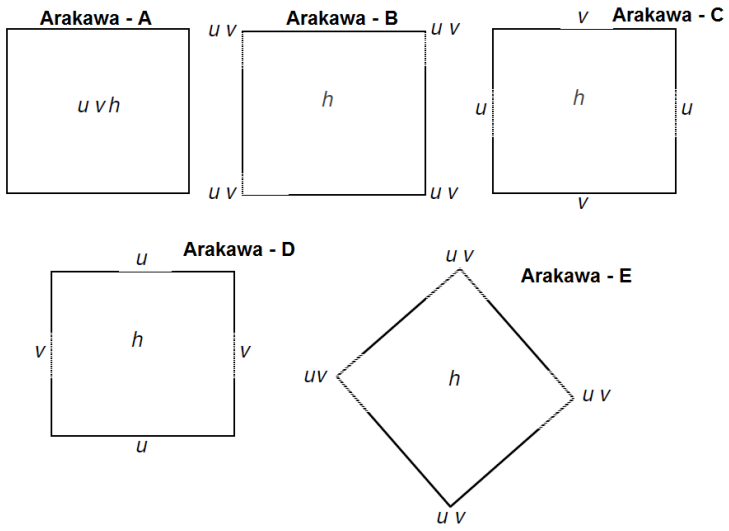


Figura 2.8: Grillas de tipo Arakawa

2.9.3. Post Procesamiento: ARWpost

El pronóstico generado por WRF-ARW, para nuestro ejemplo el archivo: *wrfout_d01:2000-01-24_12:00:00* necesita ser post-procesado para poder obtener datos en texto plano o mapas que grafiquen el pronóstico. Para ello se utiliza ARWpost que es un programa desarrollado en Fortran que toma como input la salida generada por del WRF-ARW y genera las salidas necesarias para ser interpretado por GrADS.

2.10. Grid Analysis and Display System (GrADS)

El Grid Analysis and Display System (GrADS) es una herramienta que se utiliza para la manipulación y la visualización de datos de Ciencias de la Tierra. GrADS tiene dos modelos de datos para manejar los datos de la cuadrícula y de la estación. GrADS soporta muchos formatos de archivo de datos, incluyendo NetCDF, HDF (versión 4 y 5). GrADS se ha implementado en todo el mundo en una variedad de sistemas operativos de uso común y se distribuye libremente a través de Internet.

Las operaciones se ejecutan interactivamente introduciendo expresiones similares a FORTRAN en la línea de comandos. Un rico conjunto de funciones integradas se proporciona, pero los usuarios también pueden agregar sus propias funciones como rutinas externas escritas en cualquier lenguaje de programación.

Los datos pueden mostrarse utilizando una variedad de técnicas gráficas: gráficos de líneas y barras, diagramas de dispersión, contornos suavizados, contornos sombreados, líneas de corriente, vectores de viento, cuadros de rejilla, cuadros de rejilla sombreada y diagramas de modelo de estación. Los gráficos pueden imprimirse en PostScript o en formatos de imagen. GrADS tiene una interfaz programable (lenguaje de scripting) que permite sofisticadas aplicaciones de análisis y visualización.

GrADS genera además meteogramas. Un meteograma es una gráfica donde se representan la evolución temporal de una o varias variables meteorológicas de un punto determinado. Las variables a presentar pueden ser de superficie o de altura del punto considerado. Inicialmente, los meteogramas se utilizaron para representar la evolución de los datos de superficie suministrados por una estación donde se realizaban observaciones convencionales (temperatura, nubosidad, viento, etc.). Un meteograma también puede ser utilizado para representar datos en la vertical de estaciones que realizan tomas de información en altura mediante, por ejemplo, radiosondeos. La llegada de los modelos numéricos y la mejora de las técnicas de post-proceso. Mediante técnicas de interpolación y ajustes estadísticos y climatológicos es posible obtener salidas sobre puntos conocidos y asociados a ciudades, aeropuertos, etc.
A continuación, se puede ver en la figura 2.9 un diagrama general de la estructura de WRF y sus dependencias. Se remarcaron los componentes utilizados en el presente trabajo.

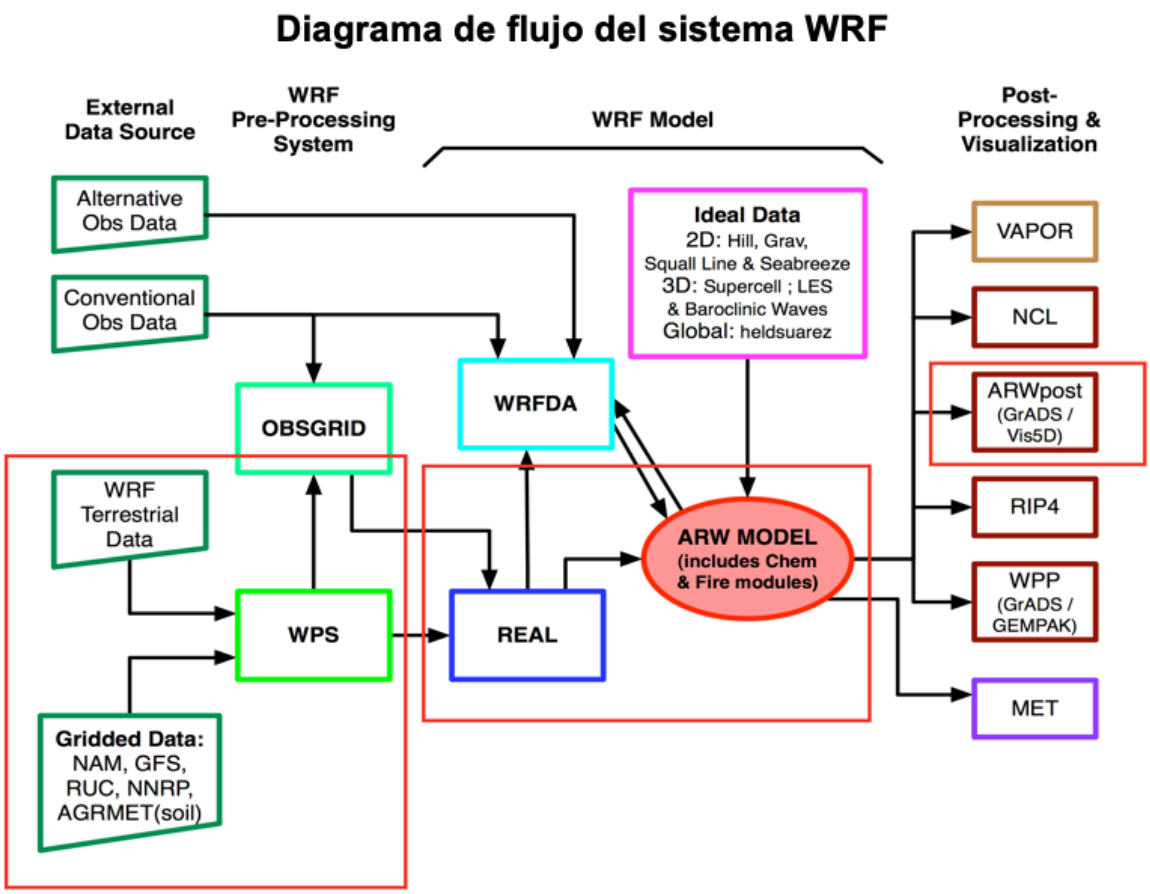


Figura 2.9: Arquitectura de WRF + ARWPost + GrADS

3. Fundamentos de computación paralela & HPC

3.1. Fundamentos

La idea de rendimiento computacional puede ser entendida como la rapidez o la mejora en el desempeño que se puede obtener en la ejecución de un programa. Se basa en la mejoras y técnicas computacionales aplicadas de manera tal que los resultados que dicho programa genera sean accesibles en un tiempo que resulte útil.

Dichas técnicas consistían inicialmente en el incremento en la cantidad de transistores contenidos en una unidad de procesamiento (chip-CPU). El escalamiento en la cantidad de transistores proporcionó un incremento en la performance y fue básicamente posible debido a que el costo del incremento en performance (aumentando la densidad de los transistores) era más bajo que el costo del desarrollo computacional (software), es decir, si la performance escala más rápido que el costo computacional, es posible computar grandes volúmenes de cálculos donde antes no era económicamente rentable o donde el tiempo de cómputo era extremadamente alto.

Esta idea de incremento en la performance computacional fue plasmada en 1965 por Gordon E. Moore en su famosa Ley de Moore [23], la cual establece que la cantidad de transistores por chip se duplicaría cada 12 o 24 meses. El reducir estas escalas (o aumentar la densidad) en los transistores deriva en una mayor rapidez en el flujo de datos que son transferidos en el transistor.

La figura 3.1 muestra que en la actualidad la cantidad de transistores en un procesador ronda los dos mil millones. Esto es posible reduciendo la superficie de dichos transistores al orden de los nanómetros. Intel alcanzó en 2014 la escala de los 14nm (14 milmillonésima parte de un metro) Obteniendo una mejora de 0.7x respecto a la tecnología anterior (22nm) y esta compañía ha anunciado el desarrollo de transistores en la escala de los 10nm para el año 2022.

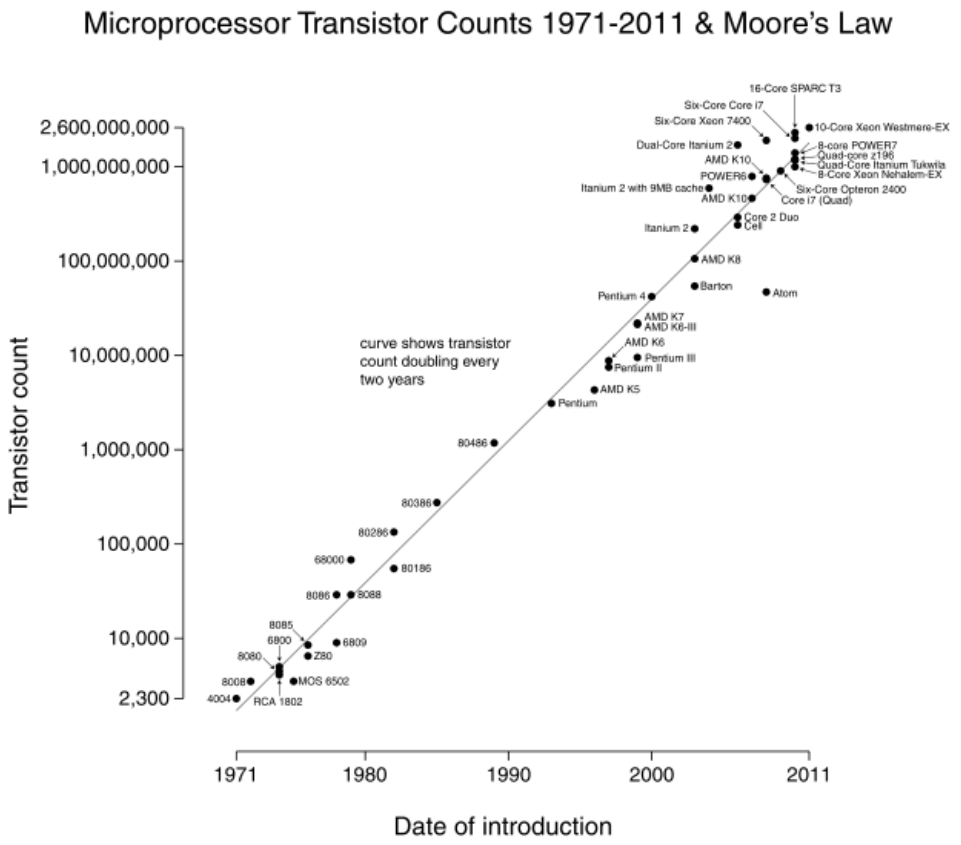


Figura 3.1: Evolución de la ley de Moore

Sin embargo, desde mediados de la primera década del siglo XXI se hizo visible claramente que la ley de Moore se encontraba frente un fin cercano, debido a dos causas obvias: 1) la imposibilidad de reducir el área de los transistores a escalas sumamente pequeñas, es decir se está alcanzando los límites físicos en cuanto a la reducción de los transistores y 2) Debido al problema del aumento de la temperatura en los transistores o la degradación de la eficiencia energética. Esto indica que la performance que se requiere ya no puede obtenerse únicamente a través del incremento en la densidad de transistores. Esto impulsó la implementación de nuevas técnicas, entre las cuales, predomina actualmente incrementar la cantidad de unidades de procesamiento de manera masiva. Es decir, incrementar el paralelismo del cómputo. Esto se logra en primera instancia agregando más núcleos a un procesador (multicore) y en segunda instancia proveyendo múltiples procesadores multicore en la computadora.

En la figura 3.2 se puede ver la estructura de un procesador multicore moderno (XEON E5, serie 2600 - utilizado en Mendieta) el cual cuenta con 8 unidades de procesamiento (núcleos).

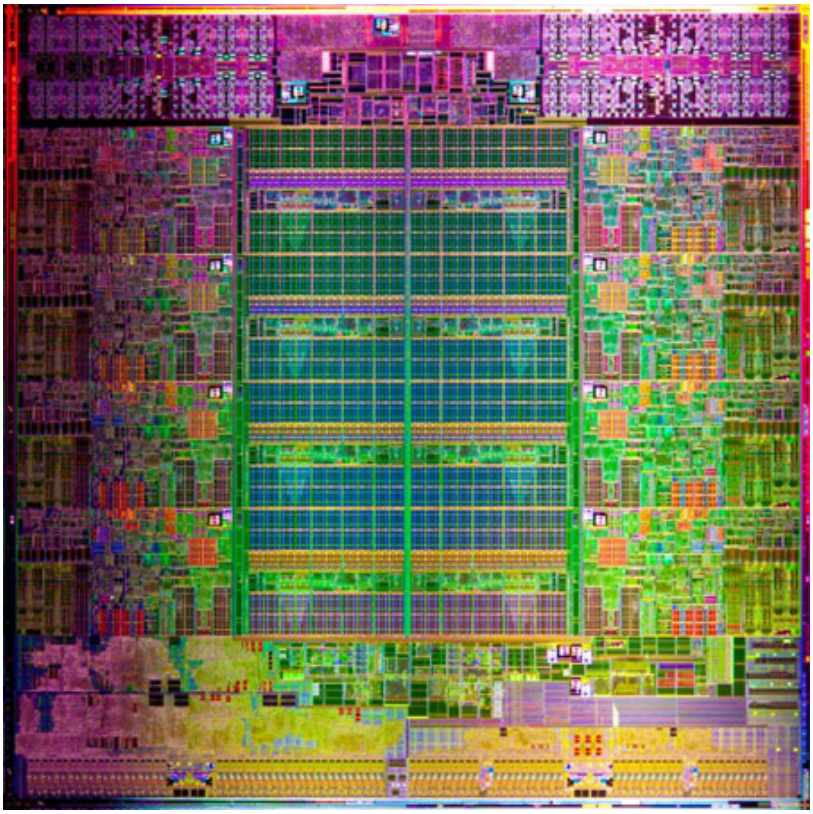


Figura 3.2: Die del Procesador Xeon E5 serie 2600[4]

Cada unidad de procesamiento tiene una estructura como la siguiente (figura 3.3) donde se puede apreciar que hay un importante uso de espacio dedicado a memorias caches. Esto es para reducir la latencia que generan los accesos a memoria principal.

[4] Un Die (en ingles pronunciado [dai]), en el contexto de circuitos integrados, es un pequeño bloque de material semiconductor en el que se fabrica un circuito funcional dado. Normalmente, los circuitos integrados se producen en grandes lotes en una sola oblea de silicio de grado electrónico. La oblea se corta en muchas piezas, cada una de las cuales contiene una copia del circuito. Cada una de estas piezas es un Die.

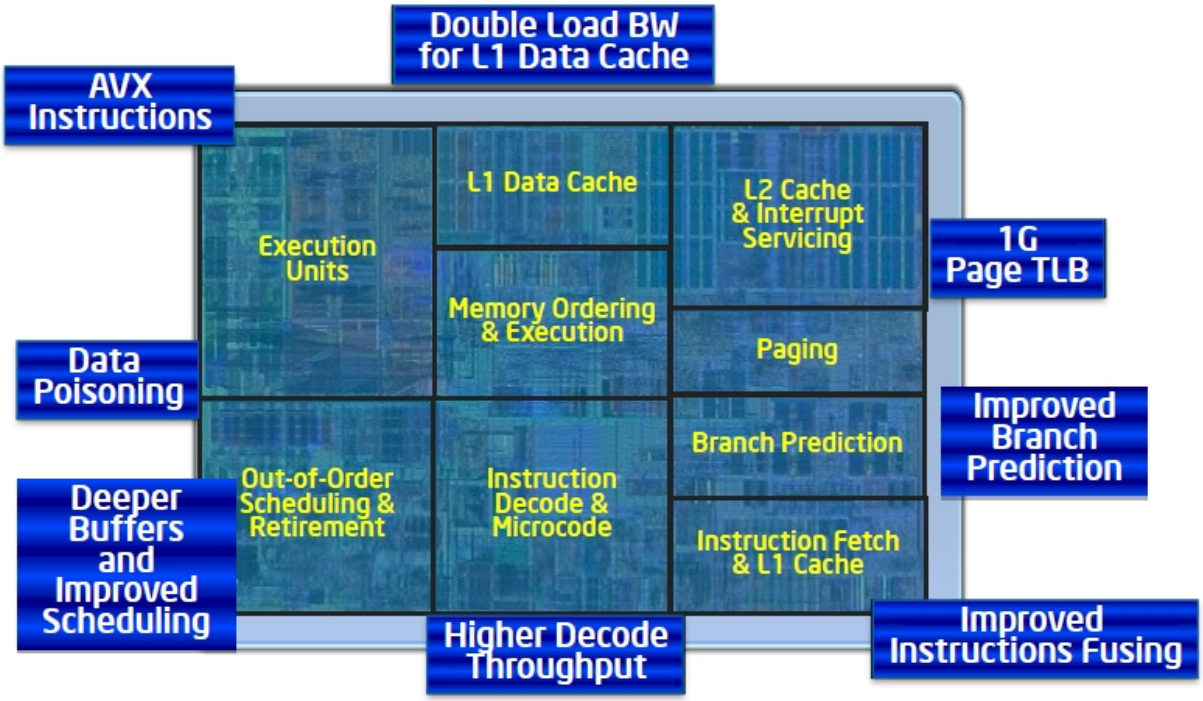


Figura 3.3: Core de un Procesador Xeon E5 2600-V2

Un paso siguiente en el nivel de paralelismo consistió en lograr desarrollar tecnologías de comunicación más eficientes para poder conectar múltiples computadoras a través de un bus de alta velocidad y de esa manera lograr ejecutar un software de manera distribuida y concurrente. Esto es conocido como computación distribuida o clúster de computadoras.

Un clúster dedicado de cómputo tiene en el orden de centenas a millones de procesadores como los señalados en el párrafo anterior. En la actualidad un clúster dedicado a cómputo puede realizar operaciones en el orden de los TFLOPS (10^{12} FLOPS) alcanzando picos en el orden de los PFLOPS (10^{15} FLOPS).

La lista de supercomputadoras más potentes del mundo reportó a noviembre de 2016 que la supercomputadora más potente; la china *Sunway TaihuLight* soporta un rendimiento teórico pico de 125 PFLOPS y ha alcanzado al día de la fecha un rendimiento real de 93 PFLOPS. Para dicho cálculo se utilizó como benchmarking[5] el software para medir rendimiento computacional LINPACK [24].

[5] Benchmarking: Es una evaluación comparativa que se utiliza para medir el rendimiento de un sistema mediante un indicador específico (Ejemplo: tiempo de ciclo de por unidad de medida o defectos por unidad de medida).

Ranking	Ubicación/ País	Sistema	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Energía (kW)
1	National supercomputing Center in Wuxi/ China	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway	10649600	93014.6	125435.9	15371
2	National Super Computer Center in Guangzhou/ China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P	3120000	33862.7	54902.4	17808
3	DOE/SC/Oak Ridge National Laboratory/ EEUU	Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x	560640	17590	27112.5	8209
4	DOE/NNSA/LLNL/ EEUU	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	1572864	17173.2	20132.7	7890
5	DOE/SC/LBNL/NERSC/ EEUU	Cori - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect	622336	14014.7	27880.7	3939

Tabla 3.1: Ranking con las 5 supercomputadoras más poderosas del mundo a Noviembre de 2016

La cantidad de FLOPS puede ser calculada con la siguiente fórmula:

$$\text{FLOPS} = \text{sockets} \times \frac{\text{cores}}{\text{socket}} \times \text{clock} \times \frac{\text{FLOPs}}{\text{cycle}}$$

Estos clústeres dedicados a cómputo también han ido incorporando nuevas arquitecturas computacionales tales como coprocesadores de cómputo masivo, placas aceleradoras, GPU, etc., y también desarrollando nuevos algoritmos, lenguajes y bibliotecas de computación paralela para dichas arquitecturas.

Con el advenimiento de nuevas demandas en diversos sectores empresariales e industriales sumado al desarrollo tecnológico fue posible realizar computaciones que años atrás hubiesen sido física y económicamente imposibles.
La disciplina que desarrolla la computación paralela en clústeres es conocida como *computación de alto desempeño* (high performance computing/HPC).

Algunas áreas de aplicación del HPC:

- Simulaciones de fenómenos físicos
- Simulaciones de fenómenos ambientales
- Industria militar
- Industria petrolífera

3.2. Niveles de paralelismo

Teniendo en cuenta que el paralelismo es la fuente que proporciona performance, es necesario entender las técnicas existentes para explotarlo al máximo. El mismo se puede extraer en varios niveles al momento de la ejecución de un programa.

3.2.1. Paralelismo a nivel de instrucciones (ILP: Instruction level parallelism)

El paralelismo a nivel de instrucción (ILP) consiste en el conjunto de técnicas que posibilitan que múltiples instrucciones independientes entre sí sean ejecutadas de forma concurrente en la unidad de procesamiento (core).
Existen dos enfoques para el paralelismo del nivel de instrucción:

- Hardware
- Software

El nivel de hardware funciona en paralelismo dinámico mientras que el nivel de software trabaja en paralelismo estático. Paralelismo dinámico significa que el procesador decide en tiempo de ejecución qué instrucciones ejecutar en paralelo, mientras que el paralelismo estático significa que el compilador decide qué instrucciones ejecutar en paralelo.
Considere el siguiente programa:

1: e = a + b
2: f = c + d
3: m = e * f

La operación 3 depende de los resultados de las operaciones 1 y 2, por lo que no puede calcularse hasta que ambos se completen. Sin embargo, las operaciones 1 y 2 no dependen de ninguna otra operación, por lo que se pueden calcular simultáneamente. Si asumimos que cada operación puede ser completada en una unidad de tiempo, entonces estas tres instrucciones pueden ser completadas en un total de dos unidades de tiempo, dando un ILP de 3/2.
Un objetivo de los diseñadores de compiladores y procesadores es identificar y aprovechar tanto ILP como sea posible. Los programas ordinarios se escriben normalmente bajo un modelo de ejecución secuencial en el que las instrucciones se ejecutan una tras otra y en el orden especificado por el programador. ILP permite al compilador y al procesador superponer la ejecución de varias instrucciones o incluso cambiar el orden en que se ejecutan las instrucciones.

Las técnicas que se utilizan para explotar el ILP incluyen:

- Instruction pipeling: La ejecución de múltiples instrucciones puede ser parcialmente solapada.
- Ejecución superscalar: Se utilizan múltiples unidades de ejecución para ejecutar múltiples instrucciones en paralelo.
- Ejecución Fuera de orden: Las instrucciones se ejecutan en cualquier orden que no viole dependencias de datos. Esta técnica es independiente tanto de pipeling y superscalar. Las implementaciones actuales de ejecución fuera de orden de forma dinámica (es decir, mientras el programa se ejecuta y sin ninguna ayuda del compilador) extraen ILP de programas ordinarios. Una alternativa es extraer este paralelismo en tiempo de compilación y de alguna manera transmitir esta información al hardware. Debido a la complejidad de escalar la técnica de ejecución fuera de orden, la industria ha

reexaminado conjuntos de instrucciones que codifican explícitamente múltiples operaciones independientes por instrucción.

- Cambio de nombre de registros: Se refiere a una técnica utilizada para evitar la serialización innecesaria de las operaciones del programa impuestas por la reutilización de registros por esas operaciones, que se utiliza para permitir la ejecución fuera de orden.
- Ejecución especulativa: Permite la ejecución de instrucciones completas o partes de instrucciones antes de estar seguro de si esta ejecución debe tener lugar. Una forma comúnmente utilizada de ejecución especulativa es la especulación de flujo de control en la que se ejecutan instrucciones pasadas una instrucción de flujo de control (por ejemplo, una rama) antes de que se determine el objetivo de la instrucción de flujo de control. Varias formas de ejecución especulativa han sido propuestas y están en uso incluyendo la ejecución especulativa impulsada por predicción de valores, predicción de dependencia de memoria y predicción de latencia de caché.
- Branch prediction: Se utiliza para evitar el bloqueo para que las dependencias de control sean resueltas. Branch prediction se utiliza junto con la ejecución especulativa.

ILP es explotado tanto por el compilador como por el procesador. Algunas técnicas de optimización para extraer ILP disponible en tiempo de compilación incluyen programación, como loop-unrolling, o estrategias que favorecen la localidad espacial y temporal de los accesos a memoria.

Tal como se detalló en la sección 3.1. ILP encuentra límites tanto en las optimizaciones de compilación como en las optimizaciones provistas por el procesador, tales como la creciente disparidad entre las frecuencias operativas del procesador y los tiempos de acceso a la memoria (este problema es conocido como *memory wall*). Aunque en principio es posible utilizar ILP para tolerar incluso tales latencias de memoria, los costes asociados de disipación de energía son desproporcionados. Además, la complejidad y frecuentemente la latencia de las estructuras de hardware subyacentes reducen la frecuencia de operación y reducen aún más los beneficios. Por lo tanto, las técnicas mencionadas resultan inadecuadas para impedir que la CPU se bloquee mientras se accede a los datos en memoria. En su lugar la industria se dirige hacia la explotación de niveles más altos de paralelismo que pueden ser explotados a través de técnicas como el multiprocesamiento y el multithreading. Es decir, el máximo posible del performance es extraído a partir del paralelismo masivo que proporcionan los niveles de paralelismo de datos (DTL) y paralelismo de threads (TLP).

3.2.2. Paralelismo a nivel de datos (DLP: Data level Parallelism o SIMD)

En computación, DLP o SIMD (del inglés Single Instruction, Multiple Data, en español: "una instrucción, múltiples datos") es una técnica empleada para conseguir paralelismo a nivel de datos. La técnica SIMD consiste en ejecutar las mismas instrucciones sobre conjuntos grandes de datos. Es una organización en donde una única unidad de control común despacha las instrucciones a diferentes unidades de procesamiento. Todas éstas reciben la misma instrucción, pero operan sobre diferentes conjuntos de datos. Es decir, la misma instrucción es ejecutada de manera sincronizada por todas las unidades de procesamiento.
La figura 3.4 muestra como el mismo conjunto de datos puede ser procesado en paralelo por varias unidades de procesamiento aplicando las mismas instrucciones sobre distintos datos.

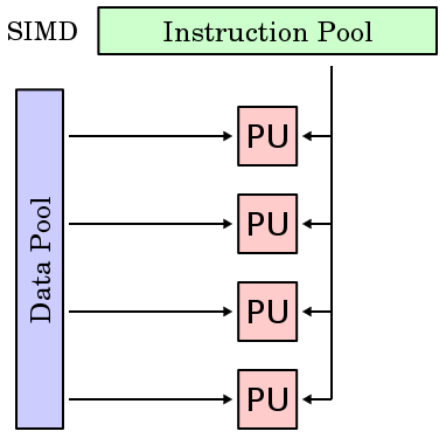


Figura 3.4: Simple Instruction Multiple Data (SIMD)

Este nivel de paralelismo puede obetenerse en la ejecución de WRF. Como se mencionó en la sección 2.3. WRF discretiza el dominio de simulación en grillas o matrices. Durante la etapa de procesamiento de datos WRF aplica a cada una de las celdas de las grillas las mismas ecuaciones de diferenciación que producen valores que representan el estado pronosticado para cada una de esas celdas. Estas operaciones (ecuaciones) son aplicadas en paralelo a toda la grilla. Mendieta cuenta además con unidades de procesamiento masivo como placas GPU y coprocesadores Xeon-Phi, los cuales están diseñados para explotar al máximo el paralelismo a nivel de datos, sin embargo, la implementación de WRF para ejecutar en dichas arquitecturas está fuera del alcance del presente trabajo.

