Paralelización de la obtención de datos de entrada del modelo de concentraciones de HYSPLIT

Serie de notas técnicas digitales del
Centro de Investigación Atmosférica de Izaña (CIAI)

Nota técnica digital 2 del CIAI

Autora:

Sonia González Cairós

Escuela Técnica Superior de Ingeniería Informática, Universidad de La Laguna

Directores:

Coromoto León Hernández y Gara Miranda Valladares

Escuela Técnica Superior de Ingeniería Informática, Universidad de La Laguna

Colaboradores:

Carlos Luis Marrero de la Santa Cruz

Centro de Investigación Atmosférica de Izaña (AEMET)

Ariel Stein

Air Resources Laboratory (NOAA)

GOBIERNO DE ESPAÑA · MINISTERIO DE MEDIO AMBIENTE Y MEDIO RURAL Y MARINO · AEMet Agencia Estatal de Meteorología · ULL Universidad de La Laguna · NOAA · Gobierno de Canarias

Catálogo de Publicaciones de la Administración General del Estado:
https://cpage.mpr.gob.es/

NIPO: 784-09-011-7
https://doi.org/10.31978/784-09-011-7

Agencia Estatal de Meteorología (AEMET)
C/ Leonardo Prieto Castro, 8
28040 Madrid
http://www.aemet.es/

@Aemet_Esp

https://www.facebook.com/AgenciaEstataldeMeteorologia

Prólogo

El Centro de Investigación Atmosférica de Izaña (CIAI) lleva trabajando desde el año 2004 en el desarrollo e implementación de herramientas numéricas aplicadas a la caracterización y pronóstico de la calidad del aire en zonas de Canarias afectadas por la emisión industrial de contaminantes. En ese año se creó el actual Grupo de Calidad del Aire y Meteorología, en colaboración con La Consejería de Medio Ambiente del Gobierno de Canarias, comenzando con el desarrollo de métodos estadísticos que permitieran prever condiciones atmosféricas favorables al aumento de la contaminación. Poco a poco, las investigaciones que se realizaban dieron lugar al desarrollo de herramientas cada vez más sofisticadas y fiables. Actualmente, el grupo tiene implementados modelos numéricos meteorológicos mesoscales de alta (2 km) y altísima resolución (100 m) que se acoplan con modelos de dispersión para generar predicciones de concentración de contaminantes con un alcance de 72 h dos veces al día. Uno de los modelos utilizados, el HYbrid Single-Particle Lagrangian Integrated Trajectory (HYSPLIT), requiere de recursos computacionales muy altos a la hora de realizar experimentos de dispersión para un número de fuentes y de partículas muy altos, y a la hora de realizar experimentos de predicción por conjuntos en periodos históricos largos. Estos modelos se ejecutan en un cluster principal compuesto por 20 procesadores en el modo rutinario, y en otro secundario, compuesto por 12 procesadores, que se utiliza para la realización de experimentos a tiempo pasado.

Teniendo en cuenta que el volumen de trabajos relacionados con el preproceso y postproceso de los resultados que se obtenían consumían cada vez más recursos de programación, se solicitó financiación al Gobierno de Canarias para la contratación de un informático a través de la Fundación Empresa Universidad (FEU) de la Universidad de La Laguna. Establecidos los contactos con el Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna, se llegó a un acuerdo para la contratación de un alumno de la Escuela Técnica Superior de Ingeniería Informática en su último año de carrera. Como motivación formativa para el alumno, el CIAI solicitó una tarea relacionada con la mejora de HYSPLIT al Air Resources Laboratory (ARL) de la NOAA, con la que tiene establecido acuerdos a través de la Universidad de Huelva como Unidad Asociada al CSIC. Fruto de esta solicitud surgió la necesidad de mejorar la programación paralela del módulo de concentraciones del HYSPLIT que ha sido la finalidad principal del proyecto de fin de carrera aquí publicado.

En nombre del CIAI y del ARL-NOAA, me gustaría agradecer principalmente a su autora, Sonia González Cairós, y a sus directoras, Coromoto León Hernández y Gara Miranda Valladares, por haber aceptado y llevado a cabo con tanta ilusión y rigor científico este proyecto que aportará tantos beneficios a miles de usuarios del HYSPLIT en todo el mundo.

Carlos Luis Marrero de la Santa Cruz
Investigador Principal del Grupo de Calidad del Aire y Meteorología
Centro de Investigación Atmosférica de Izaña
Agencia Estatal de Meteorología (AEMET)

Resumen

En esta memoria se presenta un estudio del funcionamiento del programa HYSPLIT realizado con el objetivo de detectar puntos débiles que permitan mejorar su eficiencia. HYSPLIT es un software que implementa un modelo de cálculo de trayectorias, concentraciones y dispersión de partículas atmosféricas y contaminantes ampliamente utilizado en meteorología. El trabajo se ha desarrollado en el marco de un convenio de colaboración entre la Universidad de La Laguna y la Agencia Estatal de Meteorología (AEMET). Esta institución mantiene proyectos de investigación con la National Oceanic Atmospheric Administration (NOAA) que ha desarrollado el código del programa. HYSPLIT está escrito en FORTRAN y cuenta tanto con versiones secuenciales como paralelas. Las versiones para redes de ordenadores están implementadas usando el paradigma de paso de mensajes con la herramienta MPI (Message Passing Interface). Sin embargo, los usuarios del software paralelo observan que a medida que se aumenta el número de procesadores el rendimiento del programa empeora. Así pues, el principal objetivo al iniciar este trabajo era detectar las causas de este decremento de las prestaciones del programa.

El estudio realizado del código fuente, así como de los resultados obtenidos con un conjunto de problemas reales, permite afirmar que la adquisición de los datos de entrada se realiza en paralelo y al estar el fichero correspondiente compartido mediante NFS (Network File System) se provoca una secuencialización del proceso. La solución que se propone y presenta en esta memoria está basada en el paradigma de programación paralela Maestro-Esclavo. Específicamente, consiste en hacer que la lectura la realice el procesador maestro y éste la difunda (broadcast) al resto de los procesadores participantes, denominados esclavos. Los resultados computacionales obtenidos demuestran la validez de la solución planteada.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La atmósfera es un sistema extremadamente complejo y reactivo donde tienen lugar de manera simultánea numerosos procesos físicos y químicos. Por ello, la evaluación del impacto potencial que sobre la calidad del aire pueden tener las emisiones provenientes de una fuente o conjunto de fuentes, así como el diseño de estrategias costo-efectivas orientadas a su control, demandan un conocimiento preciso de los procesos que determinan la dispersión, la transformación química y el destino final de los contaminantes en la atmósfera. Esto ha supuesto un reto a los meteorólogos de todo el mundo [31], que buscan soluciones en la creación de modelos o herramientas utilizando como base los avances tecnológicos, principalmente de la informática. Una de las aplicaciones más efectivas que ha mostrado excelentes resultados es la modelación numérica de la atmósfera que permite conocer las variables meteorológicas con muchas horas de anticipación, lo que ayuda a las autoridades, entre otras circunstancias, a tomar decisiones importantes ante un posible fenómeno desfavorable. En este sentido, los modelos de calidad del aire [52] son una herramienta de gran valor, ya que en su formulación se incorporan los conocimientos más recientes sobre dinámica atmosférica para modelar, con cierto grado de confianza, los patrones de dispersión, transformación química y deposición de los contaminantes, con lo que se obtiene una estimación de su concentración en la atmósfera. La complejidad de estos modelos, que exigen gran capacidad de cómputo, requiere de herramientas de paralelización para su implementación en modo operativo.

En este capítulo, se presenta una breve descripción de los procesos atmosféricos que tienen mayor impacto en el transporte y el destino de los contaminantes, así como de las características más sobresalientes de los diferentes tipos de modelos que se usan para simular estos procesos. Posteriormente, se realiza una pequeña introducción a la computación paralela, desribiendo las arquitecturas paralelas existentes, así como las principales técnicas y paradigmas de programación paralela y sus correspondientes estándares. Finalmente, y puesto que ha sido la librería utilizada en el presente proyecto, se explica con detalle algunas de las principales características de MPI.

1.1. Modelización de los procesos atmosféricos

La contaminación atmosférica se produce de forma inherente a la actividad humana, siendo una consecuencia no deseable del progreso tecnológico con la que se ha de convivir. Sólo en los últimos años ha alcanzado proporciones que amenazan con poner en peligro el equilibrio del planeta a escala global. No obstante, por la compleja naturaleza del sistema natural involucrado, la atmósfera, las soluciones a los problemas abordados requieren en muchos casos un conocimiento científico riguroso. La conexión entre las emisiones a la atmósfera y los receptores últimos, ya se trate de ecosistemas naturales o del propio ser humano, se produce a través de procesos atmosféricos complejos y complicados, que involucran escalas muy diferentes, con continuas transformaciones físicas y químicas de las distintas especies. En la siguiente sección, se pretende introducir una breve descripción de los principales procesos atmosféricos implicados.

1.1.1. Transporte y contaminación del aire

El término dispersión [52] generalmente se usa para referirse al conjunto de procesos que ocurren en la atmósfera y por los cuales se diluyen, transportan, remueven o transforman químicamente los contaminantes, hasta alcanzar una fuente receptora. En este contexto, la dispersión de los contaminantes está determinada tanto por variaciones locales, regionales o globales del clima, como por diversos procesos atmosféricos íntimamente ligados a la topografía. Así, el movimiento global de las masas de aire tiene su origen en el calentamiento desigual de la superficie de la Tierra; asimismo, los relieves naturales del terreno, e incluso la presencia de edificios, modifican el régimen local de los vientos. Estos fenómenos tienen un efecto directo sobre el movimiento de los contaminantes en la atmósfera. De lo anterior se desprende que si se desea entender y, en alguna medida, modelar la dispersión, la transformación química y el destino de los contaminantes emitidos a la atmósfera, es necesario comprender los procesos atmosféricos básicos que influyen en su movimiento y su transformación, y también las escalas espaciales y temporales en que se registran. Ambos aspectos se presentan a continuación.

1.1.1.1. Escalas de movimiento

La atmósfera se puede describir como un enorme reactor químico al que se introducen y del que se remueven miles de especies químicas sobre un gran conjunto de escalas temporales y espaciales. Por ello, todos los procesos atmosféricos de importancia para la problemática de la contaminación del aire tradicionalmente se estudian sobre la base de esta gama de escalas. En general, dependiendo del autor y del criterio que se utilice para definirlas, el número de escalas de movimiento en la atmósfera varía; para efectos de este capítulo y considerando su simplicidad, se hará referencia a tres escalas: microescala, mesoescala y macroescala.

Los movimientos a microescala son aquellos que pueden ocurrir en una escala espacial del orden de un par de kilómetros y en periodos del orden de segundos a minutos, y cuya causa es, principalmente, la interacción de la atmósfera con la superficie subyacente. Ejemplos de eventos que ocurren en esta escala son los truenos, los relámpagos y las ráfagas

de viento. En cambio, los movimientos a mesoescala tienen una influencia del orden de decenas de kilómetros y ocurren en periodos de unos minutos hasta varios días; se trata, por ejemplo, de islas de calor urbanas, brisas de mar-tierra, brisas de valle-montaña y tormentas eléctricas. Por último, los movimientos a macroescala abarcan la escala global y la sinóptica. En este caso se encuentran la circulación general de los vientos, los huracanes, los sistemas de alta presión (anticiclón) y de baja presión (ciclón), las corrientes de chorro, etcétera, que se desarrollan en una longitud de cientos a miles de kilómetros, y pueden tener una duración de entre un par de días hasta semanas.

1.1.1.2. Transporte atmosférico de contaminantes

Una vez en la atmósfera, los contaminantes experimentan complejos procesos de transporte, mezcla y transformación química, que dan lugar a una distribución espacial y temporalmente variable, tanto en lo que respecta a su concentración, como en términos de su composición en el aire. Así, una vez emitidos, los contaminantes se someten a procesos de transporte por advección, transporte por difusión, transformación química y remoción seca o húmeda, y son afectados, en mayor o menor grado, por las condiciones meteorológicas que prevalecen durante su emisión.

La dispersión de los contaminantes emitidos depende de la cantidad de turbulencia en la atmósfera cercana, turbulencia que se puede crear por el movimiento horizontal y vertical de la atmósfera. Al movimiento horizontal se le llama viento. Así, cuando el transporte de los contaminantes se da con la misma velocidad y en la misma dirección que el viento que los transporta, se le conoce como transporte por advección. Por lo general, una mayor velocidad del viento reduce las concentraciones de los contaminantes al nivel del suelo, ya que facilita la dilución.

1.1.1.3. Deposición seca y deposición húmeda

Al transporte y la dispersión de los contaminantes por efecto del movimiento vertical de la atmósfera se le conoce como transporte por difusión, y puede ser de tipo molecular o turbulento. El primero se refiere al movimiento de las moléculas en el aire por diferencias de concentración entre dos puntos del espacio (gradiente de concentraciones), y tiene poca importancia para fines del estudio de la contaminación del aire. El segundo se debe básicamente a la existencia de remolinos en el aire, que se producen por irregularidades en el terreno (turbulencia mecánica) o por diferencias de temperatura entre las capas atmosféricas (turbulencia térmica).

La turbulencia mecánica se produce por la fricción de las masas de aire en movimiento con la superficie terrestre, y puede afectar a una capa de aire de hasta 1000 metros de altitud. La turbulencia térmica, por su parte, se genera por el intercambio de calor entre la atmósfera y la superficie terrestre. Ambos procesos contribuyen al movimiento vertical de las masas de aire y definen las condiciones de estabilidad atmosférica.

Al mismo tiempo que los contaminantes son transportados en la atmósfera (por difusión o advección), pueden experimentar reacciones químicas que los lleven a formar nuevos contaminantes con propiedades físicas y químicas que, en algunos casos, podrían

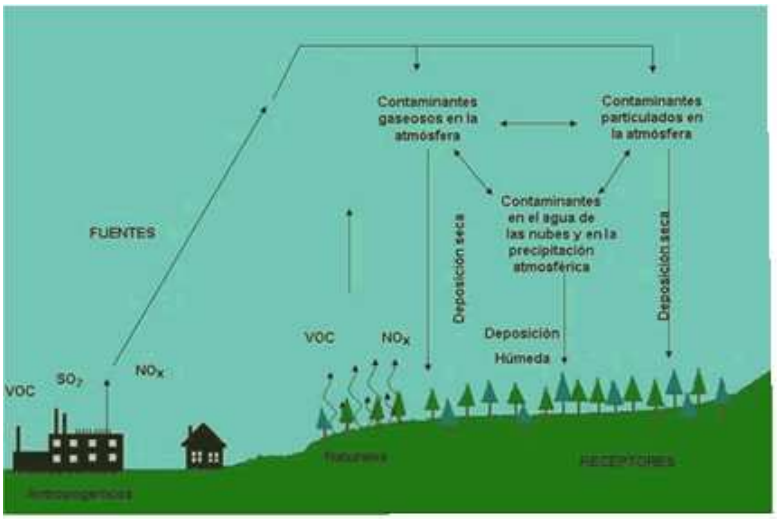

Figura 1.1: Representación esquemática de la deposición húmeda y la deposición seca

significar un mayor riesgo para el ambiente y la salud de la población, que los contaminantes que les dieron origen. Así, los contaminantes secundarios son aquellos que se forman en la atmósfera por reacciones químicas entre contaminantes o entre los contaminantes y sustancias que se encuentran en la atmósfera de manera natural; los ejemplos más característicos son el O_3 y algunos tipos de $PM_{2,5}$, como son los sulfatos y nitratos.

Si bien la transformación química de los contaminantes es un factor importante para determinar su destino final en la atmósfera, también es cierto que la deposición física en la superficie de la Tierra es de gran importancia para muchos contaminantes primarios y secundarios. En general, tanto gases como partículas pueden depositarse sobre la superficie a través de los procesos de deposición seca y deposición húmeda, dependiendo de la fase en que el contaminante haga contacto con la superficie y sea absorbido por ésta. Estos procesos pueden verse de forma esquématica en la Figura 1.1. Cuando un contaminante se disuelve en el agua de una nube, de lluvia o de nieve, y posteriormente las gotas impactan la superficie de la Tierra (incluyendo árboles, edificios, etc.), se dice que el contaminante fue depuesto por vía húmeda. Por el contrario, si el contaminante se transporta a nivel del suelo y se absorbe por los materiales sin que antes se haya disuelto en las gotas de agua de la atmósfera, entonces se trata de deposición seca. Esto es, la distinción entre los dos procesos se refiere al mecanismo de transporte a la superficie y no a la naturaleza de la superficie misma.

De acuerdo con lo anterior, los factores que determinan la importancia relativa del mecanismo mediante el cual se deponen los contaminantes del aire son los siguientes: la naturaleza física del contaminante (gas o partícula); su reactividad química; su morfología; su solubilidad en el agua; y las características climáticas y fisiográficas de la región.

1.1.2. Modelos de simulación de calidad del aire

Un modelo de simulación de la calidad del aire [52] es una herramienta de análisis que permite simular de manera integral, a través de expresiones matemáticas, los procesos atmosféricos que intervienen en el transporte, la dispersión, la deposición y, en algunos casos, la transformación química de los contaminantes. Con estos modelos es posible relacionar directamente las concentraciones ambientales de los contaminantes con sus fuentes de emisión (en el caso de los contaminantes primarios como el CO), o con la emisión de sus precursores (en el caso de los contaminantes secundarios como el O_3, los sulfatos y nitratos), incluyendo en la modelación variables tales como las condiciones topográficas, el uso del suelo y la meteorología de una región determinada.

Dada su gran variedad y los grados de detalle con que tratan los procesos atmosféricos, los modelos actualmente se usan para simular una diversidad de fenómenos atmosféricos que abarcan desde la química atmosférica global hasta la dispersión de contaminantes locales. En general, con los modelos de simulación de la calidad del aire es posible responder o ayudar a responder preguntas tales como:

- ¿Cuál es la contribución de una fuente de emisión a la concentración ambiental de un contaminante?
- ¿Cuál es la estrategia más efectiva para reducir la concentración ambiental de un contaminante?
- ¿Cuál será el efecto sobre la calidad del aire al aplicar una medida de control?
- ¿En dónde se debería colocar en el futuro una nueva fuente (por ejemplo, un complejo industrial) para minimizar su impacto ambiental?
- ¿Cuál será la calidad del aire el día de mañana o de pasado mañana?
- ¿Dónde debe ubicarse una nueva estación o red de estaciones de monitorización?

1.1.2.1. Estructura y entradas de un modelo de simulación

Las diferencias entre los distintos tipos de modelos de la calidad del aire radican fundamentalmente en el número de procesos atmosféricos considerados, el nivel de profundidad con que son tratados, y los métodos utilizados para resolver las ecuaciones que los describen. En general, operan con un conjunto de datos de entrada que caracterizan las emisiones, la topografía y la meteorología de una región, y producen salidas que describen la calidad del aire en dicha región, tal como se aprecia en la Figura 1.2. En el caso de los modelos más avanzados, también se incluye un mecanismo químico que describe las transformaciones químicas de los contaminantes. Es importante destacar que la cantidad y el grado de detalle de la información necesaria para alimentar un modelo de la calidad del aire varían de acuerdo con el tipo de modelo y con la naturaleza del estudio que se pretenda realizar.

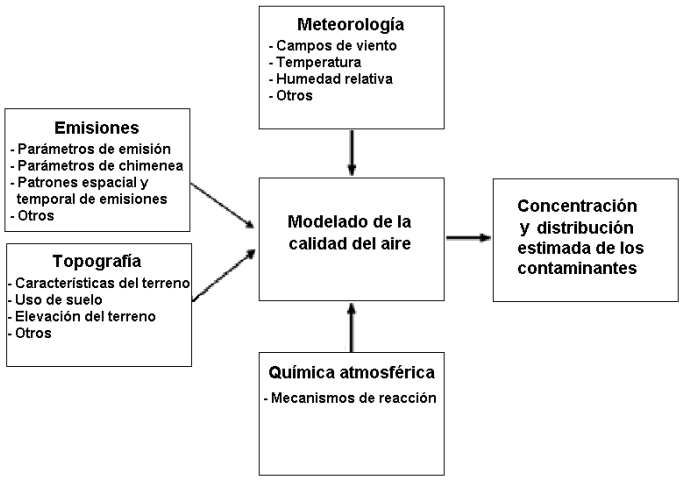

Figura 1.2: Entradas requeridas por un modelo de calidad del aire

Los modelos matemáticos de la calidad del aire se basan en la descripción fundamental de los procesos atmosféricos o en el análisis estadístico de datos. Por ello, se clasifican en modelos estadísticos y modelos deterministas (véase la Figura 1.3). Los modelos estadísticos se basan en las relaciones estadísticas existentes entre los datos históricos y las mediciones disponibles, en tanto que los modelos deterministas lo hacen en una descripción matemática de los procesos atmosféricos, estableciendo una relación causa (emisiones) - efecto (contaminación del aire).

Un ejemplo de un modelo estadístico es el pronóstico de la concentración de un contaminante como una función estadística de las mediciones actuales disponibles y de sus tendencias históricas. Por otra parte, un ejemplo de un modelo determinista es un modelo de difusión, en el cual las concentraciones ambientales de los contaminantes se calculan a partir de la simulación de los procesos atmosféricos, utilizando como entrada la información sobre la fuente de emisión (por ejemplo, tasas de emisión) y su entorno (por ejemplo, parámetros meteorológicos y topografía).

Los modelos deterministas son los más importantes para aplicaciones prácticas dado que, si son apropiadamente calibrados y usados, proporcionan una relación fiable entre la fuente de emisión de contaminantes y las áreas receptoras (o de impacto). Dicho en otras palabras, solamente un modelo determinista puede evaluar la fracción con la que cada fuente emisora participa en las concentraciones ambientales de cada contaminante en el área receptora o de impacto, permitiendo así el diseño o la evaluación de estrategias de control de emisión.

Los modelos atmosféricos deterministas pueden clasificarse de diferentes maneras de acuerdo con el criterio que se tome como referencia; por ejemplo, su escala espacial, o bien, la forma en que plantean las ecuaciones que describen el comportamiento de los contaminantes en la atmósfera. Por su escala espacial, los modelos deterministas se pueden clasificar en modelos a microescala, mesoescala, regionales, sinópticos y globales. En la

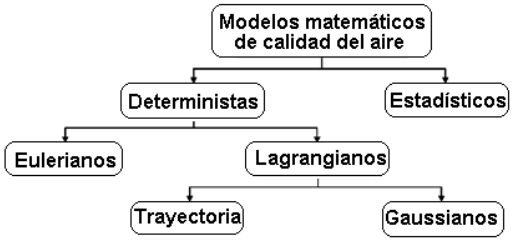

Figura 1.3: Tipos de modelos matemáticos de calidad del aire

Tabla 1.1 se presenta el domino y resolución típicos asociados a los modelos deterministas, según su escala espacial. Por la forma en que plantean las ecuaciones que describen el comportamiento de los contaminantes en la atmósfera, se les puede clasificar como modelos eulerianos (que usan un sistema de coordenadas fijo con respecto a la tierra) y lagrangianos (que usan un sistema de coordenadas que sigue el movimiento de la atmósfera). Puede apreciarse esta clasificación en la Figura 1.3.