3.2.3. Paralelismo a nivel de threads (TLP: Thread level Parallelism)

En un sistema multiprocesador, el paralelismo de tareas se logra cuando cada procesador ejecuta un proceso o thread diferente sobre los mismos o diferentes datos. Los threads pueden ejecutar el mismo código o código diferente. En el caso general, los hilos de ejecución diferentes se comunican entre sí a medida que se ejecutan. La comunicación se puede realizar a través de espacios de memoria compartida o a través envío de mensajes entre procesos.

Como ejemplo simple, si un sistema está ejecutando código en un sistema de 2 procesadores (CPUs "a" y "b") en un entorno paralelo y deseamos hacer las tareas "A" y "B", suponiendo que no existe dependencia de datos entre ellas, es posible decir CPU "a" para hacer la tarea "A" y la CPU "b" para hacer la tarea "B" simultáneamente, reduciendo así el tiempo de ejecución de la ejecución.

En Mendieta este tipo de paralelismo se manifiesta claramente cuando se ejecutan procesos MPI entre los nodos de la partición *multi* y también cuando se ejecutan threads de OpenMP en la partición *Mono.*

3.3. Speedup y eficiencia

Además de considerar la reducción del tiempo como valor principal en la obtención de mejora en la performance de la ejecución de un programa, también es interesante analizar con qué comportamiento se produce dicha mejora [25].

Speedup: Es una medida que indica de qué manera mejora la velocidad de ejecución de un programa cuando se ejecuta sobre una arquitectura paralela. Para el caso concreto de Mendieta calculamos el speedup tomando como unidad de procesamiento los nodos de la partición *multi*. Esto es el tiempo de procesamiento partiendo de un nodo sobre N nodos. La fórmula general tiene la siguiente forma: donde T_1 es el tiempo de ejecución en unidad de procesamiento y T_p es el tiempo de ejecución en P unidades de procesamiento.

$$\text{speedup} = S_p = \frac{T_1}{T_p}$$

Eficiencia: Es una medida relativa que permite la comparación de desempeño en diferentes entornos de computación paralela. Valores de eficiencia inferiores a 1 denotan un speedup sublineal. Valores superiores a 1 denotan speedup supralineal.

$$\text{efficiency} = \frac{S_p}{P} = \frac{T_1}{PT_p}$$

3.4. Non-Uniform Memory Access

Como se mencionó en la sección 3.1. la performance en cómputo es actualmente obtenida en base a paralelismo masivo, el cual es obtenido a través de arquitecturas con múltiples unidades de procesamiento.
Sin embargo, este tipo de arquitectura de procesamiento genera muchos más accesos a memoria, incrementando el fenómeno de *memory wall*. Para mitigar este problema actualmente existen dos tipos de arquitectura para memoria de acceso en paralelo.

- Arquitectura de memoria distribuida: es la arquitectura usada en clúster, en la cual múltiples nodos comparten datos a través de una conexión en una red de alta velocidad.
- Arquitectura de memoria compartida: es la memoria usada internamente en un único nodo. Dicho nodo puede estar compuesto por varias CPUs y cada CPU tiene su propia memoria. Existen dos tipos de implementaciones: *Uniform Memory Access* (UMA) y *Non-Uniform Memory Access* (NUMA).

  Con UMA el acceso a memoria se realiza a través de un controlador denominado *Memory Controller Hub* (MCH). Este tipo de arquitectura limita la escalabilidad, el ancho de banda de memoria (bandwidth) y genera más latencia, debido a que el MCH está conectado a un controlador a través de un bus, el cual, para cada comunicación, los CPUs necesitan tomar el control generando que otros CPUs deban esperar para acceder a memoria. Esto es un cuello de botella para el cómputo.

  Para resolver este problema surgió la arquitectura NUMA [26]. Este tipo de arquitectura no usa un sistema centralizado de acceso a memoria, sino que permite a cada CPU

utilizar su propia memoria, además de acceder a secciones de memoria remota que pertenecen a otra CPU. La figura 3.5 muestra que tanto la CPU 1 como la CPU 2 pueden acceder a su propia memoria como así también a la memoria remota del otro CPU.

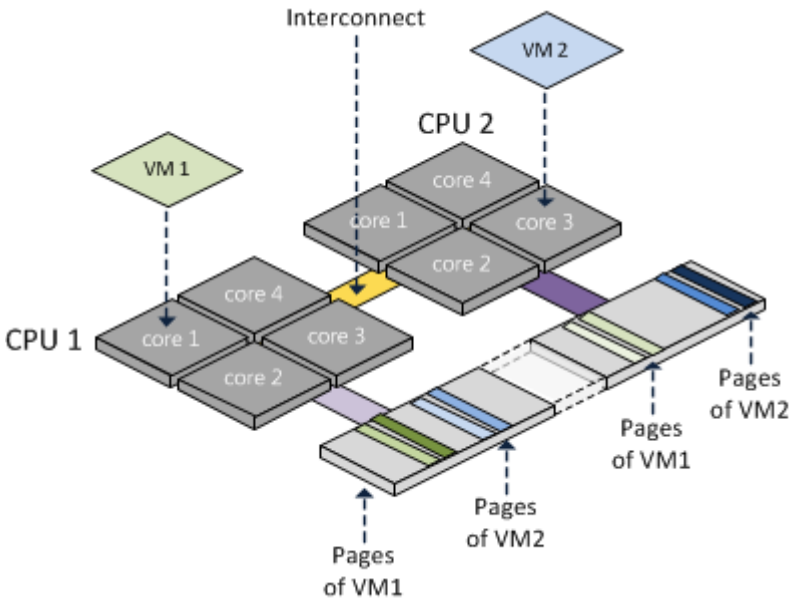


Figura 3.5: Arquitectura NUMA

De esta manera existen dos clasificaciones de memoria relativa a cada CPU: local y remota. Los accesos a memoria local proporcionan un mayor rendimiento y se reduce considerablemente la latencia al reducir los accesos a memoria remota. Estos tiempos de acceso son considerablemente diferentes, lo cual explica la no uniformidad. Para evitar problemas de coherencia en las cache de cada memoria, se utilizan algoritmos sofisticados. De este modo NUMA es además en la actualidad conocido como ccNUMA (Cache-Coherent Non-Uniform Memory Access).

WRF está implementado para aprovechar tanto la arquitectura de memoria distribuida como así también la arquitectura de memoria compartida. Mendieta tiene arquitectura ccNUMA.

La figura 3.6. muestra la arquitectura de memoria NUMA de un nodo de Mendieta.

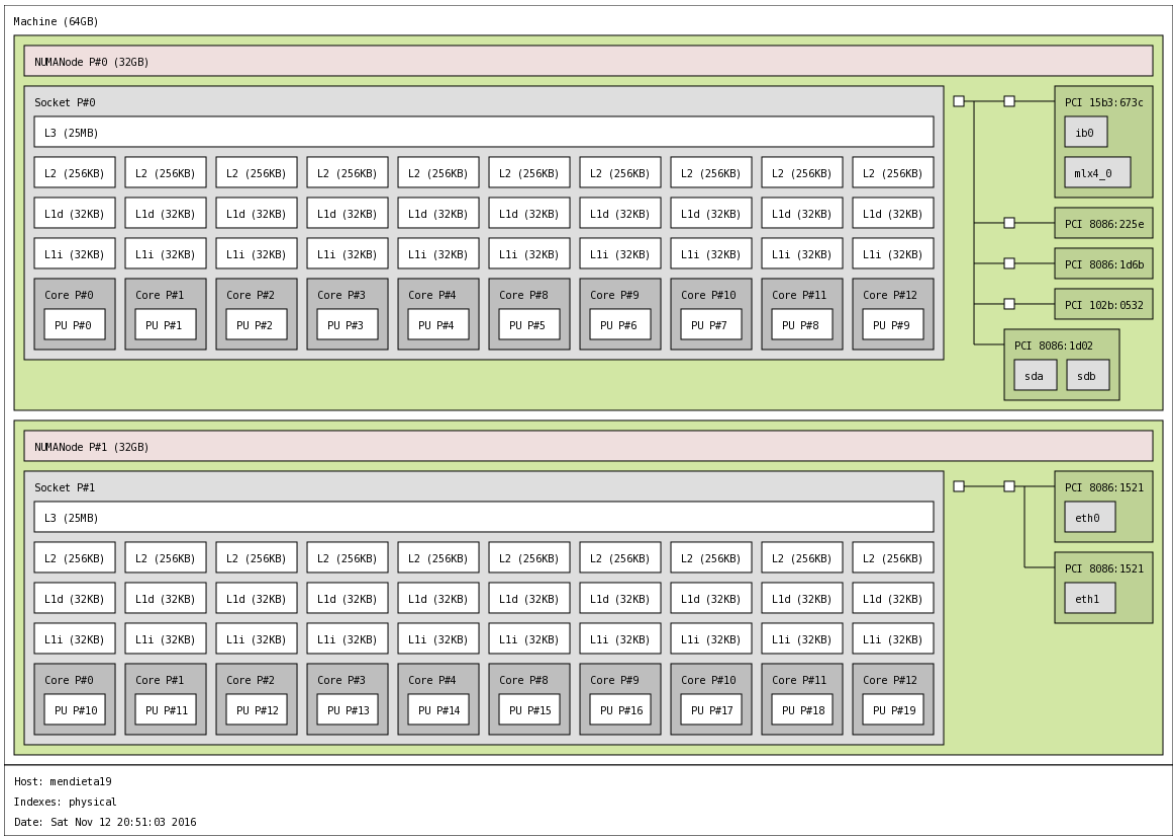


Figura 3.6: Topología de un nodo de la partición muti de Mendieta

4. Herramientas de paralelismo aplicados

4.1. Mendieta

A continuación, se detalla brevemente la estructura del clúster de cómputo de Mendieta.
La estructura actual de Mendieta (diciembre - 2017) es similar a la presentada en la figura 4.1.

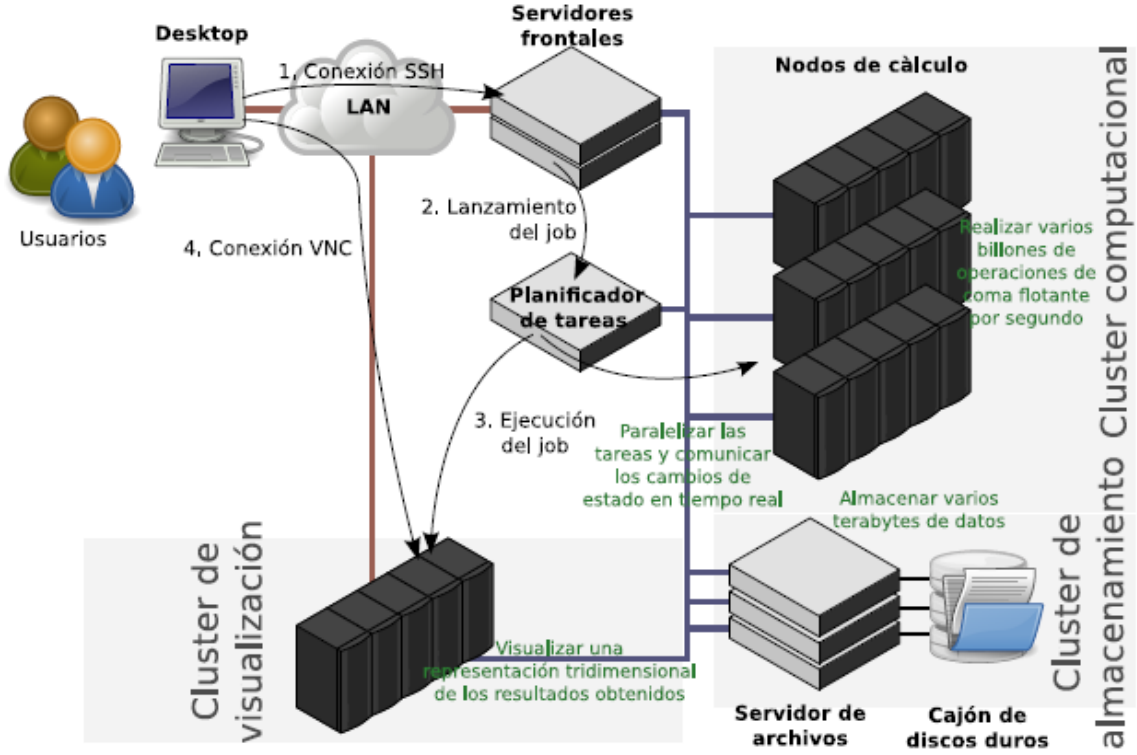


Figura 4.1: Estructura de un clúster similar a Mendieta

Mendieta esta compuesto actualmente por 23 nodos dedicados a cómputo [27], denominados Mendieta01 hasta Mendieta23.
Se accede al clúster remotamente a través de SSH a el servidor header (Mendieta), en el cual se encuentran las cuentas de cada usuario.
Los nodos de Mendieta fueron incorporados en dos etapas [28]:

Primera etapa, 8 nodos:

- 2 procesadores Intel Xeon E5-2680 de 8 cores cada uno a 2.7 GHz.
- 64 GiB DDR3 a 1333 MHz.

Segunda etapa, 14 nodos:

- 2 procesadores Intel Xeon E5-2680 v2 de 10 cores cada uno a 2.8 GHz.
- 64 GiB DDR3 a 1600 MHz.

Aceleradoras:

- 12 aceleradoras NVIDIA Tesla M2090 con 6 GiB de memoria GDDR5.
- 8 aceleradoras NVIDIA Tesla K20x con 6 GiB de memoria GDDR5.
- 14 aceleradoras Intel Xeon Phi 31S1P, con 8 GiB de memoria GDDR5.

Cuenta además con servidores de archivos (NAS), los cuales guardan los datos producidos por las computaciones. Para ejecutar programas que requieran más de un nodo, estos se comunican entre sí a través de una red de alta velocidad Infiniband QDR que proporciona una tasa pico de transferencia de datos de 40Gb/s.

La primer etapa del mismo entró en funcionamiento el 7 de Mayo de 2013 con fondos aportados por la Facultad de Matemática, Astronomía y Física (FaMAF) de la Universidad Nacional de Córdoba (UNC), y los Instituto de Física Enrique Gaviola (IFEG) e Instituto de Astronomía Teórico y Experimental (IATE), dependientes del CONICET.
La segunda etapa licitada y adjudicada por el Centro de Computación de Alto Desempeño (CCAD) de la UNC, con fondos provenientes del Sistema Nacional de Computación de Alto Desempeño (SNCAD) del Ministerio de Ciencia y Tecnología, fue puesta en funcionamiento el 6 de junio de 2014.

4.2. Lista de herramientas y tecnologías utilizadas en Mendieta

A continuación, se detalla la lista de herramientas y tecnologías que se utilizaron en Mendieta para poder instalar, ejecutar y analizar performance de WRF.

- SLURM
- Environment Modules

- MPI
- OpenMP
- ZLIB
- HDF5
- NetCDF
- NetCDF-Fortran
- WPS
- WRF
- ARWPost
- GrADS
- Perf
- Numactl
- Numastat

4.2.1. Simple Linux Utility for Resource Management (SLURM)

Este software está instalado en Mendieta y es la interface entre el usuario y el sistema de cómputo. Provee un conjunto de instrucciones y comandos que permiten pedir recursos en el clúster.
SLURM utiliza el concepto de *partición* para agrupar los nodos que el clúster provee.
Luego de definir qué partición usar, se deben setear algunas configuraciones en archivos de *shell scripts* ***.sh*** que determinan la ejecución del programa. Esta ejecución se denomina ***job***.
Dependiendo de la arquitectura de los nodos, SLURM define para Mendieta las siguientes particiones:[6]

- Multi: 14 nodos con 20 cores por nodo (máximo hasta 8 nodos por job).
- Phi: 14 nodos con 57 cores por nodo, 7 GiB de RAM.
- Mono: 9 nodos con 16 cores por nodo (sólo se puede pedir un nodo por job).
- GPU: 8 nodos de la partición Mono, los cuales cuentan además con placas GPU.

[6] Las particiones no son disjuntas.

Los recursos y comandos de SLURM a ejecutarse en el ***job*** se deben definir tambien en un *shell script.* Un ejemplo de pedido de recursos para WRF en 2 nodos (con un total de 40 cores) se puede ver a continuación en el script ***job_wrf.sh***:

```
#SBATCH --mail-user=lvc0107@famaf.unc.edu.ar
#SBATCH --job-name=WRF
#SBATCH --partition=multi
#SBATCH --exclusive

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20

### Execution Time. Format: days-hours:minutes. Max time: Four days.
#SBATCH --time 0-2:30
```

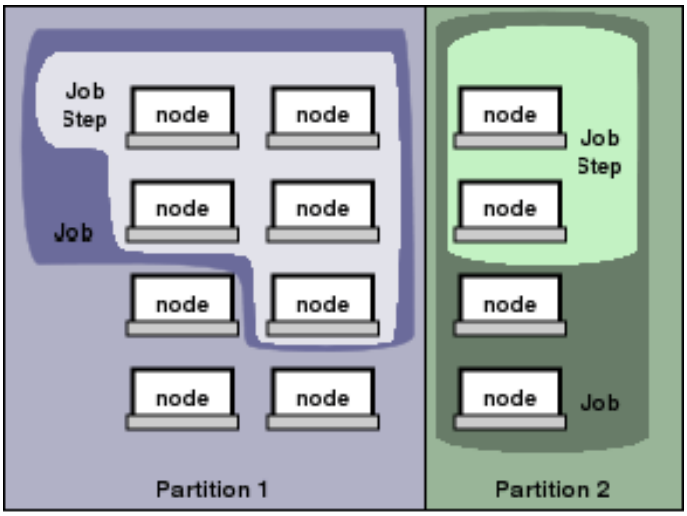


Figura 4.2: Modelo de particiones que utiliza SLURM

SLURM utiliza el concepto de cola de trabajos. Cuando un job solicita recursos del clúster, SLURM chequea si existen nodos libres para ser utilizados. En caso de que existan nodos disponibles se los asigna al job que los solicitó, en caso contrario deja el job en una cola de espera hasta que los recursos estén disponibles.
Los comandos utilizados en este trabajo son los siguientes:

sbatch: Se ejecuta desde CLI pasando como parámetro el Shell script con la secuencia de comandos que SLURM debe ejecutar.

srun: Ejecuta el job en la partición definida en el Shell script.
squeue: Muestra el estado de los Jobs en la cola de trabajo.
salloc: Este comando posibilita pedir recursos de manera interactiva a través de la CLI, en lugar de hacerlo a través de un Shell script.

4.2.2. Environment Modules

Este software también se encuentra instalado en Mendieta. Es una herramienta que permite cargar variables de entorno de manera dinámica. Es necesaria utilizarla para cargar en el entorno de ejecución variables con los path que apunten a las herramientas instaladas en Mendieta como NetCDF, HDF5 y OpenMPI. Si no se cargan dichas variables no se pueden usar las dependencias que necesita WRF.

4.2.3. MPI

MPI [29] es la biblioteca que posibilita realizar la computación distribuida a través de todos los nodos del clúster. Es una especificación estándar basada en el modelo comunicación de procesos a través de pasaje de mensajes. Cuenta con varias implementaciones tales como OpenMPI, Intel MPI, MVAPICH entre otras. En este momento Mendieta cuenta con la implementación OpenMPI compilada con gcc.

La figura 4.3 muestra un esquema de cómo se comunican dos procesos ejecutados en máquinas A y B a través de la APIs *send()* y *recv()*.

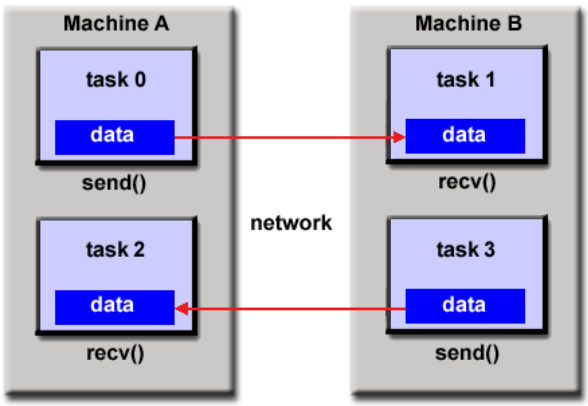


Figura 4.3: Modelo de MPI

4.2.4. OpenMP

OpenMP provee una biblioteca y lenguaje de decoración para dotar de paralelismo a nivel de datos usando threads de una manera eficiente y rápida utilizando directivas de compilación: `#pragma omp parallel` para el language C o `!$OMP PARALLEL DO` para el caso del lenguage Fortran, en secciones de código que ejecuten loops. Se basa en el modelo fork-join, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K threads (fork) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (join). El código que se ejecuta en paralelo es sincronizado a través de barreras.

Los compiladores instalados en Mendieta (gcc, icc, gfortran, ifort, etc) implementan su biblioteca de OpenMP.

Es importante destacar que, en un clúster compuesto de múltiples nodos, la única forma de distribuir las tareas, es a través del uso de procesos MPI, el modelo de OpenMP no escala a más de un nodo.

Un ejemplo de código OpenMP escrito en Fortran es el siguiente:

```
program omp_par_do
  implicit none

  integer, parameter :: n = 100
  real, dimension(n) :: dat, result
  integer :: i

  !$OMP PARALLEL DO
  do i = 1, n
     result(i) = my_function(dat(i))
  end do
  !$OMP END PARALLEL DO

contains

  function my_function(d) result(y)
    real, intent(in) :: d
    real :: y

    ! do something complex with data to calculate y
  end function my_function

end program omp_par_do
```

El código descrito ejecuta de forma paralela la función *my_function(d)* para cada elemento *d* de la estructura de datos *dat*.

4.2.5. WRF y sus dependencias

Para la implementación de WRF se usaron los siguientes programas:

- Pre-procesamiento: WPS3.8
- Procesamiento: WRF3.8
- Post-procesamiento: ARWpost_V3

Cuyas dependencias son las siguientes:

Instaladas en Mendieta:

- Zlib
- NetCDF/NetCDF-Fortran
- HDF5
- OpenMPI

No instaladas en Mendieta:

- JasPer: Herramienta adicional para pre-procesamiento
- GrADS: Herramienta adicional para post-procesamiento

WRF manipula sus datos utilizando formatos provistos por el conjunto de bibliotecas NetCDF [30]. Estos formatos son provistos a través de bibliotecas estáticas escritas en C *(libnetcdf.a*) y fortran (libnetcdff.a). Para la implementación de WRF necesitamos linkear contra las bibliotecas escritas en fortran usando el flag de linkeo **-lnetcdff**.
NetCDF soporta tres tipos de formatos, entre los cuales se destaca el formato para archivos de datos HDF5 que es el que utiliza WRF. En la figura 4.2 se puede visualizar la estructura de NetCDF, en a cuál se destacan los módulos y dependencias que se utilizaron para WRF. Este conjunto de bibliotecas se encuentra disponible en Mendieta y puede accederse a través del gestor de módulos ejecutando el siguiente comando:

```
module load netcdf-fortran/4.4.4
```

La arquitectura de NetCDF, junto a las dependencias utilizadas se remarcan en la siguiente figura:

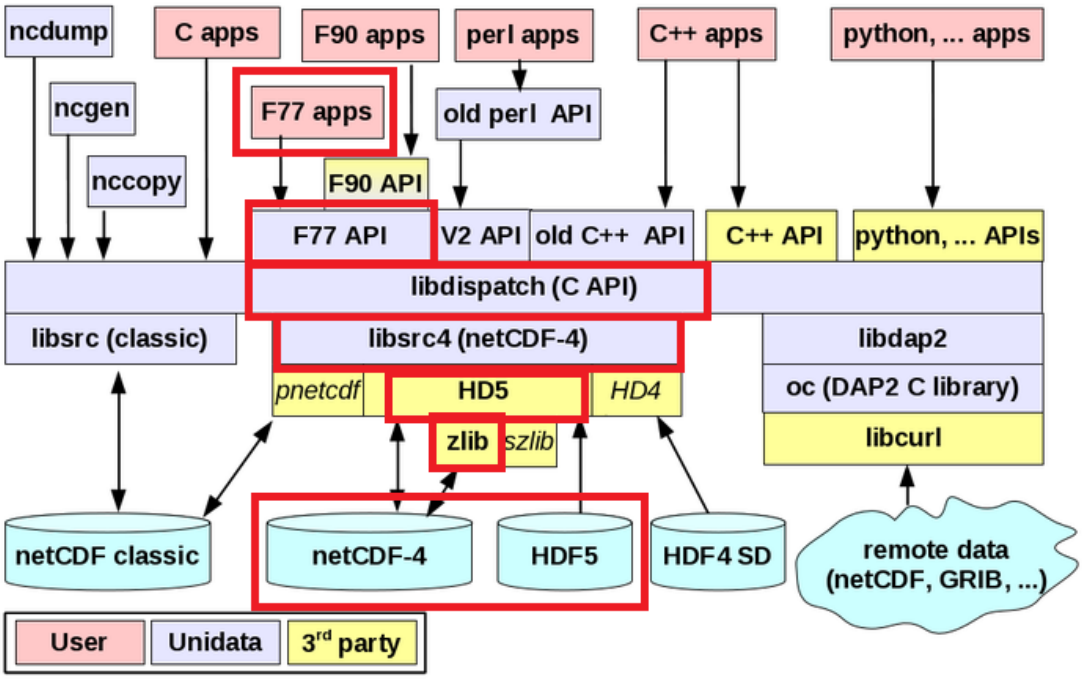


Figura 4.4: Arquitectura de NetCDF

De todas maneras, en el trabajo realizado en la presente tesis se instaló manualmente para obtener el conocimiento necesario para una instalación de WRF sin contar con herramientas provistas por Mendieta. De la misma manera se procedió con la instalación manual del resto de las dependencias.

Las dependencias de NetCDF instaladas son las siguientes:

- NetCDF-Fortran (F77/F90) [31]: Es el módulo de NetCDF que provee las bibliotecas escritas en fortran las que son finalmente utilizadas desde WRF.
- HDF5: Hierarchical Data Format Version 5, es el formato de archivos diseñado para almacenar y organizar grandes cantidades de datos. Es una dependencia necesaria para NetCDF. Provee el formato para manipulación de datos.
- ZLIB: Es una herramienta de compresión utilizada para comprimir datos con formato HDF5. Esta herramienta es necesaria para la manipulacion de datos de WRF, dedido al gran volumen de datos que procesa.

JasPer: Este software es una dependencia utilizada por HDF5 y es utilizada en etapa de pre-procesamiento de WRF (WPS). El proyecto JasPer es una iniciativa de código abierto para proporcionar una implementación basada en software del códec especificado en el estándar JPEG-2000 Parte-1 (es decir, ISO / IEC 15444-1). El software JasPer está escrito en lenguaje de programación C. Más detalles sobre este software se pueden encontrar en el Manual de Referencia de JasPer Software.

GrADS: Como se señaló en la sección 2.10. se necesita instalar esta herramienta para poder procesar y visualizar los pronósticos generados por WRF/ARWpost.

4.2.6. Herramientas de análisis de performance:

Las herramientas mencionadas son utilidades para analizar la performance de WRF durante y al final de la ejecución. Se utilizaron las siguientes herramientas:

- Perf: herramienta de profiling para obtener métricas de la ejecución como cantidad de instrucciones ejecutadas, aciertos de cache, saltos mal predichos, etc.
- Numactl: Herramieta que controla la asigación de memoria utilizada por cada nodo NUMA.
- Numastat: Herramienta para medir el uso de memoria en una arquitectura NUMA.

5. Implementación de WRF en Mendieta

5.1. Definición del dominio de simulación

Para las condiciones iniciales y de contorno se utilizaron datos obtenidos desde GFS con una resolución horizontal (longitud x latitud) de grilla de 0.25º. La resolución 0.25° provista por GFS se corresponde a celdas de aproximadamente a 25km^2. Según la tabla siguiente provista por el NCEP, esta es la máxima resolución disponible que provee el sistema GFS en la actualidad.