Los modelos eulerianos representan la clase más elaborada de modelos atmosféricos. En ellos, las ecuaciones que describen el movimiento y la transformación química de los contaminantes en la atmósfera se resuelven considerando un sistema fijo de coordenadas, y la región a modelar se puede dividir en celdas o cajas, tanto horizontal como verticalmente. La concentración de los contaminantes en cada celda se estima a intervalos específicos de tiempo, teniendo en cuenta la información sobre campos meteorológicos tridimensionales, así como las concentraciones iniciales de los contaminantes, las emisiones, el transporte, la dilución y las transformaciones químicas. La aplicación de estos modelos resulta más conveniente cuando existen patrones complejos de emisión (por ejemplo, numerosas y diversas fuentes de emisión, dispersas en un área geográfica amplia), o cuando los procesos de transformación química desempeñan un papel relevante en la generación y el destino de los contaminantes (por ejemplo, para contaminantes secundarios). En general, cuando se cuenta con información suficientemente detallada sobre el inventario de emisiones, la calidad del aire y la meteorología, estos modelos pueden aplicarse para evaluaciones detalladas de la calidad del aire, urbanas o regionales.

Los modelos lagrangianos se caracterizan por hacer uso de un sistema de referencia que se ajusta al movimiento atmosférico. Es decir, tanto las emisiones y reacciones, como la deposición y el mezclado de los contaminantes, se analizan para un volumen de aire que va cambiando su posición de acuerdo con la velocidad y la dirección del viento (y no para una región entera, como en los eulerianos). Con este esquema general, los modelos lagrangianos se pueden clasificar en modelos de trayectoria y modelos gaussianos, como se puede apreciar en la Figura 1.3, de acuerdo con la geometría del sistema de modelación. Los procesos antes mencionados se pueden simular para una columna hipotética de aire, como en los modelos de trayectoria; cuando la simulación se hace para una pluma de emisión (masa de aire que contiene contaminantes vertida a la atmósfera), continua o discreta (como paquetes denominados *puffs*), se trata de modelos gaussianos.

Modelo	**Dominio típico**	**Resolución típica**
Microescala	200 x 200 x 100 m	5 m
Mesoescala (urbano)	100 x 100 x 5 km	2 km
Regional	1000 x 1000 x 10 km	20 km
Sinóptico (continental)	3000 x 3000 x 20 km	80 km
Global	65000 x 65000 x 20 km	555 x 555 km

Tabla 1.1: Modelos atmosféricos de acuerdo con su escala espacial

En los modelos de trayectoria se define una columna hipotética de aire que se desplaza bajo la influencia de los vientos dominantes, y se asume que no hay intercambio de masa entre la columna y sus alrededores, excepto por las emisiones que ingresan a la columna por la base durante su recorrido. La columna se mueve continuamente, de tal forma que el modelo estima la concentración de los contaminantes en diferentes lugares y momentos a partir de las concentraciones iniciales, las emisiones y las transformaciones químicas. Su aplicación es recomendable en evaluaciones de la calidad del aire que consideren el transporte a grandes distancias, para modelar el comportamiento de masas individuales de aire, e incluso para evaluar la calidad del aire en casos donde existan limitaciones de información para caracterizar las emisiones y la meteorología de una región completa.

Finalmente, en los modelos gaussianos se describe el transporte y la mezcla de los contaminantes asumiendo que las emisiones presentan, en las direcciones horizontal y vertical, una distribución normal o de curva gaussiana, con una concentración máxima en el centro de la pluma. Generalmente estos modelos se aplican para evaluar la dispersión de contaminantes provenientes de fuentes puntuales, aunque en ocasiones también se aplican para simular emisiones de fuentes de área y de línea. Otra característica de este tipo de modelos es que normalmente son aplicados para evaluar la dispersión de contaminantes primarios no reactivos, aunque existen versiones que incluyen en su formulación consideraciones especiales para poder simular procesos de deposición y transformación química.

1.1.2.2. Criterios de selección y aplicación de un modelo

La decisión sobre el modelo más adecuado a utilizar en un estudio específico depende de varios factores, entre los que se pueden mencionar los siguientes:

- *El problema a resolver*: depende en gran medida de si se analizará la dispersión de un contaminante primario o secundario; si el contaminante es reactivo o no reactivo; si se estudiarán las emisiones de una o varias fuentes.
- *La extensión geográfica del área de estudio*: se debe considerar que haya transporte de corto o de largo alcance y que se cuente con información suficiente para caracterizar la meteorología, la topografía y las emisiones de una zona industrial, de un valle o de una ciudad.
- *La complejidad topográfica y meteorológica del área de estudio*: es fundamental analizar las características de la topografía (plana o accidentada) y de la meteorología, de

forma que sea posible caracterizarla adecuadamente con datos de superficie, o si se requieren datos de altura.

- *El grado de detalle y la exactitud requeridos para el análisis*: es necesario decidir si los resultados deben de tener una resolución espacial de unos cuantos kilómetros o de una región completa.
- *Los recursos técnicos y humanos disponibles*: estas consideraciones prácticas incluyen las características del equipo de cómputo (alta o baja capacidad de memoria y procesamiento), y la experiencia del personal, tanto en la aplicación de modelos, como en el procesamiento de los datos que se utilizan como entradas para los modelos y en la interpretación de los resultados de la simulación.
- *El detalle y la calidad de las bases de datos disponibles*: es indispensable analizar el tipo de datos de entradas con que se cuenta, como son los datos de las emisiones (para una sola fuente o para una región completa), la fiabilidad, el grado de detalle y la precisión de la información disponible.

1.1.2.3. Modelos de simulación

Algunos de modelos de simulación [33] más utilizados son:

- **MM5 (*Fifth-Generation Mesoscale Model*)**
  Es un modelo numérico de predicción meteorológica para áreas limitadas [8] que fue desarrollado por la Universidad Estatal de Pennsylvania y el NCAR (*National Center for Atmospheric Research*) [19], en Estados Unidos. Este último se encarga del soporte operativo. MM5 puede ser utilizado para realizar simulaciones de fechas anteriores (reanálisis), actuales o a manera de pronóstico. Se trata de un modelo de tipo euleriano que hace uso de la asimilación de datos en cuatro dimensiones o FDDA (*Four-Dimensional Data Assimilation*), es decir, en los ejes X, Y, Z, y el tiempo. Se puede aplicar a cualquier zona del mundo, ya que es posible desarrollar la información de entrada (campos meteorológicos, topografía y uso de suelo) necesaria para su funcionamiento.

  MM5 es un modelo complejo pero muy completo. Los fenómenos meteorológicos y su parametrización están descritos en más de 100000 líneas de código fuente y cerca de 1000 subrutinas (funciones específicas de un lenguaje de programación). El código fuente esta escrito en lenguaje de programación FORTRAN 90/77, y cuenta con una amplia gama de parametrizaciones de capa límite planetaria, convección, física de nubes, etc. Algunas de las principales características del modelo MM5 se describen a continuación:

  - Capacidad de anidamiento múltiple con interacción (hasta nueve dominios corriendo simultáneamente e intercambiando información entre ellos) en ambas direcciones (*two-way*) entre los dominios de una forma hidrostática o no hidrostática, lo que facilita el estudio de fenómenos atmosféricos bajo distintas escalas espaciales y el diseño de predicciones a muy alta resolución.

- Formulación de una dinámica no hidrostática, la cual permite que el modelo pueda ser empleado eficazmente para representar fenómenos con dimensiones de muy pocos kilómetros.
- Inicialización automática con diferentes fuentes de análisis meteorológicos y observaciones, incluyendo su capacidad de asimilación 4-dimensional de datos.
- Asimilación variacional de datos convencionales y de satélite durante la predicción.
- Incorporación de los más modernos y realistas esquemas de parametrización de los procesos físicos relacionados con la radiación atmosférica, microfísica de nubes y precipitación, convección por cúmulos, turbulencia, y flujos de energía y momento sobre la superficie terrestre.
- Adaptación informática para múltiples plataformas y para su ejecución en modo multitarea sobre computadoras de memoria compartida o distribuida.

Se puede consultar más información sobre este modelo en su página oficial [17].

- **WRF (*Weather Research and Forecasting*)**
  Es un modelo atmosférico de área limitada de última generación [7], [29], [64] desarrollado en NCAR [19]. Está diseñado tanto para la predicción operativa como para tareas de investigación de la dinámica atmosférica. Se puede considerar como el sucesor evolucionado del modelo MM5. El código fuente esta escrito en FORTRAN 90 y dispone de dos módulos dinámicos para la resolución de las ecuaciones primitivas de la atmósfera, de una amplia variedad de parametrizaciones físicas y de un módulo de asimilación variacional (3DVar y 4DVar). Incorpora los últimos avances en la representación física de la atmósfera, en integración numérica y en asimilación de datos. Su arquitectura permite aprovechar diversas formas de paralelismo computacional. Está diseñado para ser un sistema de simulación atmosférica de última generación, flexible, portable y eficiente. Esto hace que sea adecuado para un amplio rango de aplicaciones en diferentes escalas espaciales y temporales.

  WRF es actualmente empleado en forma operacional por el NCEP (*National Centers for Environmental Prediction*) [20]. Este sitio proporciona la información sobre los trabajos de desarrollo sobre el WRF y su organización, referencias a proyectos y pronósticos en los que se usa dicho modelo, y se vincula a la página de los usuarios WRF, usos en tiempo real y acontecimientos relacionados con el mismo.

- **HYSPLIT *(HYbrid Single Particle Lagrangian Integrated Trajectory)***
  Este modelo [12] es una herramienta para determinar la emisión, transporte, dispersión y deposición de partículas. El método de cálculo usado por HYSPLIT se basa en un modelo híbrido entre el euleriano y el lagrangiano. Ha sido desarrollado conjuntamente por el *Bureau of Meteorology of Australia* [4] y el ARL (*Air Resources Laboratory*) [1] de la NOAA (*National Oceanic and Atmospheric Administration*) [21] en Estados Unidos. La versión inicial fue desarrollada por Draxler y Taylor en 1982 [45]. La última versión disponible del modelo es la 4.9.

El código fuente de HYSPLIT está escrito en FORTRAN 77/90 y no es de dominio público. Sin embargo, sí posible hacer uso de HYSPLIT de forma gratuita, estando disponible la versión para PC, que puede ser descargada desde su página oficial para Windows o Mac. Asimismo, puede ejecutarse HYSPLIT de forma interactiva a través de la website READY (*Real-time Environmental Applications and Display sYstem*) [25].

Algunas de las múltiples características del modelo HYSPLIT son las siguientes:

- Esquema de advección: predictor-corrector
- Interpolación lineal espacial y temporal de datos meteorológicos provenientes de fuentes externas
- Emisión simultánea de múltiples fuentes
- Dispersión de *puffs* y partículas simples calculada a partir de la varianza en las velocidades
- Concentraciones calculadas con distribuciones Top-Hat/Gaussianas o partículas-en-celda
- Meteorología y/o mallas de concentración simultáneas y múltiples
- Herramientas para visualizar y manipular información meteorológica
- Utilidades para la conversión de datos a otros formatos: GIF, GrADS, ArcView, Vis5D, postscript, etc

Se ofrece información más detallada sobre el modelo HYSPLIT en el capítulo 2.

1.2. Computación paralela

Actualmente se están planteando en numerosos y diferentes campos de estudio, problemas tratables (problemas donde el número de operaciones sea una función que crezca moderadamente con el tamaño del problema y que pueden ser resolubles en un computador en un tiempo razonable), pero con características que lo hacen de difícil solución con los computadores secuenciales disponibles hoy en día. Este tipo de problemas suponen una demanda continua de una potencia computacional superior que especialmente se pone de manifiesto en problemas que requieren costosos cálculos iterativos sobre grandes cantidades de datos y fuertes restricciones temporales (predicción meteorológica, biocomputación, astrofísica, etc), así como en el modelado y simulación numérica de problemas en ciencia e ingeniería. Los problemas anteriormente citados son especialmente susceptibles de beneficiarse de la computación paralela y representan su ámbito de aplicabilidad fundamental.

La computación paralela [51], [61] permite la ejecución simultánea de múltiples instrucciones. Se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas que pueden resolverse de forma concurrente ("en paralelo"), permitiendo:

- Reducir el tiempo de cómputo
- Procesar gran cantidad de datos
- Resolver problemas complejos
- Proveer concurrencia
- Ampliar los límites de memoria para resolver problemas grandes
- Utilizar recursos remotos de cómputo cuando los locales son escasos
- Reducir costos usando múltiples recursos "baratos" en lugar de costosas supercomputadoras

Entre las consideraciones más importantes que deben tomarse a la hora emplear la computación paralela, destacan: la arquitectura paralela y el modelo de programación paralela a utilizar. En las siguientes subsecciones, se presenta una breve introducción a ambas.

1.2.1. Arquitecturas paralelas

La clasificación de los computadores [35], [65] puede realizarse atendiendo a múltiples factores, sin embargo, quizás la clasificación más conocida y citada sea la taxonomía de Flynn. En ella se utiliza la ausencia o presencia de un control global que permite regular el flujo de instrucciones y datos del computador. De esta forma aparecen cuatro clases de computadores:

- **SISD** *(Single Instruction Single Data)*: un flujo único de instrucciones se ejecuta sobre un único conjunto de datos. A esta clase pertenecen los computadores secuenciales.
- **SIMD** *(Single Instruction Multiple Data)*: un único flujo de instrucciones se ejecuta sobre diferentes conjuntos de datos. Las máquinas vectoriales pertenecen a esta clase.
- **MISD** *(Multiple Instruction Single Data)*: diferentes flujos de instrucciones se ejecutan sobre el mismo conjunto de datos. Es difícil encontrar máquinas reales que respondan a este modelo, debiéndose considerar su existencia exclusivamente a nivel conceptual.
- **MIMD** *(Multiple Instruction Multipe Data)*: diferentes flujos de instrucciones se ejecutan sobre diferentes conjuntos de datos. Desde el punto de vista teórico, representa el modelo más versátil de los cuatro. La mayoría de los sistemas multicomputadores y multiprocesadores (definidos más abajo) pertenecen a esta clase, como: IBM SP, Intel Paragon y Cray T3D.

Atendiendo a la clasificación de Flynn, las arquitecturas paralelas pueden ser esencialmente de dos tipos: SIMD y MIMD.

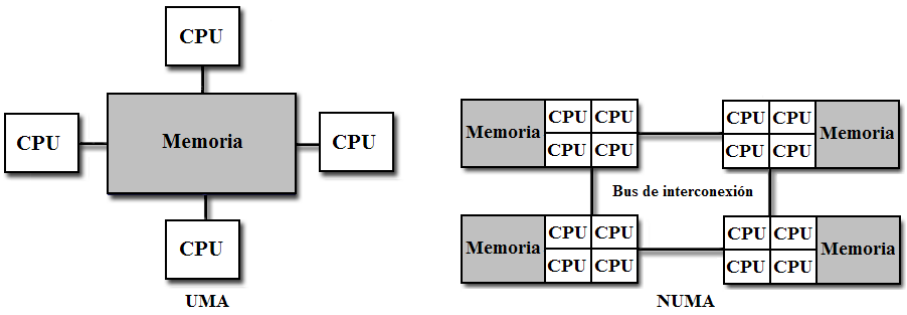

Figura 1.4: Distintos sistemas de memoria compartida

Otra forma de clasificar a los computadores paralelos se basa en la distribución de la memoria. En este caso se puede distinguir entre sistemas de memoria compartida y sistemas de memoria distribuida.

- **Sistemas de memoria compartida (*multiprocesadores*)**
  En los sistemas de memoria compartida [35], todos los procesadores acceden al mismo espacio físico de memoria. Las restricciones de acceso a la memoria son únicamente de tipo físico y no de tipo lógico, en el sentido de que el tiempo de acceso a una posición de memoria puede ser diferente en función del módulo en que se encuentre la posición de memoria. Una red de interconexión une los distintos procesadores con los módulos de esta memoria compartida. La comunicación entre los procesadores se consigue utilizando un espacio de direcciones común. Dentro de los sistemas de memoria compartida podemos encontrar:

  - **UMA** *(Uniform Memory Access)*: multiprocesadores con acceso a memoria uniforme. El tiempo de acceso a cualquier posición de memoria es constante.
  - **NUMA** *(Non-Uniform Memory Access)*: multiprocesador con acceso a memoria no uniforme. El tiempo de acceso a unas posiciones de memoria es mayor que a otras.

  El esquema de básico de las estructura de los sistemas UMA y NUMA pueden verse en la Figura 1.4. Cabe destacar que los accesos a memoria y la gestión de los correspondientes conflictos de acceso normalmente se convierten en un cuello de botella que condicionan el rendimiento de estas máquinas. Es por ello que la red de interconexión es determinante para la eficacia de este tipo sistemas.

- **Sistemas de memoria distribuida (*multicomputadores*)**
  En los sistemas de memoria distribuida (Figura 1.5), cada procesador cuenta con su propia memoria local, no existiendo una memoria global común. Una red de interconexión une los distintos elementos de procesamiento entre sí. La comunicación entre los procesadores se realiza mediante paso de mensajes. Como ejemplo de este tipo de sistemas podemos encontrar:

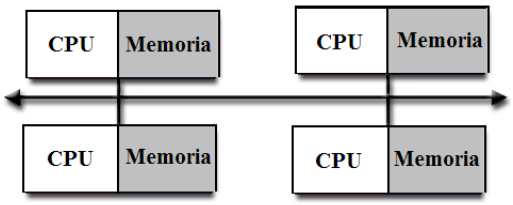

Figura 1.5: Sistema de memoria distribuida

- **Procesadores masivamente paralelos (*MPP*)**: un único computador con múltiples procesadores conectados por un bus de datos.
- **Clusters**: múltiples computadores conectados mediante una red de interconexión rápida. Existen múltiples tipos de clusters, como los Beowulf, que tienen computadores dedicados o los NOW (*Network Of Workstations*), que están constituidos por computadores no dedicados.

Es importante tener en cuenta que el rendimiento de los multicomputadores viene determinado por las comunicaciones y su gestión, las cuales juegan un papel decisivo, similar al que representan los accesos a memoria en el caso de los multiprocesadores de memoria compartida.

1.2.2. Modelos de programación paralela

Los paradigmas de programación [34], [60] no están unívocamente ligados a las arquitecturas paralelas, aunque un determinado modelo puede ser más eficiente que otro al programar sobre determinadas arquitecturas paralelas. En esta sección se presentan los principales paradigmas de programación paralela: programación mediante paso de mensajes y programación mediante memoria compartida, utilizados normalmente en los sistemas paralelos de memoria distribuida y los sistemas paralelos de memoria compartida, respectivamente.

1.2.2.1. Programación mediante memoria compartida

Uno de los modelos de programación usados en sistemas con memoria compartida [43] es la programación multithread. Antes de explicar en qué consiste, es preciso introducir el concepto de *thread*. Un *thread* o hilo de ejecución puede definirse como un flujo de instrucciones independiente que permite ser planificado para su ejecución como tal por el sistema operativo. Desde la perspectiva del programador, se asocia a un proceso que se ejecuta de forma independiente de su programa principal. Esta idea puede generalizarse de modo que si el programa principal tiene más de un procedimiento, todos los procedimientos se podrían planificar para su ejecución simultáneamente e independientemente por sistema, lo que describiría un programa multithread. Un *thread* normalmente comparte su memoria con otros *threads*, mientras que para los procesos, normalmente se tienen diferentes áreas

de memorias para cada uno de ellos. La ventaja de utilizar un grupo de *threads* en lugar de un grupo de procesos es que el cambio de contexto entre *threads* es mucho más rápido que el cambio de contexto entre procesos. Además, las comunicaciones entre dos *threads* son, generalmente, más sencillas y rápidas de implementar que la comunicación entre dos procesos.

El modelo de programación multithread proporciona a los desarrolladores una abstracción bastante útil para la ejecución concurrente. Sin embargo, la aplicación más interesante de esta tecnología aparece cuando se aplica con el fin de permitir la ejecución de múltiples tareas en un sistema paralelo. Cuando el sistema dispone de más de una CPU o CPU con múltiples núcleos, los *threads* se ejecutan independientemente en un marco de auténtico paralelismo. En tal caso, el programador es responsable de evitar las condiciones de carrera (*race conditions*) y otros comportamientos no intuitivos y no deseados. Para obtener una correcta manipulación de los datos, los *threads* deben sincronizarse cada cierto tiempo y así procesar los datos en orden correcto. Los *threads* también pueden realizar operaciones atómicas (implementadas frecuentemente mediante semáforos) con el fin de prevenir que los datos comunes sean modificados de forma simultánea o leídos por un *thread* al mismo tiempo que están siendo modificados por otro *thread*.

Dos ejemplos de estándares en memoria compartida son POSIX y OpenMP.

- **POSIX** (*POrtable Operating System Interface (UNIX)*)
  Un estándar en memoria compartida para sistemas UNIX es IEEE POSIX 1003.1c-1995, una interfaz para programación mediante threads utilizando el lenguaje C. Las implementaciones que se adhieren a este estándar se conocen como librerías de *threads* POSIX [28] o de *Pthreads*. Se considera que las primitivas que se proporcionan son de bajo nivel y que el esfuerzo y conocimiento requeridos por el programador es elevado.

- **OpenMP**
  OpenMP [22], [40] constituye un esfuerzo de estandarización de la programación paralela en máquinas de memoria compartida que comenzó en 1997. Se trata de una API (*Aplication Programming Interface*) que puede ser utilizada para diseñar explícitamente programas paralelos multithread en memoria compartida, permitiendo al usuario, a alto nivel, introducir las directivas adecuadas. OpenMP ofrece estandarización, facilidad de uso y portabilidad, y está compuesto por tres elementos: directivas de compilación, variables de entorno y rutinas de librería.

1.2.2.2. Programación mediante paso de mensajes

El modelo de paso de mensajes [59] es uno de los paradigmas más antiguo y más utilizado en la programación de máquinas paralelas. Parte de su éxito se debe a los mínimos requerimientos que impone el hardware para poder implementarse. El paralelismo es expresado explícitamente por el programador, buscando formas de extraer concurrencia del código secuencial, lo que constituye, probablemente, uno de sus mayores inconvenientes, siendo responsabilidad de éste resolver cuestiones como las dependencias de datos y evitar

interbloqueos (*deadlocks*) y condiciones de carrera (*race conditions*). El modelo de paso de mensajes presenta las siguientes características:

- Existe un conjunto de procesos o tareas que utilizan su propia memoria durante la computación.
- En un mismo procesador pueden residir físicamente varios procesos, sin embargo, es bastante común la ejecución de un único procesos por procesador.
- Los procesos se comunican mediante el envío y recepción de mensajes.
- La transferencia de los datos entre procesos requiere operaciones de tipo cooperativo que deben ser ejecutadas en cada proceso (una operación de envío se corresponde con una operación de recepción).

Dos ejemplos de especificaciones de librerías de pasos de mensajes [39], [48] son:

- **PVM** (*Parallel Virtual Machine*)
  La primera librería ampliamente adoptada y aceptada como librería de paso de mensajes fue PVM [24], desarrollada por el Laboratorio Nacional Oak Ridge de la Universidad de Tennessee, Estados Unidos, a inicios de 1990.

  PVM está compuesto de dos partes [39]: un proceso demonio (denominado `pvmd3`) y varias librerías basadas en rutinas. Cada vez que el usuario ejecuta una aplicación PVM, primero se crea una máquina virtual. Es posible que un usuario ejecute varios programas PVM simultáneamente sobre su propia máquina virtual. Además, se pueden configurar varias máquinas virtuales (una por cada nodo del cluster) para formar una sola máquina virtual de mejores características.

  La interfaz de la librería PVM contiene las primitivas necesarias para la cooperación entre las tareas de una aplicación. Aquí se definen todas las rutinas para paso de mensajes, sincronización de tareas, creación de procesos, y configuración de la máquina virtual. El sistema PVM actualmente soporta los lenguajes C, C++ y FORTRAN.