Época	Resolución (grados)	Resolución (km)
1970s - 1980s	2-3	200-500
1990s - 2010	1	100
2010s - actualidad	0.2-0.5	25-60

Tabla 5.1: Mapeo de grados a kilómetros en resoluciones horizontales

El dominio de simulación determinado para las pruebas de WRF en Mendieta abarca la región central de la República Argentina, haciendo enfoque en la provincia de Córdoba. El dominio se compone de una grilla de 300 * 250 celdas de 4 km de resolución. Es decir que durante el proceso de pre-procesamiento de WRF la resolución se incrementa aún más al pasar de un modelo global (GFS) a uno local (WRF), pasando de celdas de aproximadamente 25 Km^2 a celdas de 4 km^2.

La imagen 5.1 muestra el dominio de simulación seleccionado para los experimentos en la presente tesis. La misma fue generada con la herramienta GrADS.

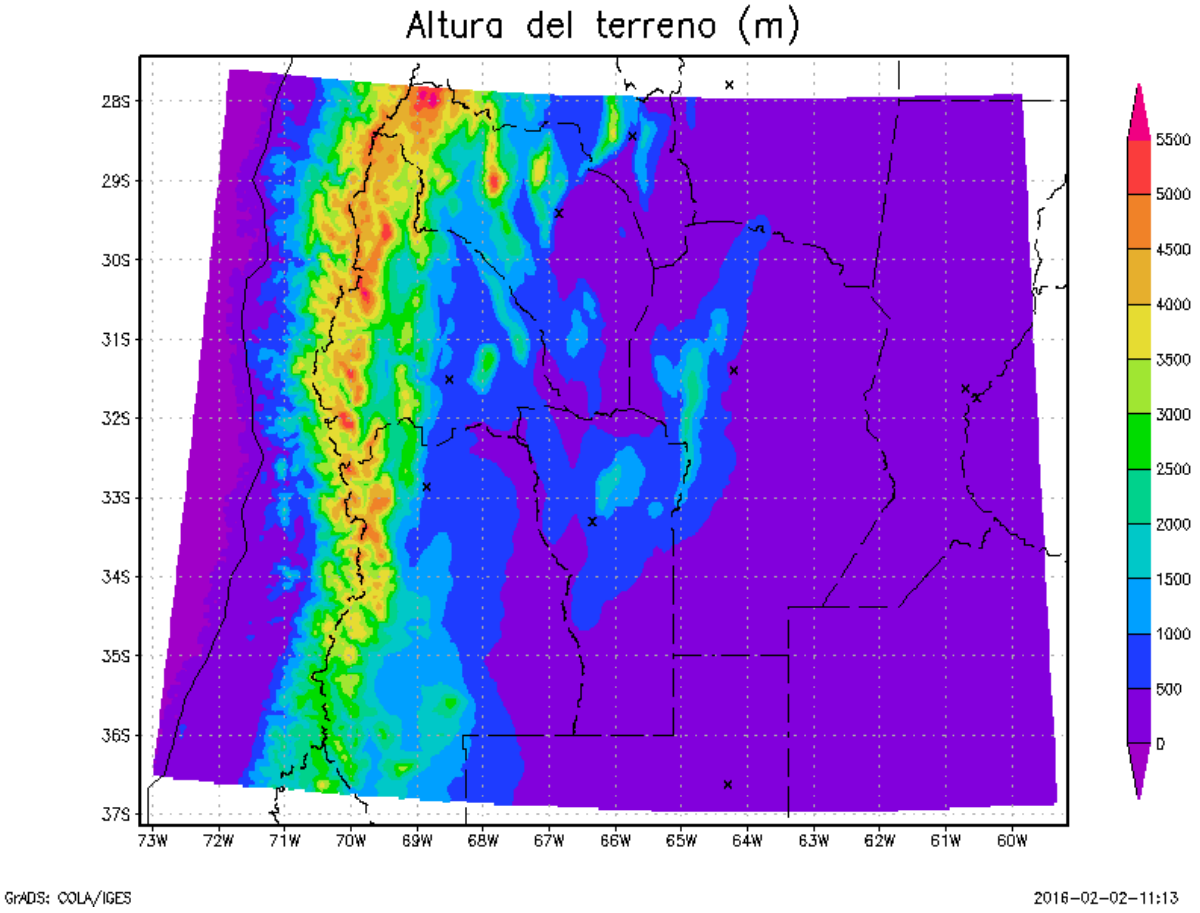


Figura 5.1: Dominio de simulación

WRF permite modificaciones en las parametrizaciones de cada módulo, los cuales modelan los fenómenos físicos y sus interacciones. WRF tiene a disposición una amplia gama de opciones para cada modulo parametrizado [32] y como se explicó en la sección 2.7 simular una situación on diferentes combinaciones de parametrizaciones generará diferentes resultados.

Estos diferentes resultados pueden ser tratados como diferentes miembros de un ensamble.

Las parametrizaciones son definidas al inicio de la simulación a través de una asignación de valores. Cada asignación de valor se corresponde con una diferente parametrización dada por el manual de WRF.[45]

En esta tesis se trabajó con 5 diferentes conjuntos de parametrizaciones (5 miembros), que si bien es un número pequeño para considerarlo como un ensamble, las técnicas que se desarrollaron son extrapolables a un número mayor de miembros.

El motivo de la selección de los miembros no es relevante para este trabajo ya que se apunta a analizar la performance de la ejecución paralela de estos en el clúster.

Para la presente simulación se realizaron pruebas variando las siguientes variables, aunque no existió un motivo especial en la selección de dichos parámetros, sino que se seleccionaron con el propósito de analizar la performance de múltiples miembros del ensamble ejecutando en paralelo en el clúster:

- mp_physics: Opción para modelar microfísica
- sf_sfclay_physics: Opción para modelar capa superficial
- bl_pbl_physics: Opción para modelar capa límite planetaria

Para el modelado de la microfísica se seleccionaron los siguientes esquemas:

- Esquema de microfísica de Thompson [33] (G. Thompson, P. Field, R. Rasmussen, y W. Hall).
- Esquema de microfísica de Morrison de dos momentos. [34] (H. Morrison y G. Thompson).
- Esquema de microfísica de momento doble – clase 6. [35] (WRF Double Moment 6 class, WDM6, Hong y Lim).
- Esquema de microfísica de momento simple – clase 6. [36] (WRF Single moment 6 class, WSM6, Hong y Lim).

Para el modelado de la la capa límite planetaria se seleccionaron los siguientes esquemas:

- Esquema de Mellor-Yamada-Janjic (MYJ, 1994) [37].
- Esquema de Eliminación de escala cuasi-normal (QNSE Sukoriansky, Galperin y Perov, 2005) [38].
- Esquema de Mellor-Yamada- Nakanishi Niino, nivel 3 (MYNN3) [39].

Para el modelado de las capas superficiales se seleccionaron los siguientes los siguientes esquemas:

- Esquema de similaridad Eta (A. Monin y A. Obukhov) [40].
- Esquema de Eliminación de escala cuasi-normal (QNSE Sukoriansky, Galperin y Perov, 2005) [38].
- Esquema de Mellor-Yamada- Nakanishi Niino, nivel 3 (MYNN3) [39].

Finalmente, el ensamble definido quedó conformado por las configuraciones descritas en la tabla 5.1. Cada fila en la tabla es la configuración de un miembro del ensamble y esta identificado alfabéticamente: [A-E].

Miembro de ensamble	mp_physics/ Opción en WRF	bl_pbl_physics/ Opción en WRF	sf_sfclay_physics/ Opción en WRF
A	Thompson/8	MYJ/2	Eta/2
B	Morrison/10	MYJ/2	Eta/2
C	WDM6/16	QNSE/4	QNSE/4
D	WSM6/6	MYJ/2	Eta/2
E	WDM6/16	MYNN3/6	MYNN3/5

Tabla 5.2: Parametrizaciones de miembros del ensamble para WRF

5.2. Estructura de WRF en Mendieta

Una vez definido el ensamble a través de cada uno de sus miembros se procedió a definir la estructura de la instalación de WRF en Mendieta. Como se ha explicado anteriormente la posibilidad de contar con una arquitectura de computación paralela como el clúster Mendieta posibilita poder ejecutar cada miembro de ensamble concurrentemente.

La ejecución de WRF aloca recursos para un job por cada miembro de ensamble, de esa manera, si existen los suficientes recursos disponibles cada miembro de ensamble puede ejecutarse paralelamente en los nodos alocados por su job.

La siguiente figura muestra un ejemplo de una posible ejecución de cuatro miembros de ensamble en dos nodos de la partición *multi* cada uno.

Notar que uno de los miembros de ensamble (C_WDM6_MYJ) no se ejecuta y queda encolado a la espera de recursos.

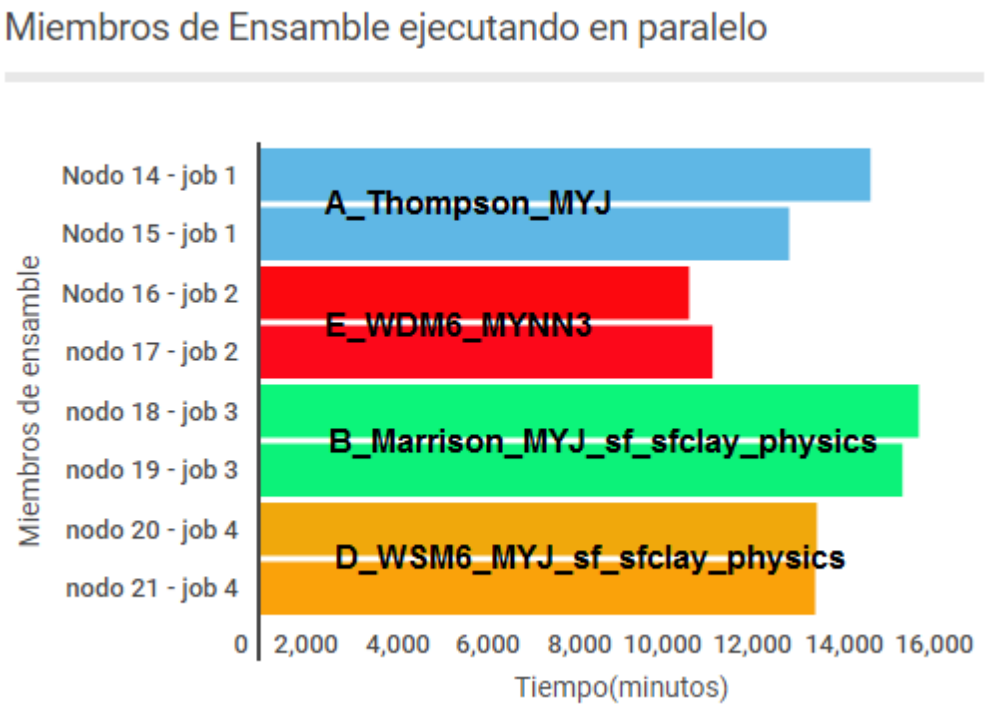


Figura 5.2: Miembros de ensamble ejecutando en paralelo

Así como hay un nivel de paralelismo entre los miembros del ensamble, es decir jobs ejecutando en paralelo, cada job a su vez extrae paralelismo en los niveles TLP, DLP e ILP.

El proceso completo de ejecución de los miembros de ensambles se detalla en la figura 5.3. Inicia con la descarga de inputs desde GFS, donde se definen las condiciones iniciales y de contorno comunes a todos los miembros de ensambles. Luego los jobs se van ejecutando a medida que Mendieta cuente con recursos disponibles.

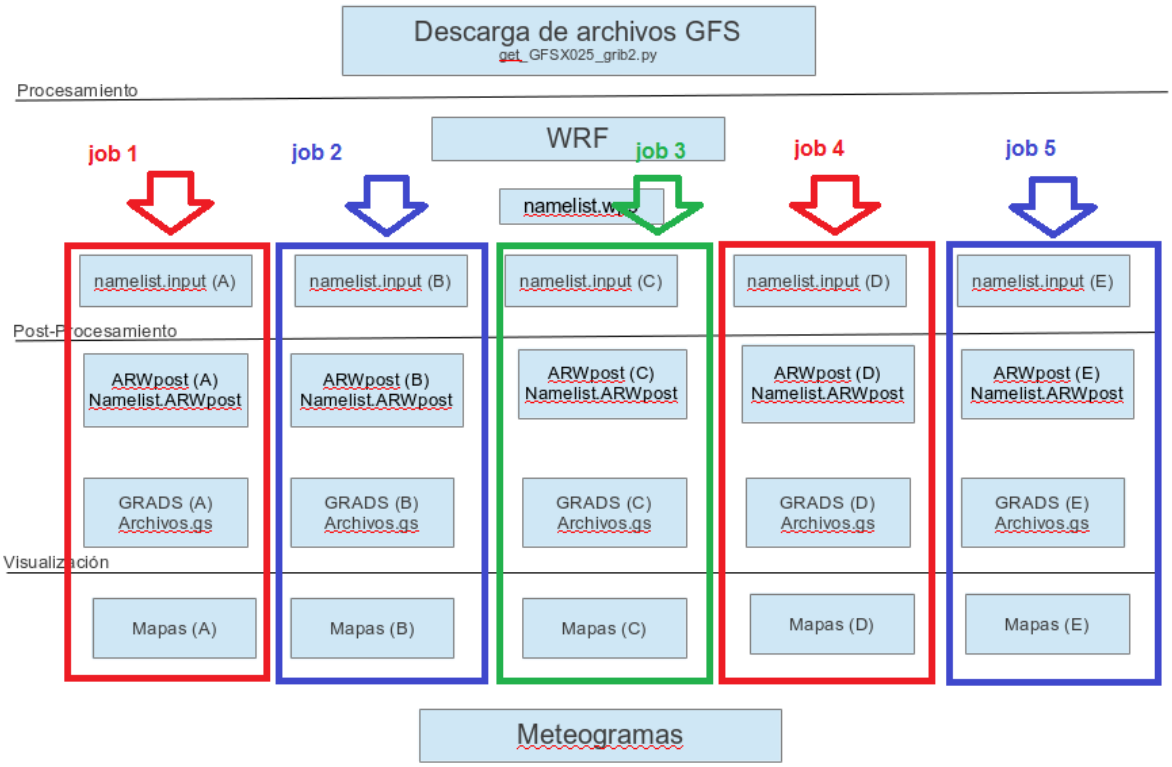


Figura 5.3: Esquema de la implementación de WRF en Mendieta

La ejecución de los miembros de ensamble está automatizada, aunque la configuración de parámetros iniciales de cada miembro del ensamble y alcance del pronóstico, debe ser definida manualmente por parte del usuario antes de ejecutar WRF. En la siguiente sección se detalla cómo modificar los archivos necesarios manualmente.

El código desarrollado para automatizar la ejecución del modelo WRF en Mendieta está versionado y documentado en el repositorio de acceso público:

https://github.com/lvc0107/wrf_mendieta

Dicho repositorio contiene las instrucciones detalladas para la instalación de WRF en Mendieta junto con todas sus dependencias.

Como parte de la investigación del presente trabajo, WRF también se instaló y puso a prueba en otros entornos a efectos de tener puntos de comparación con respecto a la performance en distintas arquitecturas computacionales, por lo que también se encuentra disponible en la documentación del repositorio instrucciones para la instalación de WRF en otras arquitecturas.

5.3. Archivos configurables por el usuario

5.3.1. Creación de entorno para el ensamble

El primer paso para iniciar la ejecución de WRF consta de ejecutar el script ***set_configurarion.sh*.** El mismo está destinado a cargar los módulos y variables correspondientes tanto como para la etapa de compilación como para la etapa de ejecución de WRF.

El script crea el directorio *ensamble,* el cual se encuentra inicialmente vacío y tiene como propósito alojar templates de archivos de configuración para cada miembro del ensamble. Para cada miembro de ensamble se deben crear subdirectorios dentro del directorio *ensamble* con la siguiente estructura:

```
ensamble
├── Miembro1
│   ├── namelist.ARWpost
│   └── namelist.input
├── Miembro2
│   ├── namelist.ARWpost
│   └── namelist.input
├── Miembro3
│   ├── namelist.ARWpost
│   └── namelist.input
│   .
│   .
│   .
├── MiembroN
│   ├── namelist.ARWpost
│   └── namelist.input
├── gradfile1.gs
├── gradfile2.gs
├── .
├── .
├── .
├── gradfileN.gs
└── namelist.wps
```

Los archivos ***namelist.{wps, input, arwpost}*** creados en la estructura de directorios anterior son inputs de configuración necesarios para cada una de las siguientes etapas respectivamente:

- Pre-procesamiento: utiliza el archivo de configuración ***namelist.wps***
- Procesamiento: utiliza el archivo de configuración ***namelist.wrf***
- Post-procesamiento: utiliza el archivo de configuración ***namelist.arwpost***

Tal como han sido creados en la estructura de directorios anterior funcionan como templates. Se deben configurar cada vez que se considere necesario, pero dejándolos siempre dentro del subdirectorio correspondiente al miembro del ensamble. El script que lanza los jobs genera una copia de estos templates, actualiza las fechas de inicio y fin de la simulación y los despliega en los directorios necesarios para que WRF los procese. También crea los siguientes directorios:

- gribfiles: directorio para descarga de inputs desde GFS
- outputs: directorio para resultados de los pronósticos
- logs: directorio para archivos de logs

En el directorio ***ensamble*** también deben alojarse los scripts a ser utilizados por GrADS.
La estructura usada para miembros del ensamble para las simulaciones realizadas es la siguiente:

```
ensamble
├── A_Thompson_MYJ
│   ├── namelist.ARWpost
│   └── namelist.input
├── B_Marrison_MYJ_sf_sfclay_physics
│   ├── namelist.ARWpost
│   └── namelist.input
├── cbar.gs
├── C_WDM6_QNSE_sf_sfclay_physics
│   ├── namelist.ARWpost
│   └── namelist.input
├── D_WSM6_MYJ_sf_sfclay_physics
│   ├── namelist.ARWpost
│   └── namelist.input
├── E_WDM6_MYNN3
│   ├── namelist.ARWpost
│   └── namelist.input
├── HPC_CBA_Rain.gs
├── HPC_CBA_Tmax_Min.gs
├── meteogramas_Preciptation.gs
├── meteogramas_rh.gs
├── meteogramas_Temp.gs
├── meteogramas_WindDir.gs
├── meteogramas_WindSpeed.gs
├── namelist.wps
└── rGiBset.gs
```

5.3.2. namelist.wps

Este es el archivo de configuración para la etapa de pre-procesamiento. Es compartido por todos los miembros de ensamble. Las fechas son actualizadas automáticamente por el script ***run_wrf_model.py*.** Se debe indicar el path donde son descargados los inputs (gribfiles) que toma WPS (ver Apéndice B.3).

```
cd $WRF_BASE/ensamble
cat namelist.wps


&share
 wrf_core = 'ARW',
 max_dom = 1,
 start_date = 2016-10-20_00:00:00
 end_date = 2016-10-21_12:00:00
 interval_seconds = 10800
 io_form_geogrid = 2,
/

&geogrid
 parent_id         =   1,   1,
 parent_grid_ratio =   1,   3,
 i_parent_start    =   1,   37,
 j_parent_start    =   1,   83,
 e_we              =   300,  61,
 e_sn              =   250,  91,
 geog_data_res     =   '30s','30s',
 dx = 4000,
 dy = 4000,
 map_proj = 'lambert',
 ref_lat   = -31.4,
 ref_lon   = -64.1,
 truelat1  = -60.0,
 truelat2  = -30.0,
 stand_lon = -63.6,
 geog_data_path = '/home/lvargas/wrf_mendieta/geog'
/

&ungrib
 out_format = 'WPS',
 prefix = 'GFS25',
/

&metgrid
 fg_name = 'GFS25'
 io_form_metgrid = 2,
/
```

Notar que es en este archivo donde se debe setear parámetros importantes donde se destacan:

- e_we: Dimensión en unidad de km este-oeste. Debe ser menor a la coordenada definida en el script de descarga de gribfiles. Para la simulación realizada se definió el valor de 300km. La segunda columna no es usada.

- e_sn: Dimensión en unidad de km sur-norte. Deben ser menor a la coordenada definida en el script de descarga de gribfiles. Para la simulación realizada se definió el valor de 250km. La segunda columna no es usada.
- ref_lat: coordenada latitud del centro de la simulación (Valor para ciudad de Córdoba).
- ref_lon: coordenada longitud del centro de la simulación (Valor para ciudad de Córdoba).
- geog_data_path: Directorio para instalación de datos estáticos para física de la tierra y utilizados por geogrid.exe

5.3.3. namelist.input

Los archivos ***namelist.input*** son la base para la definición de un miembro de ensamble. En él se definen las parametrizaciones discutidas previamente. Deben configurarse manualmente en la sección *physics* usando las configuraciones definidas en la tabla 5.2.
Las fechas son actualizadas automáticamente por el script ***run_wrf_model.py*.**

Ejemplo de la configuración para el miembro de ensamble *A_Thompson_MYJ*. La segunda y tercer columnas no son utilizadas en la presente ejecución.

Las fechas son actualizadas automáticamente por el script ***run_wrf_model.py*.**

```
&physics
 mp_physics                          = 8,     2,     2,
 ra_lw_physics                       = 1,     1,     1,
 ra_sw_physics                       = 2,     1,     1,
 radt                                = 4,    30,    30,
 sf_sfclay_physics                   = 2,     1,     1,
 sf_surface_physics                  = 2,     2,     2,
 bl_pbl_physics                      = 2,     1,     1,
```

5.3.4. miembro*i*/namelist.ARWpost

Son los archivos de configuración para la etapa de post-procesamiento. (Para cada miembro del ensamble). Las fechas son actualizadas automáticamente por el script **r*un_wrf_model.py***.

Este script también actualiza los path a los directorios inputs y output para ARWpost.

Los campos *field* e *interp_levels* deben configurarse manualmente y contienen parámetros que producen outputs que son tomados por GrADS en la última etapa de post procesamiento.

```
cd $WRF_BASE/ensamble
cat ensamble/A_Thompson_MYJ/namelist.ARWpost

&datetime
 start_date = 2016-10-20_00:00:00
 end_date = 2016-10-21_12:00:00
 interval_seconds = 3600,
 tacc = 0,
 debug_level = 0,
/

&io
 input_root_name = '../wrf_run/wrfout_d01_2016-10-20_00:00:00',
 output_root_name = './output/output'
 plot = 'all_list'
 fields = 'height,pressure,tk,tc,rh2,wd10,ws10'
 mercator_defs = .true.
/
 split_output = .true.
 frames_per_outfile = 2

 plot = 'all'
 plot = 'list'
 plot = 'all_list'
! Below is a list of all available diagnostics
 fields =
'height,geopt,theta,tc,tk,td,td2,rh,rh2,umet,vmet,pressure,u10m,v10m,wdir,wspd,wd10
,ws10,slp,mcape,mcin,lcl,lfc,cape,cin,dbz,max_dbz,clfr'


&interp

 interp_method = 0,
 interp_levels =
1000.,950.,900.,850.,800.,750.,700.,650.,600.,550.,500.,450.,400.,350.,300.,250.,20
0.,150.,100.,
/
extrapolate = .true.

 interp_method = 0,     ! 0 is model levels, -1 is nice height levels, 1 is user
specified pressure/height
```

```
 interp_levels =
1000.,950.,900.,850.,800.,750.,700.,650.,600.,550.,500.,450.,400.,350.,300.,250.,20
0.,150.,100.,
 interp_levels = 0.25, 0.50, 0.75, 1.00, 2.00, 3.00, 4.00, 5.00, 6.00, 7.00, 8.00,
9.00, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0,
```

5.3.5. set_configuration.sh

Como se mencionó anteriormente, este archivo es un script que tiene como propósito cargar los módulos provistos por *environment modules* para poder compilar y ejecutar WRF. Dichos módulos cargan entre otras cosas: compilador, bibliotecas de MPI, etc. Por defecto el compilador cargado es *gcc* y la librería de MPI utilizada es OpenMPI, aunque eventualmente se podrían usar otras opciones como *mvapich* e *icc* respectivamente. Sin embargo, si se desea probar otro compilador, deben compilarse todas las dependencias de nuevo, cargando el modulo correspondiente en el script *set_configuration.sh* y volver a realizar todos los pasos desde el paso 1 (Instalación de WRF y dependencias).

El script también crea directorios para outputs, meteogramas y gribfiles en caso de que no existan.

Para ver los módulos que cargados por el script *set_configuration.sh* se ejecuta la siguiente línea de comando:

```
[lvargas@mendieta wrf_mendieta]module
Currently Loaded Modulefiles:
  1) /opt/modules/sistema/libgcc/5
  2) hwloc/1.11.4
  3) netcdf/4.4.1.1
  4) binutils/2.27
  5) openmpi/2
  6) netcdf-fortran/4.4.4
  7) gcc/5
  8) hdf5/1.10.0p1
```

5.3.6. Configuración para ejecutar OpenMP

Para ejecutar WRF en un único nodo o por ejemplo en una máquina como JupiterAce[7] existe la posibilidad de usar threads de OpenMP en lugar de procesos MPI.
Para ejecutar WRF utilizando OpenMP es necesario compilar previamente seleccionando la opción 33 de memoria compartida (smpar: shared memory parallelism), en etapa de configuración.

```
[lvargas@mendieta WRFV3]./configure

  33.  x86_64 Linux, gfortran compiler with gcc   (smpar)
```

También es necesario pedir recursos con SLURM usando una configuración especial para activar y declarar el uso de OpenMP.
El archivo a modificar para la ejecución de WRF usando OpenMP en un nodo o en una máquina como JupiterAce para correr por ejemplo 12 threads es ***job_run_1_nodes.sh***

```
mvargas@jupiterace:~/wrf_mendieta$ cat job_wrf_1_nodes.sh

#SBATCH --nodes=1
#SBATCH --ntasks=1

### Hilos por proceso
### Poner el mismo valor acá que en OMP_NUM_THREADS/MKL_NUM_THREADS
#SBATCH --cpus-per-task=12

export OMP_NUM_THREADS=12
export MKL_NUM_THREADS=12
```

5.3.7. Archivos de post-procesamiento

En el directorio $WRF_BASE/ensamble se deben crear además los archivos de scripts necesarios para visualizar los resultados producidos por *ARWpost.exe*. Para este trabajo se utilizó la herramienta GrADS la cual ejecuta scripts con extensión *.gs.*

El usuario de Mendieta debe determinar en estos scripts las variables a extraer, denominados meteogramas y los mapas o demás archivos de visualización del pronóstico.

Ejemplo: Archivo *HPC_CBA_rain.gs*

[7] La arquitectura de esta máquina se detalla en la siguiente sección.

Este script genera un mapa del pronóstico de la lluvia para la provincia de Córdoba. Se ejecuta una vez por cada miembro del ensamble, es decir, genera cinco mapas con pronósticos de lluvias en Córdoba. Es ejecutado automáticamente en etapa de post-procesamiento. El script ***run_wrf_model.sh*** ejecuta todos los archivos de GrADS y genera sus outputs en directorios temporales.

En este script se destaca la definición del dominio de simulación (latitud y longitud) y la definición de arrays de valores que GrADS necesita para generar sus mapas. Está fuera del alcance de este trabajo el estudio de esta herramienta y es precondición por parte del usuario conocerla para poder interpretar los datos que genera WRF.

```
[lvargas@mendieta ensamble]cat HPC_CBA_Rain.gs
'reset'
'clear'
'open output.ctl'
'set lat -35.6 -29.4'
'set lon -66.2 -61.0'

**********precipitacion horas previas
'set t 10'
'lluvia1 = rainnc'

**************************Precipitacion día
'set t 34'
'lluvia = rainnc -lluvia1'
'set gxout Shaded'
'./rGiBset'
'set clevs 0 0.5 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 100 150 200 300 500'
'set ccols 0 0 43 81 39 38 37 36 35 34 22 23 24 26 27 28 86 87 82 88 89 49'
'd lluvia'
'draw shp CBA_Linea'
'./cbar'
'draw title Precipitacion acumulada 36 h'
"set display color white"
'printim ./meteogramas/rain36h.png'
'set geotiff ./meteogramas/rain36h'
'set gxout geotiff'
'd lluvia'
'quit'
```

Es importante destacar que como se mencionó en la primera sección de este capítulo, las instrucciones para el uso de estas herramientas se encuentran documentadas en un repositorio público. Para el caso de GrADS también se debe descargar un set de datos (inputs) necesarios para que los scripts de GrADS funcionen correctamente.