- **MPI** (*Message Passing Interface*)
  La estandarización de MPI [27] comenzó en abril de 1992, apareciendo la primera versión del estándar en mayo de 1994, diseñada para los lenguajes C y FORTRAN 77. En Marzo de 1995, se extendió la versión original, y culminó con la creación del estándar MPI-2 que incluye la implementación para C++ y FORTRAN 90.

  MPI no es un lenguaje de programación, es un conjunto de funciones y macros que conforman una librería estándar de C y C++, y subrutinas en FORTRAN. Ofrece un API, junto con especificaciones de sintaxis y semántica que explican como sus funcionalidades deben añadirse en cada implementación que se realice (tal como almacenamiento de mensajes o requerimientos para entrega de mensajes). Actualmente están disponibles diversas implementaciones de la librería MPI. Obviando las versiones proporcionadas por los fabricantes de computadores, específicas de sus propias plataformas, se puede mencionar un amplio conjunto de versiones que suelen proporcionarse como librerías de libre distribución: LAM/MPI [15], MPICH [18], OpenMPI [23], etc.

1.2.3. Métricas de rendimiento

El rendimiento obtenido en un sistema paralelo viene dado por una compleja relación en la que intervienen una gran cantidad de factores: el tamaño del problema, la arquitectura de soporte, la distribución de los procesos en los procesadores, etc. Como en cualquier área, en la computación paralela es necesario disponer de métricas que ayuden a evaluar objetivamente el rendimiento de un programa paralelo. Entre las métricas más conocidas se encuentran la aceleración y la eficiencia, cada una de las cuales se define a continuación.

1.2.3.1. Aceleración (*speedup*)

Formalmente, la aceleración es la relación entre el tiempo de ejecución sobre un procesador secuencial y el tiempo de ejecución en múltiples procesadores. Por tanto, la aceleración representa la ganancia de velocidad que se consigue con un programa paralelo respecto a su correspondiente programa secuencial, los cuales resuelven un mismo problema. La ecuación asociada a la aceleración se muestra a continuación:

$$S = \frac{T_s}{T_p} \tag{1.1}$$

donde S es el valor de la aceleración, T_s representa el tiempo de ejecución del programa secuencial y T_p es tiempo de ejecución del programa paralelo usando p procesadores.

La aceleración normalmente tiene un valor entre cero y p (sublineal), siendo lo ideal que sea igual a p, ya que, en teoría, la aceleración tiene una cota superior dado por el número de procesadores p. Una aceleración mayor que p es posible sólo si cada elemento de procesamiento consume menos de $\frac{T_s}{T_p}$ resolviendo el problema. A esto se le conoce como superlinealidad. Una razón de esto es que la versión paralela realiza menos trabajo que el correspondiente algoritmo secuencial. Puede deberse también a factores relacionados con los recursos que utiliza, a diferencias en el hardware (cachés, memorias, etc) o diferencias en el programa secuencial y paralelo (por ejemplo, inicializaciones).

1.2.3.2. Eficiencia (*efficiency*)

La eficiencia puede definirse como la porción de tiempo que los procesadores dedican a trabajo útil. Normaliza el valor de la aceleración dividiendo por el número de procesadores. La fórmula que define la eficiencia es la siguiente:

$$E = \frac{S_p}{p} \tag{1.2}$$

donde E es el valor de la eficiencia, S_p representa la aceleración asociada a la ejecución del programa paralelo usando p procesadores y p es el número de procesadores.

La eficiencia tiene un valor entre cero y uno, siendo lo ideal que sea igual a uno. Experimentalmente se puede obtener una eficiencia mayor que uno cuando hay superlinealidad.

1.3. MPI (*Message Passing Interface*)

MPI [27] es un estándar que define la sintaxis y la semántica de las funciones contenidas en una librería para la realización de aplicaciones paralelas bajo el modelo de paso de mensajes utilizando C, C++ o FORTRAN. MPI ha sido desarrollado por el MPI Forum [16], un comité formado por investigadores de universidades, laboratorios y empresas involucrados en la computación de altas prestaciones. El MPI Forum encaminó sus esfuerzos hacia los siguientes objetivos:

- Definir un entorno de programación único que garantice la portabilidad de las aplicaciones paralelas.
- Definir totalmente el interfaz de programación, sin especificar cómo debe ser la implementación del mismo.
- Ofrecer implementaciones de calidad, de dominio público, para favorecer la extensión del estándar.
- Convencer a los fabricantes de computadores paralelos para que ofrezcan versiones de MPI optimizadas para sus máquinas (lo que ya han hecho fabricantes como IBM y Silicon Graphics).

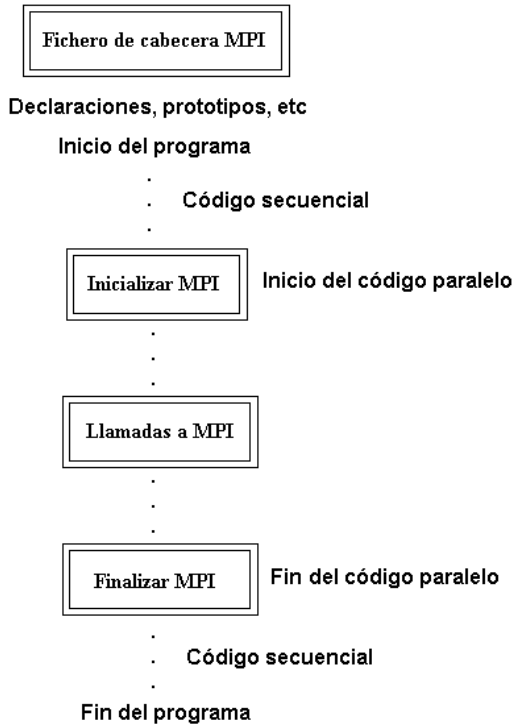

Figura 1.6: Estructura básica de un programa MPI

La primera versión estándar de MPI [49] se finalizó y publicó en 1994. En el año 1997 se presentó la especificación MPI-2 que extiende MPI sin introducir cambios en la versión 1. En esta nueva versión se introdujeron características diseñadas para incrementar la robustez de MPI, como operaciones en memoria remota, entrada/salida paralela, gestión dinámica de procesos, etc. Este proceso de especificación ha permitido que MPI sea la primera librería estándar portable con buenos rendimientos. Cabe destacar que:

- MPI es una librería, no es un lenguaje. Especifica los nombres, secuencias de llamadas y resultados de las rutinas que serán invocadas desde programas FORTRAN, de las funciones invocadas desde programas C, y de las clases y métodos que componen la librería MPIC++.
- MPI es una especificación, no una implementación particular. Todos los proveedores de computadores paralelos ofrecen una implementación de MPI para sus máquinas. Además, es posible encontrar muchas implementaciones de libre distribución. Los programas que los usuarios escriben en FORTRAN, C y C++ se compilan utilizando los compiladores habituales y se enlazan con la librería MPI.
- MPI ofrece estandarización, portabilidad, rendimiento y funcionalidad. Actualmente están disponibles diversas implementaciones de libre distribución como MPICH [18], LAM/MPI [15], FT-MPI [9], LA-MPI [14], OpenMPI [23], etc.

1.3.1. Programas MPI

Un programador que quiera emplear MPI para sus proyectos trabajará con una implementación concreta de MPI, que constará de, al menos, estos elementos:

- Una biblioteca de funciones para C y otra para FORTRAN, más el fichero de cabecera correspondiente (`mpi.h` y `mpif.h`, respectivamente) con las definiciones de esas funciones y de una colección de constantes y macros.
- Comandos para compilación, típicamente `mpicc` y `mpif77`, que son versiones de los comandos de compilación que incorporan automáticamente las bibliotecas MPI.
- Comandos para la ejecución de aplicaciones paralelas, típicamente `mpirun`.
- Herramientas para monitorización y depuración.

La estructura básica de un programa MPI se muestra en la Figura 1.6. Todo programa MPI [47] debe incluir el fichero de cabecera correspondiente (para poder hacer uso de las rutinas MPI), comenzar con una llamada a la rutina `MPI_INIT` (para inicializar MPI) y finalizar con una llamada a `MPI_FINALIZE` (finaliza MPI). No se puede invocar ninguna función MPI antes de la llamada a `MPI_INIT` ni después de la llamada a `MPI_FINALIZE`. Sólo el código dispuesto entre las llamadas a dichas rutinas se ejecuta en paralelo.

Para la compilación y ejecución de programas MPI suele ser habitual disponer de scripts, los cuales compilan el código y enlazan rutinas utilizadas con las de la librería MPI y que permite también lanzar ejecuciones del programa en la máquina paralela. Para

el caso de la compilación, una forma típica de hacerlo es (programa fuente en FORTRAN 77 y FORTRAN 90, respectivamente):

```
$> mpif77 -o ejemplo ejemplo.f -lmpi
$> mpif90 -o ejemplo ejemplo.f -lmpi
```

que genera el programa ejecutable `ejemplo`, y para la ejecución del programa:

```
$> mpirun -np 4 ejemplo
```

que lanza cuatro copias del ejecutable `ejemplo` siguiendo el modo de programación SPMD (*Single Program Multiple Data*), pudiendo ser asignadas cada una de ellas a un procesador diferente.

1.3.2. Comunicaciones MPI

Toda comunicación MPI [35] se produce dentro de un comunicador, que es, básicamente, un conjunto de procesos y un contexto de comunicación. El comunicador universal `MPI_COMM_WORLD` está siempre definido e incluye a todos los procesos que forman parte de la aplicación. Los comunicadores permiten que diferentes grupos de procesos actúen sin interferir, así como dividir la aplicación en fases no solapadas. MPI siempre garantiza la entrega ordenada de mensajes dentro de un mismo comunicador. Algunas de las funciones sobre comunicadores más utilizadas son:

- `MPI_Comm_size`: devuelve el número de procesos perteneciente al comunicador por defecto (`MPI_COMM_WORD`).
- `MPI_Comm_rank`: devuelve el identificador del proceso (*rank*) en un comunicador.
- `MPI_Comm_dump`: permite crear un nuevo comunicador, con el mismo grupo de procesos, pero diferente contexto.
- `MPI_Comm_split`: crea, a partir de un comunicador inicial, varios comunicadores diferentes, cada uno con un conjunto disjunto de procesos.
- `MPI_Comm_free`: destruye un comunicador.

Los mensajes gestionados por MPI son secuencias de elementos con un tipo asociado. MPI define una colección de tipos de datos primitivos correspondientes a los tipos de datos existentes en C y otra para FORTRAN. Los principales tipos de datos MPI se listan en la Tabla 1.2.

1.3.2.1. Comunicación punto a punto

MPI define dos modelos de comunicación [36]: bloqueante (*blocking*) y no bloqueante (*nonblocking*). El modelo de comunicación tiene que ver con el tiempo que un proceso pasa bloqueado tras llamar a una función de comunicación, sea ésta de envío o de recepción.

Tipo de dato en MPI	**Tipo de dato en FORTRAN**
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE_PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_UNSIGNED_CHARACTER	CHARACTER
MPI_BYTE	
MPI_PACKED	

Tabla 1.2: Tipos de datos básicos en MPI y FORTRAN

Una función bloqueante (`MPI_Send, MPI_Recv`) mantiene a un proceso bloqueado hasta que la operación solicitada finalice. Una no bloqueante (`MPI_Isend, MPI_Irecv`) simplemente "encarga" al sistema la realización de una operación, recuperando el control inmediatamente. El proceso tiene que preocuparse, más adelante, de averiguar si la operación ha finalizado.

Al margen de si la función invocada es bloqueante o no, el programador puede tener un cierto control sobre la forma en la que se realiza y completa un envío. MPI define, en relación a este aspecto, cuatro modos de envío:

- Básico *(basic)*: no especifica la forma en la que se completa la operación, depende de la implementación. Normalmente equivale a un envío con buffer para mensajes cortos y a un envío síncrono para mensajes largos.
- Con buffer *(buffered)*: tras el envío se guarda inmediatamente en un buffer una copia del mensaje. La operación se da por completa cuando se ha efectuado esta copia. En caso de que no haya espacio en el buffer, el envío fracasa.
- Síncrono *(synchronous)*: la operación se da por terminada sólo cuando el mensaje ha sido recibido en el destino.
- Listo *(ready)*: sólo se puede realizar el envío si antes, el otro extremo está preparado para una recepción inmediata. No hay copias adicionales del mensaje.

1.3.2.2. Comunicaciones colectivas

Muchas aplicaciones requieren de la comunicación entre más de dos procesos. Esto se puede hacer combinando operaciones punto a punto, pero para que le resulte más cómodo al programador, y para posibilitar implementaciones optimizadas, MPI incluye una serie de operaciones colectivas. El esquema asociado a cada una de las comunicaciones colectivas m´as importantes se muestran en la Figura 1.7. A continuación se explican detalladamente cada una de ellas:

- `MPI_Broadcast`: permite que un único proceso envíe un mensaje a todos los procesos del comunicador.
- `MPI_Gather`: realiza una recolección de los datos en un único proceso (proceso maestro) que recopila un vector de datos al que contribuyen todos los procesos del comunicador con la misma cantidad de datos. El proceso maestro almacena las contribuciones de forma consecutiva.
- `MPI_Allgather`: equivale a `MPI_Gather`, sólo que la recolección de los datos se distribuyen a todos los procesos del comunicador en lugar de sólo al proceso maestro.
- `MPI_Scatter`: realiza la operación simétrica a `MPI_Gather`.
- `MPI_Reduce`: permite hacer una operación de forma cooperativa entre todos los procesos de un comunicador, de tal forma que se obtiene un resultado final que se almacena en un único proceso.
- `MPI_Allreduce`: realiza un operación de forma cooperativa entre todos los procesos de un comunicador, obteniendo un resultado final que se almacena en todos los procesos del comunicador.
- `MPI_Scan`: es similar a `MPI_Allreduce`, salvo que cada proceso del comunicador recibe un resultado parcial de la reducción en lugar del final.

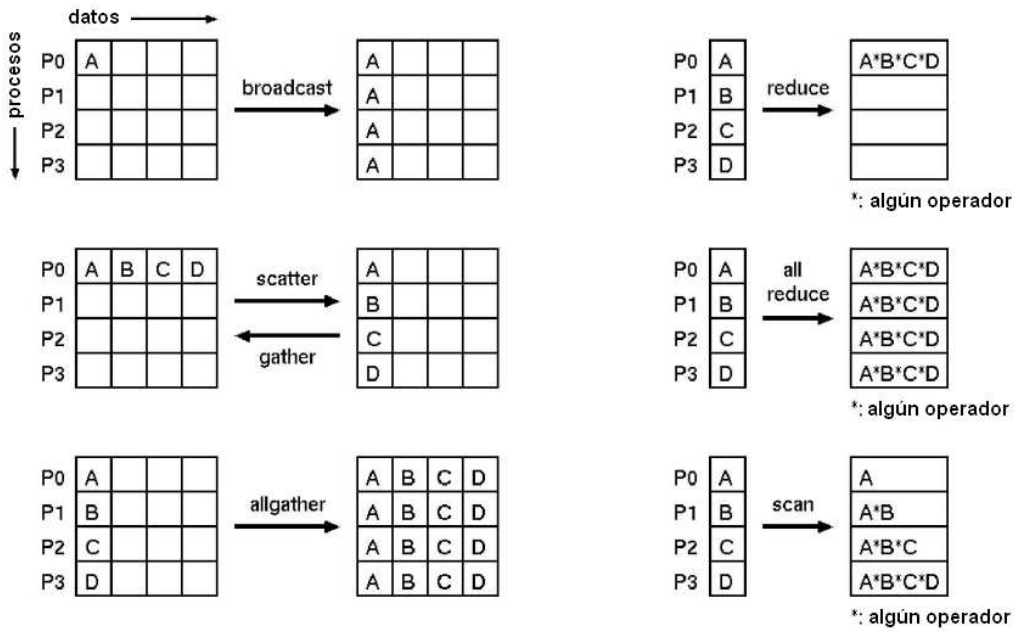

Figura 1.7: Esquema de las principales comunicaciones colectivas en MPI

Capítulo 2

HYSPLIT

El modelo de transporte y dispersión HYSPLIT (*HYbrid Single-Particle Lagrangian Integrated Trajectory*) [12] está diseñado para soportar un amplio rango de simulaciones relacionadas con el transporte de partículas atmosféricas y la dispersión de contaminantes, así como la deposición de esos materiales sobre la superficie terrestre. Los cálculos se realizan a partir de una partícula simple o *puff*, o siguiendo el movimiento dispersivo de un gran número de partículas. Ha ido desarrollándose conjuntamente por el *Bureau of Meteorology of Australia* [4] y el ARL (*Air Resources Laboratory*) [1] de la NOAA (*National Oceanic and Atmospheric Administration*) [21], en Estados Unidos, a lo largo de numerosas etapas durante la última década y media. La versión inicial fue desarrollada por Draxler y Taylor en 1982 [45]. La última versión disponible del modelo es la 4.9. El código fuente de HYSPLIT está escrito en FORTRAN 77/90 y aunque no es de dominio público, es posible obtener la versión para PC del programa de forma gratuita en su página web, o bien ejecutarlo online en el website READY [25].

En este capítulo, inicialmente se presentan las múltiples aplicaciones que tiene el modelo HYSPLIT y a continuación se realiza una breve explicación del funcionamiento general del programa. Posteriormente, se comenta su modo de uso, indicando las diversas plataformas sobre las que se puede ejecutar HYSPLIT, cómo instalarlo y cómo realizar una ejecución (describiendo previamente los ficheros de entrada y configuración que se precisan). Finalmente, se comentan algunas de las utilidades adicionales más importantes que incorpora HYSPLIT.

2.1. Aplicaciones

Los efectos intencionados o accidentales de los agentes químicos, biológicos y nucleares pueden afectar de forma significativa a la salud, la economía, la seguridad nacional e internacional y por supuesto, tener implicaciones ecológicas. HYSPLIT es una herramienta que ayuda a explicar cómo, dónde y cuándo sustancias químicas y materiales son transportados, dispersados y depositados. Conocerlo es esencial a la hora de responder de forma apropiada y prevenir catástrofes, permitiendo, por ejemplo, que los sistemas de emergencia puedan realizar una correcta evacuación de los afectados.

HYSPLIT cuenta con un número considerable de aplicaciones [26], [30], entre las que destacan, la predicción del transporte y dispersión, así como el seguimiento de:

- Contaminación procedente de varias fuentes de emisión (tanto móviles como estacionarias) [46]
- Emisiones de materiales radioactivos [53]
- Incendios forestales [42]
- Intrusiones de polvo sahariano [6]
- Emisiones de ceniza volcánica [38]
- Tormentas de arena y polvo [55]
- Partículas de polen [56], [62]

Actualmente, el modelo es usado ampliamente por múltiples centros de meteorología y centros de investigación, como el Servicio Nacional de Meteorología de la NOAA [1] (Estados Unidos), AEMET [5] (España), así como en departamentos de investigación de diversas universidades (Univ. de Huelva, Univ. de Houston, Univ. de Hawaii, Univ. de York, etc).

A nivel regional, HYSPLIT es usado para responder a las peticiones sobre previsión de dispersión derivadas de emergencias de carácter local. En Estados Unidos, el modelo se aplica a las necesidades en la industria de la aviación y a los reguladores calidad del aire. Internacionalmente, NOAA colabora a través de su participación con la Organización Mundial de Meteorología y la Agencia Internacional de Energía Atómica proporcionando su modelo de predicción de dispersión en caso de producirse un incidente nuclear a gran escala [32].

2.2. Funcionamiento general

La última versión de HYSPLIT [45] realiza un amplio rango de simulaciones relacionadas con el transporte a gran escala, dispersión y deposición de contaminantes. Entre los numerosos modelos que implementa, destacan: el modelo de concentraciones `hymodelc` y el modelo de trayectorias `hymodelt` que permiten monitorizar y pronosticar el movimiento y la concentración, respectivamente, que presentan las partículas de aire o contaminación en sucesos tales como incendios, contaminación atmosférica, tornados, huracanes y diversos acontecimientos naturales. Ambos modelos cuentan además, con su correspondiente versión paralela (`hymodelm` y `hymodelp`).

Las entradas y salidas de HYSPLIT se muestran de forma esquemática en la Figura 2.1. La ejecución secuencial de cualquiera de los modelos que implementa HYSPLIT recibe como entrada uno o varios (hasta un máximo de doce) ficheros de datos meteorológicos en formato ARL. Además, requiere la especificación de un fichero de control (`CONTROL`) donde se definen varios parámetros del modelo, así como el nombre y localización de los

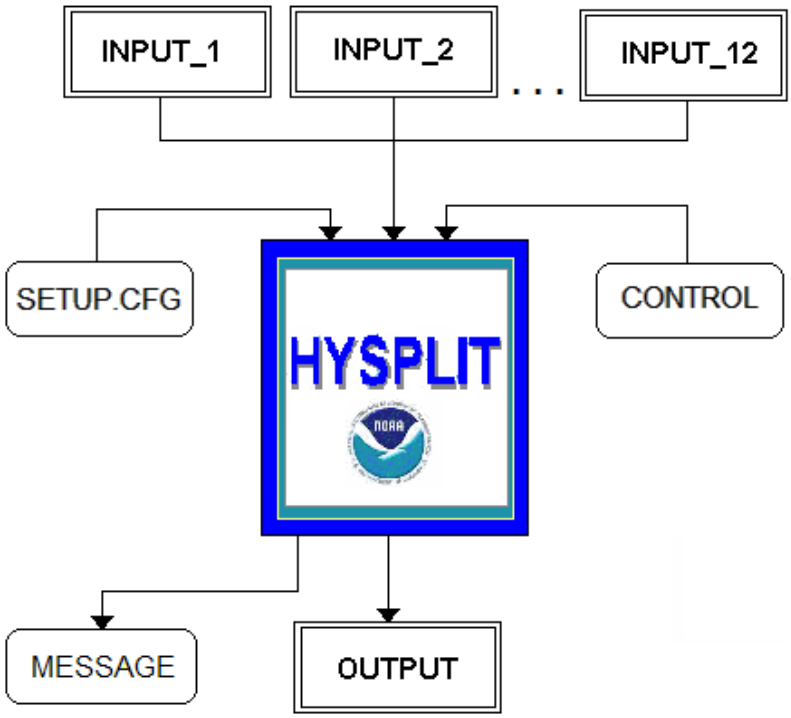

Figura 2.1: Esquema de las entradas y salidas de HYSPLIT

ficheros con datos meteorológicos y el nombre del fichero de salida. También es necesario incluir un fichero de configuración (`SETUP.CFG`) donde se definen algunas características de simulación más avanzadas (como número máximo de partículas emitidas en la simulación, tamaño inicial de la malla meteorológica, etc). Una vez aportados los datos necesarios, se puede realizar una simulación del modelo elegido (trayectorias o concentraciones) que generará como salida un fichero binario (`OUTPUT`) con los resultados obtenidos. Es posible realizar representaciones gráficas de la salida, así como convertir el fichero a otros formatos. La simulación también genera un fichero de texto `MESSAGE` que contiene información de diagnóstico acerca de la simulación realizada.