5.4. Ejecución del modelo

La ejecución de WRF se realiza con el siguiente comando:

```
./run_wrf_model.py --start_date=STARTDATE --offset=OFFSET --nodes=2
```

Ejemplo: Para ejecutar todos los miembros de ensamble en dos nodos de *multi* (20 cores p/nodo) para un pronóstico de 36 h.[8]

```
./run_wrf_model.py --start_date=2018012000 --offset=36 --nodes=2
```

Este script realiza las siguientes tareas:

1. Descarga gribfiles dada una fecha en el directorio gribfiles creado en la sección anterior.
2. Descarga los datos del día 20/01/2018 a las 00:00 UTC-3 (00:00 h Argentina).

```
[lvargas@mendieta wrf_mendieta]ls -lh gribfiles/2018-01-20_00\:00\:00/
total 336M
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:45 GFS_2018012000+000.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:45 GFS_2018012000+003.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:46 GFS_2018012000+006.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:46 GFS_2018012000+009.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:46 GFS_2018012000+012.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:46 GFS_2018012000+015.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:47 GFS_2018012000+018.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:47 GFS_2018012000+021.grib2
```

[8] Ajustar el tiempo de ejecución del modelo en el script ***job_wrf.sh*** de la forma más precisa possible. Es decir si tras varias ejecuciones se observa por ejemplo un tiempo promedio de 2 h de ejecución, se deberia solicitar un tiempo de ejecución de 2 h mas un tiempo offset acorde (2:30 h, 2:40 h) y no por ejemplo 10 h pues en ese caso se estaría desperidiciando recursos computacionales.

```
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:48 GFS_2018012000+024.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:48 GFS_2018012000+027.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:49 GFS_2018012000+030.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:49 GFS_2018012000+033.grib2
-rw-rw-r-- 1 lvargas lvargas 26M Mar  7 21:49 GFS_2018012000+036.grib2
```

Cada archivo descargado pesa en promedio 26 MB. Dado que se descargan 13 archivos se tiene un aproximado 336 MB descargados en concepto de inputs para WRF.

3. Actualiza fecha en el archivo template *namelist.wps* en el directorio *ensamble.*

4. Actualiza fecha en los archivos template *namelist.input* dentro de cada directorio ensamble/miembro*i* con i:{1..N}.

5. Actualiza fecha en *namelist.ARWpost* dentro de cada directorio ensamble/miembro*i* con i:{1..N}.

6. Asigna un *job id* a cada miembro de ensamble *y s*olicita recursos (dos nodos en este caso) para la ejecución en los nodos de la partición *multi*. Si no hay recursos, las peticiones de recursos quedan encoladas hasta que algunos nodos de la partición *multi* se liberen. Si hay recursos disponibles, SLURM asigna dichos recursos (dos nodos) para cada miembro del ensamble y lanza la ejecución del job.

El output de la ejecución del script ***run_wrf_model.py*** es el siguiente:

```
[lvargas@mendieta wrf_mendieta] ./run_wrf_model.py --start_date=2018012000 --
offset=36 --nodes=2

    __          _______  ______
     \ \        / /  __ \|  ____|
      \ \  /\  / /| |__) | |__
       \ \/  \/ / |  _  /|  __|
        \  /\  /  | | \ \| |
         \/  \/   |_|  \_\_|

Start forecast date: 2018-01-20_00:00:00
End forecast date: 2018-02-21_12:00:00
------------------------------------------------------------------------------
sbatch job_wrf.sh A_Thompson_MYJ 2018-01-20_00:00:00 2018-02-21_12:00:00 2
Submitted batch job 55066
------------------------------------------------------------------------------
```

```
sbatch job_wrf.sh B_Marrison_MYJ_sf_sfclay_physics 2018-01-20_00:00:00 2018-02-
21_12:00:00 2
Submitted batch job 55067
--------------------------------------------------------------------------------
sbatch job_wrf.sh C_WDM6_QNSE_sf_sfclay_physics 2018-01-20_00:00:00 2018-02-
21_12:00:00 2
Submitted batch job 55068
--------------------------------------------------------------------------------
sbatch job_wrf.sh D_WSM6_MYJ_sf_sfclay_physics 2018-01-20_00:00:00 2018-02-
21_12:00:00 2
Submitted batch job 55069
--------------------------------------------------------------------------------
sbatch job_wrf.sh E_WDM6_MYNN3 2018-01-20_00:00:00 2018-02-21_12:00:00 2
Submitted batch job 55070
squeue -u $USER
PARTITION   JOBID PRIO       NAME     USER ST       TIME NO CPU  GRES
NODELIST(REASON)
multi       55066 5002        WRF  lvargas R        0:13  2  40 (null mendieta[17-
18])
multi       55067 5002        WRF  lvargas R        0:13  2  40 (null mendieta[20-
21])
multi       55068 5002        WRF  lvargas PD       0:00  2  40 (null (Resources)
multi       55069 5002        WRF  lvargas PD       0:00  2  40 (null (Resources)
multi       55070 5002        WRF  lvargas PD       0:00  2  40 (null (Resources)
```

Para este ejemplo se ve que al momento de la ejecución del script había cuatro nodos de la partición *multi* disponibles para ser usados y fueron asignados dos por cada job a los miembros de ensamble:

- A_Thompson_MYJ: Usa los nodos *mendieta17* y *mendieta18* en el job 55066.
- B_Marrison_MYJ_sf_sfclay_physics: Usa los nodos *mendieta20* y *mendieta21* en el job 55067.

El resto de los miembros de ensamble queda encolados a la espera de que nodos de la partición multi se liberen.

El ouptut de la ejecución del script ***run_wrf_model.py*** muestra también que se ejecuta el comando squeue -u $USER luego de hacer submit de los jobs (petición de recursos para los miembros de ensamble). Estos jobs están en estado PD (pending) de obtener recursos. Cuando haya nodos disponibles, los jobs que obtengan recursos van a pasar a estado R (running).

El log proporciona también información relevante:

- PARTITION: Partición a la que pertenecen los nodos.
- JOBID: Identificador único del job (ejecución del miembro del ensamble).
- USER: Usuario que lanzó la ejecución.
- NAME: Nombre e identificador del job.
- TIME: Cuando el job está en estado R este valor se actualiza mostrando el tiempo transcurrido de ejecución. Importante: si el tiempo de ejecución es mayor al estimado en SBATCH --time el job se cancela. Por lo tanto, es necesario actualizar ese valor en el script ***job_wrf.sh*** de manera que ese valor sea mayor y correr nuevamente.
- NO: números de nodos asignados.
- CPU: número de cores asignados.
- NODELIST: lista de nodos asignados al job.

En caso de que se haya realizado la descarga de los inputs (gribfiles) correctamente es posible ejecutar solo un miembro de ensamble (por ejemplo, A_Thompson_MYJ) en dos (o más) nodos de multi (20 cores p/nodo) para las mismas fechas de inicio y periodo de 36 h.

```
sbatch job_wrf.sh A_Thompson_MYJ 2018-01-20_00:00:00 2018-02-21_12:00:00 2
```

La ejecución genera logs en los directorios:[9]

```
$WRF_BASE/logs/$RUN_PARAMETERS/$SLURM_JOB_ID
```

La ejecución genera outputs en los siguientes directorios:

```
cd $WRF_BASE
ls -l output/2_nodes_A_Thompson_MYJ/meteogramas/
total 3.1M
-rw-rw-r-- 1 lvargas lvargas  16K Nov  5 06:13 temp_max_A.png
```

[9] Dónde RUN_PARAMETERS está definido en el script ***job_wrf.sh*** con parametro N en {2, ... ,8}.

```
-rw-rw-r-- 1 lvargas lvargas  15K Nov  5 06:13 temp_min_A.png
-rw-rw-r-- 1 lvargas lvargas  26K Nov  5 06:13 rain36h_A.png
-rw-rw-r-- 1 lvargas lvargas  367 Nov  5 06:13 rain_COLONIA_CAROYA_A.txt
-rw-rw-r-- 1 lvargas lvargas  312 Nov  5 06:13 rain_CAPILLA_DEL_MONTE_A.txt
-rw-rw-r-- 1 lvargas lvargas  314 Nov  5 06:13 rain_CANALS_A.txt
-rw-rw-r-- 1 lvargas lvargas  341 Nov  5 06:13 rain_BRINCKMANN_A.txt
-rw-rw-r-- 1 lvargas lvargas  326 Nov  5 06:13 rain_BIALET_MASSE_A.txt
-rw-rw-r-- 1 lvargas lvargas  314 Nov  5 06:13 rain_ARROYO_CABRAL_A.txt
-rw-rw-r-- 1 lvargas lvargas  341 Nov  5 06:13 rain_ARROYITO_A.txt
-rw-rw-r-- 1 lvargas lvargas  324 Nov  5 06:13 rain_ALTA_GRACIA_A.txt
-rw-rw-r-- 1 lvargas lvargas  368 Nov  5 06:13 rain_ALMAFUERTE_A.txt
-rw-rw-r-- 1 lvargas lvargas  338 Nov  5 06:13 rain_ALICIA_A.txt
-rw-rw-r-- 1 lvargas lvargas  314 Nov  5 06:13 rain_ALEJO_LEDESMA_A.txt
```

Para solicitar más nodos (entre dos y ocho), el script ***run_wrf_model.py*** se debe ejecutar de la siguiente manera.

Ejemplos que ejecutan los miembros de ensamble usando 3, 4 y 5 nodos de 20 cores c/u respectivamente:

```
./run_wrf_model.py --start_date=2016102000 --offset=36 --nodes=3
./run_wrf_model.py --start_date=2016102000 --offset=36 --nodes=4
./run_wrf_model.py --start_date=2016102000 --offset=36 --nodes=5
```

Importante: La quota por usuario es de 500 GiB. La instalación de WRF ocupa aproximadamente 100 GiB (mayormente debido a los ~85 GiB al directorio *geog* en $WRF_BASE) Por lo tanto quedan disponibles ~400 GiB. Es necesario entonces borrar los resultados que se van generando periódicamente, luego de su procesamiento.

La etapa de post procesamiento finaliza ejecutando scripts de GrADS, los cuales generan mapas y meteogramas (datos en texto plano) para ser graficados finalmente en una interfaz web.

Mapas: Se generan cinco grupos de mapas distintos correspondientes a cada miembro de ensamble, para las siguientes variables observadas:

- Lluvia (precipitación)
- Temperatura máxima
- Temperatura minima

Meteogramas: Se generan cinco archivos por cada miembro de ensamble con la siguiente información para cada ciudad de Córdoba.

- Lluvia (precipitación)
- Temperatura
- Humedad
- Dirección del viento
- Velocidad del viento

En total se generarán 475 archivos .txt para los meteogramas (5 variables por ciudad - para 95 ciudades).

6. Resultados

6.1. Sistema computacional convencional

Para contrastar las mejoras obtenidas en performance en la ejecución del modelo WRF se tomó como referencia los resultados obtenidos en la implementación de WRF en la unidad CAEARTE. Dicha implementación fue realizada en una máquina con arquitectura convencional[41].

WRF en CAEARTE:

Recurso	Detalles
Sistema operativo	Linux-Ubuntu 14.4
CPU info	Procesador Intel core i7-2600K CPU @ 3.40GHz
Número de cores	4 (Reales: sin HT activado)
Memoria	4GiB Tipo: DDR3 Velocidad:1333 MHz

Tabla 6.1: Arquitectura de máquina en CAEARTE

Arquitectura del procesador core i7 que utiliza el equipo de CAEARTE para ejecutar WRF:

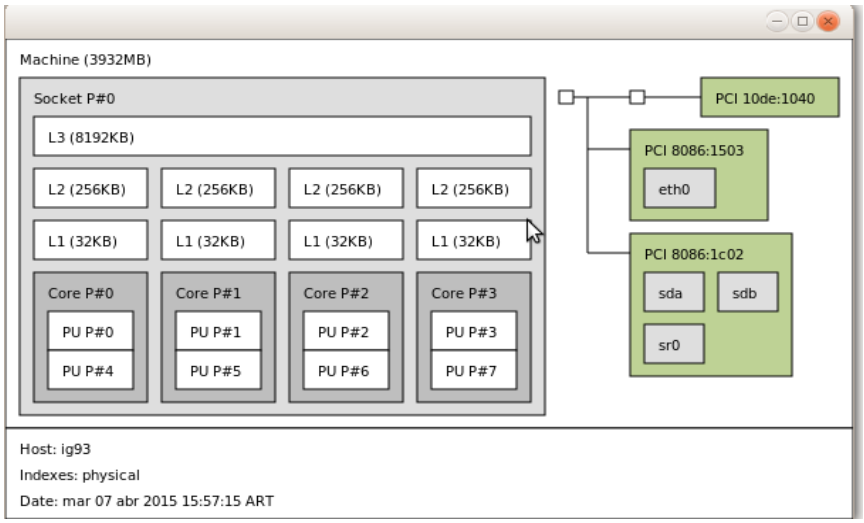


Figura 6.1: Arquitectura de máquina de CAERTAE

Observar la diferencia de arquitectura con respecto a un nodo de la partición ***multi*** de Mendieta detallada en la figura 3.6.
Para una ejecución del Modelo con un pronóstico de 36 h. en la ciudad de Córdoba y usando las configuraciones descritas en la sección 5.2. se obtuvieron los siguientes resultados:

```
real 693m9.766s
user 5200m19.736s
sys 162m0.339s
```

Es decir, el modelo WRF ejecuta en aproximadamente 10.65 h un pronóstico de 36 h de lluvia, temperaturas y humedad para la provincia de Córdoba en la máquina i7 de CAEARTE.

6.2. Pruebas para determinar software más perfomante

Mendieta es constantemente actualizada en términos de hardware y software. En este trabajo se denomina como Mendieta2 al clúster luego de una importante actualización realizada a comienzos de 2014 y la cual utiliza hardware más potente: nodos de la partición *multi* con 2 procesadores Intel Xeon E5-2680 v2 de 10 cores cada uno a 2.8 GHz. - 64 GiB DDR3 a 1600 MHz.
Se denomina al clúster como Mendieta1 cuando hagamos referencia a las pruebas realizadas antes dicha actualización. Y cuyo hardware se compone de nodos con 2 procesadores Intel Xeon E5-2680 de 8 cores cada uno a 2.7 GHz. - 64 GiB DDR3 a 1333 MHz.

La primera etapa de experimentación se basó en pruebas utilizando como dominio el territorio de Córdoba para un pronóstico de 48 h.

Se realizó una serie de pruebas para determinar la mejor combinación de bibliotecas instaladas en Mendieta para la ejecución de procesos en paralelo de WRF. Se realizaron pruebas para las siguientes bibliotecas compiladas con gcc.4-9:

- OpenMP: Ejecución en un único nodo de la partición *mono.*
- Mvapich: instalada en Mendieta. WRF corriendo procesos MPI en memoria distribuida todos los nodos. El resto de las dependencias usadas son las preinstaladas en Mendieta.

- OpenMPI: Misma prueba que la realizada con mvapich.
- OpenMPI + OpenMP: configuración híbrida entre memoria compartida (intra nodo) y memoria distribuida (inter nodos). WRF corriendo procesos MPI en memoria distribuida todos los nodos y luego internamente en cada nodo los procesos son mapeados a thread de OpenMP que usan memoria compartida. El resto de las dependencias usadas son las preinstaladas en Mendieta.
- Mvapich + OpenMP: misma prueba que con OpenMPI + OpenMP.
- Mpich + OpenMP: misma prueba que con OpenMPI + OpenMP, pero usando las dependencias NetCDF/HDF5/Zlib compiladas e instaladas manualmente.

Los resultados obtenidos son los que se muestran en las siguientes figuras:

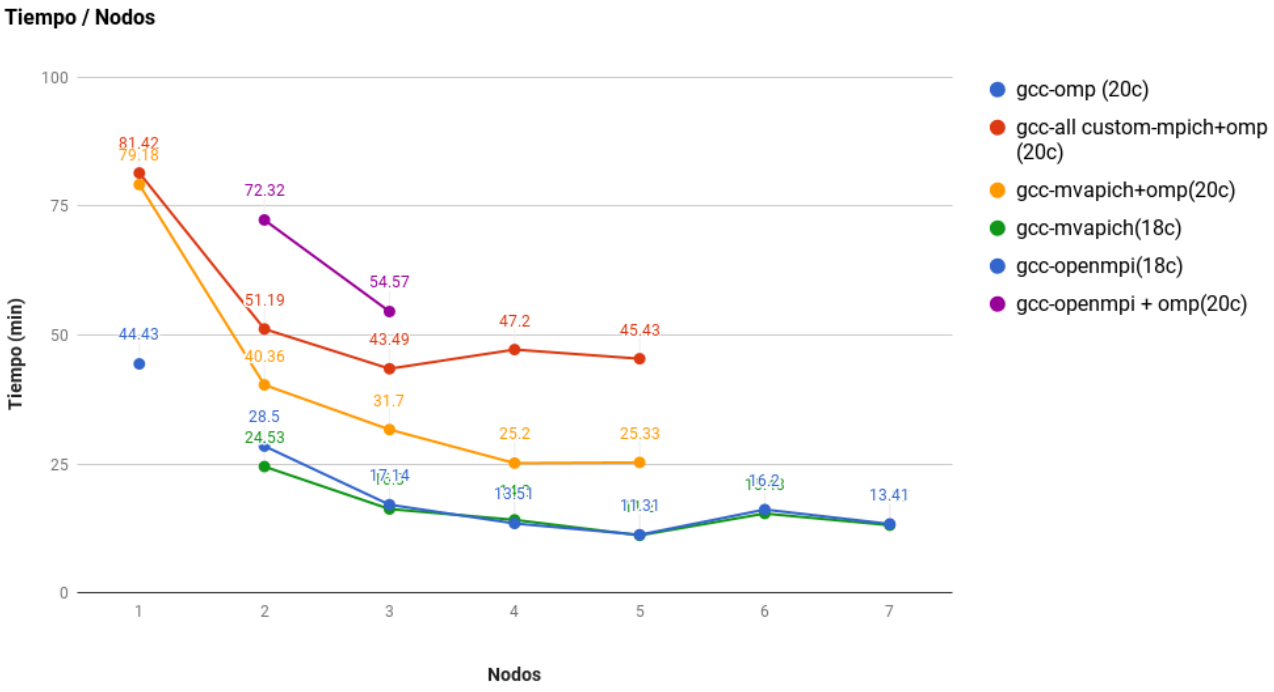


Figura 6.2: Mediciones para múltiples nodos. Tiempo

La figura 6.2 indica que la mayor eficiencia en el tiempo de cómputo es obtenida con las bibliotecas pre instaladas en Mendieta. Se puede apreciar un desempeño muy similar usando las bibliotecas de OpenMPI como así también con las bibliotecas de MVAPICH. También es notorio que la opción de uso de procesos que usan memoria distribuida tiene mejor desempeño que la opción híbrida MPI + OpenMP la cual utiliza procesos de MPI para la comunicación

inter-nodos y threads de OpenMP que comparten memoria para la comunicación intra-nodo. Una posible explicación es que faltó realizar alguna configuración adicional a la opción híbrida para alcanzar una mejor performance.

La figura 6.3 muestra el speedup[10] obtenido para esta prueba, el cual es levemente sublineal para las bibliotecas preinstaladas en Mendieta.

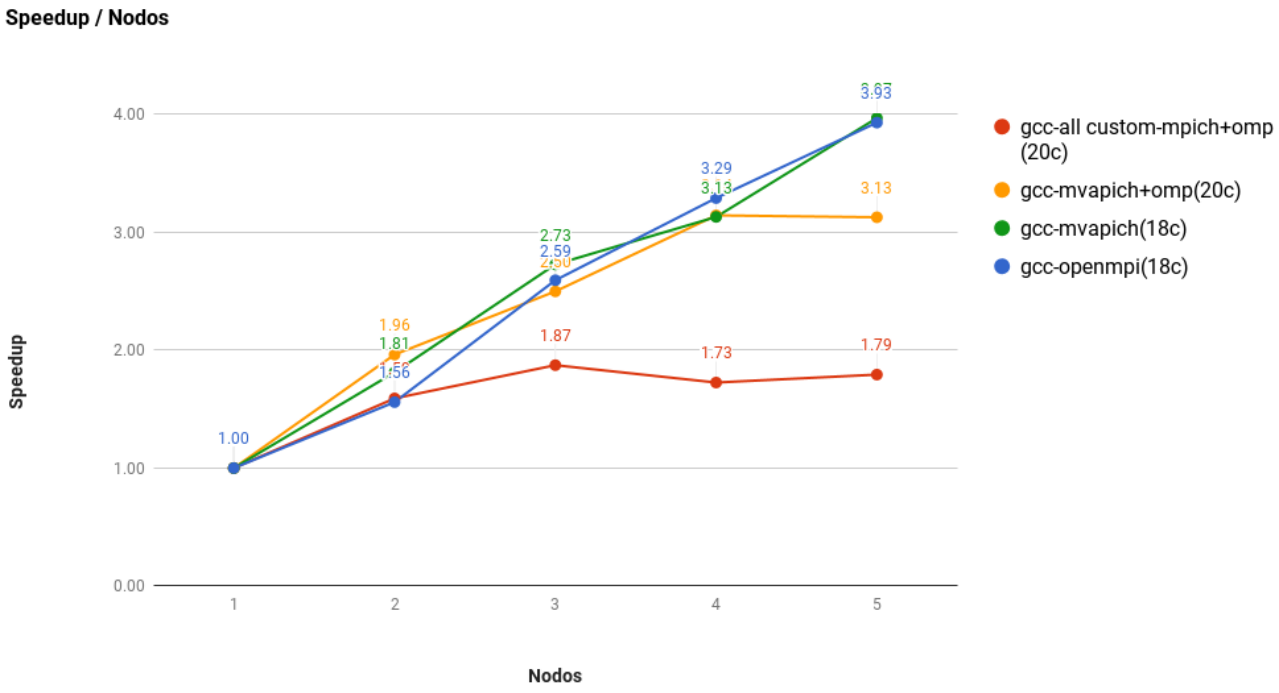


Figura 6.3: Mediciones para múltiples nodos. Speedup

Nuevamente se observa un speedup lineal para las gcc + Mvapich y gcc + Open MPI.

Finalmente, el análisis de eficiencia indica valores próximos a 1 (el ideal) para las bibliotecas preinstaladas, las gcc + Mvapich y gcc + OpenMPI, concluyendo que son la mejor opción a elegir para la ejecución de WRF.

[10] Speedup y eficiencia estan definidos en sección 3.3

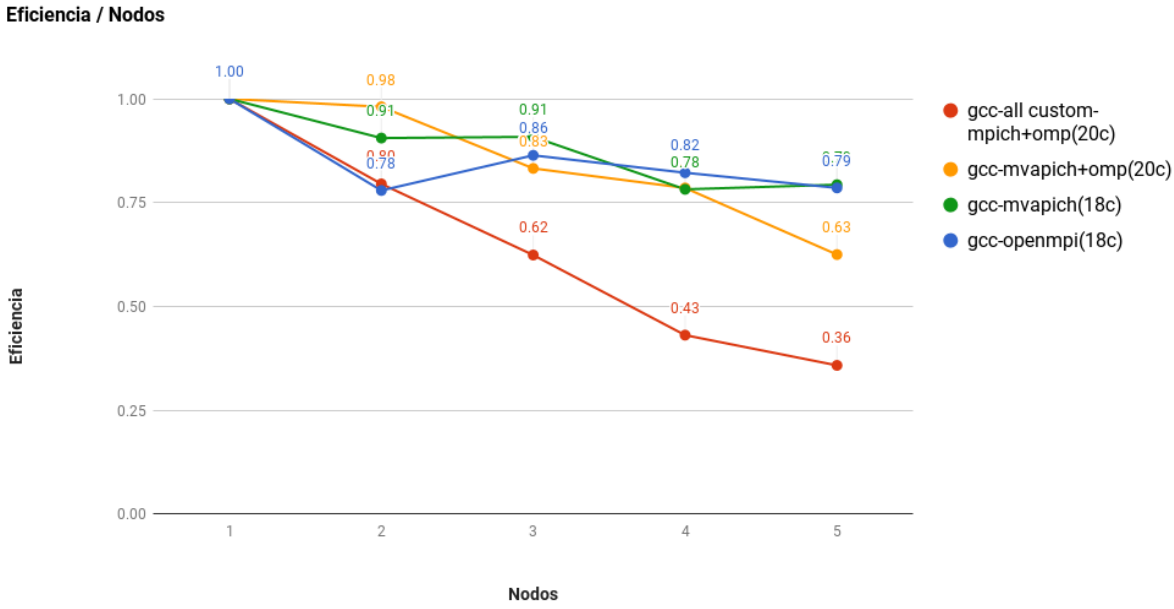


Figura 6.4: Mediciones para múltiples nodos. Eficiencia

6.3. Resultados para ensambles con distintas parametrizaciones

Los resultados previos permiten definir una configuración inicial para realizar las pruebas contra un pronóstico real, del tipo requerido por el grupo CAEARTE.

Para las pruebas realizadas ejecutando miembros de ensamble definidos en la tabla 5.2 y para un pronóstico de 36 h para la fecha 2018-01-20_00:00:00 y usando el dominio de simulación definido en la sección 5.1.

Se seleccionó la siguiente configuración:

Herramienta	Información
Compilador	gcc-4.9.2
Librería de MPI	Open MPI
Bibliotecas instaladas en Mendieta	NetCDF/HDF5
Bibliotecas instaladas manualmente	JasPer/GrADS

Tabla 6.2: Herramientas utilizadas para la ejecución de WRF

A continuación, se detallan los resultados obtenidos para la ejecución de cada uno de estos miembros de ensamble sobre distintas arquitecturas.

6.3.1. Resultado para tiempo de ejecución para cada uno de los miembros del ensamble

La figura 6.5 muestra las métricas de tiempo de cada uno de los miembros del ensamble para todas las arquitecturas donde se ejecutó WRF. La primera observación que se puede realizar es que los cinco miembros tienen un comportamiento similar en todas las arquitecturas donde fueron probados.

La ejecución del miembro del ensamble con parametrizaciones:

C_WDM6_QNSE_sf_sfclay_physics consume más tiempo de cómputo en todas las arquitecturas probadas. El resto de las parametrizaciones se procesan en aproximadamente el mismo tiempo. Se puede observar que el tiempo se reduce de manera importante comparando la ejecución de WRF en la máquina de CAEARTE (Core i7) con respecto a la ejecución en un nodo de Mendieta1.

La mejora es de aproximadamente de entre 2x-3x partiendo del Sistema computacional convencional con respecto a la ejecución en 1 nodo de Mendieta1 (De ~640 minutos a 330-224 minutos). Luego es notable un segundo salto en la mejora de tiempos pasando a dos nodos de Mendieta2. En este punto la mejora es de aproximadamente 3x con respecto a la ejecución en único nodo de Mendieta1.

A medida que se escaló en la cantidad de nodos de Mendieta2 se puede notar que se continúa obteniendo mejoras en la performance inclusive hasta ejecutar los ensambles en 8 nodos de la

partición *multi*. También es observable que el paralelismo escala sublinealmente a partir del uso de dos nodos, e incluso la performance comienza a "plancharse" a partir del uso de cuatro nodos, lo cual implica que la relación costo/beneficio entre recursos computacionales y tiempo ya no es justificable. Es decir, probablemente no se justifica usar un nodo más para ganar un par de minutos en la ejecución del modelo (ejemplo pasar de 7 a 8 nodos solo reduce el tiempo en unos 2-3 minutos).