En cuanto al método de cálculo que usa HYSPLIT [13] se basa en un modelo híbrido entre el euleriano y el lagrangiano. El cálculo de la advección y difusión lo realiza en un marco lagrangiano, mientras que para la concentración utiliza un marco euleriano. HYSPLIT puede simular una distribución de contaminantes a partir de una partícula simple o *puff*, o siguiendo el movimiento dispersivo de un gran número de partículas. El método euleriano describe la concentración estadística en términos de las velocidades eulerianas del fluido, es decir, velocidades medidas en puntos fijos. Como se aprecia en la parte izquierda de la Figura 2.2, bajo la aproximación euleriana, la concentración en cada celda de la malla se calcula integrando el flujo de contaminantes correspondiente a cada interfaz de cada celda proveniente de la dispersión y advección de dicho contaminante.

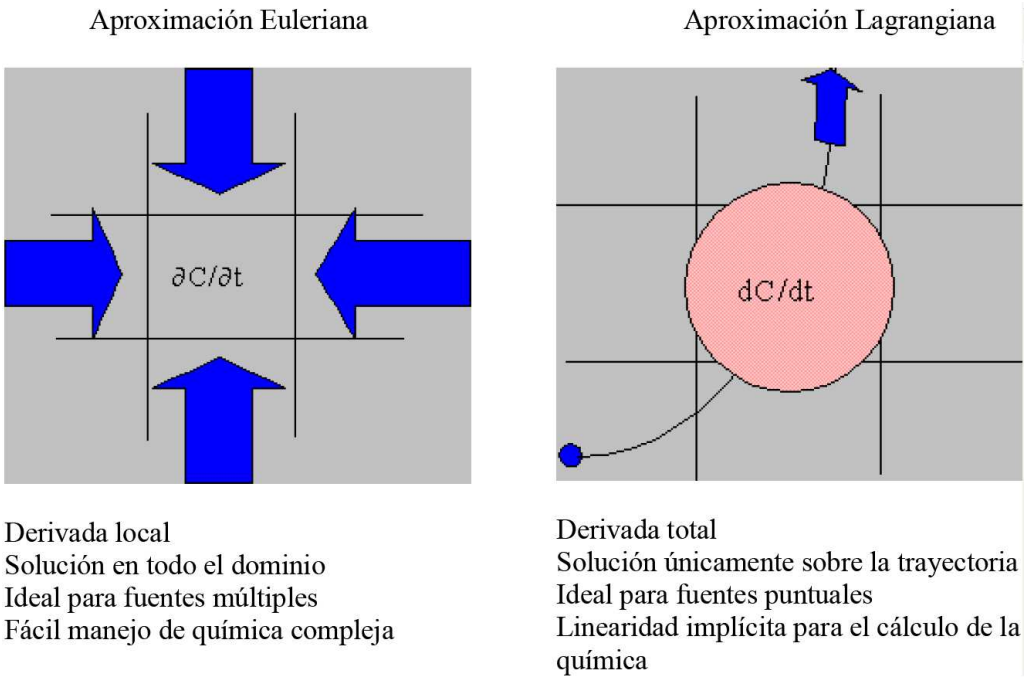

Figura 2.2: Métodos de cálculo de advección y dispersión

La aproximación lagrangiana describe la concentración de un contaminante en términos de las propiedades estadísticas de los desplazamiento de las partículas del fluido. Cuando se utiliza la metodología lagrangiana, las concentraciones se computan sumando la contribución de cada *puff* de contaminantes que se transporta a través de la malla siguiendo su trayectoria, tal como se observa en la parte derecha de la Figura 2.2.

Una vez conocido el funcionamiento general de HYSPLIT, se explica cómo hacer uso del programa en función de la plataforma elegida para su ejecución. En la siguiente sección se presentan cada una de las plataformas sobre las que se puede ejecutar HYSPLIT, indicando cómo realizar su instalación, así como la preparación del entorno de trabajo para la realización de simulaciones.

2.3. Modo de uso

En esta sección se presenta una breve guía sobre el uso de HYSPLIT. Inicialmente, se comentan brevemente los pasos básicos para la correcta instalación de HYSPLIT. Posteriormente, se detallan las características y formato de cada uno los ficheros de entrada y configuración que son necesarios para la realización de una simulación y luego se explica cómo se realiza la ejecución de un modelo en función de la plataforma elegida. Seguidamente, se indican cuáles son los ficheros de salida de una simulación y finalmente, se exponen algunas de las utilidades adicionales más importantes de las que dispone HYSPLIT.

2.3.1. Instalación

Es posible obtener HYSPLIT de forma gratuita para la realización de simulaciones, a pesar de que el código fuente no es de dominio público. HYSPLIT está disponible sobre las siguientes plataformas:

- ***PC***: se puede descargar HYSPLIT desde su página oficial [12] y ejecutarlo sobre un PC bajo Windows o Mac.
- ***Online***: HYSPLIT puede ejecutarse online, a través del website READY [25] sin necesidad de instalar ningún paquete.
- ***Cluster***: es necesario solicitar el código fuente para compilar e instalar HYSPLIT adecuadamente. Es la única vía que permite ejecutar los modelos paralelos.

La instalación de la versión para PC de HYSPLIT requiere tener previamente instalado el siguiente software:

- Tcl/Tk (versión 8.5.4) para poder utilizar la interfaz gráfica (GUI).
- Ghostscript (versión 8.63) y Ghostview (versión 4.9) para visualizar e imprimir una salida postscript.
- ImageMagick (versión 6.4) para convertir archivos postscript a otros formatos gráficos.

Cuando se realiza la instalación, se crea en la máquina local una carpeta denominada *hysplit4* que contiene los directorios siguientes:

- `bdyfiles`: archivos de altura de terreno y usos de suelo.
- `browser`: personaliza los scripts tcl para la interfaz del GUI de ayuda.
- `concmdl`: scripts y archivos para automatizar y personalizar las simulaciones.
- `csource`: archivos dll requeridos para visualizar y editar partículas.
- `data2arl`: programas para convertir datos meteorológicos al formato HYSPLIT.
- `document`: versión mas reciente de documentos técnicos y guía de usuario.
- `exec`: todos los ejecutables se encuentran en este directorio.
- `grads`: código fuente para convertir el archivo de salida de HYSPLIT y los datos meteorológicos a formato GRADS.
- `graphics`: archivos de mapas de fondo y mapas personalizados.
- `guicode`: scripts tcl requeridos para ejecutar el GUI.
- `html`: archivos de ayuda.

- `metdata`: archivo de datos meteorológicos de muestra y programa para lectura de datos.
- `source`: subrutinas para compilar programas de conversión de datos meteorológicos.
- `trajmdl`: scripts y archivos para personalizar simulaciones de trayectoria.
- `utilities`: herramientas de conversión y chequeo de datos.
- `vis5d`: código fuente del programa para convertir archivos de salidas de concentración a vis5d.
- `working`: directorio de trabajo, donde se encuentran los ficheros de entrada, configuración y control.

Se puede acceder al programa a través del acceso directo que se crea en el escritorio tras la instalación. El aspecto de la aplicación al abrirla puede verse en la Figura 2.3.

Para la instalación de HYSPLIT en un cluster, es necesario disponer del código fuente. Se deben configurar los ficheros *makefiles* para compilarlo de forma adecuada sobre el cluster y realizar la instalación en función de la arquitectura disponible. La instalación requiere de más software adicional, como por ejemplo, la librería OpenMPI [23]. No obstante, esta es la única forma en la que se puede ejecutar las versiones paralelas de los modelos de HYSPLIT, como `hymodelm` y `hymodelp`. La jerarquía de directorios creada tras la instalación es muy similar a la indicada anteriormente, incluyendo además varias carpetas más con código fuente. Finalizada la instalación, todas las órdenes se indican a través de línea de comandos.

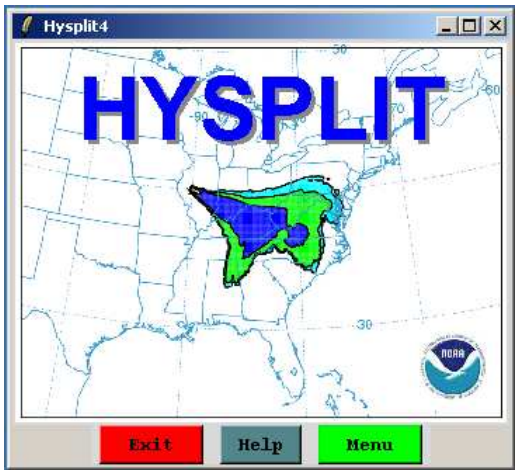

Figura 2.3: Versión de HYSPLIT para PC

2.3.2. Ficheros de entrada y configuración

La ejecución de una simulación en HYSPLIT requiere la presencia de los ficheros siguientes [13]:

- ***Fichero de control***: en él se definen varios parámetros del modelo y otros ficheros de entrada y salida.
- ***Fichero de configuración***: en este fichero se pueden definir algunas características más avanzadas de la simulación.
- ***Ficheros con datos meteorológicos***: los ficheros de entrada contienen datos meteorológicos en un formato compatible con HYSPLIT.

A continuación se explica con detalle los requisitos que deben cumplir cada uno de estos ficheros de entrada para poder realizar una simulación exitosamente.

2.3.2.1. Ficheros de datos meteorológicos

HYSPLIT recibe como entrada [44] uno o varios ficheros de datos meteorológicos. Solamente acepta como entrada ficheros de datos meteorológicos en formato ARL [2]. Si no se dispone de datos en dicho formato, proporciona una utilidad para convertir el fichero al formato adecuado. Una de las características fundamentales de una simulación en HYSPLIT es la de poder seleccionar los mejores archivos de datos meteorológicos disponibles. En la última versión se permiten definir hasta 12 archivos simultáneamente.

ARL proporciona un servidor de ficheros de datos meteorológicos que el usuario puede utilizar para realizar simulaciones. Estos datos cubren tres dominios geográficos: América del Norte, el hemisferio norte y el hemisferio sur. En la versión para PC de HYSPLIT, se incorpora una opción en el menú que permite descargar estos ficheros desde el servidor. Está disponible en la opción $Meteorology \rightarrow ARL\ Data\ FTP$. Asimismo, incorpora otras utilidades adicionales que permiten chequear los datos de entrada y visualizar el dominio meteorológico.

Cada uno de los ficheros de entrada que se incluya en la simulación debe especificarse en el fichero de control, el cual se presenta con detalle en la siguiente subsección.

2.3.2.2. Fichero de control

El fichero de control (denominado `CONTROL`) permite la definición de varios parámetros del modelo: número de fuentes de emisión, duración de la simulación, resolución de la malla, esquinas inferior izquierda y superior derecha del dominio de emisiones deseado, etc. Asimismo, permite definir cuáles son los ficheros con datos meteorológicos de entrada y el nombre y ubicación del fichero de salida de la simulación.

El formato general del fichero de control se indica a continuación, apareciendo a la derecha de cada uno de los campos un comentario que describe su función:

```
00 00 00 00          Año, mes, día, hora de inicio
1                    Número de localidades de inicio
40.0 -90.0 10.0      Latitud, longitud, altura de inicio
48                   Duración de la simulación en horas
0                    Opción de movimiento vertical
10000.0              Tope del dominio del modelo en metros
1                    Número de archivos meteorológicos de entrada
../working/          Directorio del 1er archivo meteorológico
oct1618.BIN          Nombre del 1er archivo meteorológico
1                    Número de contaminantes emitidos por cada fuente
TEST                 Nombre del contaminante (4 caracteres)
1.0                  Unidades de velocidad de emisión por hora
1.0                  Duración de la emisión en horas
00 00 00 00 00       Año, mes, día, hora, minuto de inicio de emisión
1                    Número de mallas de salida de concentración
0.0 0.0              Latitud, longitud centrales de la malla
0.5 0.5              Resolución de la malla de concentración en grados
30.0 30.0            Latitud y longitud abarcadas por la malla
./                   Directorio de salida
cdump                Nombre del archivo de salida
1                    Número de niveles de salida de concentración
100                  Altura de los niveles de salida de concentración
00 00 00 00 00       Tiempo inicial de salida de datos (a,m,d,h,m)
00 00 00 00 00       Tiempo final de salida de datos (a,m,d,h,m)
00 12 00             Flag de promediación, intervalo de promediación horas, minutos
1                    Número de contaminantes
0.0 0.0 0.0          Diámetro de partícula, densidad, forma
0.0 0.0 0.0 0.0 0.0  Velocidad,peso molecular,cocientes de actividad,difusividad,Henry
0.0 0.0 0.0          Coef. de Henry para nubes, coef. de Scavenging, cociente
0.0                  Vida media para decaimiento radiactivo
0.0                  Factor de resuspensión
```

El fichero de control puede editarse manualmente desde el directorio `working`. En la versión para PC, también es posible hacerlo a través de la opción *Concentration* → *Setup Run* o *Trajectory* → *Setup Run* para editar el fichero de control asociado al modelo de concentraciones o trayectorias, respectivamente. Los pasos a seguir se muestran en la Figura 2.5.

2.3.2.3. Fichero de configuración

El fichero de configuración (denominado `SETUP.CFG`) permite definir algunas características de simulaciones más avanzadas como: establecer el número máximo de partículas emitidas en la simulación, el tamaño inicial de la malla meteorológica, la vida máxima de las partículas, etc. Este fichero se encuentra disponible en el directorio `working`, desde donde puede editarse manualmente. En la versión para PC, también es posible editarlo a través de la opción *Advanced* → $Configuration\ Setup$ → $Concentration \mid Trajectory$ del menú (si se desea editar el fichero de configuración asociado al modelo de concentraciones o trayectorias, respectivamente). En la Figura 2.4 se ilustra el proceso.

El formato general del fichero de configuración se indica a continuación, apareciendo a la derecha de cada uno de los campos un comentario que describe su función:

```
&SETUP
tratio = 0.75,   Lapso de tiempo del cociente de estabilidad
initd = 4,       Distribución inicial de partículas
khmax = 9999,    Vida máxima de las partículas
numpar = 500,    Partículas emitidas durante 1 ciclo de emisión
qcycle = 0.0,    Intervalo del ciclo de emisión en horas
isot = 0,        Flag de turbulencia isotrópica
ndump = 0,       Inicio de archivo de salida de partículas en horas
ncycl = 0,       Intervalo del archivo de salida de partículas en horas
mgmin = 10,      Tamaño inicial de la malla meteorológica
kmsl = 0,        Flag para tratar las alturas como si estuvieran arriba del nivel del mar
maxpar = 10000,  Número máximo de partículas permitidas
cpack = 1,       Flag de empaquetamiento de concentracion
dxf = 1.0,       Opción de conjunto (Ensemble) offset de malla en la dirección x
dyf = 1.0,       Opción de conjunto (Ensemble) offset de malla en la dirección y
dzf = 0.01,      Opción de conjunto (Ensemble) offset de malla en la dirección z
ichem = 0,       Flag de módulos de química
/
```

En la realización de simulaciones online con HYSPLIT, la configuración se hace introduciendo los datos en un formulario que establecerá los parámetros necesarios de la misma forma en que se hacía en los ficheros de control y configuración. En la Figura 2.6 se muestra la pantalla de configuración de la simulación online.

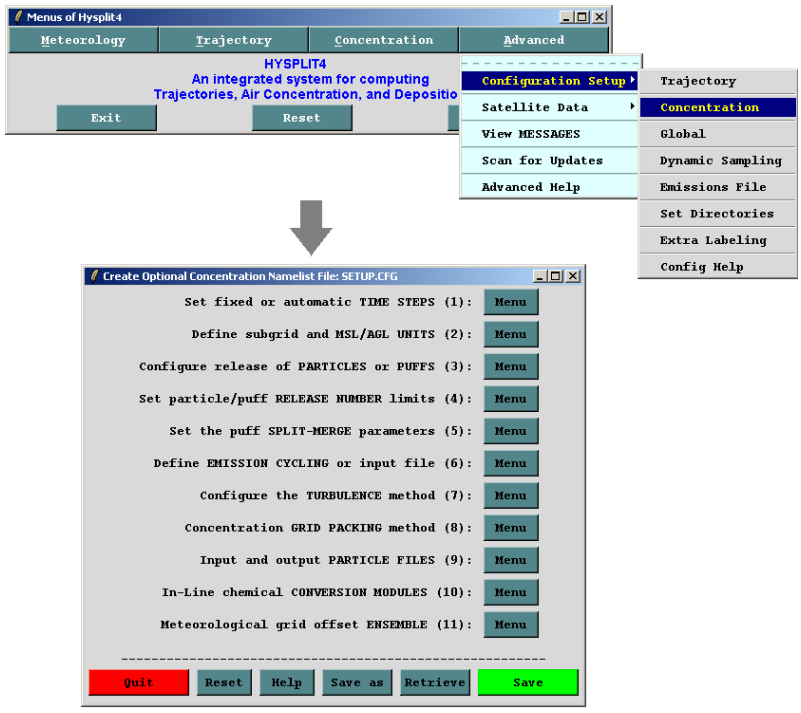

Figura 2.4: Edición del fichero de configuración (versión PC)

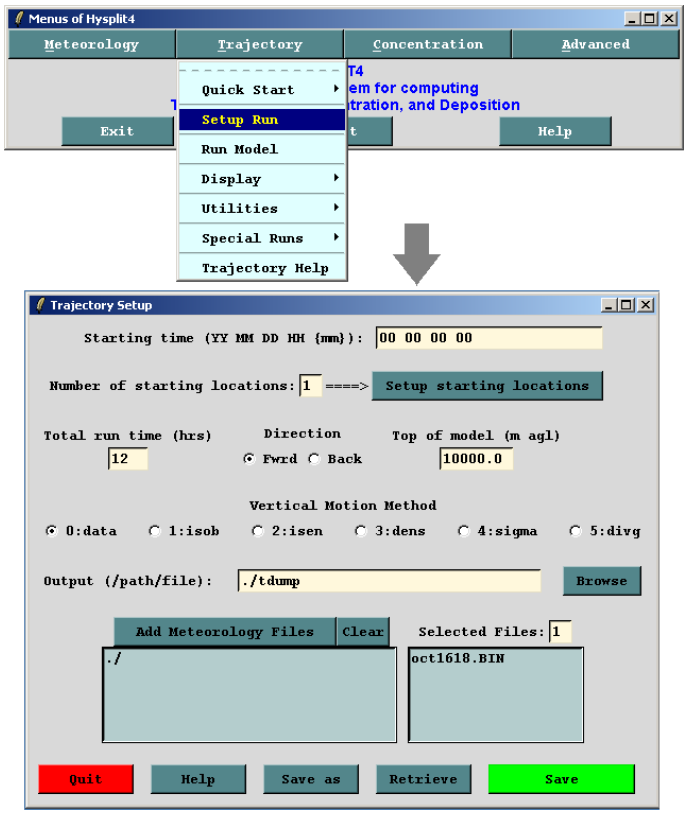

Figura 2.5: Edición del fichero de control (versión PC)

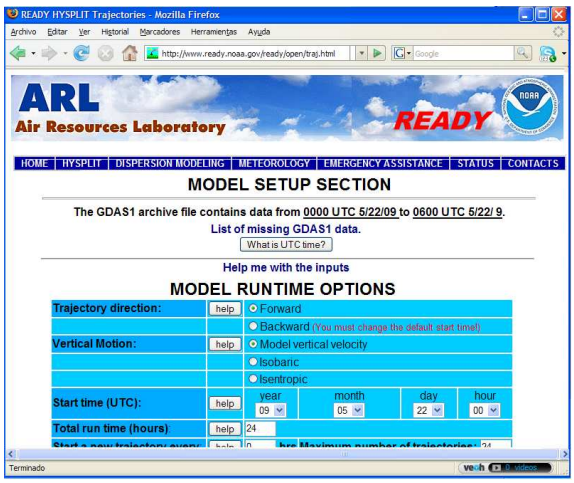

Figura 2.6: Edición del fichero de control y configuración (versión online)

2.3.3. Ejecución

Una vez aportados los datos meteorológicos necesarios y editado de forma adecuada el fichero de control y el fichero de configuración, se está preparado para ejecutar cualquiera de los modelos que implementa HYSPLIT. La realización de una simulación en HYSPLIT dependerá de la plataforma sobre la que se realice la ejecución. Para la versión online y de PC de HYSPLIT sólo es posible ejecutar los modelos en secuencial. Para realizar simulaciones en paralelo es necesario disponer de un cluster.

A continuación se indican los pasos a seguir para realizar una simulación sobre las diferentes plataformas en las que puede ejecutarse HYSPLIT.

- **PC (Mac o Windows)**
  La versión para PC de HYSPLIT provee una interfaz gráfica que permite realizar simulaciones de forma sencilla. Si se desea ejecutar el modelo de concentraciones, basta con elegir la opción *Concentration* → *Run Model* disponible en la barra de menú del programa. Para la ejecución del modelo de trayectorias, se elegirá la opción *Trajectory* → *Run Model*, tal como se muestra en la Figura 2.7.

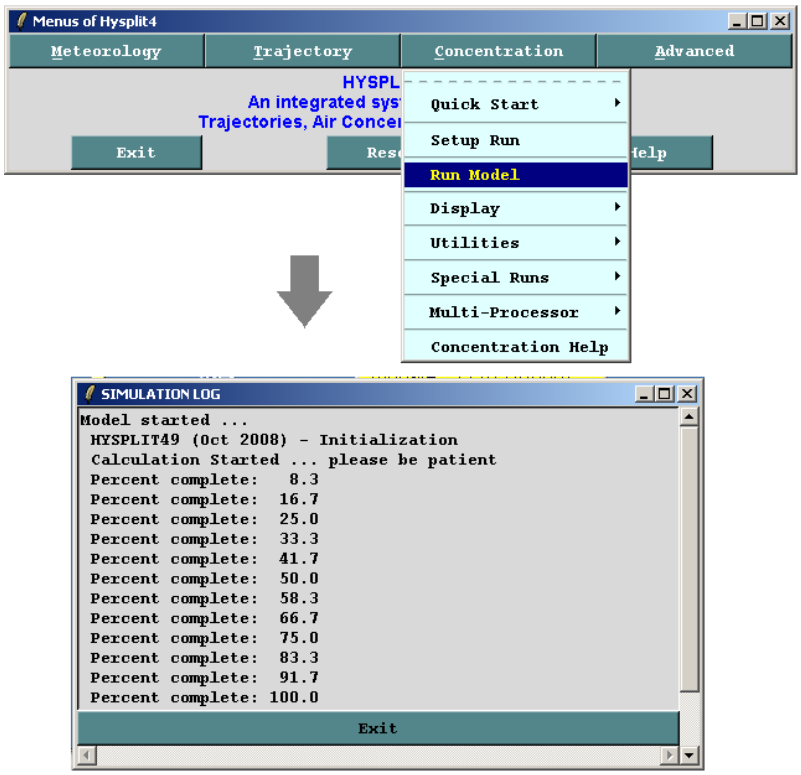

Figura 2.7: Ejecución de una simulación (versión PC)

- **Online (READY website)**
  A través del website READY, es posible ejecutar varios modelos. Si se quiere comenzar la simulación, primero se debe elegir el modelo a ejecutar. Para ilustrar este proceso, se eligió el modelo de trayectorias. Una vez elegido el modelo, tan sólo hay que seguir los pasos para seleccionar los ficheros de meteorología de entrada (disponibles online) y las opciones de configuración y control. El primer paso (elección de los ficheros de entrada) se muestra en la Figura 2.8. Una vez finalizada la simulación, se muestran gráficamente los resultados obtenidos tras la ejecución del modelo elegido.

- **Cluster**
  Si se dispone de un cluster, es posible realizar tanto ejecuciones en secuencial como en paralelo. Para realizar una simulación, por ejemplo, del modelo de concentraciones, se debe dar la siguiente orden en la línea de comandos (suponiendo que nos encontramos en el directorio `working`):

  ```
  $> ../exec/hymodelc
  ```

  Si se desea ejecutar la versión en paralelo del modelo de concentraciones para 4 procesadores, se debe escribir:

  ```
  $> mpirun -machinefile file -np 4 ../exec/hymodelm
  ```

  donde la opción `-machinefile` indica el fichero que contiene la lista de nodos sobre los que se ejecutará (`file`) y la opción `-np` permite especificar el número de procesos que ejecutarán el programa.

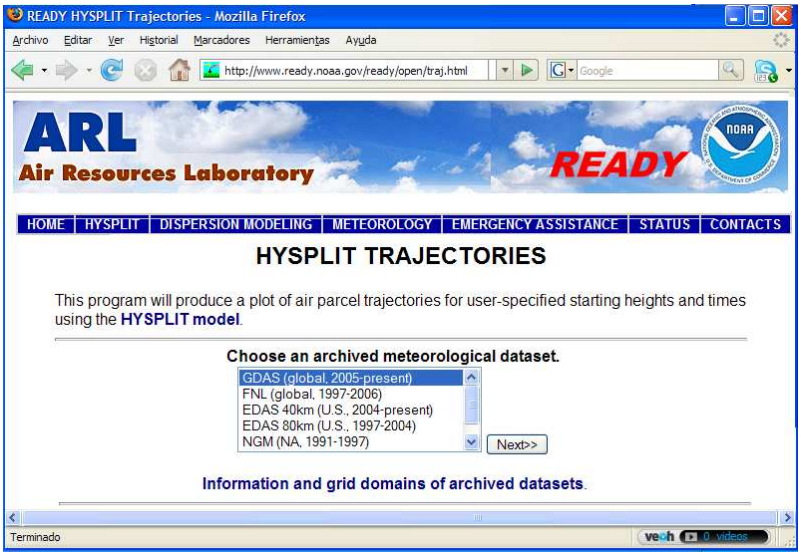

Figura 2.8: Ejecución del modelo de trayectorias (versión online)

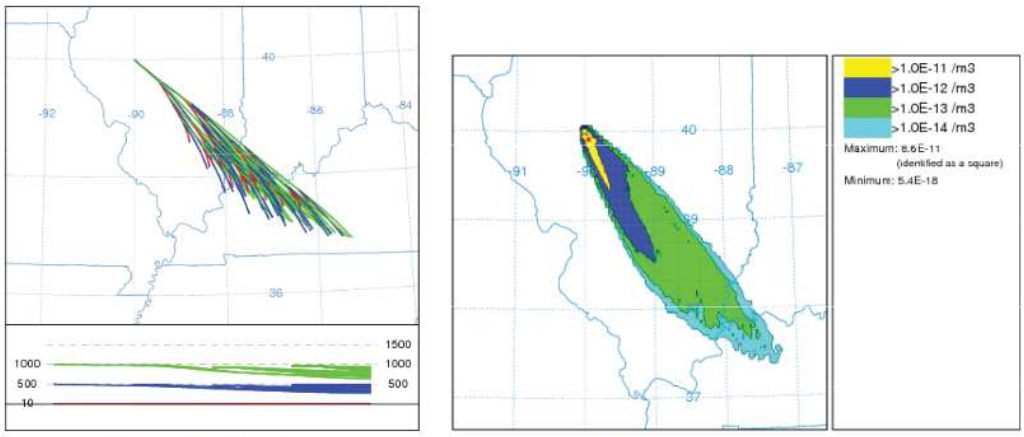

Figura 2.9: Salidas de HYSPLIT en modo gráfico

2.3.4. Ficheros de salida

Una vez aportados los datos necesarios y finalizada la simulación, HYSPLIT devuelve como salida un fichero binario con los resultados obtenidos. Existen varias opciones para transformar estos datos en otros formatos de interés para el usuario como postscript, ASCII, GRADS, ArcView, Vis5D, etc.

En la Figura 2.9 se pueden ver dos ejemplos de ficheros de salida convertidos en formato postscript. A la izquierda se muestra la salida correspondiente a una simulación del modelo de trayectorias y a la derecha del modelo de concentraciones.

Por otro lado, toda simulación realizada en HYSPLIT genera un fichero de texto `MESSAGE.txt` que contiene información de diagnóstico acerca de la simulación realizada. Este fichero está disponible en el directorio `working`. También puede consultarse a través de la opción de menú *Advanced* → *View Messages* de la versión para PC de HYSPLIT.

2.3.5. Utilidades

HYSPLIT incorpora una serie de utilidades que permiten editar y convertir a otros formatos tantos los datos de entrada como los de salida. Algunas de la opciones más útiles se presentan a lo largo de esta subsección, mostrando cómo se utilizan en la versión para PC.

2.3.5.1. Validación de los datos de entrada

Esta opción es especialmente útil, ya que permite chequear un fichero de datos meteorológicos para saber si se encuentra en un formato compatible con HYSPLIT. Esto se hace a través de la herramienta `check_file`. Se encarga de desempaquetar y leer cada uno de elementos de cada registro contenidos en un fichero meteorológico empaquetado en formato ARL y escribe sus valores en un fichero de salida `chk_file.txt` (en el directorio `working`).

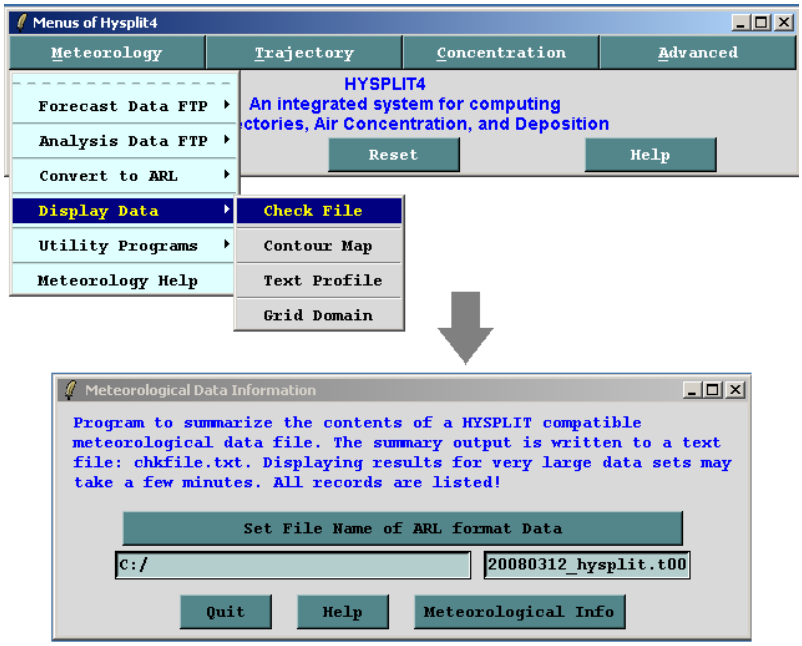

Figura 2.10: Validación del formato de un fichero de entrada

El contenido de este fichero se muestra por pantalla al usuario. Se puede acceder a esta utilidad eligiendo la opción del menú *Meteorology* → *Display Data* → *Check File*, tal como se muestra en la Figura 2.10. Se selecciona el archivo de meteorología que se desea chequear y se pulsa el botón *Meteorological Info*. Se mostrará por pantalla toda la información asociada. En caso de que el proceso falle, significará que el fichero de datos elegidos no está en formato ARL y por tanto, no puede usarse como entrada para HYSPLIT.

2.3.5.2. Visualización del dominio meteorológico

El dominio meteorológico puede ser visualizado a través de la opción *Display Data* → *Grid Domain* disponible en la opción de menú *Meteorology*. Los pasos a seguir se indican en la Figura 2.11. Primero se selecciona el archivo de meteorología cuyo dominio se desea visualizar, se establecen el número de nodos y el valor de los intervalos en grados de latitud y longitud. En el ejemplo de la Figura 2.11 se ha establecido que se muestre 1 de cada 10 nodos de la malla y las líneas de latitud y longitud cada 5 grados. Una vez configurado, se debe pulsar sobre el botón *Create Map* para generar el mapa de dominio.

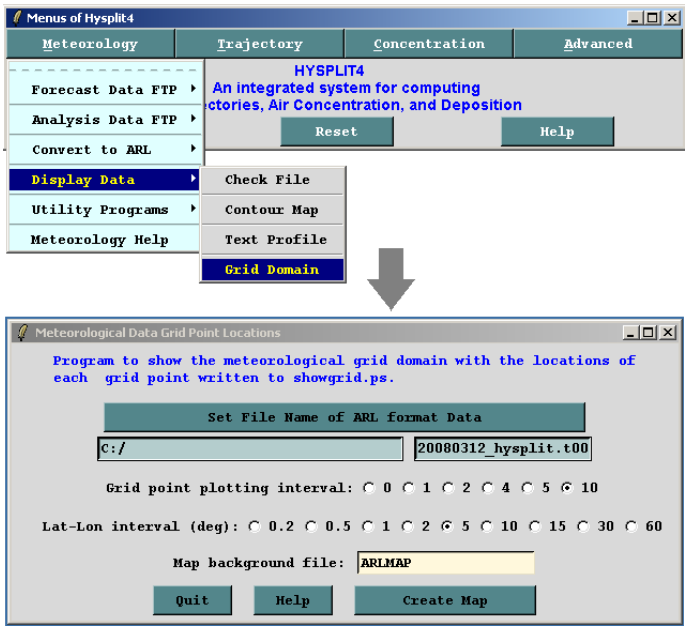

Figura 2.11: Pasos a seguir para visualizar el dominio meteorológico

2.3.5.3. Conversión a formato ARL

Como se vio anteriormente, los datos de entrada de HYSPLIT deben estar en formato ARL. Teniendo en cuenta los formatos de datos meteorológicos más comunes, HYSPLIT incorpora una utilidad para convertir los datos a formato ARL. Esta utilidad está disponible a través de la opción *Meteorology* → *Convert to ARL*, como se muestra en la Figura 2.12 donde además se aprecian los distintos formatos de fichero a partir de los cuales se puede realizar la conversión.

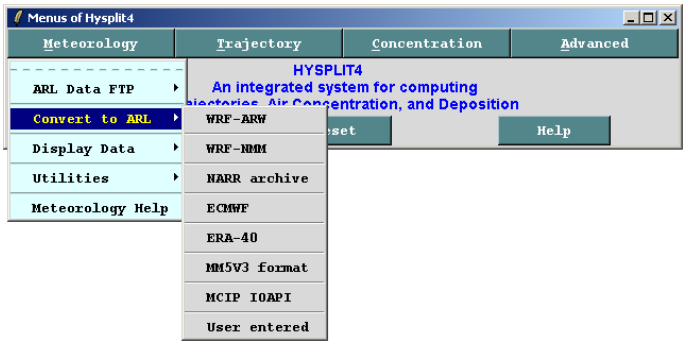

Figura 2.12: Conversión de un fichero de entrada al formato ARL

2.3.5.4. Conversión de los datos de salida

HYSPLIT permite la conversión de los datos de salida a otros formatos de interés para el usuario como postscript, ASCII, GRADS, ArcView, Vis5D, etc. Si por ejemplo, se desea convertir a postscript la salida obtenida con la ejecución del modelo de concentraciones, se deben seguir los pasos indicados en la Figura 2.13. Inicialmente, se selecciona la opción *Concentration* → *Display* → *Concentration* → *Contours* del menú. Luego, se cambian las opciones que se desean o se dejan las que están por defecto y finalmente, se pulsa sobre el botón *Execute Display*. Se abrirá una nueva ventana con la representación gráfica de la salida en formato postscript (muy similar al mostrado en la parte derecha de la Figura 2.9). Si se desea obtener la salida en postscript para el modelo de trayectorias, se debe seleccionar la opción *Trajectory* → *Display*.

También es posible cambiar el formato del postscript generado (mediante alguna de las opciones anteriores) a otros formatos como por ejemplo, GIF. Para ello, basta con seleccionar la opción *Meteorology* → *Utilities* → *Convert Postscript* y elegir el formato deseado.

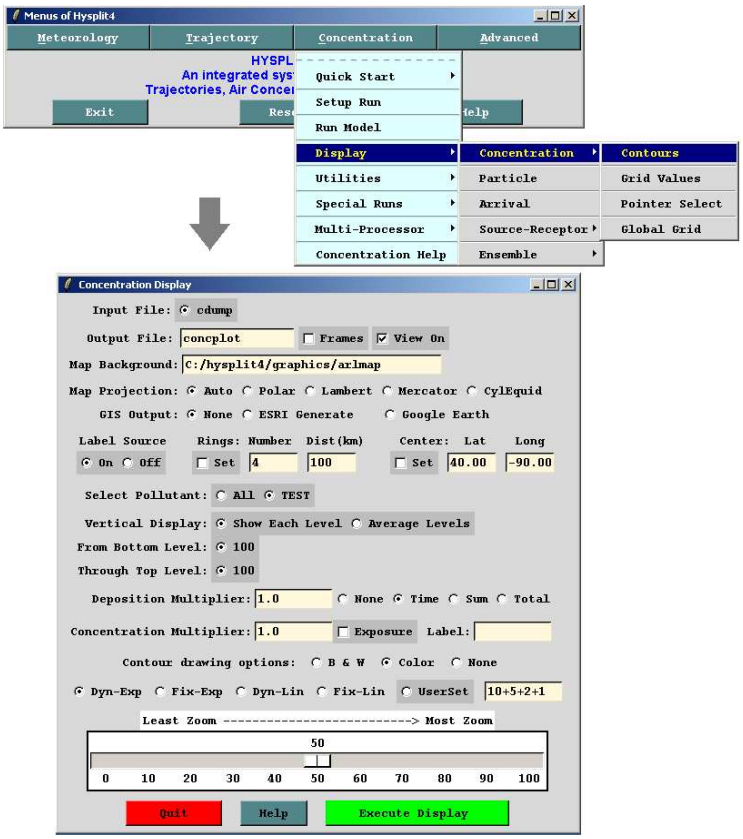

Figura 2.13: Conversión de los datos de salida a postscript (versión PC)

Capítulo 3

Análisis del rendimiento de HYSPLIT para cluster

En los problemas meteorológicos se debe computar un gran número de datos en un tiempo relativamente corto, puesto que el objetivo principal es la predicción y no tendría sentido obtenerla en un momento muy cercano o incluso posterior al que se desea predecir. HYSPLIT [12] cuenta con versiones paralelas (en MPI) de algunos de los modelos que implementa. Concretamente, la versión paralela del modelo de concentraciones es una de las más usadas en el ARL [1] de la NOAA. Generalmente es ejecutada sobre un cluster formado por más de una treintena de procesadores que utilizan NFS (*Network File System*) como sistema para compartir archivos. Sin embargo, esta implementación presenta un comportamiento poco adecuado que se pone de manifiesto en la ralentización del programa a medida que el número de procesadores con que se ejecuta aumenta, por lo que no escala convenientemente. Se sospecha que el origen de este comportamiento está en la inadecuada gestión de la obtención de los datos de entrada. Así pues, el objetivo de este trabajo es estudiar el comportamiento de la versión paralela del modelo de concentraciones, determinar las causas de la degradación del rendimiento y proponer soluciones con su correspondiente implementación.

En este capítulo se describe el problema que presenta la versión paralela de HYSPLIT para cluster. Para ello, se realiza un análisis del código fuente, así como diversos experimentos sintéticos que reproducen el problema y permiten proponer una solución abstracta al mismo.

3.1. Análisis del código fuente

En esta sección se presenta la estructura general del código fuente de HYSPLIT, así como los pasos seguidos durante el análisis al que fue sometido. El código fuente utilizado permite realizar ejecuciones en paralelo sobre un cluster y corresponde a la versión 4.8. Como única documentación se proporcionó la Guía de Usuario de HYSPLIT [44]. El objetivo inicial es obtener el diagrama de flujo de la versión paralela del modelo de concentraciones (`hymodelm`). Asimismo, se desea conocer dónde y cómo se realiza la obtención de los datos

de entrada ante la sospecha de que los problemas de escalabilidad provienen de ahí, ya que se ejecuta sobre un cluster con NFS y es sabido que los sistemas paralelos con NFS producen un cuello de botella que impide alcanzar una escalabilidad lineal [37], [54], [63].

3.1.1. Estructura del código fuente

La versión 4.8 de HYSPLIT para cluster se distribuye en un total de 19 carpetas incluyendo código fuente, utilidades, documentación, ejemplos, tests, etc. Existen más de 200 rutinas distribuidas en 196 ficheros distintos (con extensión *.f), así como 17 módulos donde se definen variables, constantes y los prototipos de algunas rutinas. Asimismo, HYSPLIT cuenta con dos ficheros *main*: uno para el modelo de concentraciones (en el fichero `hymodelc.F`) y otro para el modelo de trayectorias (en el fichero `hymodelt.F`). A partir de dichos ficheros, se generan distintos ejecutables. El código fuente está contenido principalmente en las siguientes carpetas:

- `./source`: incluye los ficheros *main* del modelo de concentraciones y el de trayectorias, así como otros programas de utilidades para conversión de formatos de ficheros de entrada, creación de representaciones gráficas, chequeo de datos de entrada, etc.
- `./library/source`: en este directorio se encuentran cerca de 200 rutinas usadas en los distintos modelos de simulación, así como varios módulos con definiciones de datos y prototipos.

Como `hymodelm` es la versión paralela del modelo de concentraciones, en el presente proyecto sólo se sometió a estudio el fichero *main* `hymodelc.F`. También formó parte del análisis el código fuente incluido en la carpeta `./library/source` que cuenta con la implementación de todas las rutinas, así como los ficheros `makefiles`. Debido a la naturaleza del programa, orientado a la realización de cálculos meteorológicos para la obtención de pronósticos, fue necesario obtener una explicación previa sobre su funcionamiento antes de comenzar con el análisis del código. Sin embargo, teniendo en cuenta que el proyecto se estaba desarrollando en el CIAI-AEMET (Tenerife, España) y el único vínculo más directo con los desarrolladores trabaja en ARL (Maryland, Estados Unidos) se encontraron ciertos problemas de comunicación debido principalmente, al gran desfase horario. Por esta razón, la comunicación con ARL se acabó estableciendo únicamente vía e-mail. Por otro lado, dada la gran complejidad del programa, así como el enorme tamaño del código fuente, no era viable una explicación detallada del mismo sin una reunión presencial. No obstante, debido a la imposibilidad de una reunión presencial hasta cinco meses después de la fecha de inicio del presente proyecto, se partió de la idea muy general acerca del funcionamiento del programa y se optó por comenzar un análisis del código aún disponiendo de escasa documentación. Únicamente se proporcionó como documentación la Guía de Usuario de HYSPLIT en donde no se establece ningún tipo de aclaración sobre el código del programa. Así que fue solamente el propio código fuente, junto a los comentarios de cabecera de las rutinas y comentarios técnicos en algunas líneas de código, la información con la que se contó acerca del código de HYSPLIT para su análisis.

3.1.2. Determinación del código asociado a `hymodelm`

El primer fichero a estudiar para comenzar el análisis del código fue el *main* del modelo de concentraciones, que se encuentra disponible en el fichero `hymodelc.F` del directorio *source*. Cuenta con un total de 2400 líneas de código. En él se implementan además otros modelos que se compilan a partir de dicho fichero realizando la definición de las constantes correspondientes en tiempo de compilación (para ser procesadas por el preprocesador). Es por ello que a lo largo de todo el código fuente se presentan innumerables líneas con directivas del preprocesador del tipo:

```
#ifdef cte
  code_1
#else
  code_2
#endif
```

Estas directivas se usan en el código para realizar las compilaciones condicionales anteriormente mencionadas y poder obtener así, a partir de un único fichero fuente, varios ejecutables. El problema que se encontró es que se hace un gran uso de estas directivas a lo largo de todo el código, estando incluso anidadas en algunos casos y separando secciones de código en intervalos irregulares. A esto hay que añadir, que es posible definir en tiempo de compilación numerosas constantes que permiten la generación de hasta diez programas distintos a partir del fichero *hymodelc.F*.

Como el estudio sólo se centra en `hymodelm`, se accedió al fichero *makefile* de HYSPLIT para saber qué constantes se definían en tiempo de compilación para obtener este programa. Para ello, se localizó la línea de compilación correspondiente y se buscó el símbolo que iba a continuación del flag `-D` de compilación. La constante que se define es `MPI`. Una vez encontrada la constante, fue posible intentar filtrar en el *main* las zonas de código que pertenecían a la implementación de `hymodelm`. En principio, bastaba con descartar todo aquel código que estuviera rodeado por las directivas condicionales:

```
#ifdef cte
  code
#endif
```

donde `cte` fuera distinta de `MPI`. Sin embargo, no se pudo aplicar esto tan directamente como se pensaba, pues se descubrió que en el propio fichero `main` se definen nuevas constantes en función de las que se han definido previamente en tiempo de compilación, como se muestra a continuación:

```
#ifdef MPI
#define MPIENSVAR
  code
#endif
```

Por tanto, había que tener en cuenta también estas nuevas constantes definidas, lo que hizo aún más tedioso el análisis del código, ya que en algunos casos se encontraron muchas ramificaciones provocadas por el anidamiento de directivas condicionales del preprocesador. Un ejemplo se muestra a continuación:

```
#ifdef MPI
#ifndef ENS
#ifndef CB4
  code_1
#else
  code_2
#endif
#else
  code_3
#endif
#else
  code_4
#endif
```

A esto hay que añadir, que el código que no está dentro de una secuencia de directivas condicionales del preprocesador también es ejecutado por `hymodelm`. Tras salvar todas estas dificultades mediante un minucioso análisis del código, se consiguió determinar las zonas de código correspondientes a `hymodelm`.

3.1.3. Determinación del flujo de `hymodelm`

Una vez determinado el código fuente asociado a `hymodelm` el siguiente paso consistió en realizar un estudio del flujo del programa. De esta forma, se tendría una idea general de su comportamiento que ayudaría a determinar las zonas más importantes en el desarrollo del presente proyecto, como la obtención de los datos de entrada. Sin embargo, nuevamente se encontraron algunas dificultades como:

- La implementación de cada una de las rutinas llamadas desde el *main* está en un fichero fuente aparte.
- La mayoría de las definiciones de las variables no están disponibles en el fichero fuente sino en los módulos que se incluyen. Hay un total de 17 módulos distintos.
- Se desconoce el funcionamiento detallado de los algoritmos implementados. No son "algoritmos estándar" que tengan documentación disponible.
- Todos los cálculos realizados en el programa pertenecen al ámbito meteorológico, por lo que son de difícil comprensión.

Debido a todo ello, era realmente complicado poder realizar una traza de ejecución sobre el código fuente, así que se decidió intentar buscar directamente la zona donde se leían los datos de entrada por las siguientes razones:

1. La zona de código de interés para este estudio corresponde a donde se realiza la lectura de los datos de entrada.
2. La lectura de datos es el punto previo a la realización de los cálculos meteorológicos.
3. El punto anterior ayudaría a separar el código perteneciente a inicializaciones (de bajo interés para determinar el flujo principal del programa) del código en el cual se invierte mayor tiempo (cálculos meteorológicos).

Así que se decidió empezar a buscar en el código fuente donde se obtienen los datos de entrada. En principio se desconocía la forma en que se realiza la lectura de datos en paralelo. Inicialmente, se supuso que cada proceso (en función de su identificador) lee una zona determinada del fichero de entrada de datos. Por tanto, lo lógico era comenzar a filtrar las zonas de código donde hubiera llamadas a rutinas MPI. Para ello, desde los directorios `./source` y `./library/source` se utilizó el comando:

```
$> grep -rin ‘‘MPI_’’ *
```

donde `-r` busca recursivamente, `-i` ignora mayúsculas/minúsculas y `-n` muestra las líneas donde está cada ocurrencia. Este comando proporcionó una forma rápida y eficaz de localizar las llamadas a MPI, indicando la línea y el fichero correspondiente.