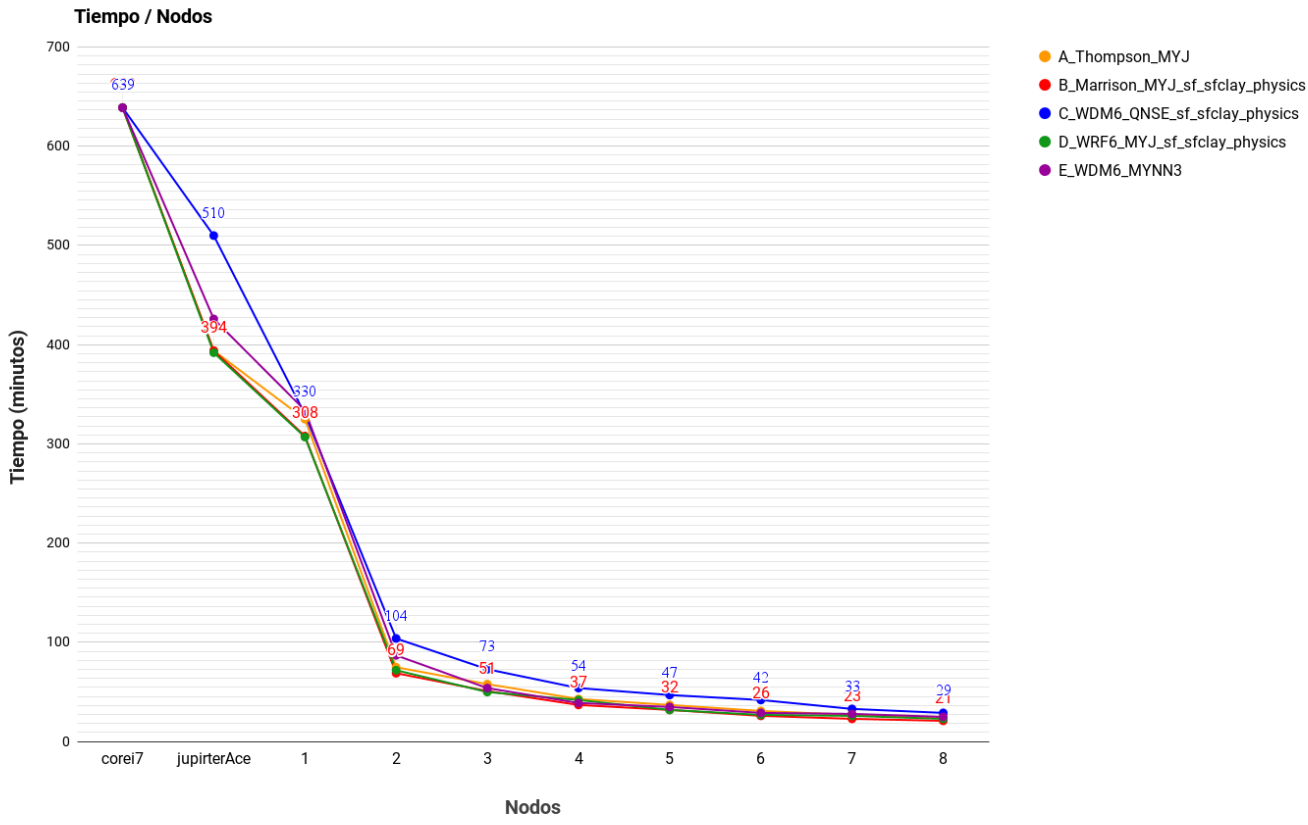


Figura 6.5: Tiempo de cómputo para pronóstico de 36 h. en Córdoba.

6.3.2. Resultado para speedup de ejecución para cada uno de los miembros del ensamble

Para este análisis se tuvo en cuenta sólo las ejecuciones realizadas en Mendieta. Se considera como unidad de procesamiento a un nodo (20 cores) de la partición *multi*, exceptuando para la ejecución en único nodo, que se realizó sobre la partición *mono* (16 cores), es decir que se analizó como escala WRF cuando se incrementa el número de nodos en la computación. Esta prueba deja en evidencia el carácter paralelizable de WRF. Se puede extraer mucho paralelismo distribuyendo el cómputo en múltiples nodos de 20 cores cada uno. El paralelismo obtenido al ejecutar WRF de manera distribuida en múltiples nodos está soportado a nivel software en la herramienta OpenMPI y a nivel hardware a través de la red de alta velocidad *Infiniband*.
Se observa que todos los miembros de ensamble escalan de manera aproximadamente lineal obteniendo una mejora de entre 11x y 13x respecto la ejecución en un nodo. Se debe tener en cuenta que el speedup debe en realidad ser menor ya que la ejecución en un nodo se realizó sobre un nodo de la partición *mono* (2 Intel Xeon E5-2680 -16 núcleos).

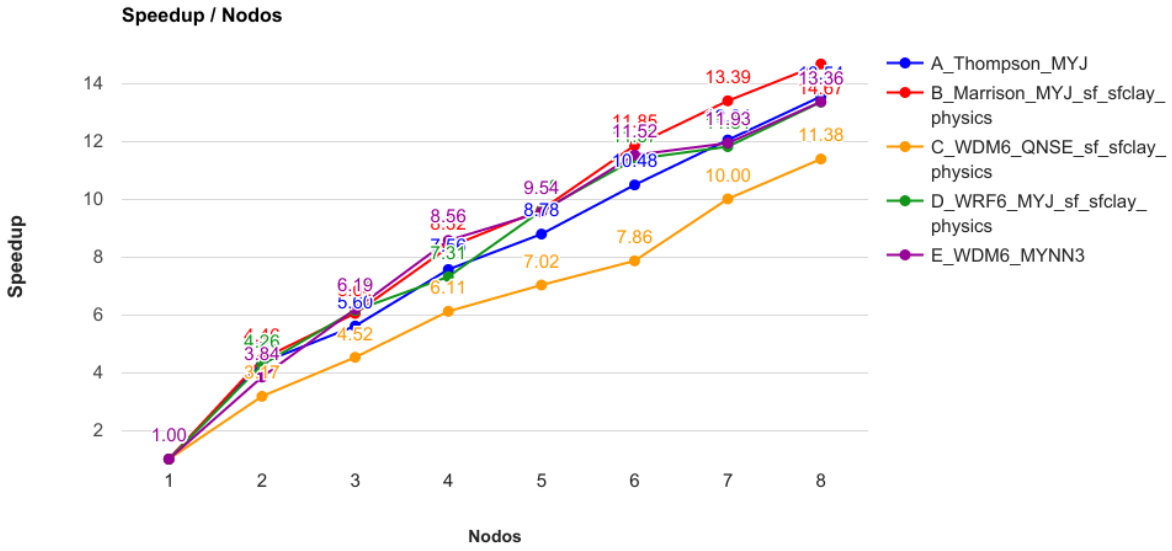


Figura 6.6: Speedup de cómputo para pronóstico de 36 h. en Córdoba.

6.3.3. Resultados de Ejecución en JupiterAce

Con el propósito de investigar la performance que se puede obtener en una máquina similar a un nodo de Mendieta1, se realizaron pruebas de performance en la máquina JupiterAce, perteneciente a FaMAF-UNC.

En la figura 6.8 se muestra el tiempo de cómputo para el pronóstico en el core i7 de CAEARTE, los tiempos de ejecución en JupiterAce y los tiempos de ejecución para un nodo de la partición *mono* (Mendieta1).

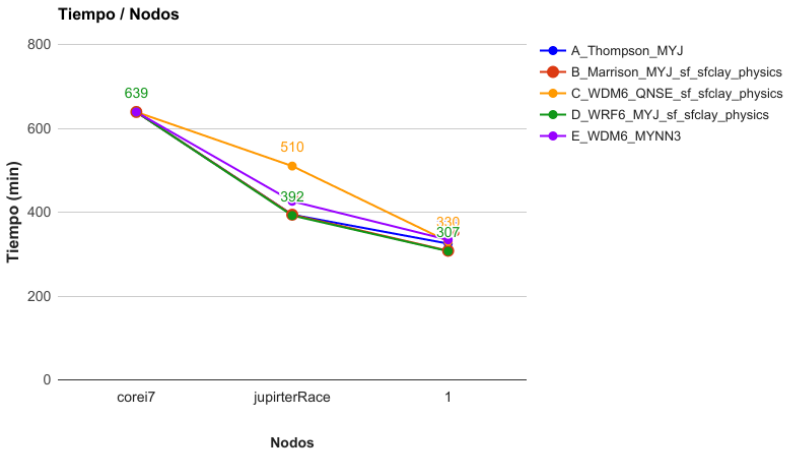


Figura 6.7: Tiempo de ejecución incluyendo JupiterAce.

Como se detalló en la sección 4.4. y 4.5. WRF soporta tres modos de ejecución paralela: procesos MPI (memoria distribuida), threads OpenMP (memoria compartida) y omp+mpi (modelo híbrido)

Para la ejecución en JupiterAce WRF fue compilado y testeado en los dos primeros modos: La mejor performance se alcanzó con OpenMP. No se obtuvieron mejoras usando el modo de MPI, de hecho, los resultados de tiempos de ejecución para este modo fueron similares a los mismos tiempos registrados para la máquina de CAEARTE, lo cual refleja que en un sistema no clúster conviene ejecutar WRF utilizando memoria compartida en lugar de memoria distribuida.

Se observa también que la mejora en tiempo de ejecución de aproximadamente 1.5X (de ~640 min. a ~400 min.) entre la máquina de CAEARTE vs JupiterAce radica principalmente en hardware, ya que ambas ejecuciones usaron OpenMP.

Recordamos el hardware de ambos sistemas.

	Máquina de CAEARTE	JupiterAce
Sistema operativo	Linux-Ubuntu 14.4	Debian 4.9.13-1
Compilador	gcc-5.4.0	gcc-6.3
Memoria	4GiB Tipo: DDR3 Velocidad:1333 MHz	128GiB tipo: DDR3- 2133 MT/s
procesador	Procesador Intel I7-2600K CPU @ 3.40GHz	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz [42]
Número de cores	4(8 lógicos)	6
Nodos NUMA	1	2(6 cores c/u)

Tabla 6.3: JupiterAce vs máquina de CAEARTE

6.3.4. Mapas de lluvias, y temperaturas

Un aspecto importante a considerar, además de los resultados que miden la performance de la ejecución de WRF en Mendieta, es la verificación de los pronósticos. Una real verificación y validación escapan al alcance de esta tesis, sin embargo es posible hacer una primera inspección comparando visualmente con un caso real.

A continuación se muestran los resultados de precipitación acumulada registrada (36 h) para el día 20 de Enero de 2018 según las estaciones meteorologicas activas de la Red de Estaciones de Recursos Hídricos de la provincia de Córdoba (Fig 6.8).

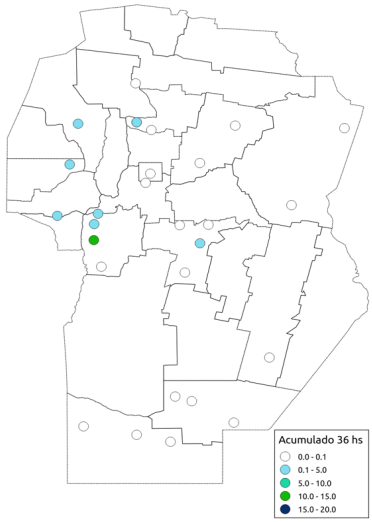
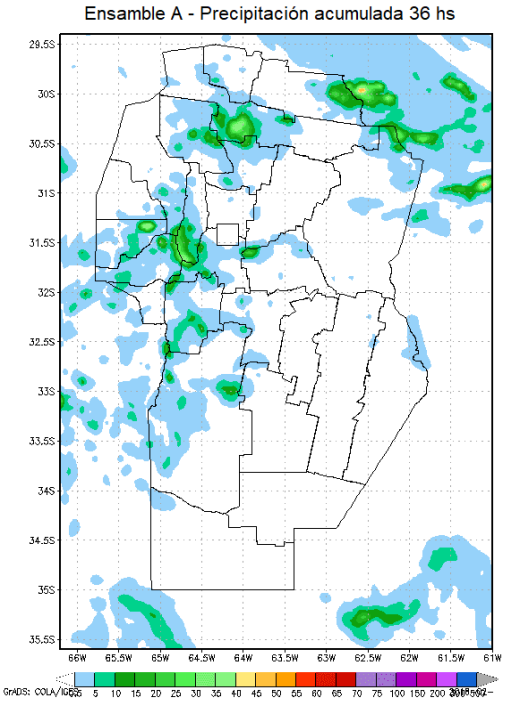
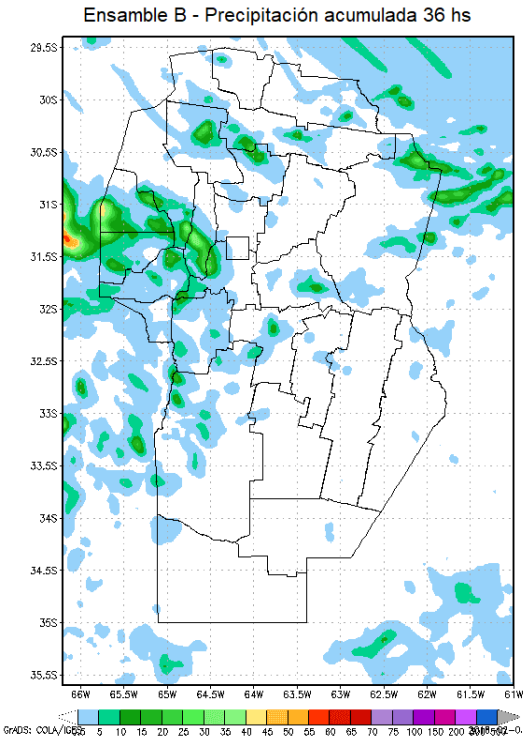
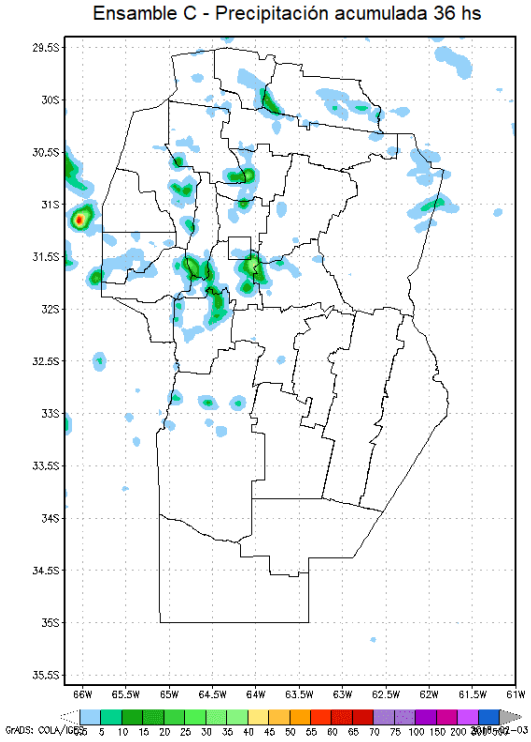


Figura 6.8: Precipitaciones registradas para el dia 20 de Enero de 2018.

A continuación se muestran los resultados de precipitación acumulada simulada (36 h) para el mismo dia y para cada uno de los cinco miembro del ensamble (Fig 6.9).







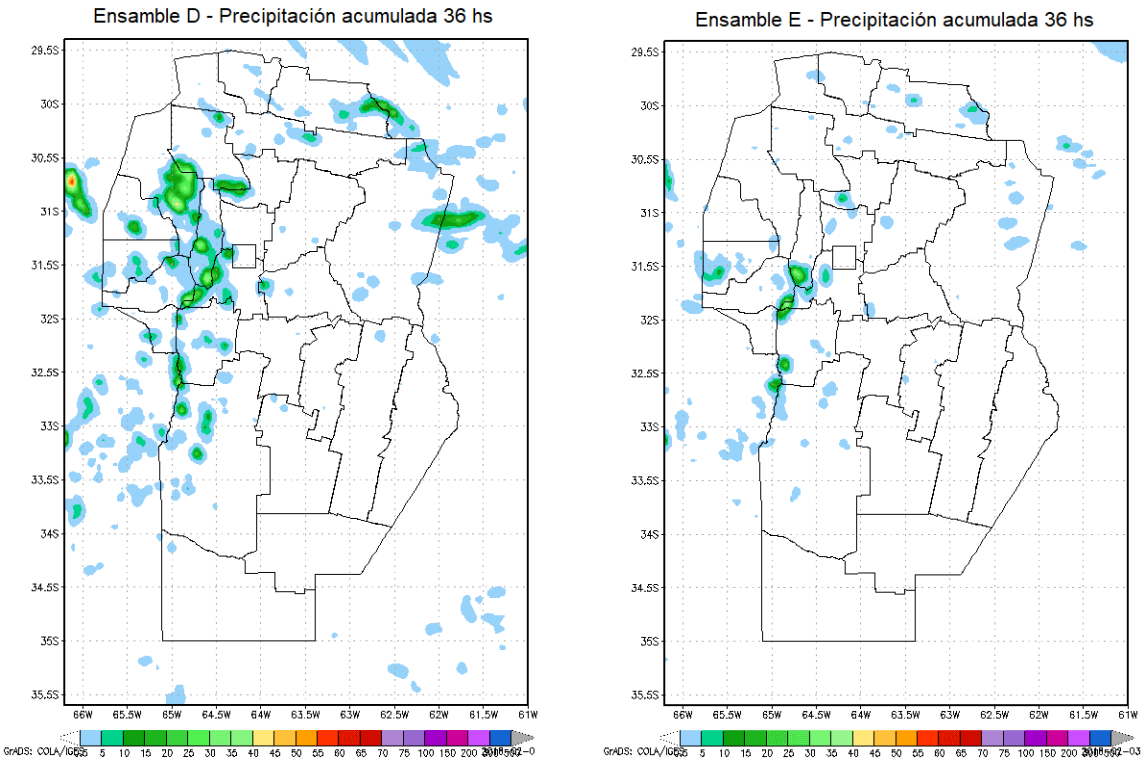


Figura 6.9: Mapas para pronósticos de precipitación de 36 h. en Córdoba.

Se puede observar que luego del procesamiento, los mapas generados (a través de GrADS) muestran variaciones importantes entre si, aunque en todos los casos las zonas donde se concentraron las precipitaciones pronosticadas se asemejan a las zonas donde efectivamente se registraron precipitaciones (visto en la figura 6.8). Esto proporciona las siguientes conclusiones:

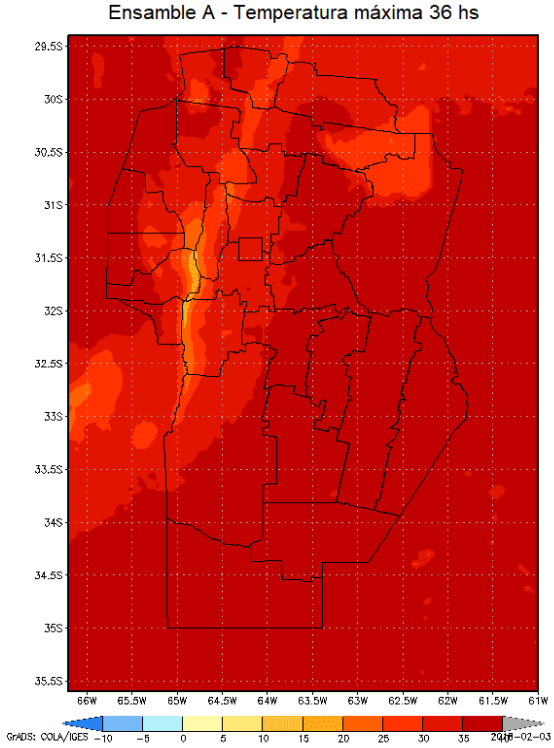
- El evento elegido como ejemplo, presenta precipitaciones aisladas en diversas zonas en la región Oeste de la provincia de Córdoba. Cada uno de los miembros muestra un patrón similar en cuanto a precipitaciones aisladas, sin embargo la intensidad y ubicación de las zonas con precipitación difieren uno de otro.

- Los cinco gráficos de la figura 6.9 muestran una sensibilidad de los resultados para la precipitación acumulada en cuanto a la física elegida. Esta clase de eventos, el de las precipitaciones aisladas, son dificultosas en su predicción pero mediante el uso de pronósticos por ensamble o como en este caso con 5 corridas se espera acotar la incertidumbre obtenida por un único pronóstico.

- Los pronosticos realizados para las precipitaciones por cada uno de los miembros del ensamble indican que el modelado fue correcto, pues las zonas con mayor precipitaciones coincidieron con las que fueron registradas en la fecha dada.

A continuación, se pueden observar los mapas obtenidos por cada miembro del ensamble para temperaturas máximas de Córdoba, para un pronóstico de 36 h.

Para el caso de las temperaturas, que no son variables puntuales como la precipitación, los patrones espaciales son homogéneos, mostrando ligeras diferencias en los lugares donde se pronostico precipitación y en los lugares con orografía compleja.



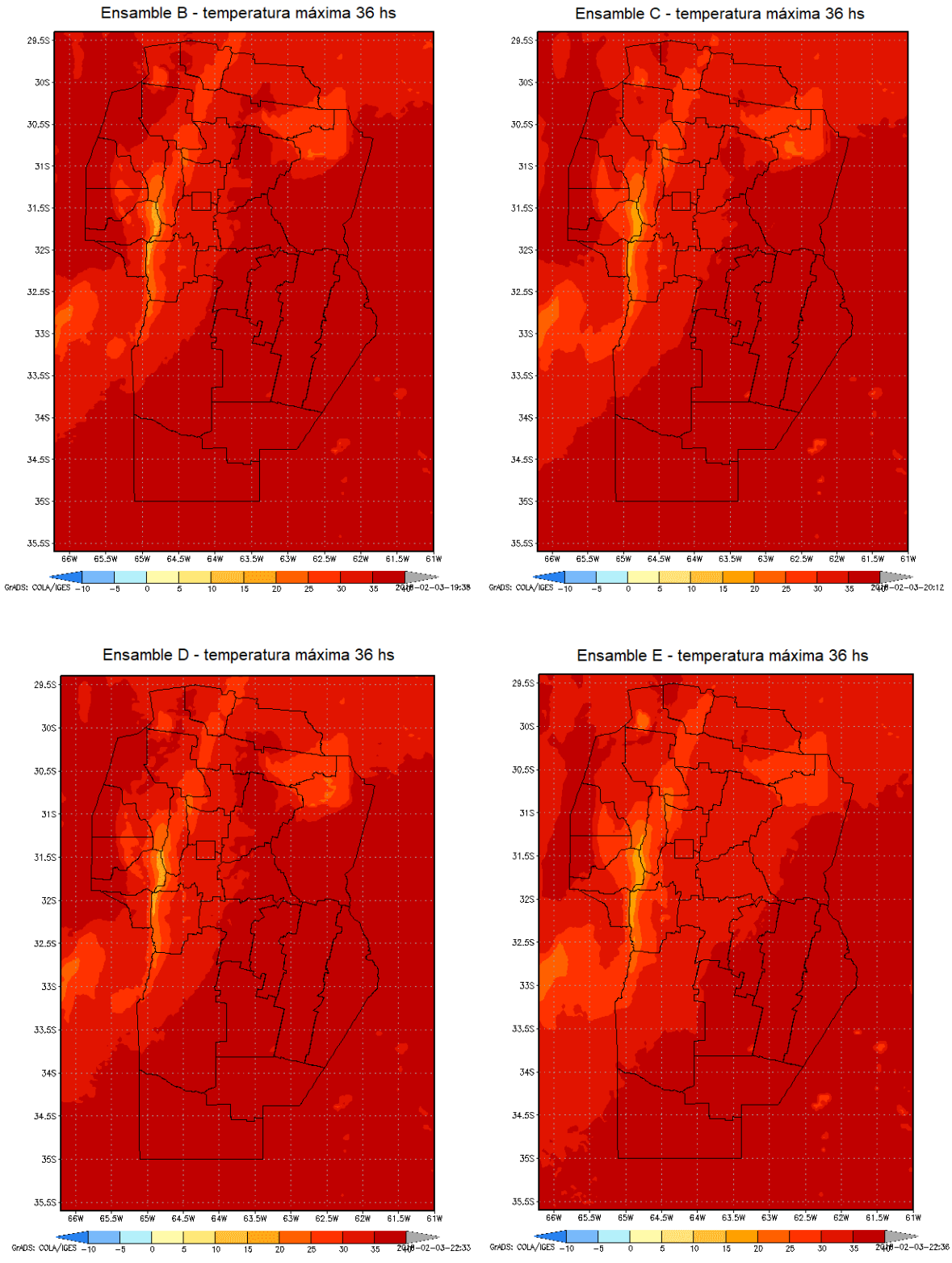
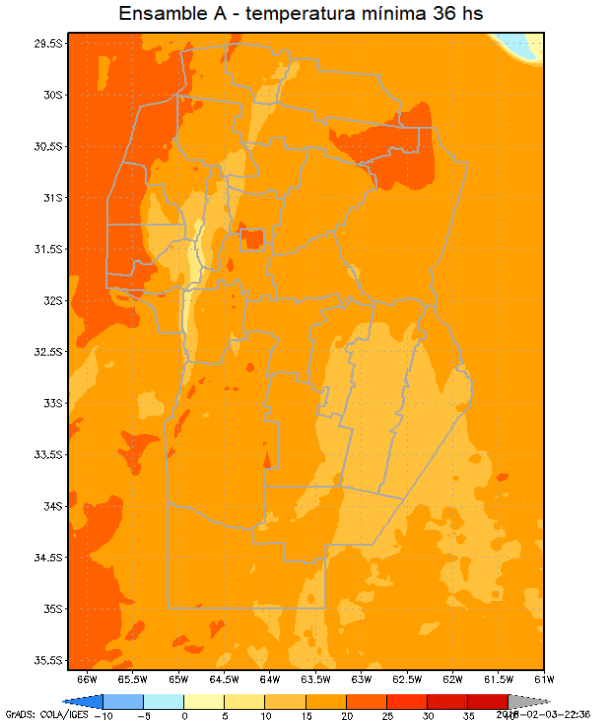
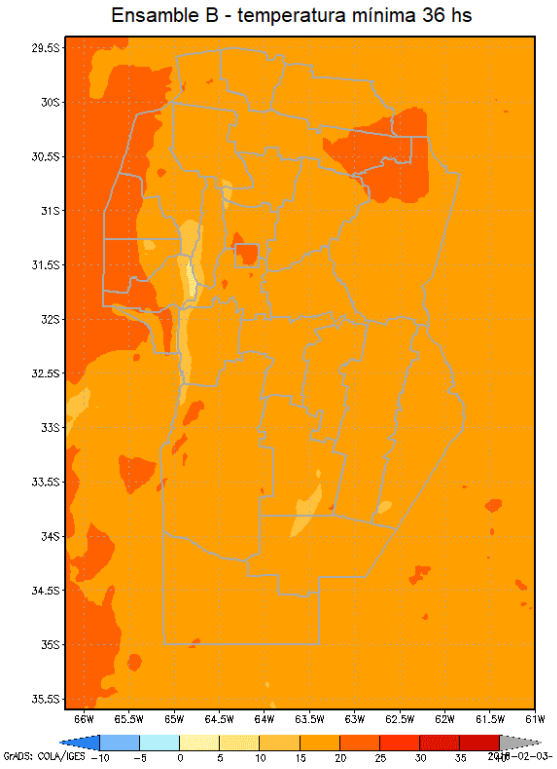
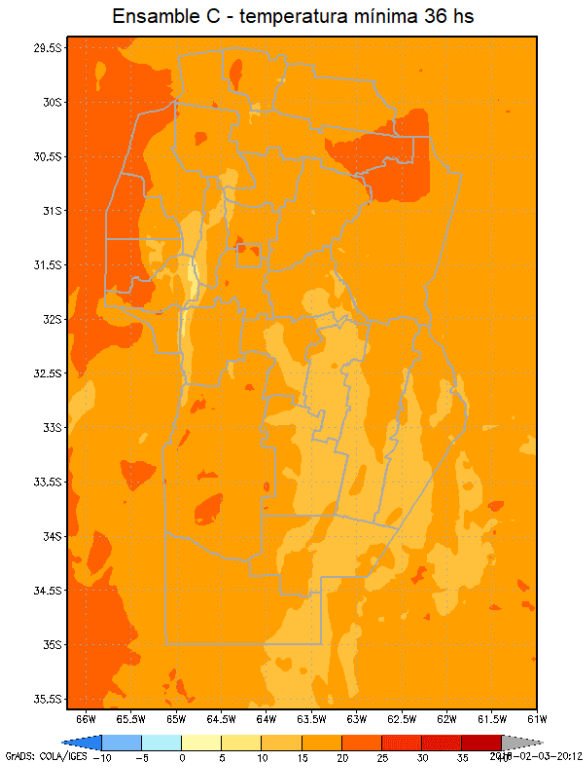


Figura 6.10: Mapas para pronósticos de temperatura máxima de 36 h. en Córdoba.

Por último, se observan los mapas obtenidos por cada miembro del ensamble para temperaturas mínimas de Córdoba, para un pronóstico de 36 h.







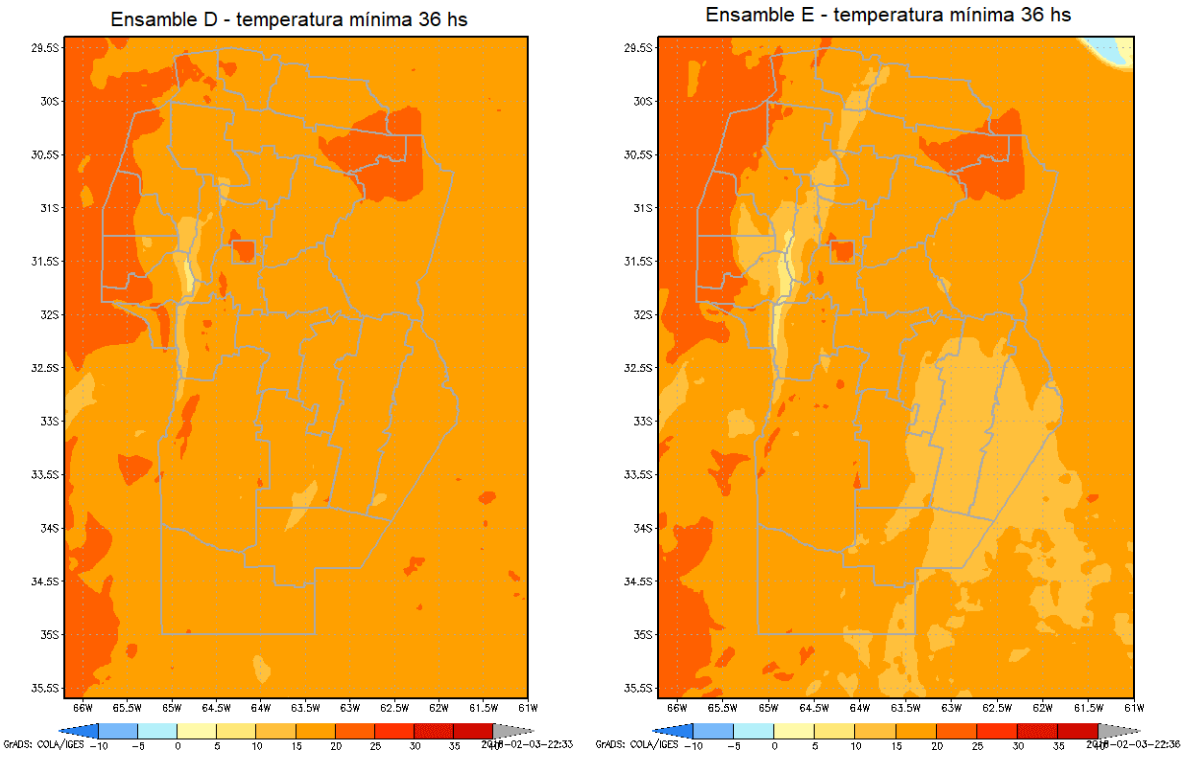


Figura 6.11: Mapas para pronósticos de temperatura mínima de 36 h. en Córdoba.

Los pronósticos para los miembros de ensamble de temperaturas mínimas presentan una divergencia un poco más pronunciada con respecto a las temperaturas máximas pronosticadas.

6.3.5. Inputs para meteogramas

Como se mencionó en la sección 2.10. Además de los mapas, GrADS genera archivos .txt con datos en texto planos a partir de los inputs que provee ARWpost. Dichos archivos contienen datos en texto plano con información de las 5 variables pronosticadas.
Estos datos pueden ser de utilidad para la visualizacion gráfica en alguna aplicación web.

Recordemos que se generan cinco pronósticos por cada una de las localidades de Córdoba.
Por ejemplo, se pueden visualizar los 36 valores por cada hora pronosticada de precipitación correspondiente al miembro de ensamble *A_Thompson* para la localidad de Alejo Ledesma.

```
[lvargas@mendieta meteogramas]cat rain_ALEJO_LEDESMA.txt
Notice:  Automatic Grid Interpolation Taking Place
Printing Grid -- 36 Values -- Undef = -9.99e+08
0
0
0
0
0
0
2.02418e-06
0.00507394
0.00507713
0.150332
1.06391
1.08192
1.08473
1.08507
1.08562
1.09133
1.09248
1.09248
1.09248
1.09248
1.09248
1.09248
1.09248
1.09248
1.06391
1.06391
1.06391
1.06391
1.00233
1.00233
0.04554
0.04554
0.04554
0.04554
0.04554
0.00345
```

6.3.6. Visualización web

Como se dijo anteriormente los archivos .txt son generados para cada ciudad (o coordenada del dominio) mediante la herramienta GRADS. Estos valores se utilizan para generar reportes gráficos en el archivo meteogramas.html. Para cada miembro del ensamble habrá un meteograma por variable, considerando que hay 5 miembros, se obtendrán 5 meteogramas por variable. Cada uno de estos tendrá la evolución temporal de esta variable con valores que pueden diferir debido a las distintas configuraciones de los miembros.

Un ejemplo de dicha visualización puede verse en la figura 6.12.

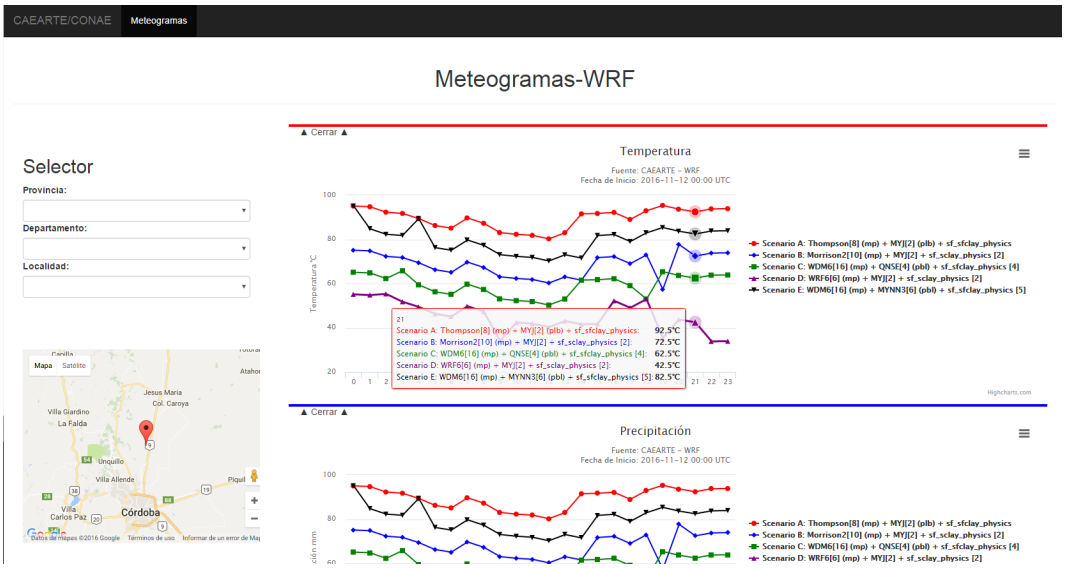


Figura 6.12: Reporte de meteograma.

6.3.7. Análisis durante la ejecución

Durante la ejecución de WRF en nodos de Mendieta se realizaron algunos análisis para chequear que el procesamiento se realizaba de manera eficiente.

1) Se pudo constatar que el total de los núcleos involucrados en procesamiento realizaron cómputo al máximo de su potencia durante la etapa paralelizable, esto es mientras se ejecutaba *wrf.exe*.

   La figura 6.13 muestra el uso intensivo de cada uno de los 100 núcleos para una ejecución que utilizo 5 nodos de la partición *multi* utilizando las herramientas tmux + htop.

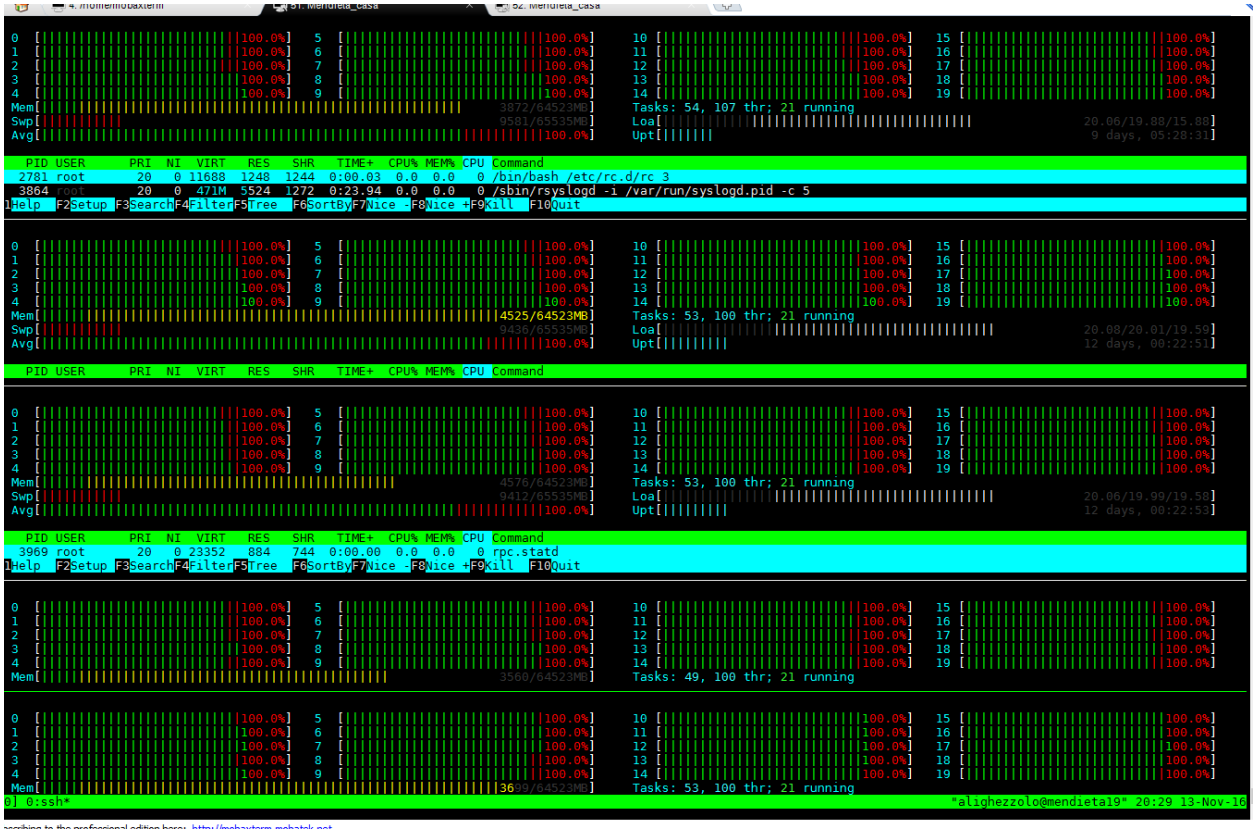


Figura 6.13: 100 procesos MPI ejecutando WRF en 5 nodos.

Este resultado es muy positivo pues indica que se están aprovechando al máximo las unidades de procesamiento en el cómputo de WRF. No se detectó que durante la ejecución de *wrf.exe*, hubiera alguno de los núcleos ejecutando algún otro proceso que no fuera *wrf.exe*. Esto deja claramente manifiesto lo que significa un sistema de cómputo *dedicado*.

2) Un análisis del uso de memoria con la herramienta ***numactl*** muestra que la memoria compartida por ambos procesadores en un nodo fue accedida de una manera eficiente. Una muestra tomada para una ejecución de WRF en los nodos mendieta09 y mendieta13 muestra los siguientes resultados:

   La CPU 0 (nodo 0) de Mendieta09 tiene durante un momento de la ejecución ocupada casi la totalidad de su memoria: aproximadamente 31GiB, mientras que la CPU 1 tiene ocupado aproximadamente el 50% de su capacidad.

```
[lvargas@mendieta09 ~]numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
node 0 size: 32733 MB
node 0 free: 955 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19
node 1 size: 32768 MB
node 1 free: 15098 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

Análisis sobre Mendieta13: En este nodo la distribución fue más pareja aproximadamente 20 GiB para cada CPU.

```
[lvargas@mendieta13 ~]numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
node 0 size: 32733 MB
node 0 free: 11665 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19
node 1 size: 32768 MB
node 1 free: 10061 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

Un primer aspecto básico pero importante a destacar, que proporciona un clúster como Mendieta, a diferencia de un sistema computacional convencional, es que al contar con mucha memoria (64GiB) todo el dominio de simulación se encuentra alojado en memoria, lo que evita el uso de acceso a memoria virtual (disco). Los tiempos de acceso a disco son prohibitivos en HPC. Aunque el modelado realizado por WRF consume una considerable cantidad de memoria: aproximadamente 40 GiB.

Para interpretar mejor la información obtenida se complementa con las siguientes herramientas:

i. Se solicita a través del comando *salloc* de SLURM un nodo de la partición *muti* y se ejecuta WRF bajo la herramienta *perf*, la cual como se explicó en la sección 4.2.6. es un profiler que recolecta métricas durante tiempo de ejecución. Las métricas pedidas son instructions, cycles, cache-misses y cache-references.

```
[lvargas@mendieta ~] cd $WRF_BASE[11]
[lvargas@mendieta wrf_mendieta] salloc -p multi -n 20 srun --exclusive --pty --
preserve-env $SHELL
salloc: Pending job allocation 63241
salloc: job 63241 queued and waiting for resources
[lvargas@mendieta13 wrf_mendieta] cd WRF3.8/WPS
[lvargas@mendieta13 WPS] ./link_grib.csh $WRF_BASE/gribfiles/2018-01-
20_00\:00\:00/GFS*
[lvargas@mendieta13 WPS] ./ungrib.exe
[lvargas@mendieta13 WPS] ./geogrid.exe
[lvargas@mendieta13 WPS] ./metgrib.exe
[lvargas@mendieta13 WPS] cd ../WRF3.8/WRFV3/test/em_real/
[lvargas@mendieta13 em_real] ./real.exe
[lvargas@mendieta13 em_real] time srun perf stat -e instructions, cycles, cache-
references, cache-misses ./wrf.exe
```

El resultado es obtenido para cada uno de los 20 núcleos que ejecutaron WRF y brindan la siguiente información[12].

```
 Performance counter stats for './wrf.exe':

   32545983155507      instructions              #    1.24  insns per cycle
   26346678055134      cycles
     125676007246      cache-references
      39025273377      cache-misses              #   31.052 % of all cache refs
   8504.378669569 seconds time elapsed
```

La información relevante es que aproximadamente hubo un 30% de cache-misses. Lo que indica que en aproximadamente un 70%, los accesos a cache fueron exitosos. El 30% de cache-misses derivan en accesos a RAM.

[11] Ver Apendice B Sección 2. Para tener detalles de la variable de entorno WRF_BASE
[12] La información detallada es para solo un núcleo.

ii. Aplicando un análisis con la herramienta ***numastat*** durante un periodo de la ejecución de WRF se obtiene la siguiente información:

```
[lvargas@mendieta20 ~]for i in {1..100}; do numastat -c ; sleep 5;  done

Per-node numastat info (in MBs):
                 Node 0  Node 1   Total
                ------- ------- -------
Numa_Hit        3070988 3553197 6624185
Numa_Miss             0       0       0
Numa_Foreign          0       0       0
Interleave_Hit      111     112     224
Local_Node      3070909 3553118 6624026
Other_Node           79      80     159

Per-node numastat info (in MBs):
                 Node 0  Node 1   Total
                ------- ------- -------
Numa_Hit        3071133 3553377 6624510
Numa_Miss             0       0       0
Numa_Foreign          0       0       0
Interleave_Hit      111     112     224
Local_Node      3071053 3553297 6624351
Other_Node           79      80     159
.
.
.
.
Per-node numastat info (in MBs):
                 Node 0  Node 1   Total
                ------- ------- -------
Numa_Hit        3071346 3553646 6624992
Numa_Miss             0       0       0
Numa_Foreign          0       0       0
Interleave_Hit      111     112     224
Local_Node      3071267 3553566 6624833
Other_Node           79      80     159
```

Se observa que, del 30% de accesos a RAM (NUMA) debido a cache-misses, no se obtuvieron NUMA-miss, esto significa que wrf.exe ejecutado en el nodo 0, no tuvo necesidad de acceder a la memoria remota alocada por wrf.exe ejecutado en el nodo 1. Esto explica los valores 0 para las métricas Numa-hits y Numa-Foreign.

iii. Se verificó que efectivamente los accesos a memoria por parte de WRF estaban efectuándose de manera eficiente utilizando el soporte de "afinidad" que proporciona *numactl*. Esto significa que se explicitó que cada uno de los 20 procesos *wrf.exe* estuvieran "bindeados" (atados) a un núcleo en particular. Esta técnica evita la perdida de cache al migrar procesos entre cores, ya que este mecanismo implica cargar el estado del proceso nuevamente en la cache y RAM del nuevo core asignado, generando perdida de performance.
La instrucción para proporcionar afinidad es la siguiente:

```
time srun numactl --physcpubind=0-19 ./wrf.exe
```

Sin embargo, no se obtuvieron mejoras significativas en tiempo de ejecución lo que confirma el análisis realizado en ii. Que sin esta técnica WRF ya estaba accediendo de forma eficiente a su memoria.

7. Conclusiones

Existe una necesidad importante de que las áreas de estudio y pronóstico del clima en el país se desarrollen de manera de obtener pronósticos climáticos que ayuden en la implementación de sistemas de alertas tempranas ante siniestros ambientales. Por otro lado tambien existe una creciente tendencia en el uso de pronosticos numericos regionalizados debido a que estos ofrecen la posibilidad de adaptacion a una region en particular. Para paises como Argentina, la regionalización es fundamental, ya que las características que modelan el clima son muy variadas entre las distintas regiones. Es decir, una única implementación de un pronóstico numérico no puede describir de buena manera a todas las regiones que componen a nuestro País. El presente trabajo intenta contribuir en ese sentido estudiando la implementación de WRF en el clúster Mendieta de la región de la Provincia de Córdoba.

El objetivo principal planteado para esta tesis fue logrado a través de la implementación del modelo WRF en el entorno del clúster Mendieta. importante de que las áreas de estudio y pronóstico del clima en el país se desarrollen de manera de obtener pronósticos climáticos que ayuden en la implementación de sistemas de alertas tempranas ante siniestros ambientales.

La instalación, configuración y puesta en marcha fueron logradas. La información generada mediante las simulaciones fue convertida en archivos visuales para su interpretación.

Todas las acciones fueron documentadas y puestas a disposición en el repositorio publico: https://github.com/lvc0107/wrf_mendieta.

Este esfuerzo permitirá a futuros usuarios acelerar los procesos previos y focalizarse en los experimentos correspondientes al modelado con WRF.

Como se dijo anteriormente, el modelo fue corrido en alta resolución para la región de la Provincia de Córdoba. Esta es una región clave a nivel mundial ya que en la misma se producen tormentas intensas. La alta resolución podría permitir estudiar y comprender estas tormentas. Además en el año 2018 a través del proyecto RELAMPAGO la región y sus tormentas serán estudiadas detalladamente. Con lo cual la implementación lograda podría ser un importante aporte a estos estudios.

Se simuló un caso de prueba para evaluar, de manera simple, el desempeño del modelo en su representación de la realidad. Los resultados fueron satisfactorios.

En la actualidad si bien existen pronósticos de alta resolución implementados operativamente, no existen pronósticos de alta resolución por ensambles.

Lo desarrollado en esta tesis permite tener una perspectiva de los recursos computacionales y los tiempos de cálculo necesarios para tener un pronóstico por ensambles de manera operativa.

En lo que respecta específicamente a las implementaciones los primeros resultados obtenidos demuestran que la implementación del modelo WRF en Mendieta entrega una performance muy superior a los resultados de correr WRF en una máquina convencional como un procesador Core i7 con cuatro núcleos o incluso una máquina como JupiterAce. Aunque la arquitectura de JupiterAce es mucho más potente que la máquina de CAEARTE no se obtuvo una mejora significativa en los tiempos de ejecución usando compilaciones de WRF presumiblemente similares. Las causas pueden darse en qué WRF no escala linealmente agregando más núcleos y usando memoria compartida en un single-node o probablemente la implementación de WRF en JupiterAce careció de todos los parámetros necesarios para ejecutar de forma más performante. Los resultados de WRF ejecutado en una arquitectura single-node, en el cual no hay latencia de I/O durante el cómputo, muestran una leve mejora por parte de los threads de OpenMP por sobre la biblioteca de procesos de MPI. Esto se puede deber a que en el modelo de memoria compartida, en donde todos los threads comparten el mismo espacio de memoria, existe una mayor eficiencia por sobre los procesos de MPI que deben compartir información a través del envío de mensaje entre ellos usando el modelo de memoria distribuida (donde cada nodo cuenta con su propia memoria). El paralelismo que se puede extraer corriendo WRF en un solo nodo de la partición mono demuestra una mejora en el speedup de 3x con respecto a la implementación en hardware convencional. La performance mejora aún más ejecutando WRF en múltiples nodos de la partición de multi, alcanzando mejoras significativas ejecutando hasta en 7 nodos y obteniendo una mejora final de aproximadamente 10x en tiempo de cómputo con respecto al Sistema computacional convencional inicial. (Partiendo de ~360 minutos en máquina convencional a ~21 minutos en 8 nodos de Mendieta). 95 También vemos que la performance comienza a escalar sublinealmente distribuyendo el cómputo en más de cuatro nodos. Por otro lado también se nota que la diferencia en performance entre Mendieta1 y Mendieta2 no es muy significativa, aunque se nota una leve mejora (en el orden del 5%) en los tiempos de ejecución. Es un desafío futuro o parte de una línea de investigación a seguir el uso de WRF en aceleradores de cómputo tales como con las que cuenta Mendieta (placas NVIDIA Tesla y los coprocesadores Xeon Phi). En la actualidad el equipo de desarrollo de WRF (NCAR entre otros) se encuentra

trabajando en la migración de código a estas arquitecturas, aunque aún no ha publicado oficialmente el código.

La conclusión final a la que se puede arribar tras evaluar los resultados obtenidos, es que es posible obtener pronósticos sobre aspectos climáticos de grandes porciones de regiones y con un buen grado de precisión y de resolución en un tiempo que permite poder actuar en consecuencia, es decir en tiempos relativamente cortos que permitan anticiparse a fenómenos climáticos adversos. De otra manera si no se contase con una herramienta de pronóstico climático como WRF en un centro de cómputo dedicado como el caso de Mendieta y en su lugar se ejecutase este modelo en arquitecturas convencionales el tiempo de cómputo sería demasiado alto e impracticable si se desean realizar múltiples pronósticos en paralelos (modelo de ensambles).

Apéndice A: Variables de entorno necesarias para construir WRF y sus dependencias

La siguiente tabla describe ciertas variables de entorno que son necesarias declarar o setear para la implementación de WRF en Mendieta.

Variable	Uso	Descripción
CC	Compilador C	La versión utilizada fue gcc-5.4.0
FC	Compilador Fortran	Carga mpif90
CFLAGS	Flags para compilador C (optimización)	Carga -w -O3 –c
CPPFLAGS	Opciones para preprocesador C	Este flag debe contener el path de $NETCDF/lib al momento de compilar NetCDF-Fortran También debe contener los path de $HDF5/lib y $ZlIB/lib. Ver el archivo $WRF_BASE/set_custom_configuration.h
FCFLAGS	Flags para compilador Fortran 90	La optimización para código fortran se alcanza con estos flags que se auto generan al compilar WRF: -O2 -ftree-vectorize -funroll-loops Proveen autoparalización y unrolling en loops También carga -ffree-form -ffree-line-length-none y -fconvert=big-endian -frecord-marker=4
LDFLAGS	Flags para bibliotecas dinámicas	Cargamos path de bibliotecas de las dependencias de WRF: -L${NETCDF}/lib -L${HDF5}/lib
DM_CC		Es un requerimiento necesario para poder utilizar Open MPI. Esta variable debe ser seteada con los siguientes valores: mpicc -DMPI2_SUPPORT

CFLAGS_LOCAL		La optimización para código C se alcanza con estos flags que se auto generan al compilar WRF: -w -O3 –c
LD_LIBRARY_PATH	Path de bibliotecas dinámicas	Este path debe contener todas las bibliotecas dinámicas (shared objects) cargadas en tiempo de ejecución. Por ejemplo, ARWpost ejecutado en Mendieta utiliza la librería de netcdf ubicada en /opt-old/netcdf/4.3.3.1-gcc_4.9.2/lib: libnetcdf.so.7
WRF_LIB	Variable para bibliotecas de WRF	-lgomp: Flag necesario para ejecutar OMP. Debe setearse en tiempo de compilación de WRF para correr en nodos de la partición *mono*.

Tabla A.1: Variables de entorno usadas

Apéndice B: Instalación de WRF en Mendieta u otro sistema Linux

1. Documentación en repositorio
2. Descarga de WRF/WPS/ARWpost
3. Instalación de WRF + dependencias
4. Obtención de datos terrestres
5. Pruebas durante la ejecución

1. Documentación en repositorio

Las siguientes instrucciones de ejecución de comandos, junto con el código fuente se encuentra versionado en el repositorio público https://github.com/lvc0107/wrf_mendieta.
Esta sección detalla los pasos necesarios para instalar WRF tanto en Mendieta como en otro entorno Linux por ejemplo JupiterAce.
Se instaló la versión 3.8 de WRF.