En el directorio con el código fuente de las rutinas no se encontró ninguna llamada a funciones MPI, aunque en el fichero *main* se localizaron un total de cinco llamadas (dentro de la zona de código correspondiente a `hymodelm`):

- `MPI_INIT`
- `MPI_COMM_SIZE`
- `MPI_COMM_RANK`
- `MPI_REDUCE`
- `MPI_FINALIZE`

Por otro lado, como era posible que algunos datos relacionados con la utilización de MPI (como número de procesadores o el identificador asociado a un proceso) podían ser pasados como argumentos a otras rutinas, fue necesario buscar en el código estas posibles variables. Se encontraron:

- `job_id`: identificador (*rank*) asociado a un proceso.
- `num_job`: número de procesadores que están ejecutando el programa.

Una vez localizadas, se analizaron las zonas de código donde se emplean estas variables, ya que, si la lectura se realizaba de forma diferente según el proceso (como se había supuesto), serían utilizadas. Para ello se usó de nuevo el comando `grep` y se encontraron un total de cinco rutinas que reciben como argumento alguna de las variables anteriormente citadas: `CB4CON`, `CB4GRD`, `EMSPNT`, `EMSTMP` y `TRJSET`. Se estudiaron los ficheros fuente asociados a cada una de estas rutinas con el fin de determinar si alguna de ellas realizaba la lectura de los datos de entrada. Como primer paso se leyeron los comentarios de cabecera que describían en un par de líneas la funcionalidad de la rutina. A pesar de que en ninguno de los comentarios indicaba que se realizaba la obtención de los datos de entrada, se buscó alguna llamada a la sentencia `READ` de FORTRAN, pero en ninguno de los casos se encontró. Por tanto, todas estas rutinas están destinadas exclusivamente a la realización de cálculos, con lo que se clasificaron como rutinas de bajo interés en el estudio. Tampoco se encontró ninguna llamada explícita a `READ` en las zonas del *main* donde se hacía uso de

alguna de las variables `job_id` o `num_job`, con lo que se concluyó que la lectura de datos la realizaban todos los procesos simultáneamente.

El siguiente paso consistió en buscar las llamadas a `READ` realizadas en todo el resto del código, para determinar cuál es la rutina encargada de obtener los datos de entrada. No se encontró ninguna llamada explícita a `READ` en el *main*, sin embargo, se obtuvo una amplia lista de ficheros donde sí se realizaba alguna llamada. Esta lista resultó ser demasiado grande como para plantearse analizar cada fichero independientemente, leyendo su correspondiente cabecera y localizando las llamadas a `READ` en el cuerpo de la rutina. El hecho de haber obtenido una lista tan amplia es lógico. Esto se debe a que algunas de las rutinas son únicamente usadas en utilidades que incorpora HYSPLIT al margen de la implementación del modelo, como conversión del fichero binario de salida a otros formatos, chequeo del fichero de entrada, etc. Teniendo en cuenta la naturaleza de las utilidades, las rutinas usadas para su implementación cuentan con múltiples llamadas a `READ`. Además, como en el directorio `./library/source` no existe ningún tipo de separación entre las rutinas usadas en la implementación de las utilidades y las usadas en la implementación de los modelos, no era viable realizar un análisis fichero a fichero. Así que se decidió buscar otro método que facilitara el encuentro de la rutina encargada de la obtención de los datos de entrada.

Si se parte del supuesto de que la lectura de los datos ocupa gran parte del tiempo de ejecución, lo lógico es que la rutina encargada de ello sea llamada desde la rutina que mayor tiempo de ejecución consume. En este caso, la mejor forma de determinarla es realizando un perfil de ejecución (*profile*). Para la obtención del *profile* se hizo uso de la herramienta `gprof` [10] y se ejecutó la versión secuencial del modelo de concentraciones (`hymodelc`). Como entrada se utilizó el fichero con datos meteorológicos `20080312_hysplit.t00z.namsa.AK` de 606 MB cuyo dominio meteorológico corresponde a Alaska. Este fichero fue obtenido desde el ftp de ARL [3]. El fichero de control utilizado está disponible en el Apéndice A.2 y cuenta con cinco fuentes de emisión. El fichero de configuración que se usó establece una simulación con 10000 partículas y puede consultarse en el Apéndice B.3. A partir de los resultados del *profile* se elaboró la Tabla 3.1. En ella se muestran las cuatro rutinas que consumen mayor tiempo de ejecución en la columna izquierda, mientras que en la columna derecha se especifica el porcentaje de tiempo de ejecución consumido por cada rutina, incluyendo el tiempo invertido en las funciones llamadas desde el cuerpo de la propia rutina.

Nombre de la rutina	**Tiempo total de ejecución usado por la rutina y sus hijas (%)**
ADVPNT	76.1
ADVMET	49.2
ADV3NT	38.8
PARDSP	18.4

Tabla 3.1: Resumen del *profile*

Como se puede observar, la rutina que con diferencia, consume más tiempo de ejecución es `ADVPNT`, llegando a un 76.1 % con respecto al tiempo total. Por tanto, existía una alta probabilidad de que la lectura de datos de entrada se realizara en algún punto de dicha rutina. Se analizaron los comentarios de cabecera de `ADVPNT` que describían su funcionalidad. La información aportada se indica a continuación:

"*`ADVPNT` (ADVection of one PoiNT in space) es la principal rutina que se usa en las simulaciones de trayectorias y dispersión. Se encarga de calcular la advección de un punto del espacio. Es llamada cada periodo de tiempo y chequea la información meterológica almacenada en arrays. Si el punto a calcular coincide con el periodo de tiempo y los límites espaciales actuales, el cálculo continúa, en otro caso, solicita la lectura de nuevos datos a partir del fichero de entrada.*"

En la última línea de esta descripción se indica que desde `ADVPNT` se obtienen los datos de entrada, así que tan sólo había que encontrar dónde. Primero se buscó la presencia de llamadas a `READ` en el cuerpo de la rutina, pero no se encontró ninguna, así que la lectura

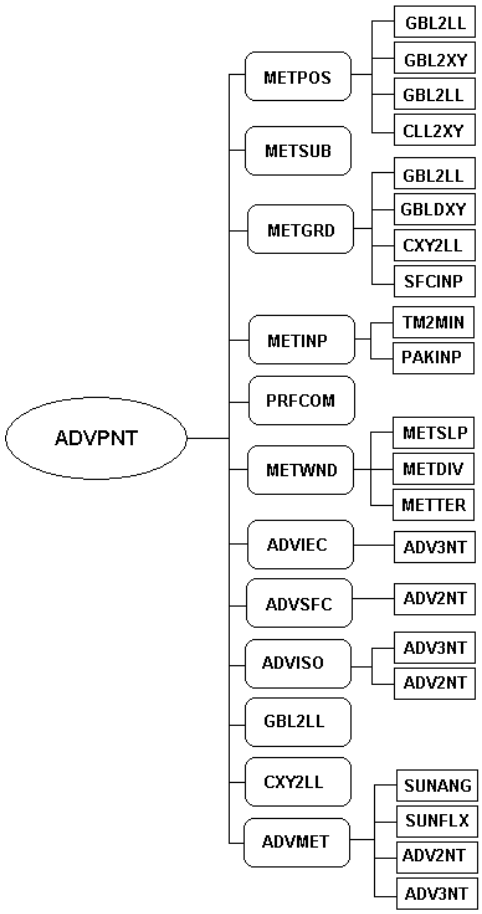

Figura 3.1: Árbol de rutinas llamadas desde `ADVPNT`

de datos debía hacerse a través de una de las múltiples rutinas a las que llama `ADVPNT`. Se realizó un árbol de llamadas para facilitar la búsqueda de la rutina de lectura, el cual se muestra en la Figura 3.1.

Analizando los ficheros de código fuente asociados a las rutinas llamadas desde `ADVPNT` se logró encontrar la rutina encargada de la lectura de los datos de entrada: `METINP`. La descripción de la misma disponible en los comentarios de cabecera se indica a continuación:

"*`METINP` (METeorological INPut): en cada llamada, lee los datos meteorológicos correspondientes a un único periodo de tiempo y los almacena en las variables correspondientes (arrays). Los datos que no se encuentren se ignoran y los valores pertenecientes al paso de tiempo anterior permanecen almacenados.*"

Una vez localizada la rutina de lectura, se determinó desde qué zonas de código se realizan llamadas a la misma. Para ello, se usó de nuevo el comando `grep`. Se obtuvo que `METINP` sólo es llamada desde la rutina `ADVPNT`, la cual a su vez, sólo es llamada desde el *main*. Se estudió el cuerpo de ambas rutinas con el fin de comprender cuándo y de qué forma se realizaba la lectura de los datos de entrada. Se llegaron a las siguientes conclusiones:

- No se lee todo el fichero de entrada de una sola vez. La lectura se hace bajo demanda, leyéndose los registros correspondientes en función de los datos que se necesitan para realizar los cálculos en cada paso de tiempo.
- Todos los procesos leen los mismos datos de entrada y de forma simultánea.
- No se realiza una lectura selectiva del fichero en función del identificador asociado a cada proceso, como se pensó en un primer momento.

Analizando las llamadas realizadas desde el *main* a la rutina `ADVPNT`, así como a las rutinas MPI, se pudo determinar a grandes rasgos el flujo principal del programa. En la Figura 3.2 se muestra un diagrama de flujo de `hymodelm` para n procesos. Cada línea roja representa a un proceso. A la izquierda se presentan los ficheros principales que intervienen en la simulación: el fichero de control (`CONTROL`) y el de configuración (`SETUP.CFG`), los ficheros con datos de entrada (`INPUT_1`, `INPUT_2`,... `INPUT_12`), el fichero de mensajes de diagnóstico de la simulación (`MESSAGE`) y el fichero de salida (`OUTPUT`). Asimismo se representa en qué momento de la ejecución intervienen.

Cuando se comienza una simulación, primero se realizan la operaciones de inicialización, entre las que destaca la inicialización de MPI (llamada a `MPI_INIT`). Después se lleva a cabo el chequeo de las opciones de simulación y luego, se entra en el bucle principal del programa, donde se realizan los cálculos meteorológicos (se destaca el cálculo de la advección): para cada hora de simulación establecida, para cada paso de tiempo, para cada partícula o *puff* se calcula la advección. El cálculo de la advección requiere de la lectura de los datos de entrada por parte de todos los procesos (llamadas a la rutina `METINI`). Tras el cálculo de la advección, se recolectan los datos obtenidos por todos los procesos en un único proceso (proceso maestro) llamando a la rutina `MPI_REDUCE` y se realizan cálculos globales. El proceso maestro escribe los resultados en el fichero de salida y luego todos los procesos escriben los mensajes de diagnóstico de la simulación en el fichero `MESSAGE`. Una

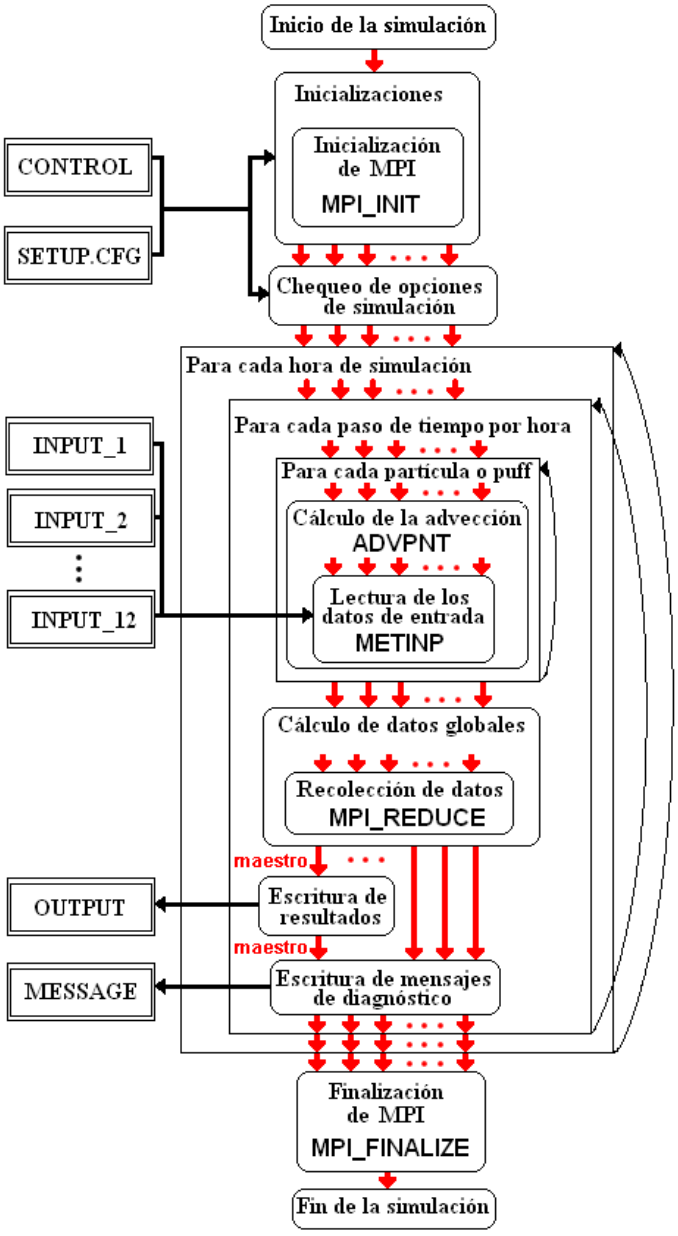

Figura 3.2: Diagrama de flujo de `hymodelm`

vez hecho esto, se vuelve a iterar hasta finalizar los pasos de tiempo, y a su vez, las horas de simulación. Posteriormente, se finaliza MPI (se llama a `MPI_FINALIZE`) y se muestra el mensaje de finalización del programa: “Complete Hysplit”.

Una vez conocido el funcionamiento general de `hymodelm` y determinado cómo y dónde se realiza la lectura de los datos de entrada, resultó de interés estimar cuánto tiempo ocupa la lectura respecto al tiempo de ejecución total. A partir del *profile* realizado con anterioridad, se pudo obtener que, en una ejecución en secuencial del modelo de concentraciones, el porcentaje de tiempo invertido por la rutina `METINP` durante la ejecución representaba un 12.6 %. Este valor es más bajo de lo que se esperaba, con lo que se hizo necesario averiguar qué factores podrían afectar al aumento del tiempo de lectura.

3.2. Estudio del tiempo invertido en la lectura

En esta sección se presenta el estudio sobre los parámetros configurables por el usuario que pueden incrementar el tiempo de obtención de los datos de entrada en HYSPLIT. Asimismo, se explican cada uno de los experimentos realizados para reproducir y analizar el comportamiento de los procesos cuando se realiza una lectura en paralelo. También se comentan los experimentos elaborados con el fin de proponer una solución abstracta al problema planteado.

3.2.1. Factores que incrementan el tiempo de lectura

Se realizó un estudio de las variables de los ficheros de control y configuración cuyos valores podrían aumentar el tiempo de obtención de datos del fichero de entrada. Para ello se analizó la Guía de Usuario de HYSPLIT [44] aportada como documentación. De entre los múltiples parámetros, se descartó la mayoría por su descripción y tan sólo quedaron tres candidatos:

- Variable `topmod` del fichero de control: también se denomina variable "tope del modelo". Representa el límite vertical de la malla meteorológica, es decir, la altura (en metros) más allá de la cual los datos meteorológicos no se procesan.
- Variable `mgmin` del fichero de configuración: esta variable representa el valor mínimo en unidades de malla horizontal de la sub-malla meteorológica para el cálculo de concentraciones. Su valor por defecto es 10. Incrementar este valor permite forzar una carga de datos de todo la malla (*full-grid*).
- Variable `numpar` del fichero de configuración: representa el número de partículas emitidas por una fuente durante una simulación.

Para determinar la influencia de la variable `topmod` sobre el tiempo de lectura, se realizaron varias ejecuciones de `hymodelc` con distintos valores superiores al que tenía esta variable por defecto (10000 m) y se observó que a medida que aumentaba este valor, se incrementaba el tiempo de lectura, ya que se procesaban un mayor número de niveles. Los meteorólogos comentaron que un valor superior a 10000 m en esta variable carece de sentido, pues 10 km representa el tope adecuado para los cálculos dentro de la troposfera, donde tienen lugar todas las simulaciones meteorológicas. Por tanto, la manipulación de la variable `topmod` quedó descartada.

Por otro lado, según la documentación consultada, la variable `mgmin` sí influye en el aumento del tiempo de lectura. Se debe cambiar el valor por defecto de esta variable a 1000 (o valores superiores) cuando realizan ejecuciones de los modelos paralelos. Esto provoca el aumento del tamaño mínimo de la malla meteorológica y fuerza una carga de datos de todo la malla. Es por ello que el tiempo de lectura se incrementa, pues se lee una mayor cantidad de datos. Se cambió el valor de `mgmin` a 1000 para garantizar un estudio más fiable, ya que es el valor mínimo que se recomienda que tenga esta variable cuando se realizan ejecuciones de los modelos paralelos en HYSPLIT.

Finalmente, para poder determinar si la variación del valor de `numpar` influye en el aumento del tiempo de lectura, se decidió realizar el experimento que se describe en la siguiente sección.

3.2.2. Ejecuciones secuenciales del modelo de concentraciones

Este experimento consistió en medir el tiempo total empleado en la lectura del fichero de entrada, así como el tiempo de ejecución total, realizando varias ejecuciones en secuencial del modelo de concentraciones (`hymodelc`) usando ficheros de entrada de diversa índole y tamaño y variando el número de partículas de cada simulación. El objetivo era determinar si el número de partículas y el tamaño del fichero de entrada influyen en el incremento del tiempo de obtención de los datos. Para desarrollar este experimento, se modificó el código fuente de `hymodelc` añadiendo dos temporizadores que midieran el tiempo de ejecución total del programa y el tiempo total de lectura de los datos, respectivamente. Para ello se hizo uso de la rutina `SYSTEM_CLOCK` disponible en FORTRAN 90.

ID del fichero	Nombre del fichero	Tamaño del fichero
1	20080830_hysplit.t00z.namsa.HI	68 MB
2	20080422_hysplit.t00z.namsa.HI	68 MB
3	20080512_hysplit.t00z.namsa.HI	68 MB
4	20080612_hysplit.t00z.namsa.HI	68 MB
5	20080711_hysplit.t00z.namsa.HI	68 MB
6	20080812_hysplit.t00z.namsa.HI	68 MB
7	20080912_hysplit.t00z.namsa.HI	68 MB
8	20081012_hysplit.t00z.namsa.HI	68 MB
9	20081112_hysplit.t00z.namsa.HI	68 MB
10	20081212_hysplit.t00z.namsa.HI	68 MB
11	20080830_hysplit.t00z.namsa.AK	606 MB
12	20080422_hysplit.t00z.namsa.AK	606 MB
13	20080512_hysplit.t00z.namsa.AK	606 MB
14	20080612_hysplit.t00z.namsa.AK	606 MB
15	20080711_hysplit.t00z.namsa.AK	606 MB
16	20080812_hysplit.t00z.namsa.AK	606 MB
17	20080912_hysplit.t00z.namsa.AK	606 MB
18	20081012_hysplit.t00z.namsa.AK	606 MB
19	20081112_hysplit.t00z.namsa.AK	606 MB
20	20081212_hysplit.t00z.namsa.AK	606 MB

Tabla 3.2: Lista de ficheros de entrada

Como ficheros con datos meteorológicos de entrada, se eligieron un total de 20. Cada fichero contiene datos meteorológicos relativos a un determinado dominio meteorológico. Los ficheros con extensión *.HI y *.AK contienen datos correspondientes a los dominios meteorológicos de Hawaii y Alaska, respectivamente. En la Tabla 3.2 se muestra el nombre de cada uno de los ficheros utilizados, así como su tamaño y un identificador que se asignó para permitir referenciarlos de forma más sencilla. Cada uno de los ficheros de entrada usados fueron descargados desde el servidor ftp de ARL [3].

Se crearon tres ficheros de configuración distintos, ya que se deseaba realizar las ejecuciones con 10000, 50000 y 100000 partículas. En cada fichero de configuración se cambió el valor por defecto de la variable `mgmin` a 1000, tal y como se había recomendado. El valor de la variable `topmod` se mantuvo a 10000. El fichero de configuración para la simulación con 10000 partículas está disponible en el Apéndice B.2. El resto de ficheros de configuración utilizados son iguales que éste, salvo que el valor de la variable `numpar` (número de partículas) se cambió de 10000 a 50000 y 100000, respectivamente.

Debido a que en el fichero de control se debe indicar el nombre y ruta del fichero de entrada, se tuvo que crear un fichero de control por cada fichero de meteorología con el que se iba a realizar el experimento. Además, fue necesario establecer el número de fuentes de emisión de partículas (un total de 5) y su localización, que se mantuvo constante para un mismo dominio meteorológico. El formato general de los ficheros de control usados para los ficheros de entrada con dominio correspondiente a Hawaii y Alaska pueden consultarse en los Apéndices A.1 y A.2, respectivamente. Para un mismo dominio, los ficheros de control tan sólo difieren en la ruta y nombre del fichero de entrada.

Para la realización de las diferentes ejecuciones del experimento se contó con la máquina *Hurricane*, que se encuentra en el CIAI (Centro de Investigación Atmosférica de Izaña) de la Agencia Estatal de Meteorología. *Hurricane* tiene Linux como sistema operativo y consta de un total de dos procesadores Intel Xeon de 3.2 GHz con 4 GB de memoria.

El desarrollo del experimento consistió en realizar tres ejecuciones en secuencial de `hymodelm` con 10000, 50000 y 100000 partículas, respectivamente, tomando como entrada varios ficheros con dominio meteorológico de Alaska y de Hawaii. Esto dio lugar a un total de 60 simulaciones. Para cada una de ellas se obtuvo el tiempo de ejecución total y el tiempo empleado en la lectura de los datos de entrada. Los resultados del experimento pueden consultarse en las Tablas 3.3 y 3.4.

En la primera columna de la Tabla 3.3 se indican los identificadores de los distintos ficheros de entrada utilizados, cuyo dominio meteorológico corresponde a Hawaii. En la segunda y tercera columna, se presentan el tiempo de ejecución total y el tiempo invertido en la lectura de los datos de entrada, respectivamente. En la cuarta columna se indica el número de partículas con que se ha ejecutado la simulación y finalmente, la última columna expresa el porcentaje de tiempo de lectura invertido respecto al tiempo de ejecución total. En esta tabla se observa que a medida que aumenta el número de partículas, el tiempo empleado en la lectura parece mantenerse constante. No obstante, el tiempo de ejecución del programa sí aumenta. Esto es debido a que, al tener que procesar un mayor número de partículas, se realiza un mayor número de cálculos, con el consecuente aumento del tiempo de ejecución. Sin embargo, esto no provoca un aumento en el tiempo de lectura.