2. Descarga de WRF/WPS/ARWpost y seteo inicial de variables

Clonar el siguiente repo:

```
ssh <USER>@mendieta.ccad.unc.edu.ar
cd $HOME
git clone https://github.com/lvc0107/wrf_mendieta.git
cd wrf_mendieta
mkdir WRF3.8
```

La ejecución del siguiente comando carga las variables de entorno necesarias.

```
. set_configuration.sh 3.8
```

En caso de utilizar una versión posterior a WRF3.8 se debe editar el archivo s*et_configuration.sh* y cambiar por la versión correspondiente. Ejemplo para usar WRF3.8:

```
export WRF_VERSION=WRF3.9
```

Para un entorno como JupiterAce es necesario agregar las siguientes variables de entornos en el archivo ***set_custom_configuration.sh*** y luego cargarlas en el sistema.

```
export CPPFLAGS="-I${NETCDF}/include -I${HDF5}/include -I${ZLIB}/include"
export LDFLAGS="-L${NETCDF}/lib -L${HDF5}/lib -L${ZLIB}/lib"
export LD_LIBRARY_PATH=${ZLIB}/lib:${HDF5}/lib:${NETCDFC}/lib:${LD_LIBRARY_PATH}

export WRFIO_NCD_LARGE_FILE_SUPPORT=1
export WRF_EM_CORE=1

### Folder for grads configuration.
export GADDIR=$WRF_BASE/library/grads-2.0.2/data
export PATH=$PATH:$WRF_BASE/library/grads-2.0.2/bin
. set_custom_configuration.sh
```

Descarga de WRF

```
cd $WRF_BASE/WRF3.8
wget http://www2.mmm.ucar.edu/wrf/src/WRFV3.8.TAR.gz
tar -xvzf WRFV3.8.TAR.gz
rm WRFV3.8.TAR.gz
```

Descarga de WPS

```
cd $WRF_BASE/WRF3.8
wget http://www2.mmm.ucar.edu/wrf/src/WPSV3.8.TAR.gz
tar -xvzf WPSV3.8.TAR.gz
rm WPSV3.8.TAR.gz
```

Descarga de ARWpost

```
cd $WRF_BASE/WRF3.8
wget http://www2.mmm.ucar.edu/wrf/src/ARWpost_V3.tar.gz
tar -xvzf ARWpost_V3.tar.gz
rm ARWpost_V3.tar.gz
```

3. Instalación de WRF y sus dependencias

3.1. JasPer[13]

```
cd $WRF_BASE
module load compilers/gcc/5
mkdir -p library/jasper/build
cd library/jasper
wget http://www.ece.uvic.ca/~mdadams/jasper/software/jasper-1.900.1.zip
unzip jasper-1.900.1.zip
rm jasper-1.900.1.zip
cd jasper-1.900.1
./configure --prefix=$(pwd)/build
make
make check
make install
```

Chequeo de la correcta instalación de JasPer:

```
ls build/bin/
imgcmp  imginfo  jasper  tmrdemo
```

3.2. Zlib

Esta herramienta ya se encuentra instalada en Mendieta. Paso válido para JupiterAce.

```
cd $WRF_BASE/library
mkdir -p zlib/build
cd zlib
wget http://fossies.org/linux/misc/zlib-1.2.8.tar.gz
tar -xvf zlib-1.2.8.tar.gz
```

[13] Estas librerías y su funcionamiento estan detalladas en la sección 4.2.5

```
rm zlib-1.2.8.tar.gz
cd zlib-1.2.8/
./configure --prefix=$(pwd)/build
make test
make install
```

3.3. HDF5

Esta herramienta ya se encuentra instalada en Mendieta. Paso válido para JupiterAce.

```
cd $WRF_BASE/library
mkdir hdf5
cd hdf5/
wget https://www.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8.13/src/hdf5-1.8.13.tar.gz
tar -xvzf hdf5-1.8.13.tar.gz
hdf5-1.8.13.tar.gz
cd hdf5-1.8.13/
mkdir build
cd build
make clean
 ./configure --prefix=$(pwd)/build
make
make test
make install
make check-install
# update $WRF_BASE/set_custom_configuration.sh with the following variable
# export HDF5=$WRF_BASE/library/hdf5-1.8.13/build
. set_custom_configuration.sh
```

3.4. NetCDF

Esta herramienta ya se encuentra instalada en Mendieta. Paso válido para JupiterAce.

```
cd $WRF_BASE/library
mkdir netcdf
wget http://pkgs.fedoraproject.org/repo/pkgs/netcdf/netcdf-
4.3.3.1.tar.gz/5c9dad3705a3408d27f696e5b31fb88c/netcdf-4.3.3.1.tar.gz
md5sum netcdf-4.3.3.1.tar.gz | grep 5c9dad3705a3408d27f696e5b31fb88c
tar -xvf netcdf-4.3.3.1.tar.gz
rm netcdf-4.3.3.1.tar.gz
cd netcdf-4.3.3.1/
```

```
mkdir build
make clean
./configure --prefix=$(pwd)/build FC=gfortran F77=gfortran CC=gcc --enable-shared
LDFLAGS="-L$HDF5/lib"  CPPFLAGS="-I$HDF5/include"
make
make check
make install
cd $WRF_BASE
# update $WRF_BASE/set_custom_configuration.sh with the following variable
# export HDF5=$WRF_BASE/library/netcdf/netcdf-4.3.3.1/build
. set_custom_configuration.sh
```

3.5. NetCDF-Fortran

Esta herramienta ya se encuentra instalada en Mendieta. Paso válido para JupiterAce.

```
cd $WRF_BASE/library/netcdf
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-fortran-4.2.tar.gz
tar -xvf netcdf-fortran-4.2.tar.gz
rm netcdf-fortran-4.2.tar.gz
# si no esta disponible el recurso intentar:
#git clone https://github.com/Unidata/netcdf-fortran.git #ultimo release (no 4.2)
cd netcdf-fortran-4.2
make clean
#install in the netcdf build directory
./configure --prefix=$(pwd)/../netcdf-4.3.3.1/build FC=gfortran F77=gfortran CC=gcc
--enable-shared 2>&1 | tee configure.log
make
make check
make install
```

3.6 MVAPICH

Paso válido para un entorno donde no esté instalada alguna implementación de MPI.

```
cd $WRF_BASE/library
mkdir mvapich
cd mvapich
wget http://mvapich.cse.ohio-state.edu/download/mvapich/mv2/mvapich2-
2.2.tar.gz
tar -xvf mvapich2-2.2.tar.gz
rm mvapich2-2.2.tar.gz
cd mvapich2-2.2
#configure: error: 'infiniband/mad.h not found. Please retry with --disable-mcast'
./configure --prefix=$(pwd)/build --disable-mcast
make
make install

# Agregar $(pwd)/build/bin a PATH
```

3.7. Instalación de WRF

```
cd $WRF_DIR
./clean -a
./configure
```

Al iniciar el script ***configure*** debe dar un mensaje como el siguiente:
De esta forma si se está usando ***set_configuration.sh*** (Herramientas provistas por Mendieta).

```
checking for perl5... no
checking for perl... found /usr/bin/perl (perl)
Will use NETCDF in dir: /opt/netcdf-fortran/4.4.2-netcdf_4.3.3.1-gcc_4.9.2
Will use PHDF5 in dir: /opt/hdf5/1.8.15-gcc_4.9.2
which: no timex in (/opt/netcdf-fortran/4.4.2-netcdf_4.3.3.1-
gcc_4.9.2/bin:/opt/netcdf/4.3.3.1-gcc_4.9.2/bin:/opt/hdf5/1.8.15-
gcc_4.9.2/bin:/opt/openmpi-cuda/1.8.8-gcc_4.9-cuda_7.0-
clean/bin:/opt/gcc/4.9.3/bin:/opt/cuda/7.0/bin:/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/opt/ibutils/i
n:/opt/mendieta/bin:/home/lvargas/bin:/home/lvargas/conae/library/grads-2.0.2/bin)
```

O de esta forma si se está usando ***set_custom_configuration.sh.***

```
checking for perl... found /usr/bin/perl (perl)
Will use NETCDF in dir: /home/<USER>/wrf_mendieta/library/netCDF
Will use PHDF5 in dir: /home/<USER>/wrf_mendieta/library/hdf5-1.8.13
which: no timex in (/opt/gcc/4.9.3/bin:/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/opt/ibutils/b
in:/opt/mendieta/bin:/home/<USER>/bin
```

Se debe verificar que las variables NETCDF y PHDF5 apunten a los path seteados en los archivos ***set_configuration.sh*** (***set_custom_configuration.sh***).

Elegir opciones 34-1 para usar procesos con memoria distribuida: Open MPI

```
34. x86_64 Linux, gfortran compiler with gcc (dmpar)
Compile for nesting? (1=basic) 1
```

Para utilizar Open MPI, se debe actualizar la variable DM_CC con el valor -DMPI2_SUPPORT en el archivo ***configure.wrf***.

```
DM_CC           =       mpicc -DMPI2_SUPPORT
```

En caso de correr WRF en un solo nodo es necesario usar procesos con memoria compartida: OpenMP.

```
33.  x86_64 Linux, gfortran compiler with gcc   (smpar)
Compile for nesting? (1=basic) 1
./compile em_real &> compile.log
```

Comprobar la generación de los siguientes archivos .exe:

```
ls -lt main/*.exe

real.exe
tc.exe
ndown.exe
wrf.exe
```

3.8. Instalación de WPS

```
cd $WPS_DIR
./clean -a
./configure
```

Notar que al iniciar debe dar un mensaje como el siguiente:

```
Will use NETCDF in dir: /home/<USER>/library/netCDF
Found Jasper environment variables for GRIB2 support...
  $JASPERLIB = /home/<USER>/wrf_mendieta/library/jasper/lib

$JASPERINC = /home/<USER>/wrf_mendieta/library/jasper/include
```

Elegir opción 1

Actualizar Vtable[14].

```
cd ungrib/Variable_Tables
wget http://www2.mmm.ucar.edu/wrf/src/Vtable.GFS_new
cd ../../
ln -s ungrib/Variable_Tables/Vtable.GFS_new Vtable
```

[14] Vtable: Tabla que lista los campos de interés a interpolar por WPS

En caso de correr WRF en un solo nodo es necesario se debe declarar explícitamente el uso de OpenMP agregando el flag **-lgomp** a la variable WRF_LIB en el archivo ***configure.wps***.

```
WRF_LIB         =       -L$(WRF_DIR)/external/io_grib1 -lio_grib1 \
                        -L$(WRF_DIR)/external/io_grib_share -lio_grib_share \
                        -L$(WRF_DIR)/external/io_int -lwrfio_int \
                        -L$(WRF_DIR)/external/io_netcdf -lwrfio_nf \
                        -L$(NETCDF)/lib -lnetcdff -lnetcdf -lgomp
./compile &> compile.log
```

Comprobar la generación de los siguientes archivos .exe:

```
ls -lt *.exe
metgrid.exe -> metgrid/src/metgrid.exe
ungrib.exe -> ungrib/src/ungrib.exe
geogrid.exe -> geogrid/src/geogrid.exe
```

Copiar este script:[15]

```
cp $WRF_BASE/link_grib.csh $WPS_DIR
```

3.9. Instalación de ARWpost

```
cd $ARWPOST_DIR
```

Agregar el flag **-lnetcdff** en src/Makefile

```
ARWpost.exe: $(OBJS)
    $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $(OBJS)
        -L$(NETCDF)/lib -I$(NETCDF)/include -lnetcdff -lnetcdf
./clean -a
./configure
```

[15] Es el script original, pero con permisos de ejecución.

Elegir opción 3.

```
./compile
```

Comprobar la generación del siguiente archivo .exe:

```
ls *.exe
ARWpost.exe
```

3.10. Instalación de GrADS

```
cd $WRF_BASE/library
wget ftp://cola.gmu.edu/grads/2.0/grads-2.0.2-bin-CentOS5.8-x86_64.tar.gz
tar -xvzf grads-2.0.2-bin-CentOS5.8-x86_64.tar.gz
rm grads-2.0.2-bin-CentOS5.8-x86_64.tar.gz
cd grads-2.0.2
mkdir data
# Este directorio esta versionado en el repositorio
```

4. Obtención de datos terrestres

Los inputs para etapa de pre-procesamieto (geogrib.exe) se deben descargar de la siguiente manera:

```
cd $WRS_BASE
wget http://www2.mmm.ucar.edu/wrf/src/wps_files/geog_complete.tar.bz2
tar -xjvf geog_complete.tar.bz2
rm geog_complete.tar.bz2

#datos adicionales
mkdir geog && cd geog
wget http://www2.mmm.ucar.edu/wrf/src/wps_files/topo_gmted2010_30s.tar.bz2
tar -xjvf topo_gmted2010_30s.tar
rm topo_gmted2010_30s.tar
```

```
wget http://www2.mmm.ucar.edu/wrf/src/wps_files/topo_30s.tar.bz2
tar -xjvf topo_30s.tar.bz2
rm topo_30s.tar.bz2

wget http://www2.mmm.ucar.edu/wrf/src/wps_files/modis_landuse_21class_30s.tar.bz2
tar -xjvf modis_landuse_21class_30s.tar.bz2
rm modis_landuse_21class_30s.tar.bz2
```

Apéndice C: Scripts para instalación y ejecución de WRF

1. Script para seteo de entorno*:* ***set_configuration.sh.***
2. Script para ejecución del modelo: ***run_wrf_model.py.***
3. Script para descarga de inputs*:* ***get_GFSX025_grib2.py.***
4. Script para solicitud de recursos para N nodos: ***job_wrf.sh.***
5. Script para envió de Jobs a los nodos de Mendieta, ejecución y recopilación de resultados: ***run_wrf_model.sh.***
6. Script para post-procesamiento con GrADS: ***grads_process.sh.***

1. Script para seteo de entorno: *set_configuration.sh*

Este script realiza dos tareas principales:

i. Setea variables de entorno con las herramientas necesarias para la ejecución de WRF.

ii. Crea directorios para alojar los resultados de la ejecución del modelo.

```
#! /bin/bash

################################## WRF Configuration ###################################
# Execution:
# . set_configuration <version_numer>
#
# Example:
# . set_configuration 3.8
########################################################################################

if [ $# -eq 0 ]
  then
    echo "No arguments supplied"
    echo "Execution mode:"
    echo "1 ) . set_configuration <WRF_VERSION_NUMBER> (must be installed)"
    echo "2 ) . set_configuration # take the default value:3.8 "
    echo "Example"
    echo " . set_configuration 3.8 "
    export WRF_VERSION=WRF3.8
  else
    export WRF_VERSION=WRF$1
fi

echo "      __           _______  ______  "
echo "     \ \         / /  __ \|  ____|"
echo "      \ \  /\  / /| |__) | |__    "
echo "       \ \/  \/ / |  _  /|  __|   "
echo "        \  /\  /  | | \ \| |      "
echo "         \/  \/   |_|  \_\_|      "
echo ""

### Environment modules

module load gcc/5
```

```
module load openmpi/2
module load hdf5/1.8.18
module load netcdf/4.4.1.1
module load netcdf-fortran/4.4.4
module list

echo ""
echo "Variables defined:"

echo WRF_VERSION=$WRF_VERSION

export WRF_BASE=$HOME/wrf_mendieta
echo WRF_BASE=$WRF_BASE

export WRF_DIR=$WRF_BASE/$WRF_VERSION/WRFV3         # Directorio principal de WRF
echo WRF_DIR=$WRF_DIR

export WPS_DIR=$WRF_BASE/$WRF_VERSION/WPS           # Directorio principal del pre-procesador WPS
echo WPS_DIR=$WPS_DIR

export ARWPOST_DIR=$WRF_BASE/$WRF_VERSION/ARWpost   # Directorio principal del post-procesador
ARWPost
echo ARWPOST_DIR=$ARWPOST_DIR

export GFS_DIR=$WRF_BASE/gribfiles                  # Directorio para grib files
echo GFS_DIR=$GFS_DIR

export ENSAMBLE_DIR=$WRF_BASE/ensamble             # Directorio para miembros del ensamble
echo ENSAMBLE_DIR=$ENSAMBLE_DIR

export JASPER=$WRF_BASE/library/jasper
echo  JASPER=$JASPER

export JASPERLIB=$JASPER/lib
echo JASPERLIB=$JASPERLIB

export JASPERINC=$JASPER/include
echo JASPERINC=$JASPERINC

export NETCDF=/opt/spack/opt/spack/linux-centos6-x86_64/gcc-5.4.0/netcdf-fortran-4.4.4-
farzkr5oqxp5k2u7bnyaps5g6pnvxffj/
echo NETCDF=$NETCDF

export NETCDF_LIB="${NETCDF}/lib -lnetcdf"
echo NETCDF_LIB=$NETCDF_LIB

export NETCDF_INC=$NETCDF/include
echo NETCDF_INC=$NETCDF_INC

export HDF5=/opt/hdf5/1.8.15-gcc_4.9.2
echo HDF5=$HDF5

export PHDF5=$HDF5
echo PHDF5=$PHDF5

export CPPFLAGS="-I${NETCDF}/include -I${HDF5}/include" cho CPPFLAGS="-I${NETCDF}/include -
I${HDF5}/include"
echo CPPFLAGS=$CPPFLAGS

export LDFLAGS="-L${NETCDF}/lib -L${HDF5}/lib"
#echo LDFLAGS=$LDFLAGS

export LD_LIBRARY_PATH=${NETCDF}/lib:${LD_LIBRARY_PATH}:/opt/netcdf/4.3.3.1-gcc_4.9.2/lib
#echo LD_LIBRARY_PATH=$LD_LIBRARY_PATH

export WRFIO_NCD_LARGE_FILE_SUPPORT=1
echo WRFIO_NCD_LARGE_FILE_SUPPORT=$WRFIO_NCD_LARGE_FILE_SUPPORT

export WRF_EM_CORE=1
echo WRF_EM_CORE=$WRF_EM_CORE

### Folder for grads configuration.
export GADDIR=$WRF_BASE/library/grads-2.0.2/data
echo GADDIR=$GADDIR

export PATH=$PATH:$WRF_BASE/library/grads-2.0.2/bin
```

```
111
112 mkdir -p ensamble
113 mkdir -p output
114 mkdir -p logs
115 mkdir -p gribfiles
```

2. Script para ejecución del modelo: *run_wrf_model.py*

Este script define realiza dos tareas principales:

i. Recibe parametros para la ejecucion del modelo y setea archivos de configuración con dichos parametros: tiempo de pronostico, fecha de inicio, y cantidad de nodos.

ii. Valida que los parametros ingresados son correctos. Para este trabajo se realizan pronosticos como masimo de una semana (168 h). La cantidad de nodos deber ser N en {2, ..., 8}.

iii. Detecta y carga los miembros de ensambles con sus configuraciones.

iv. Descarga los gribfiles necesarios desde GFS.

v. Solicita recursos a SLURM para ejecutar cada miembro de ensamble

```
1   #! /usr/bin/python
2   import os
3   import argparse
4   import sys
5   import get_GFSX025_grib2 as grib
6   from datetime import datetime, timedelta
7   import time
8
9
10  # Default values. Editable by user
11  MAX_OFFSET       = 168         # MAX_OFFSET == 168h(7 days)
12  MIN_NODES_AMOUNT = 2
13  MAX_NODES_AMOUNT = 9
14  # End Default values. Editable by user
15  SEPARATOR = '-' * 80
16
17  def update_namelist_wps(environment):
18
19      try:
20          print(SEPARATOR)
21          print("Set date for namelist.wps")
22
23          ENSAMBLE_DIR = environment["ENSAMBLE_DIR"]
24          os.chdir(ENSAMBLE_DIR)
25          start_date = " start_date = {0}\n".format(environment["start_date"])
26          end_date = " end_date = {0}\n".format(environment["end_date"])
27          patterns = {"start_date": start_date, "end_date": end_date}
28
29          namelist_wps = "namelist.wps"
30
31          with open(namelist_wps) as infile:
32              with open(namelist_wps, 'r+') as outfile:
33                  for line in infile:
34                      for k, v in patterns.iteritems():
35                          if k in line:
36                              line = v
37                              break
```

```
                    outfile.write(line)

        infile.close()
        outfile.close()
        os.system("head -15 " + namelist_wps)

    except Exception:
        raise


def update_namelist_input_output(ensamble_path, environment):

    try:
        print(SEPARATOR)
        print("Set date for namelist.input in {0}".format(ensamble_path))

        ENSAMBLE_DIR = environment["ENSAMBLE_DIR"]
        os.chdir(ENSAMBLE_DIR + "/" + ensamble_path)

        patterns = {
                    "run_days"      : " run_days            = {0}\n".format(environment["run_days"]    ),
                    "run_hours"     : " run_hours           = {0}\n".format(environment["run_hours"]   ),
                    "run_minutes"   : " run_minutes         = {0}\n".format(environment["run_minutes"] ),
                    "run_seconds"   : " run_seconds         = {0}\n".format(environment["run_seconds"] ),
                    "start_year"    : " start_year          = {0}\n".format(environment["start_year"]  ),
                    "start_month"   : " start_month         = {0}\n".format(environment["start_month"] ),
                    "start_day"     : " start_day           = {0}\n".format(environment["start_day"]   ),
                    "start_hour"    : " start_hour          = {0}\n".format(environment["start_hour"]  ),
                    "start_minute"  : " start_minute        = {0}\n".format(environment["start_minute"]),
                    "start_second"  : " start_second        = {0}\n".format(environment["start_second"]),
                    "end_year"      : " end_year            = {0}\n".format(environment["end_year"]    ),
                    "end_month"     : " end_month           = {0}\n".format(environment["end_month"]   ),
                    "end_day"       : " end_day             = {0}\n".format(environment["end_day"]     ),
                    "end_hour"      : " end_hour            = {0}\n".format(environment["end_hour"]    ),
                    "end_minute"    : " end_minute          = {0}\n".format(environment["end_minute"]  ),
                    "end_second"    : " end_second          = {0}\n".format(environment["end_second"]  )
                    }

        with open('namelist.input') as infile:
            with open('namelist.input', 'r+') as outfile:
                for line in infile:
                    for k, v in patterns.iteritems():
                        if k in line:
                            line = v
                            break
                    outfile.write(line)

        infile.close()
        outfile.close()
        os.system("head -15 namelist.input")

        print("Set date for namelist.ARWpost {0}".format(ensamble_path))

        start_date = " start_date = {0}\n".format(environment["start_date"])
        end_date = " end_date = {0}\n".format(environment["end_date"])
        input_root_name =  " input_root_name =
'../wrf_run/wrfout_d01_{0}',\n".format(environment["start_date"])

        patterns = {
                    "start_date": start_date,
                    "end_date": end_date,
                    "input_root_name": input_root_name
                    }

        namelist_awr = "namelist.ARWpost"

        with open(namelist_awr) as infile:
            with open(namelist_awr, 'r+') as outfile:
                for line in infile:
                    for k, v in patterns.iteritems():
                        if k in line:
                            line = v
                            break
                    outfile.write(line)

        infile.close()
```

```
114         outfile.close()
115         os.system("head -15 {0}".format(namelist_awr))
116 
117     except Exception:
118         raise
119 
120 
121 def download_grib_files(environment, offset):
122 
123     try:
124         print(SEPARATOR)
125 
126         GFS_DIR = environment["GFS_DIR"]
127         os.chdir(GFS_DIR)
128         start_date_dir = environment["start_date"]
129         start_date = environment["start_date_int_format"]
130 
131         if not os.path.exists(start_date_dir):
132             os.system("mkdir {0}".format(start_date_dir))
133 
134         gfs_path = GFS_DIR + "/" + start_date_dir
135         grib.download_grib_files(start_date, offset, gfs_path)
136 
137     except Exception:
138         raise
139 
140 
141 def load_configuration(environment, offset):
142 
143     try:
144 
145         update_namelist_wps(environment)
146         ensamble_names = environment["ENSAMBLE"]
147         for ensamble in ensamble_names:
148             update_namelist_input_output(ensamble, environment)
149 
150     except Exception:
151         raise
152 
153 
154 def run_process_model(environment, nodes):
155 
156     try:
157         os.chdir(environment["WRF_BASE"])
158         ensamble_names = environment["ENSAMBLE"]
159         start_date = environment["start_date"]
160         end_date = environment["end_date"]
161         import re
162         with open('job_wrf.sh') as infile:
163             with open('job_wrf.sh', 'r+') as outfile:
164                 for line in infile:
165                     if re.search(r'(--nodes=[0-9]{1})', line):
166                         line = re.sub(r'(--nodes=[0-9]{1})', '--nodes={0}'.format(nodes), line)
167                     outfile.write(line)
168         infile.close()
169         outfile.close()
170 
171         for ensamble in ensamble_names:
172             print(SEPARATOR)
173             execute_command = "sbatch job_wrf.sh {0} {1} {2} {3}".format(ensamble, start_date,
174 end_date, nodes)
175             print(execute_command§)
176             os.system(execute_command)
177 
178         check_command = "squeue -u $USER"
179         print(check_command)
180         os.system(check_command)
181     except Exception:
182         raise
183 
184 
185 def get_ensamble_names(environment):
186     """
187       This function return a list of ensamble's names:
188       [
189        ensamble1,
```

```
        ensamble2,
        .
        .
        .
        ensambleN,
      ]
    """

    try:

        ENSAMBLE_DIR = environment["ENSAMBLE_DIR"]
        os.chdir(ENSAMBLE_DIR)
        ensamble_names = []
        subdirs = [x[0] for x in os.walk(ENSAMBLE_DIR)]
        for subdir in subdirs:
            ensamble_names.append(subdir.split("/")[-1])

        return ensamble_names[1:]

    except Exception:
        raise


def define_environment(start_date, offset):
    """
    Format of start and end date:
    start_date = YYYY-MM-DD_HH:MM:SS
    end_date = YYYY-MM-DD_HH:MM:SS

    Example:
    start_date = 2015-02-24_18:00:00
    """
    try:

        start_date_int_format = int(start_date)
        start_date            = datetime.strptime(start_date, "%Y%m%d%H")
        start_year            = "{0:02d}".format(start_date.year)
        start_month           = "{0:02d}".format(start_date.month)
        start_day             = "{0:02d}".format(start_date.day)
        start_hour            = "{0:02d}".format(start_date.hour)
        start_minute          = "{0:02d}".format(start_date.minute)
        start_second          = "{0:02d}".format(start_date.second)

        end_date              = start_date + timedelta(hours = int(offset))
        end_year              = "{0:02d}".format(end_date.year)
        end_month             = "{0:02d}".format(end_date.month)
        end_day               = "{0:02d}".format(end_date.day)
        end_hour              = "{0:02d}".format(end_date.hour)
        end_minute            = "{0:02d}".format(end_date.minute)
        end_second            = "{0:02d}".format(end_date.second)

        #TODO ASK about these parameters
        run_days              = "0"
        run_hours             = offset
        run_minutes           = "0"
        run_seconds           = "0"

        start_date            = start_date.strftime("%Y-%m-%d_%H:%M:%S")
        end_date              = end_date.strftime("%Y-%m-%d_%H:%M:%S")
        print("Start forecast date: {0}".format(start_date))
        print("End forecast date: {0}".format(end_date))


        environment = {
                        "start_date_int_format" : start_date_int_format,
                        "start_date"            : start_date,
                        "end_date"              : end_date,
                        "offset"                : offset,
                        "start_year"            : start_year,
                        "start_month"           : start_month,
                        "start_day"             : start_day,
                        "start_hour"            : start_hour,
                        "start_minute"          : start_minute,
                        "start_second"          : start_second,
                        "end_date"              : end_date,
                        "end_year"              : end_year,
```

```
                        "end_month"             : end_month,
                        "end_day"               : end_day,
                        "end_hour"              : end_hour,
                        "end_minute"            : end_minute,
                        "end_second"            : end_second,
                        "run_days"              : run_days,
                        "run_hours"             : run_hours,
                        "run_minutes"           : run_minutes,
                        "run_seconds"           : run_seconds
                       }

        if not os.getenv("WRF_BASE"):
            print(SEPARATOR)
            print("Before run this script you should run: . ./set_configuration.sh")
            print(SEPARATOR)
            sys.exit(1)

        print("ENVIRONMENT VARIABLE LOADED: {0}".format(os.getenv("WRF_BASE")))
        environment["WRF_BASE"] = os.getenv("WRF_BASE")
        print("ENVIRONMENT VARIABLE LOADED: {0}".format(os.getenv("GFS_DIR")))
        environment["GFS_DIR"] = os.getenv("GFS_DIR")
        print("ENVIRONMENT VARIABLE LOADED: {0}".format(os.getenv("ENSAMBLE_DIR")))
        environment["ENSAMBLE_DIR"] = os.getenv("ENSAMBLE_DIR")
        environment["ENSAMBLE"] = get_ensamble_names(environment)

        return environment

    except Exception:
        raise


def usage(msg):

    print(SEPARATOR)
    print(SEPARATOR)
    print(msg)
    print(SEPARATOR)
    print(SEPARATOR)
    print("""
       Execution of WRF model:

       ./run_wrf_model.py -i=STARTDATE -o=OFFSET -n=2
       or:
       ./run_wrf_model.py --start_date=STARTDATE --offset=OFFSET --nodes=2

       Where STARTDATE has the follow format: YYYYMMDDHH
       and OFFSET is an integer value that represent the forecast hours
       starting from the STARTDATE and defined in the range [0-MAX_OFFSET] in hours.
       The MAX_OFFSET is currently defined in 168h(a week),
       but it can be editable by the user, changing it in this file.

       The nodes flag is the number of:
       nodes in multi partition, with nodes in [2,8].
       This values can also be changed in this file editing the
       MIN_NODES_AMOUNT/MAX_NODES_AMOUNT variables

       Example:
       ./run_wrf_model.py -i=2018020218 -o=36 -n=2
       means Forecast of 36 h starting from the date:
       year: 2018
       month: 02
       day: 02
       hour: 18
       forecast time: 36 h

       running in 2 nodes of multi partition


       Warning: The date is valid only until 14 days behind
       This is a constrain from the GFS site
      """)
    print(SEPARATOR)
    print(SEPARATOR)

    sys.exit(1)