ID del fichero	**Tiempo de ejecución (seg)**	**Tiempo de lectura (seg)**	**Número de partículas**	**Tiempo de lectura (%)**
1	230.6306000	6.6773000	10000	2.8952359
2	251.7543000	6.8490000	10000	2.7205096
3	225.5241000	6.8498000	10000	3.0372807
4	282.5821000	6.8719000	10000	2.4318242
5	218.4099000	6.6723000	10000	3.0549439
6	252.4474000	6.6720000	10000	2.6429268
7	348.2351000	6.7767000	10000	1.9460129
8	272.8972000	6.7165000	10000	2.4611831
9	238.3032000	6.6783000	10000	2.8024382
10	328.1791000	6.7341000	10000	2.0519588
1	1275.7817000	6.6454000	50000	0.5208884
2	1387.9817000	6.5558000	50000	0.4723261
3	1247.0804000	6.6443000	50000	0.5327884
4	1562.6842000	6.5757000	50000	0.4207951
5	1188.8764000	6.4028000	50000	0.5385589
6	1394.2875000	6.5404000	50000	0.4690854
7	1957.9539000	6.5871000	50000	0.3364277
8	1517.4997000	6.4375000	50000	0.4242175
9	1306.6814000	6.5679000	50000	0.5026397
10	1837.8718000	6.4716000	50000	0.3521246
1	2698.3232000	6.4995000	100000	0.2408718
2	3051.0731000	7.3797000	100000	0.2418722
3	2621.2623000	6.3526000	100000	0.2423488
4	3307.5480000	6.5850000	100000	0.1990900
5	2551.0294000	6.5345000	100000	0.2561514
6	2951.8827000	6.4641000	100000	0.2189822
7	4156.9394000	6.5921000	100000	0.1585806
8	3251.0633000	6.4707000	100000	0.1990333
9	2780.3511000	6.4589000	100000	0.2323051
10	3850.1012000	6.4340000	100000	0.1671124

Tabla 3.3: Tiempos de ejecución y lectura con los ficheros de Hawaii

En la Tabla 3.4 se presentan los resultados de las simulaciones realizadas usando diferentes ficheros con dominio meteorológico de Alaska como entrada. Esta tabla tiene el mismo formato y resultados similares a los obtenidos en la Tabla 3.3. El tiempo de lectura se mantiene constante a pesar de variar el número de partículas en las simulaciones. No obstante, el tiempo de ejecución total sí aumenta, por las mismas razones explicadas con anterioridad. Comparando los resultados de ambas tablas, se observa que el tiempo empleado en la lectura de un fichero con datos meteorológicos de Hawaii es menor que el de Alaska. Si se observa la Tabla 3.2 se comprueba que el tamaño de los ficheros con meteorología de Hawaii tienen menor tamaño que los de Alaska. Por tanto, como era de esperar, el tiempo de lectura se incrementa con el tamaño del fichero de entrada. Por otro lado, el porcentaje de tiempo de ejecución que consume la obtención de los datos de entrada supone hasta un 16.41 % en el caso de los ficheros de Alaska, mientras que apenas llega al 2.89 % en el caso de Hawaii. Estos valores máximos se alcanzan con simulaciones de 10000

ID del fichero	**Tiempo de ejecución (seg)**	**Tiempo de lectura (seg)**	**Número de partículas**	**Tiempo de lectura (%)**
11	490.8182000	73.2153000	10000	14.9169896
12	452.0824000	74.1845000	10000	16.4095085
13	539.0655000	73.9297000	10000	13.7144187
14	481.3583000	73.8604000	10000	15.3441626
15	473.3149000	73.8637000	10000	15.6056148
16	500.9904000	73.9376000	10000	14.7582868
17	492.2537000	74.0639000	10000	15.0458798
18	526.1700000	74.1042000	10000	14.0836992
19	480.3894000	73.7001000	10000	15.3417415
20	501.4946000	73.9063000	10000	14.7372075
11	2086.0626000	72.6187000	50000	3.48113714
12	1871.5337000	72.4800000	50000	3.87275954
13	2354.3349000	72.4956000	50000	3.07923907
14	2033.3373000	72.3025000	50000	3.55585372
15	1969.3306000	72.4469000	50000	3.67875764
16	2116.6431000	72.6046000	50000	3.43017677
17	2076.8407000	72.4178000	50000	3.48692126
18	2281.5021000	72.4002000	50000	3.17335671
19	2226.7163000	77.2014000	50000	3.46705146
20	2151.6054000	72.5863000	50000	3.37358793
11	4211.5540000	72.4988000	100000	1.72142634
12	3787.0485000	72.3165000	100000	1.90957417
13	4781.9636000	72.3827000	100000	1.51366062
14	4116.0231000	72.2881000	100000	1.75626079
15	3971.0031000	72.4592000	100000	1.82470772
16	4335.7044000	72.3708000	100000	1.66918206
17	4269.2488000	72.4960000	100000	1.69809733
18	4629.3511000	72.2755000	100000	1.56124473
19	4065.5608000	72.6648000	100000	1.78732538
20	4376.8788000	72.2263000	100000	1.65017820

Tabla 3.4: Tiempos de ejecución y lectura con los ficheros de Alaska

partículas. Este porcentaje baja en las simulaciones con un número de partículas mayor, debido a que el tiempo de lectura se mantiene, pero aumenta el tiempo de ejecución.

En la Tabla 3.5 se presenta un resumen de los resultados obtenidos en el experimento. En la tercera, cuarta y quinta columna se indican el tiempo medio de ejecución del programa, el tiempo medio empleado en la lectura de los datos de entrada y el porcentaje medio de tiempo de lectura empleado en relación al tiempo de ejecución total, respectivamente. Estos datos se muestran para cada simulación realizada combinando un fichero de entrada con un determinado dominio meteorológico y un número de partículas distinto. Se observa que el tiempo medio de ejecución y de lectura, así como el porcentaje medio de lectura en los ficheros de meteorología de Alaska es mayor que el de Hawaii en todos los casos. Asimismo, el valor del tiempo medio de lectura es mayor en el caso del fichero de entrada de Alaska que en el de Hawaii. Por tanto, se pudo concluir que el número de partículas no influye significativamente en el aumento del tiempo empleado en la lectura de los datos

Tamaño del fichero	**Número de partículas**	**Tiempo medio de ejecución (seg)**	**Tiempo medio de lectura (seg)**	**Tiempo medio de lectura (%)**
68 MB	10000	264.89630	6.74979	2.54808
68 MB	50000	1467.6699	6.54285	0.44579
68 MB	100000	3121.9574	6.57711	0.21067
606 MB	10000	493.79374	73.87657	14.96101
606 MB	50000	2116.79070	72.95540	3.44650
606 MB	100000	4254.43360	72.39787	1.70117

Tabla 3.5: Resumen de los resultados del experimento

de entrada, pero sí el tamaño del fichero de entrada. Según los resultados, parece que el porcentaje de tiempo empleado en la lectura respecto el tiempo de ejecución total no es tan significativo como se pensaba. Hay que tener en cuenta que las simulaciones se han ejecutado de forma secuencial, así que cabe la posiblidad de que el comportamiento cambie cuando se ejecute en paralelo. Por ello, fue necesario estudiar la lectura paralela de los datos de entrada.

3.2.3. Lectura paralela (nodos heterogéneos)

Existen grandes dificultades para medir con exactitud cuál es el tiempo empleado en la obtención de los datos de entrada de la versión en paralelo del modelo de concentraciones. Se puede medir el tiempo que ha invertido cada proceso en la lectura, pero no es posible determinar si dichos tiempos se solapan total o parcialmente, con lo que no se puede obtener de forma precisa el tiempo total invertido en la lectura. Debido a esto, se decidió reproducir el problema implementando un programa que leyera un fichero. Como la funcionalidad de dicho programa sólo es la de leer datos de entrada, el tiempo de ejecución total es aproximadamente igual al tiempo empleado en la lectura. De esta forma, se dispone de un método que ofrece la posibilidad de analizar efizcamente el comportamiento que se desea estudiar.

Por otro lado, se debe tener en cuenta que las simulaciones se realizan sobre un cluster con NFS [37], [54], [63] y que esto provoca ralentización en las ejecuciones. Como el fichero de entrada está compartido, si se desea realizar una lectura paralela del mismo por n procesadores, no se puede hacer de forma simultánea. Los procesos son gestionados por el sistema para que sólo uno acceda cada vez al recurso. Esto provoca que la lectura en paralelo por n procesadores tienda a secuencializarse, es decir, el tiempo total empleado en la lectura es aproximadamente la suma del tiempo invertido por cada proceso en leer el fichero. Esto es casi equivalente a realizar n lecturas secuenciales. Por tanto, como la funcionalidad del programa desarrollado para este experimento es la de leer un fichero, se estima que el tiempo de ejecución del mismo será:

$$T_{exec} \simeq n * T_{sec} \tag{3.1}$$

donde T_{exec} es el tiempo de ejecución total del programa paralelo con n procesadores, n es el número de procesadores y T_{sec} es el tiempo de ejecución del programa en secuencial.

Para desarrollar este experimento se implementó un programa en FORTRAN 90 [41] que realiza la lectura de un fichero en paralelo usando MPI. Además, se incluyeron temporizadores que miden el tiempo invertido por cada proceso en la lectura, así como el tiempo de ejecución del programa. Para ello se hizo uso de la función `SYSTEM_CLOCK`. El código fuente de dicho programa está disponible en el Apéndice C.1.

Como entrada al programa se eligió un fichero de tamaño considerable (2.3 GB aproximadamente), ya que los ficheros de entrada de HYSPLIT suelen ser bastantes grandes debido a la naturaleza de los datos que contienen. Así, se podría simular de forma más real el problema. La máquina utilizada para el desarrollo de este experimento fue el cluster de AEMET. Este cluster es un sistema Beowulf, el cual se caracteriza por estar dedicado al procesamiento en paralelo y construido a partir de hardware estándar. Ejecuta un sistema operativo de libre distribución (Linux), y se interconecta mediante una red privada de gran velocidad. El software que se ejecuta de forma paralela hace uso de la librería MPI para las comunicaciones. Se compone de un total de cinco nodos (siguiendo el paradigma maestro-esclavo). El nodo maestro controla todo el cluster y comparte un sistema de ficheros (NFS) con los nodos esclavos. También es la consola del cluster y su conexión al exterior. Los esclavos no tienen teclado ni monitor y son accesibles sólo a través de *login* remoto. El cluster no es homogéneo y cada uno de sus nodos son identificados como: *jet0* (maestro), *jet1*, *jet2*, *jet3* y *jet4* (esclavos). Los nodos *jet0*, *jet1* y *jet2* disponen cada uno de un procesador de cuatro núcleos Intel Quad Core de 2.40 GHz y memoria de 4 GB. Los nodos *jet3* y *jet4* cuentan con un procesador de cuatro núcleos Intel Quad Core de 2.50 GHz y 8 GB de memoria.

El experimento consistió en ejecutar el programa implementado y obtener los tiempos de ejecución y lectura realizando ejecuciones en secuencial y en paralelo con 1, 2, 4, 8, 12 y 16 procesadores.

Los resultados obtenidos pueden verse en la Tabla 3.6. En ella se representa el tiempo de lectura invertido por cada uno de los procesos en cada ejecución en paralelo, así como el tiempo de ejecución total. También se indica el tiempo de lectura y ejecución en la versión secuencial. Como se observa, los tiempos de la ejecución en secuencial y en paralelo con un único procesador son prácticamente los mismos, como era de esperar. Sin embargo, se aprecia como el tiempo de ejecución (y consecuentemente, de lectura) se incrementa a medida que el número de procesadores aumenta. Cabe destacar el gran "salto" que se establece en tiempo de ejecución cuando se pasa de dos a cuatro procesadores, siendo éste último casi diez veces superior al tiempo de ejecución con dos procesadores. En la ejecución con ocho procesadores también se aprecia un incremento del tiempo de ejecución, pero en este caso menor que el anterior. A partir de ocho procesadores, parece que el tiempo de ejecución se mantiene y se estabiliza en torno al valor de 1800 segundos. Claramente no se cumple la ecuación 3.1.

También se observa en los resultados que en una misma ejecución, algunos procesos emplean más tiempo en la lectura que otros. Esto se pone de manifiesto en las ejecuciones usando ocho o más procesadores. En el caso de la ejecución realizada con ocho procesadores, los procesos del 0 al 3 emplean unos 1293 segundos en la lectura, mientras que los procesos con identificador del 4 al 7 invierten uno 1750 segundos (más de 450 segundos de diferencia). Esto puede deberse a que los procesos, en su lucha por acceder al recurso,

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	88.14293000	88.14330000
1	0	89.28270000	89.28290000
2	0	120.6380000	122.7180000
	1	121.7112000	
4	0	1284.657800	1285.669400
	1	1284.164300	
	2	1284.499200	
	3	1284.182000	
8	0	1293.498600	1752.545300
	1	1293.091700	
	2	1292.623200	
	3	1292.977500	
	4	1751.435100	
	5	1750.693200	
	6	1750.964800	
	7	1749.872900	
12	0	1284.600100	1771.844400
	1	1284.172300	
	2	1283.329900	
	3	1284.152700	
	4	1752.155100	
	5	1752.506600	
	6	1753.548400	
	7	1753.489500	
	8	1770.753000	
	9	1767.845100	
	10	1766.944500	
	11	1770.350600	
16	0	1304.665000	1802.80260
	1	1305.706200	
	2	1306.017700	
	3	1305.639400	
	4	1745.345700	
	5	1745.372700	
	6	1745.847900	
	7	1744.825000	
	8	1801.705600	
	9	1799.568600	
	10	1797.864200	
	11	1799.545300	
	12	1376.046500	
	13	1375.533000	
	14	1375.561400	
	15	1375.840400	

Tabla 3.6: Tiempos de lectura y ejecución del programa de lectura paralela

provocan que unos acaben más tarde la lectura que otros. Por otro lado, cabe destacar la ralentización que sufre el programa paralelo: la ejecución en secuencial tarda en torno a 90 segundos, mientras que con 16 procesadores unos 1800 segundos. Esto implica que la lectura de un fichero en paralelo con 16 procesadores tarde 20 veces más que la lectura del fichero en secuencial. Estableciendo analogías con respecto al comportamiento de la versión paralela del modelo de concentraciones (`hymodelm`), estos resultados podrían reflejar las causas por las que dicho programa no escala adecuadamente.

El "salto" que se aprecia entre el tiempo de ejecución de dos a cuatro procesadores hizo pensar que probablemente, el tamaño del fichero de lectura influye en los resultados, ya que pudo haberse realizado un *swapping* debido a la limitación de memoria de los nodos. El espacio de memoria de intercambio o *swap*, es lo que se conoce como memoria virtual. La diferencia entre la memoria real y la virtual es que esta última utiliza espacio en el disco duro en lugar de un módulo de memoria. Cuando la memoria real se agota, el sistema copia parte del contenido de ésta directamente en este espacio de memoria de intercambio a fin de poder realizar otras tareas. El *swapping* con disco consume demasiado tiempo de acceso y transferencia, con lo que se pensó que probablemente esto podría ser lo que estaba ocurriendo a partir de las ejecuciones con cuatro procesadores. Por otro lado, observando las características de los nodos del cluster utilizado, se ve que no son homogéneos y que la memoria de la cual disponen varía, así que cabía la posibilidad de que esto también estuviera influyendo. Con el fin de averiguarlo, se inició un nuevo experimento, el cual se explica con detalle en la siguiente sección.

3.2.4. Lectura paralela (nodos homogéneos)

Este experimento consistió en ejecutar el mismo programa utilizado en el experimento anterior, pero sólo sobre nodos homogéneos, es decir, con el mismo procesador y cantidad de memoria diponible. Con esto se prentendía averiguar cuál es la causa de la gran diferencia (hasta diez veces más) entre los tiempos de ejecución obtenidos con la ejecución del programa paralelo con dos y cuatro procesadores, respectivamente. Se pensó que este fenómeno podría deberse a que se realiza un *swapping* con disco, el cual consume un tiempo significativo. De ser así los resultados cambiarían respecto a los obtenidos en el experimento anterior. Asimismo, también se comprobó si en este caso tampoco se cumple la ecuación 3.1.

Para la realización de este experimento se hizo uso del cluster de AEMET, cuyas características se explican en la sección 3.2.3 y como entrada al programa se eligieron ficheros de tamaños distintos: 1 GB, 2 GB, 4 GB y 8 GB. Se seleccionaron estos tamaños basándose en la memoria disponible en cada nodo, con el fin de determinar si se estaba realizando *swapping*. El tamaño máximo de los ficheros que se emplearon en las ejecuciones sobre los nodos *jet1* y *jet2* es de 4 GB, debido a que coincide con el tamaño de la memoria de la que disponen. Por esta misma razón, se eligió un fichero de hasta 8 GB para las pruebas realizadas con los nodos *jet3* y *jet4*.

El experimento consistió en la realización de varias ejecuciones del programa de lectura paralela utilizado en el experimento anterior (disponible en el Apéndice C.1), haciendo uso exclusivo de nodos homogéneos y tomando como entrada ficheros de distintos tamaños.

Las características de las pruebas realizadas se indican a continuación:

- Ejecución en secuencial con un fichero de entrada de 1 GB, 2 GB, 4 GB y 8 GB, respectivamente.
- Ejecución en paralelo con 1, 2, 4 y 8 procesadores usando los nodos:
  - *jet1* y *jet2*, con un fichero de entrada de 1 GB, 2 GB y 4 GB, respectivamente.
  - *jet3* y *jet4*, con un fichero de entrada de 1 GB, 2 GB, 4 GB y 8 GB, respectivamente.

Los resultados obtenidos se muestran en las Tablas 3.7 a la 3.13. En las Tablas 3.7, 3.8 y 3.9 se presentan el tiempo de lectura empleado por cada proceso y el tiempo de ejecución total ejecutando el programa con 1, 2, 4 y 8 procesadores sobre los nodos *jet1* y *jet2*. Como entrada se empleó un fichero de 1 GB, 2 GB y 4 GB, respectivamente. También se indican los tiempos asociados a la ejecución del programa en secuencial. En estas tablas se puede observar cómo se sigue produciendo un "salto" respecto al valor del tiempo de ejecución, entre las ejecuciones del programa en paralelo con dos y cuatro procesadores. Esto ocurre en el caso del fichero de entrada de 1 GB, de 2GB y de 4 GB. Parece ser que no se está realizando *swapping* a disco, ya que si no, no se obtendrían estos mismos resultados para los tres tamaños de los ficheros de entrada y menos aún para los de menor tamaño. Por otro lado, se observa que en la lectura en secuencial del fichero de 1 GB se emplean 49.356 segundos, en el de 2 GB un total de 97.724 segundos y en fichero de 4 GB se invierten 197.32 segundos. Analizando estos valores se aprecia que existe una relación lineal entre los tiempos de lectura en secuencial obtenidos en función del tamaño de los ficheros. Esta linealidad también se aprecia en los tiempos asociados a la lectura paralela, dependiendo además, del número de procesadores, de forma que, aproximadamente se cumple:

$$T_{p,n} \simeq T_{p,1} * n \tag{3.2}$$

donde $T_{p,n}$ es el tiempo de lectura en paralelo por p procesadores de un fichero de n GB, $T_{p,1}$ es el tiempo de lectura en paralelo por p procesadores de un fichero de 1 GB y n es el tamaño en GB del fichero leído.

En las Tablas 3.10 a la 3.13 se presentan el tiempo de lectura empleado por cada proceso y el tiempo de ejecución total ejecutando el programa con 1, 2, 4 y 8 procesadores sobre los nodos *jet3* y *jet4*. También se indican los tiempos asociados a la ejecución del programa en secuencial. Como entrada se tomó un fichero de 1 GB, 2 GB, 4 GB y 8 GB. Se observan resultados análogos a los obtenidos con las ejecuciones sobre los nodos *jet1* y *jet2*: se cumple la relación lineal 3.2 y se aprecia un "salto" en el valor de los tiempos entre la ejecución con dos a cuatro procesadores. En este caso, tampoco parece que la causa de esta diferencia sea la realización de un *swapping*, ya que se tienen los mismos resultados con todos los ficheros de entrada (independientemente de su tamaño) y no debería presentarse el "salto" en los tiempos obtenidos con ficheros de entrada de pequeño tamaño (como 1 GB y 2 GB).

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	49.3562000	49.3563000
1	0	67.8825000	67.8883000
2	0	58.4760000	59.4826000
	1	58.3792000	
4	0	734.6797000	736.4299000
	1	735.2743000	
	2	735.4190000	
	3	735.3565000	
8	0	721.5606000	722.6976000
	1	721.1490000	
	2	721.1328000	
	3	721.2192000	
	4	685.1576000	
	5	685.1537000	
	6	685.1579000	
	7	685.1569000	

Tabla 3.7: Tiempos de lectura y ejecución con fichero de 1 GB (*jet1* y *jet2*)

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	97.7240000	97.7241000
1	0	133.2673000	133.2834000
2	0	123.4058000	124.4146000
	1	123.1062000	
4	0	1472.7749000	1474.8585000
	1	1472.9618000	
	2	1473.8470000	
	3	1473.6140000	
8	0	1444.3860000	1446.1312000
	1	1444.7179000	
	2	1445.0345000	
	3	1443.5583000	
	4	1382.4037000	
	5	1382.4052000	
	6	1382.4052000	
	7	1382.4046000	

Tabla 3.8: Tiempos de lectura y ejecución con fichero de 2 GB (*jet1* y *jet2*)

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	197.3269000	197.3271000
1	0	234.9417000	234.9616000
2	0	248.5361000	249.5497000
	1	248.5421000	
4	0	2951.2020000	2952.2544000
	1	2951.1952000	
	2	2951.2225000	
	3	2951.2295000	
8	0	2932.8305000	2933.8698000
	1	2932.8155000	
	2	2932.8252000	
	3	2932.8228000	
	4	2800.5496000	
	5	2800.5498000	
	6	2800.5474000	
	7	2800.5494000	

Tabla 3.9: Tiempos de lectura y ejecución con fichero de 4 GB (*jet1* y *jet2*)

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	49.3562000	49.3563000
1	0	66.0460000	66.0698000
2	0	58.4517000	59.4601000
	1	58.0688000	
4	0	923.4110000	924.4229000
	1	923.0592000	
	2	921.8099000	
	3	923.3265000	
8	0	906.4154000	912.6326000
	1	910.5585000	
	2	910.4687000	
	3	910.0757000	
	4	910.4502000	
	5	910.2034000	
	6	910.4299000	
	7	910.4059000	

Tabla 3.10: Tiempos de lectura y ejecución con fichero de 1 GB (*jet3* y *jet4*)

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	97.7240000	97.7241000
1	0	126.2196000	126.2258000
2	0	114.1108000	116.1206000
	1	115.1122000	
4	0	1838.1686000	1839.3688000
	1	1837.3313000	
	2	1838.3564000	
	3	1837.8670000	
8	0	1847.5104000	1854.5153000
	1	1847.7081000	
	2	1846.6397000	
	3	1847.5314000	
	4	1852.4111000	
	5	1852.4886000	
	6	1852.3529000	
	7	1852.1756000	

Tabla 3.11: Tiempos de lectura y ejecución con fichero de 2 GB (*jet3* y *jet4*)

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	197.3269000	197.3271000
1	0	263.8379000	263.8443000
2	0	227.6209000	230.1590000
	1	229.1509000	
4	0	3632.8609000	3634.9661000
	1	3631.7540000	
	2	3627.2286000	
	3	3633.9541000	
8	0	3703.1164000	3705.1887000
	1	3703.1110000	
	2	3703.1206000	
	3	3703.1017000	
	4	3695.8500000	
	5	3695.8197000	
	6	3695.8095000	
	7	3695.8379000	

Tabla 3.12: Tiempos de lectura y ejecución con fichero de 4 GB (*jet3* y *jet4*)

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	394.4895000	394.4897000
1	0	523.756000	523.8085000
2	0	592.8527000	593.8085000
	1	592.8527000	
4	0	7375.6469000	7376.7405000
	1	7375.6510000	
	2	7375.6514000	
	3	7375.6403000	
8	0	7486.1931000	7488.3162000
	1	7486.1979000	
	2	7486.1935000	
	3	7486.1967000	
	4	7267.8345000	
	5	7267.4478000	
	6	7267.7941000	
	7	7267.8151000	

Tabla 3.13: Tiempos de lectura y ejecución con fichero de 8 GB (*jet3* y *jet4*)

Por otro lado, se observa que, en todas las tablas, tanto en las ejecuciones sobre *jet1* y *jet2* como en las de *jet3* y *jet4*, a partir de cuatro procesadores, se produce un incremento del tiempo de lectura en paralelo diez veces superior al tiempo de lectura con dos procesadores:

$$T_{p,n} > 10 * T_{2,n} \quad siendo\ p \geqslant 4 \tag{3.3}$$

donde $T_{p,n}$ es el tiempo de lectura en paralelo por p procesadores de un fichero de n GB, $T_{2,n}$ es el tiempo de lectura en paralelo por dos procesadores de un fichero de n GB y n es el tamaño en GB del fichero leído.