```

```
342
343 def check_parameter(init_date, offset, nodes, force=None):
344
345     try:
346         date = datetime.strptime(init_date, '%Y%m%d%H')
347         if date < datetime.now() - timedelta(days=14):
348             if not force:
349                 usage(msg="Date available until 14 days ago")
350         if not int(offset) in range(0, MAX_OFFSET + 1):
351             usage(msg="Forecast's hours grater than 168h")
352         if not int(nodes) in range(MIN_NODES_AMOUNT, MAX_NODES_AMOUNT):
353             usage(msg="Mendieta nodes out of allowed range")
354     except ValueError:
355         usage(msg="Error in the date format")
356
357
358 def main():
359
360     print("""
361     __          _______  ______
362      \ \        / /  __ \|  ____|
363       \ \  /\  / /| |__) | |__
364        \ \/  \/ / |  _  /|  __|
365         \  /\  /  | | \ \| |
366          \/  \/   |_|  \_\_|
367         """)
368
369     time.sleep(1)
370
371     start_date = None
372     offset = None
373     nodes = 2  # Default value in multi partition
374
375     parser = argparse.ArgumentParser()
376     parser.add_argument('-i', '--start_date', help='Start date for a forecast')
377     parser.add_argument('-o', '--offset', help='Amount of forecast in hours')
378     parser.add_argument('-n', '--nodes', help='Mendieta nodes')
379     parser.add_argument('-f', '--force', action='store_true', help='force execution for any date')
380
381     args = parser.parse_args()
382
383     if args.start_date and args.offset and args.nodes:
384         start_date = args.start_date
385         offset = args.offset
386         nodes = args.nodes
387         force = args.force
388         check_parameter(start_date, offset, nodes, force)
389     else:
390         usage("Insert all the parameters")
391
392
393     try:
394         environment = define_environment(start_date, offset)
395         download_grib_files(environment, offset)
396         load_configuration(environment, offset)
397         run_process_model(environment, nodes)
398     except Exception:
399         raise
400
401 if __name__ == "__main__":
402     main()
403
```

3. Script para descarga de inputs: *get_GFSX025_grib2.py*

Este script descarga los gribfiles desde GFS reciebiendo la fecha de inicio y finalización para la ejecución del modelo.

```
#0.5 x0 .5 horizontal resolution on the specified geographical domain
# and for the specified meteorological parameters only(slice, i.e.sub - area of global data)

import datetime
import urllib2
import sys
import os
import time

################################################################################

def chunk_report():
    bar_len = 60
    filled_len = 0
    percents = 0
    bar = '=' * filled_len + '-' * (bar_len - filled_len)

    sys.stdout.write('[%s] %s%s ...\r' % (bar, percents, '%'))
    sys.stdout.flush()

def request(url, file):

    try:
        print file
        if os.path.exists(file):
            # TODO here add checksum verification
            return 0
        req = urllib2.Request(url)
        response = urllib2.urlopen(req)
    except urllib2.HTTPError, e:
        print 'The server could not fulfill the request.'
        print 'Error code: ', e.code
        return e.code
    except IOError, e:
        print 'The server could not fulfill the request.'
        print 'IOError code: ', e
        return -1
    except urllib2.URLError, e:
        print 'Failed to reach a server.'
        print 'Reason: ', e.reason
        return -1
    else:
        w = open(file, 'w')
        w.write(response.read())

        chunk_size=8192
        while 1:
            chunk = response.read(chunk_size)
            if not chunk:
                break
            chunk_report()
            w.write(chunk)

        response.close()
        w.close()
        return 0


def download_grib_files(start_date, offset, grib2_dir):

    INIDATE = start_date
    DIR_OPER_DATA = grib2_dir

    # Mode of managing: automatic(WORK_MODE = 'auto': automatic retrials) or manual(WORK_MODE = 'man')
```

```
    # WORK_MODE = 'man'
    WORK_MODE = 'auto'

    # Date of forecast start(analysis)
    DATE = INIDATE / 100
    # Instant(hour, UTC) of forecast start(analysis)

    FCI = INIDATE - DATE * 100

    # Archive tar - file creation option(ARCHIVE = True or ARCHIVE = False)
    ARCHIVE = False

    if ARCHIVE:
        DIR_ARCHIVE = DIR_OPER_DATA
    LON_W = "-96"
    LON_E = "-15"
    LAT_N = "-10"
    LAT_S = "-75"

    #Data grid resolution( in degree)

    ADGRID = "0.25"
    #can be: 0.25, 0.5, 1.0, 2.5

    # Defines connection timeout

    urllib2.socket.setdefaulttimeout(30)

    # Total forecast length( in hours) for which data are requested:
    NHOUR = int(offset)
    # Interval in hours between two forecast times:
    DHOUR = 03
    # Definiton of date( in required format)
    day = datetime.datetime.today()
    ### tomorrow = day + datetime.timedelta(days = 1)### yesterday = day + datetime.timedelta(days = -
1)

    # If the download is made early in the morning, the date is that of yesterday

    #if day.hour < 6: #day = yesterday

    ### day = "%4.4i%2.2i%2.2i" % (day.year, day.month, day.day)# structure definition
    day = "%8.8i" % (DATE)

    FCIA = "%2.2i" % FCI

    print "Date and hour of GFS forecast initial time: ", day, FCI

    # Definition of servers name# The first in the list below of those available is used

    SERVER = []
    SERVER.append("nomads.ncep.noaa.gov/")

    # Definitions of server partial subdir.name

    SERV_PASS = "cgi-bin/"

    #Definition of requested levels and parameters

    LEV_LIST = ["all"]
    PAR_LIST = ["HGT", "LAND", "PRES", "PRMSL", "RH", "SOILW", "SPFH", "TMP", "UGRD", "VGRD", "WEASD",
"TSOIL"]


    if WORK_MODE == "auto":
        COUNTMAX = 50
        icountmax = 100
        S_SLEEP1 = 600
        S_SLEEP2 = 60
    else:
        COUNTMAX = 1
        icountmax = 1
        S_SLEEP1 = 10
        S_SLEEP2 = 1

    NSERVER = len(SERVER)
```

```
    N_LEV_TYPE = 1
    NINSTANT = NHOUR / DHOUR + 1
    NFILE_REQUESTED = [NINSTANT, 1]
    FILE_NAME_DOMAIN = "&subregion=&leftlon=" + LON_W + "&rightlon=" + LON_E + "&bottomlat=" + LAT_S +
"&toplat=" + LAT_N

    if ADGRID == "0.25":
        FILE_NAME_INI = "filter_gfs_0p25.pl"
        DIR_NAME = "&dir=%2Fgfs." + day + FCIA
    if ADGRID == "0.5":
        FILE_NAME_INI = "filter_gfs_hd.pl"
        DIR_NAME = "&dir=%2Fgfs." + day + FCIA + "%2Fmaster"
    if ADGRID == "1.0":
        FILE_NAME_INI = "filter_gfs.pl"
        DIR_NAME = "&dir=%2Fgfs." + day + FCIA
    if ADGRID == "2.5":
        FILE_NAME_INI = "filter_gfs_2p5.pl"
        DIR_NAME = "&dir=%2Fgfs." + day + FCIA

    # Full list of requested files

    LIST_FILE_REMOTE = []
    LIST_FLAG = []
    LIST_FILE_LOCAL_FIN = []
    NINST = NFILE_REQUESTED[0]
    for INST in range(0, NINST):
        if INST + 1 > NFILE_REQUESTED[0]:
            continue
        NLEV = len(LEV_LIST)
        NPAR = len(PAR_LIST)
        PARAMETERS = ""
        for IPAR in range(0, NPAR):
            PARAMETERS = PARAMETERS + "&var_" + PAR_LIST[IPAR] + "=on"
        LEVELS = ""
        if LEV_LIST[0] == "all":
            LEVELS = LEVELS + "&all_lev=on"
        else:
            for ILEV in range(0, NLEV):
                LEVELS = LEVELS + "&lev_" + LEV_LIST[ILEV] + "=on"
        HF = INST * DHOUR
        HFA = "%2.2i" % HF
        HFA2 = "%3.3i" % HF
        if ADGRID == "0.25":
            FILE_NAME_BASE = "?file=gfs.t" + FCIA + "z.pgrb2.0p25.f" + HFA2
        if ADGRID == "0.5":
            FILE_NAME_BASE = "?file=gfs.t" + FCIA + "z.mastergrb2f" + HFA
        if ADGRID == "1.0":
            FILE_NAME_BASE = "?file=gfs.t" + FCIA + "z.pgrbf" + HFA + ".grib2"
        if ADGRID == "2.5":
            FILE_NAME_BASE = "?file=gfs.t" + FCIA + "z.pgrbf" + HFA + ".2p5deg.grib2"
        FILE_REMOTE = FILE_NAME_INI + FILE_NAME_BASE + LEVELS + PARAMETERS + FILE_NAME_DOMAIN +
DIR_NAME
        FILE_LOCAL_FIN = "GFS_" + day + FCIA + "+" + HFA2 + ".grib2"
        FLAG = 0
        LIST_FILE_REMOTE.append(FILE_REMOTE)
        LIST_FILE_LOCAL_FIN.append(FILE_LOCAL_FIN)
        LIST_FLAG.append(FLAG)

    NFILE = len(LIST_FLAG)

    # Dowloading of requested files
    WORK = True
    while WORK:
        for ISERVER in range(0, NSERVER):
            FILE_REMOTE0 = "http://" + SERVER[ISERVER] + SERV_PASS
            print 'Request in server: ', SERVER[ISERVER]

            COUNT = 1
            while COUNT <= COUNTMAX:
                print 'Attempt number: ', COUNT

                NREQ = 0
                NFLAG = 0
                for IFILE in range(0, NFILE):
                    if LIST_FLAG[IFILE] == 0:
                        NFLAG = NFLAG + 1
```

```
                        FILE_REMOTE = FILE_REMOTE0 + LIST_FILE_REMOTE[IFILE]
                        FILE_LOCAL = DIR_OPER_DATA + '/' + LIST_FILE_LOCAL_FIN[IFILE]

                        ierr = 100;
                        icount = 0
                        while ierr != 0 and icount <= icountmax:
                            icount = icount + 1
                            ierr = request(FILE_REMOTE, FILE_LOCAL)
                            print 'dowloading error= ', ierr
                            if ierr == 0: #successeful downloading
                                LIST_FLAG[IFILE] = 1
                                NREQ = NREQ + 1
                                print "Requested remote file downloaded in local file", FILE_LOCAL

                            else:#unsuccesseful downloading
                                print 'Data file', FILE_REMOTE, 'not downloaded! sleep ', S_SLEEP2, '
s'
                                time.sleep(S_SLEEP2)
                if NFLAG == NREQ:
                    WORK = False
                if WORK:
                    print "Not all requested files downloaded, sleeping", S_SLEEP1, "s before next
trial"
                    time.sleep(S_SLEEP1)
                else:
                    print '*****************************************************'
                    print " All requested grib2 files downloaded !", day, FCI, 'UTC'
                    print '*****************************************************'
                    break
                COUNT = COUNT + 1
            if WORK:
                print "All acceptable attempts have been done in this server, sleeping ", S_SLEEP2, "s
before request other server"
                time.sleep(S_SLEEP2)
            else:
                break
        if not WORK:
            break
        else:
            if WORK_MODE == 'man':
                break

```

4. Script para solicitud de recursos para N nodos: *job_wrf.sh*

Este script setea los prametros necesarios para la ejection de un job del modelo. Llama a ***run_wrf_model.sh***. Tambien puede ser ejecutado de manera individual para ejecutar un unico miembro de ensamble.

```
#!/bin/bash

#############################################################################################
### How to run:
### sbatch job_wrf.sh f_Thompson_MYJ 2016-09-28_18:00:00 2016-09-30_06:00:00 N
### With N in {2..8}
###

################################### Slurm configuration ####################################

### Samples of job scripts in /usr/share/doc/mendieta/

#SBATCH --mail-type=ALL
#SBATCH --mail-user=lvc0107@famaf.unc.edu.ar
#SBATCH --job-name=WRF
```

```
#SBATCH --partition=multi
#SBATCH --exclusive

### This value is set with the N value dinamicaly by run_wrf_model.py
### Its default value is 2
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20

### Execution Time. Format: days-hours:minutes. Max time: Four days.
#SBATCH --time 0-3:30

### Environment setup
. /etc/profile
################################# WRF Configuration ###################################

### $1: Ensamble name
### $2: Start Date
### $3: End Date
### $4: Amount of nodes

./run_wrf_model.sh $1 $2 $3 $4
```

5. Script para envío de Jobs a los nodos de Mendieta, ejecución y recopilación de resultados: *run_wrf_model.sh*

Finalemente, este script es el encargado de lanzar la ejecución del job para un miembro de ensamble dado. Realiza las siguientes tareas:

i. Genera directorios temporales para alojar los resultados intermedios de la ejecución de WRF. Soporta una funcionalidad para debugear un JOB ejecutado manualmente y con ID arbitrario 11111.

ii. Ejecuta el preprocesamiento en el nodo master. Esto se debe a que el tiempo de preprocesamiento demora un par de segundos y es mas barato ejecutar en el nodo master que solicitar un nodo de computo, pues el tiempo en cola penaliza demasiado.

iii. Ejecuta el procesamiento del job:

```
time srun ./wrf.exe
```

iv. Una vez finalizado el procesamiento, ejecuta el post-procesamiento: (ARWPost + Grads).

v. Elimina los archivos binarios intermedios generados. Solo mantiene los resultados producidos por la etapa de post-procemamiento.

```
#!/bin/bash
################################# WRF Configuration ###################################

ENSAMBLE=$1
ACTUAL_START_DATE=$2
ACTUAL_END_DATE=$3
NODES=$4
echo ENSAMBLE: $1
echo ACTUAL START DATE: $2
echo ACTUAL END DATE: $3
echo NODES: $4

RUN_PARAMETERS=$NODES'_nodes_'$ENSAMBLE
if [ -z $SLURM_JOB_ID ]; then
    SLURM_JOB_ID=11111
    TEMP_PATH=$WRF_DIR/test/em_real/$RUN_PARAMETERS/$SLURM_JOB_ID
    while [ -d $TEMP_PATH ]
    do
        SLURM_JOB_ID=$[ $SLURM_JOB_ID + 1]
        TEMP_PATH=$WRF_DIR/test/em_real/$RUN_PARAMETERS/$SLURM_JOB_ID
        echo checking for $TEMP_PATH
    done

fi

WPS_RUN_DIR=$WRF_DIR/test/em_real/$RUN_PARAMETERS/$SLURM_JOB_ID/wps_run
WRF_RUN_DIR=$WRF_DIR/test/em_real/$RUN_PARAMETERS/$SLURM_JOB_ID/wrf_run
ARWPOST_RUN_DIR=$WRF_DIR/test/em_real/$RUN_PARAMETERS/$SLURM_JOB_ID/arwpost_run

mkdir -p $WPS_RUN_DIR
mkdir -p $WRF_RUN_DIR
mkdir -p $ARWPOST_RUN_DIR


################################# Model execution ###################################


### Pre-processing configuration

echo Entering directory $WPS_RUN_DIR
cd $WPS_RUN_DIR

cp $WPS_DIR/link_grib.csh .
cp $ENSAMBLE_DIR/namelist.wps .
ln -s $WPS_DIR/geogrid .
ln -s $WPS_DIR/geogrid.exe .

ln -s $WPS_DIR/ungrib .
ln -s $WPS_DIR/ungrib.exe .
ln -s $WPS_DIR/metgrid .
ln -s $WPS_DIR/metgrid.exe .

ln -s $WPS_DIR/ungrib/Variable_Tables/Vtable.GFS_new Vtable
./link_grib.csh $GFS_DIR/$ACTUAL_START_DATE/GFS*

echo ==================== PRE-PROCESSING STARTED ======================

./geogrid.exe
./ungrib.exe
./metgrid.exe


### Processing Configuration

echo Entering directory $WRF_RUN_DIR
cd $WRF_RUN_DIR

ln -sf $WPS_RUN_DIR/met_em.* .
cp $WRF_DIR/run/* .
echo setting $ENSAMBLE
cp $ENSAMBLE_DIR/$ENSAMBLE/namelist.input .

rm -f real.exe
ln -s $WRF_DIR/run/real.exe real.exe

rm -f wrf.exe
```

```
77  ln -s $WRF_DIR/run/wrf.exe wrf.exe
78
79  rm -f ndown.exe
80  ln -s $WRF_DIR/run/ndown.exe ndown.exe
81
82  rm -f nup.exe
83  ln -s $WRF_DIR/run/nup.exe nup.exe
84
85  rm -f tc.exe
86  ln -s $WRF_DIR/run/tc.exe tc.exe
87
88  echo ====================== PROCESSING STARTED ========================
89  echo configuration used: $RUN_PARAMETERS
90  echo JOBID: $SLURM_JOB_ID
91  echo JOBNAME: $SLURM_JOB_NAME
92  echo NODES: $SLURM_JOB_NUM_NODES
93  echo TASK PER NODES: $SLURM_TASKS_PER_NODE
94  echo Cores obteined: $CORES
95  echo SLURM_NODELIST: $SLURM_NODELIST
96
97  echo real.exe execution
98  ###srun ./real.exe
99  ./real.exe
100
101 echo wrf.exe execution
102 echo execution time
103 ###time srun numactl --physcpubind=0-19 ./wrf.exe
104 time srun ./wrf.exe
105
106 echo execution status
107 tail -5 rsl.error.0000
108
109 echo output size:
110 ls -lh wrfout_d01_$ACTUAL_START_DATE
111
112 echo ==================== POST-PROCESSING STARTED ======================
113
114 ### Post-processing configuration
115
116 echo Entering directory $ARWPOST_RUN_DIR
117 cd $ARWPOST_RUN_DIR
118
119 ### Target folder for ARWPost
120 mkdir -p output/meteogramas
121
122 cp $ENSAMBLE_DIR/$ENSAMBLE/namelist.ARWpost .
123
124 rm -f ARWpost.exe
125 ln -s $ARWPOST_DIR/ARWpost.exe ARWpost.exe
126
127 ./ARWpost.exe
128
129 cd output
130 cp $ENSAMBLE_DIR/*.gs .
131 sh $WRF_BASE/grads_process.sh
132
133 LOG_DIR=$WRF_BASE/logs/$RUN_PARAMETERS
134 echo log dir: $LOG_DIR
135 OUTPUT_DIR=$WRF_BASE/output/$RUN_PARAMETERS/meteogramas
136 echo output dir: $OUTPUT_DIR
137 mkdir -p $LOG_DIR
138 mkdir -p $OUTPUT_DIR
139 cp -avr $ARWPOST_RUN_DIR/output/meteogramas/* $OUTPUT_DIR
140 LOGFILE=$WRF_BASE/slurm-$SLURM_JOB_ID.out
141 mv $LOGFILE $LOG_DIR
142
143 ################################# Clean temporary files ###################################
144 cd $WRF_RUN_DIR/../../ ; rm -rf *
145
```

6. Script para post-procesamiento con GrADS: *grads_process.sh*

Este Script llama a los scripts de GrADS. Dischos scripts deben ser configurados por el usuario.

```
#! /bin/bash

################################# GRADS Configuration ###################################
# Execution:
# This script is executed from run_wrf_model.sh script (post_processing stage)
#
# This script should be updated by the user
#########################################################################################

echo ================================================
echo executing grads -pbcx 'run HPC_CBA_Tmax_Min.gs'
grads -pbcx 'run HPC_CBA_Tmax_Min.gs'
echo ================================================
echo executing grads -pbcx 'run HPC_CBA_Rain.gs'
grads -pbcx 'run HPC_CBA_Rain.gs'
echo ================================================
echo executing grads -pbcx 'run meteogramas_Preciptation.gs'
grads -pbcx 'run meteogramas_Preciptation.gs'
echo ================================================
echo executing grads -pbcx 'run meteogramas_rh.gs'
grads -pbcx 'run meteogramas_rh.gs'
echo ================================================
echo executing grads -pbcx 'run meteogramas_Temp.gs'
grads -pbcx 'run meteogramas_Temp.gs'
echo ================================================
echo executing grads -pbcx 'run meteogramas_WindDir.gs'
grads -pbcx 'run meteogramas_WindDir.gs'
echo ================================================
echo executing grads -pbcx 'run meteogramas_WindSpeed.gs'
grads -pbcx 'run meteogramas_WindSpeed.gs'

```

Bibliografía y referencias

[1] Mastrandrea, M. D., Field, C. B., Stocker, T. F., Edenhofer, O., Ebi, K. L., Frame, D. J., & Plattner, G. K. (2010). Guidance note for lead authors of the IPCC fifth assessment report on consistent treatment of uncertainties.

[2] IPCC, 2014: Climate Change 2014: Synthesis Report. Contribution of Working Groups I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Core Writing Team, R.K. Pachauri and L.A. Meyer (eds.)]. IPCC, Geneva, Switzerland, 151 pp

[3] Subbiah, A. R., Bildan, L., & Narasimhan, R. (2008). Background Paper on Assessment of the Economics of Early Warning Systems for Disaster Risk Reduction. World Bank Group for Disaster Reduction and Recovery.

[4] Steppeler, J., Doms, G., Schättler, U., Bitzer, H. W., Gassmann, A., Damrath, U., & Gregoric, G. (2003). Meso-gamma scale forecasts using the nonhydrostatic model LM. Meteorology and atmospheric Physics, 82(1), 75-96.

[5] Kain, J. S., Weiss, S. J., Levit, J. J., Baldwin, M. E., & Bright, D. R. (2006). Examination of convection-allowing configurations of the WRF model for the prediction of severe convective weather: The SPC/NSSL Spring Program 2004. Weather and forecasting, 21(2), 167-181.

[6] Narita, M., & Ohmori, S. (2007, August). Improving precipitation forecasts by the operational nonhydrostatic mesoscale model with the Kain-Fritsch convective parameterization and cloud microphysics. In Preprints, 12th Conf. on Mesoscale Processes, Watervillle Valley, NH, Amer. Meteor. Soc (Vol. 3).

[7] Lean, H. W., Clark, P. A., Dixon, M., Roberts, N. M., Fitch, A., Forbes, R., & Halliwell, C. (2008). Characteristics of high-resolution versions of the Met Office Unified Model for forecasting convection over the United Kingdom. Monthly Weather Review, 136(9), 3408-3424.

[8] Kain, J. S., Weiss, S. J., Bright, D. R., Baldwin, M. E., Levit, J. J., Carbin, G. W., & Thomas, K. W. (2008). Some practical considerations regarding horizontal resolution in the first generation of operational convection-allowing NWP. Weather and Forecasting, 23(5), 931-952.

[9] Weisman, M. L., Davis, C., Wang, W., Manning, K. W., & Klemp, J. B. (2008). Experiences with 0–36-h explicit convective forecasts with the WRF-ARW model. Weather and Forecasting, 23(3), 407-437.

[10] Xue, M., Kong, F., Thomas, K. W., Wang, Y., Brewster, K., Gao, J., & Coniglio, M. (2010). CAPS realtime storm-scale ensemble and convection-resolving high-resolution forecasts for the NOAA Hazardous Weather Testbed 2010 Spring Experiment. In 25th Conference on Severe Local Storms, Amer. Meteor. Soc., Denver, Colorado, October (pp. 11-14).

[11] Grimit, E. P., & Mass, C. F. (2002). Initial results of a mesoscale short-range ensemble forecasting system over the Pacific Northwest. Weather and Forecasting, 17(2), 192-205.

[12] P. Lynch. The origins of computer weather prediction and climate modeling. Journal of Computational Physics, 227(7):3431_3444, 2008.

[13] Integración de modelos numéricos de predicción meteorológica en sistemas de alerta temprana a emergencias. Tesis de Maestría de Andrés Lighezzolo.

[14] E. Kalnay. Atmospheric modeling, data assimilation, and predictability. Cambridge University Press, 2003.

[15] Stensrud, D. J., Wicker, L. J., Kelleher, K. E., Xue, M., Foster, M. P., Schaefer, J. T., & Tuell, J. P. (2009). Convective-scale warn-on-forecast system: A vision for 2020. Bulletin of the American Meteorological Society, 90(10), 1487-1499.

[16] Weisman, M. L., Skamarock, W. C., & Klemp, J. B. (1997). The resolution dependence of explicitly modeled convective systems. Monthly Weather Review, 125(4), 527-548.

[17] Romero, R., Doswell III, C. A., & Riosalido, R. (2001). Observations and fine-grid simulations of a convective outbreak in northeastern Spain: Importance of diurnal forcing and convective cold pools. Monthly weather review, 129(9), 2157-2182.

[18] Speer, M. S., & Leslie, L. M. (2002). The prediction of two cases of severe convection: implications for forecast guidance. Meteorology and Atmospheric Physics, 80(1), 165-175.

[19] Done, J., Davis, C. A., & Weisman, M. (2004). The next generation of NWP: Explicit forecasts of convection using the Weather Research and Forecasting (WRF) model. Atmospheric Science Letters, 5(6), 110-117.

[20] Richardson, D. (2011). From Observations to Forecasts–Part 11: Ensemble products for weather forecasters. Weather, 66(9), 235-241.

[21] Epstein, E. S. (1969) http://www.tandfonline.com/doi/abs/10.3402/tellusa.v21i6.10143

[22] J.J. Ruiz, C. Saulo, and J. Nogués-Paegle. WRF model sensitivity to choose of

Parameterization over South America: validation against surface variables. Monthly Weather Review, 138(8):3342_3355, 2010

[23] Ley de Moore: The Future of Computing Performance: Game Over or Next Level?

[24] TOP500: https://www.top500.org/

[25] Cátedra de Computación Paralela. FaMAF – UNC: https://cs.famaf.unc.edu.ar/~nicolasw/Docencia/CP/2016/

[26] NUMA: http://lse.sourceforge.net/numa/faq/index.html#what_does_numa_stand_for

http://frankdenneman.nl/2015/02/27/memory-deep-dive-numa-data-locality/

[27] Estructura de Clúster similar a Mendieta: http://ccad.unc.edu.ar/files/presentacion-OAC.pdf

[28] Centro de computación y alto desempeño – UNC: http://ccad.unc.edu.ar

http://ccad.unc.edu.ar/equipamiento/cluster-mendieta

[29] MPI: https://computing.llnl.gov/tutorials/mpi/#Whathttps://www.meted.ucar.edu/training_course_es.php?id=19

[30] NetCDF: http://www.unidata.ucar.edu/software/netcdf/docs/netcdf_introduction.html#architecture

[31] NetCDF-Fortran: http://www.unidata.ucar.edu/software/netcdf/docs/building_netcdf_fortran.html

[32] Opciones de física a modelar: http://www2.mmm.ucar.edu/wrf/users/phys_references.html

[33] Esquema de Microfísica de Thompson: http://www2.mmm.ucar.edu/wrf/users/phys_refs/MICRO_PHYS/Thompson.pdf

[34] Esquema de Microfísica de Morrison - 2: http://www2.mmm.ucar.edu/wrf/users/phys_refs/MICRO_PHYS/Morrison.pdf

[35] Esquema de Microfísica de momento doble – clase 6: http://www2.mmm.ucar.edu/wrf/users/phys_refs/MICRO_PHYS/WDM5_6.pdf

[36] Esquema de microfísica de momento simple – clase 6:
http://www2.mmm.ucar.edu/wrf/users/phys_refs/MICRO_PHYS/WSM6.pdf

[37] Esquema de capa límite planetaria Mellor-Yamada-Janjic:
http://www2.mmm.ucar.edu/wrf/users/phys_refs/PBL/MYJ.pdf

[38] Esquema de Eliminación de escala cuasi-normal:
http://www2.mmm.ucar.edu/wrf/users/phys_refs/PBL/QNSE.pdf

[39] Esquema de Mellor-Yamada- Nakanishi Niino, nivel 3:
http://www2.mmm.ucar.edu/wrf/users/phys_refs/PBL/MYNN_part1.pdf

[40] Modelo ETA: http://etamodel.cptec.inpe.br/history/
http://www2.mmm.ucar.edu/wrf/users/phys_refs/SURFACE_LAYER/eta_part1.pdf

[41] Arquitectura de core i7-2600K: https://ark.intel.com/es/products/52214/Intel-Core-i7-2600K-Processor-8M-Cache-up-to-3_80-GHz

[42] Arquitectura de E5-2620V3: https://ark.intel.com/es/products/83352/Intel-Xeon-Processor-E5-2620-v3-15M-Cache-2_40-GHz

[43] Instalación de WRF tomada de referencia:
http://forum.wrfforum.com/viewtopic.php?f=5&t=7099

http://www2.mmm.ucar.edu/wrf/users/docs/user_guide_V3/users_guide_chap2.htm#_Required_Compilers_and_1

http://www2.mmm.ucar.edu/wrf/users/FAQ_files/FAQ_wrf_installation.html

[44] Compilación de WPS:
http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compilation_tutorial.php#STEP5

[45] Manual de WRF:
http://www2.mmm.ucar.edu/wrf/users/docs/user_guide_V3.8/contents.html