Comparando las tablas correspondientes a este y al anterior experimento se puede apreciar que en ningún caso se cumple la ecuación 3.1 y que existen muchas similitudes entre los resultados obtenidos. Esto indica que la ejecución sobre nodos homogéneos y heterogéneos parece no influir significativamente sobre el tiempo invertido en la lectura y que la idea de que se está realizando *swapping* a disco debe ser descartada.

3.2.5. Lectura secuencial y *broadcast*

En el experimento anterior se comprobó que el *swapping* no es la causa de la gran diferencia de tiempos entre ejecuciones con dos y cuatro procesadores. Teniendo en cuenta que el cluster utiliza NFS, una posible explicación a este fenómeno es que se debe a la ralentización producida por la competición de los procesos al intentar acceder a un mismo recurso de forma simultánea. Como sólo se permite el acceso de un único proceso a la vez al recurso, el sistema debe realizar la gestión correspondiente, lo que hace incrementar el tiempo de lectura a medida que se involucra a un mayor número de procesos. Una posible solución a este problema es que tan sólo uno de los procesos acceda al fichero, lea los datos

del mismo y posteriormente los envíe al resto de procesadores. Este tipo de comunicación colectiva entre procesos se denomina *broadcast*.

Se decidió implementar un programa que realizara la lectura de la forma anteriormente explicada con el fin de estudiar su comportamiento. Para ello, se hizo uso de la rutina que provee MPI [58] para realizar *broadcast*: `MPI_BROADCAST`. Se modificó el programa utilizado en los anteriores experimentos para que fuera un único proceso el que leyera la información y fuera almacenándola en un *buffer* y enviándola al resto de procesos mediante sucesivas llamadas a `MPI_BROADCAST`. El código fuente de este programa puede consultarse en el Apéndice C.2. Como además, se quería comprobar si el *buffer* de lectura utilizado podía influir en los tiempos de lectura, se utilizaron distintos tamaños.

Como entrada al programa se eligió un fichero de 2,3 GB y las ejecuciones se realizaron sobre el cluster de AEMET (el mismo utilizado en los dos experimentos anteriores). El experimento consistió en medir el tiempo de ejecución y lectura en los siguientes casos:

- Ejecución del programa en secuencial.
- Ejecución del programa en paralelo (con 1, 2, 4, 8, 12 y 16 procesadores) variando el tamaño del *buffer* de lectura en 8 KB, 512 KB, 32 MB y 512 MB.

Los resultados obtenidos están disponibles en las Tablas 3.14, 3.15, 3.16 y 3.17. En todas ellas se indica el tiempo de lectura invertido por cada uno de los procesos en una ejecución del programa en paralelo con 1, 2, 4, 8, 12 y 16 procesadores, así como el tiempo de ejecución total. También se muestran los tiempos de la ejecución en secuencial.

En la Tabla 3.14 se observa como no existe ninguna diferencia especialmente notable entre el tiempo de ejecución con dos procesadores y el correspondiente al de cuatro procesadores (apenas unos tres segundos). Por tanto, ya no aparece ese "salto" que se producía en los experimentos anteriores. La mayor diferencia se establece a partir de ejecuciones con más de ocho procesadores, incrementándose el tiempo de lectura en torno a 25 segundos, siendo este valor poco significativo. De hecho, el mayor tiempo obtenido, en la ejecución en paralelo con 16 procesadores, llega tan sólo a 179 segundos. Por otro lado, si comparamos, por ejemplo, el tiempo de ejecución obtenido con ocho procesadores en este y el anterior experimento, a través de las Tablas 3.14 y 3.11, respectivamente, se observa que hay una diferencia considerable. En el presente experimento se invierten 131 segundos mientras que en el anterior 1854 segundos. Estos resultados se obtienen en la lectura de un fichero de tamaño similar (unos 2 GB) y refleja que el programa que implementa la lectura secuencial y luego un envío *broadcast* es más eficiente que el que realiza la lectura en paralelo.

En las Tablas 3.15 y 3.16 se presentan unos resultados muy similares a los de la Tabla 3.14, a pesar de que se utilizan *buffers* de envío de mayor tamaño, en este caso de 512 KB y 32 MB, respectivamente. El tiempo de lectura se incrementa muy poco a medida que aumenta el número de procesadores no llegando a alcanzar, en ninguno de los casos, los 180 segundos.

En la Tabla 3.17 se indican los tiempos de lectura y ejecución usando un *buffer* de envío de datos de 512 MB. Estos tiempos son ligeramente mayores a las correspondientes ejecuciones mostradas anteriormente (con *buffer* de menor tamaño) llegando a alcanzar como

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	117.9923000	117.9923000
1	0	118.1943000	118.1943000
2	0	118.3028000	119.3103000
	1	118.3027000	
4	0	121.5239000	122.5352000
	1	121.5241000	
	2	121.5239000	
	3	121.5239000	
8	0	129.7724000	131.4328000
	1	129.7724000	
	2	129.7724000	
	3	129.7727000	
	4	129.7731000	
	5	129.7729000	
	6	129.7728000	
	7	129.7730000	
12	0	154.7102000	156.1014000
	1	154.7102000	
	2	154.7102000	
	3	154.8994000	
	4	154.8995000	
	5	154.7109000	
	6	154.7108000	
	7	154.8996000	
	8	154.8946000	
	9	154.7070000	
	10	154.7068000	
	11	154.8943000	
16	0	177.1538000	179.1674000
	1	177.1539000	
	2	177.1540000	
	3	177.1538000	
	4	177.1459000	
	5	177.1458000	
	6	177.1455000	
	7	177.1454000	
	8	177.1518000	
	9	177.1514000	
	10	177.1513000	
	11	177.1516000	
	12	177.1526000	
	13	177.1522000	
	14	177.1522000	
	15	177.1528000	

Tabla 3.14: Tiempos de lectura y ejecución con *buffer* de 8 KB

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	118.0125000	118.0125000
1	0	117.1299000	117.1299000
2	0	117.0241000	118.0406000
	1	117.0241000	
4	0	119.9443000	121.0375000
	1	119.9443000	
	2	119.9443000	
	3	119.9442000	
8	0	148.8928000	150.1387000
	1	148.8930000	
	2	148.8930000	
	3	148.8992000	
	4	148.9000000	
	5	148.8989000	
	6	148.8988000	
	7	148.8996000	
12	0	156.4036000	157.4486000
	1	156.4021000	
	2	156.4019000	
	3	156.4272000	
	4	156.4257000	
	5	156.4130000	
	6	156.4129000	
	7	156.4237000	
	8	156.4119000	
	9	156.4061000	
	10	156.4060000	
	11	156.4013000	
16	0	174.1313000	176.1996000
	1	174.1314000	
	2	174.1313000	
	3	174.1313000	
	4	174.1336000	
	5	174.1335000	
	6	174.1335000	
	7	174.1336000	
	8	174.1296000	
	9	174.1297000	
	10	174.1296000	
	11	174.1295000	
	12	174.1318000	
	13	174.1318000	
	14	174.1318000	
	15	174.1318000	

Tabla 3.15: Tiempos de lectura y ejecución con *buffer* de 512 KB

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	119.0833000	119.0833000
1	0	119.8397000	119.8397000
2	0	120.6098000	121.6175000
	1	120.6098000	
4	0	122.2506000	123.2628000
	1	122.2507000	
	2	122.2509000	
	3	122.2510000	
8	0	153.3357000	154.3776000
	1	153.3392000	
	2	153.3428000	
	3	153.3465000	
	4	153.3489000	
	5	153.3490000	
	6	153.3491000	
	7	153.3491000	
12	0	171.4759000	172.5274000
	1	171.4821000	
	2	171.4884000	
	3	171.4945000	
	4	171.4976000	
	5	171.4978000	
	6	171.4978000	
	7	171.4982000	
	8	171.4941000	
	9	171.4944000	
	10	171.4946000	
	11	171.4945000	
16	0	177.4703000	178.5361000
	1	177.4764000	
	2	177.4824000	
	3	177.4887000	
	4	177.4919000	
	5	177.4921000	
	6	177.4921000	
	7	177.4925000	
	8	177.4881000	
	9	177.4884000	
	10	177.4884000	
	11	177.4886000	
	12	177.4921000	
	13	177.4922000	
	14	177.4925000	
	15	177.4924000	

Tabla 3.16: Tiempos de lectura y ejecución con *buffer* de 32 MB

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	117.8132000	117.8132000
1	0	117.9289000	117.9289000
2	0	119.7791000	120.7861000
	1	119.7790000	
4	0	125.8375000	126.8557000
	1	125.8375000	
	2	125.8377000	
	3	125.8376000	
8	0	168.8620000	169.9308000
	1	168.8658000	
	2	168.8693000	
	3	168.8743000	
	4	168.8763000	
	5	168.8765000	
	6	168.8766000	
	7	168.8765000	
12	0	197.6539000	198.7517000
	1	197.6600000	
	2	197.6662000	
	3	197.6723000	
	4	197.6752000	
	5	197.6754000	
	6	197.6755000	
	7	197.6760000	
	8	197.6710000	
	9	197.6713000	
	10	197.6714000	
	11	197.6713000	
16	0	197.3816000	199.4738000
	1	197.3876000	
	2	197.3936000	
	3	197.3996000	
	4	197.4026000	
	5	197.4028000	
	6	197.4031000	
	7	197.4047000	
	8	197.3999000	
	9	197.4000000	
	10	197.4001000	
	11	197.4004000	
	12	197.4042000	
	13	197.4043000	
	14	197.4045000	
	15	197.4044000	

Tabla 3.17: Tiempos de lectura y ejecución con *buffer* de 512 MB

máximo 199 segundos con 16 procesadores. Aún así, sigue siendo un tiempo muchísimo menor que el obtenido en el experimento anterior bajo ficheros de entrada de similar tamaño.

A partir de los resultados de este experimento se concluye que, en general, el tamaño del *buffer* de envío no supone una ralentización importante de las comunicaciones entre los procesos para obtener los datos leídos por el proceso maestro. Por lo que, en principio, el coste asociado al envío de datos es aproximadamente equivalente en todos los casos. Asimismo, se observa que el tiempo de lectura aumenta ligeramente con el número de procesadores. Sin embargo, este incremento es, comparado con el obtenido con el programa de lectura paralela, muchísimo menor. Todo esto pone de manifiesto que la lectura secuencial de un fichero y el posterior envío *broadcast* de los datos se podría presentar como una posible solución al problema planteado en el presente proyecto.

3.2.6. Lectura secuencial y *broadcast* con acceso a datos

En el experimento anterior se comprobó que, con respecto a la lectura paralela, se produce una considerable reducción del tiempo de lectura cuando ésta se realiza por un único proceso que comunica luego los datos leídos al resto de procesos. Sin embargo, fue necesario realizar otro tipo de comprobaciones antes de proponer esto como solución como, por ejemplo, estudiar el comportamiento del programa utilizado en el experimento previo si además, se realiza un acceso a los datos. Para ello, se modificó el programa anterior con el fin de que todos los procesos hicieran un acceso a los datos leídos. Los cambios introducidos dieron lugar a un nuevo programa cuyo código fuente está disponible en el Apéndice C.3.

Como fichero de entrada al programa se usó el del experimento de la sección 3.2.5. Además, las ejecuciones se realizaron sobre la misma máquina y con idénticos supuestos en cuanto a número de procesadores y tamaño de *buffers*.

Los resultados obtenidos usando un *buffer* de tamaño 8 KB, 512 KB, 32 MB y 512 MB se presentan en las Tablas 3.18, 3.19, 3.20 y 3.21, respectivamente. En todas ellas se indica el tiempo de lectura invertido por cada uno de los procesos en una ejecución del programa paralelo con 1, 2, 4, 8, 12 y 16 procesadores, así como el tiempo de ejecución total. También se muestran los tiempos de la ejecución en secuencial.

Si comparamos la Tabla 3.18 con la Tabla 3.14 del experimento anterior, se aprecia que los resultados son prácticamente iguales, tanto los tiempos de ejecución como los de lectura, para todas las ejecuciones realizadas sobre distinto número de procesadores. Los mismo ocurre con las Tablas 3.19, 3.20 y 3.21, lo que indica que el acceso a los datos leídos no implica un aumento del tiempo de lectura. En las ejecuciones donde se utilizaba un *buffer* de envío de 512 MB es donde se presentó mayor tiempo de ejecución, llegando a los 206 segundos con 16 procesadores, tal como se observa en la Tabla 3.21. Aún así, este tiempo sigue siendo muy inferior a los obtenidos en los experimentos mostrados en las secciones 3.2.3 y 3.2.4 donde se hacía una lectura en paralelo. En general, los tiempos reflejan un buen comportamiento del programa a medida que se aumenta el número de procesadores con que se ejecuta, no produciéndose ningún "salto" significativo como ocurrió en casos anteriores.

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	119.9661000	119.9661000
1	0	121.9577000	121.9632000
2	0	119.2695000	120.2774000
	1	119.2696000	
4	0	120.2926000	121.3035000
	1	120.2926000	
	2	120.2927000	
	3	120.2925000	
8	0	130.9667000	132.0492000
	1	130.9669000	
	2	130.9669000	
	3	130.9672000	
	4	130.9676000	
	5	130.9672000	
	6	130.9673000	
	7	130.9674000	
12	0	159.4228000	160.4549000
	1	159.4230000	
	2	159.4230000	
	3	159.1912000	
	4	159.1914000	
	5	159.4236000	
	6	159.4235000	
	7	159.1915000	
	8	159.1857000	
	9	159.4194000	
	10	159.4193000	
	11	159.1855000	
16	0	179.2722000	180.9149000
	1	179.2729000	
	2	179.2729000	
	3	179.4466000	
	4	179.4473000	
	5	179.2734000	
	6	179.2733000	
	7	179.4472000	
	8	179.4392000	
	9	179.2695000	
	10	179.2695000	
	11	179.4389000	
	12	179.4423000	
	13	179.2718000	
	14	179.2718000	
	15	179.4422000	

Tabla 3.18: Tiempos de lectura y ejecución con *buffer* de 8 KB

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	117.9836000	117.9836000
1	0	118.1122000	118.1177000
2	0	116.6868000	117.7015000
	1	116.6940000	
4	0	117.7881000	118.8066000
	1	117.7953000	
	2	117.7952000	
	3	117.7952000	
8	0	154.7306000	155.799000
	1	154.7250000	
	2	154.7249000	
	3	154.7413000	
	4	154.7431000	
	5	154.7319000	
	6	154.7325000	
	7	154.7436000	
12	0	159.5136000	160.5928000
	1	159.5186000	
	2	159.5184000	
	3	159.5339000	
	4	159.5335000	
	5	159.5351000	
	6	159.5351000	
	7	159.5324000	
	8	159.5164000	
	9	159.5276000	
	10	159.5276000	
	11	159.5166000	
16	0	176.6606000	178.7409000
	1	176.6678000	
	2	176.6684000	
	3	176.6677000	
	4	176.6706000	
	5	176.6699000	
	6	176.6698000	
	7	176.6699000	
	8	176.6658000	
	9	176.6657000	
	10	176.6658000	
	11	176.6657000	
	12	176.6681000	
	13	176.6681000	
	14	176.6681000	
	15	176.6681000	

Tabla 3.19: Tiempos de lectura y ejecución con *buffer* de 512 KB

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	118.0712000	118.0712000
1	0	117.0057000	117.0112000
2	0	119.8267000	121.2978000
	1	120.2900000	
4	0	121.8385000	123.3153000
	1	122.3030000	
	2	122.3032000	
	3	122.3030000	
8	0	149.1183000	150.6503000
	1	149.5870000	
	2	149.5899000	
	3	149.5936000	
	4	149.5954000	
	5	149.5956000	
	6	149.5955000	
	7	149.5949000	
12	0	170.6141000	172.1570000
	1	171.0868000	
	2	171.0905000	
	3	171.0966000	
	4	171.1003000	
	5	171.0995000	
	6	171.0997000	
	7	171.1001000	
	8	171.0836000	
	9	171.0852000	
	10	171.0809000	
	11	171.0842000	
16	0	170.7631000	173.4654000
	1	171.4006000	
	2	171.2396000	
	3	171.2458000	
	4	171.2498000	
	5	171.2492000	
	6	171.2490000	
	7	171.2491000	
	8	171.2330000	
	9	171.2317000	
	10	171.2407000	
	11	171.2351000	
	12	171.2343000	
	13	171.2361000	
	14	171.2372000	
	15	171.2369000	

Tabla 3.20: Tiempos de lectura y ejecución con *buffer* de 32 MB

Número de procesadores	**ID del proceso**	**Tiempo de lectura (seg)**	**Tiempo de ejecución (seg)**
Secuencial	----	118.0949000	118.0949000
1	0	118.5320000	118.5377000
2	0	127.5199000	129.9913000
	1	128.9838000	
4	0	131.3997000	132.8427000
	1	131.8252000	
	2	131.8302000	
	3	131.8295000	
8	0	175.2645000	179.4359000
	1	175.6975000	
	2	175.7022000	
	3	175.7030000	
	4	175.7041000	
	5	175.3822000	
	6	175.7042000	
	7	175.7044000	
12	0	194.1035000	195.6128000
	1	194.5402000	
	2	194.5431000	
	3	194.5471000	
	4	194.5518000	
	5	194.5525000	
	6	194.5539000	
	7	194.5534000	
	8	194.3494000	
	9	194.3204000	
	10	194.3171000	
	11	194.4716000	
16	0	203.7826000	206.2952000
	1	204.2181000	
	2	204.2193000	
	3	204.2307000	
	4	204.2313000	
	5	204.2292000	
	6	204.2276000	
	7	204.2316000	
	8	203.9728000	
	9	204.0008000	
	10	204.0046000	
	11	204.2078000	
	12	204.0059000	
	13	203.9889000	
	14	204.0499000	
	15	204.0102000	

Tabla 3.21: Tiempos de lectura y ejecución con *buffer* de 512 MB

Se concluyó que una posible solución al problema de escalabilidad que presenta `hymodelm` es la implementación de la lectura de los datos de entrada en secuencial y el envío de datos mediante *broadcast*, ya que de forma experimental, se han obtenido excelentes resultados. En el siguiente capítulo se abordan los temas relacionados con la implementación de la solución propuesta.

Capítulo 4

Paralelización de la obtención de datos de entrada

En base a los estudios y experimentos realizados se ha propuesto una solución a la ralentización que se produce en la versión paralela del modelo de concentraciones a medida que el número de procesadores con que se ejecuta aumenta. La solución consiste en la obtención de los datos de entrada por parte de un único proceso, el cual se encarga de enviar los datos leídos al resto de procesadores (*broadcast*). A continuación, se explica cómo se llevó a cabo su implementación, así como los pasos seguidos para la validación de la misma.

4.1. Implementación

Para la implementación de la solución fue necesario, en primer lugar, determinar cada una de las variables que almacenan los datos de entrada y elegir el tipo de *broadcast* más apropiado. La difusión de datos o *broadcast* es un tipo de comunicación colectiva definida en MPI [66] cuyo esquema de funcionamiento se muestra en la Figura 4.1. Finalmente, se estableció el número y tipo de datos adecuado para cada comunicación *broadcast* y se realizó la implementación en código. En esta sección se describen cada uno de los pasos indicados anteriormente.

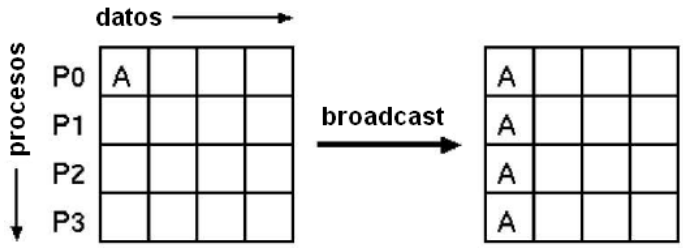

Figura 4.1: Esquema de la comunicación colectiva *broadcast*

4.1.1. Variables con los datos de entrada

El análisis al que fue sometido el código fuente (sección 3.1) permitió conocer cúal es la rutina que realiza la lectura de los datos de entrada: `METINP`. Esta rutina recibe un total de 33 argumentos entre los que se encuentran aquéllos que almacenan los datos de entrada leídos. La definición de cada uno de ellos, así como el prototipo de la rutina se indican a continuación:

```
SUBROUTINE METINP(BACK,KG,KT,KUNIT,KREC,LX1,LY1,NXS,NYS,NZS,MC,KEND,    &
                  IFHR,ZT,DS,P0,T0,U0,V0,UF,VF,HF,RT,ZI,U,V,W,A,Q,P,E,H,X)
!-------------------------------------------------------------------------------
! Lista de argumentos
!-------------------------------------------------------------------------------
LOGICAL,      INTENT(IN)    :: back      ! integration direction
INTEGER,      INTENT(IN)    :: kg        ! number of active grid
INTEGER,      INTENT(IN)    :: kt        ! number of active time
INTEGER,      INTENT(IN)    :: kunit     ! input device unit number
INTEGER,      INTENT(IN)    :: krec      ! record number of index record
INTEGER,      INTENT(IN)    :: lx1,ly1   ! lower left  of subgrid FG unit
INTEGER,      INTENT(IN)    :: nxs,nys   ! dimensions of sub-grid
INTEGER,      INTENT(IN)    :: nzs       ! number of data levels to read
INTEGER,      INTENT(INOUT) :: mc        ! accumulated minutes of data read
INTEGER,      INTENT(IN)    :: kend      ! last valid record number of file
INTEGER,      INTENT(OUT)   :: ifhr      ! current forecast hour
REAL,         INTENT(OUT)   :: zt (:,:)  ! terrain height
REAL,         INTENT(OUT)   :: ds (:,:)  ! downward shortwave
REAL,         INTENT(OUT)   :: p0 (:,:)  ! surface pressure
REAL,         INTENT(OUT)   :: u0 (:,:)  ! low level u wind
REAL,         INTENT(OUT)   :: v0 (:,:)  ! low level v wind
REAL,         INTENT(OUT)   :: t0 (:,:)  ! low level temperature
REAL,         INTENT(OUT)   :: uf (:,:)  ! u momentum flux
REAL,         INTENT(OUT)   :: vf (:,:)  ! v momentum flux
REAL,         INTENT(OUT)   :: hf (:,:)  ! sensible heat flux
REAL,         INTENT(OUT)   :: rt (:,:)  ! rainfall total
REAL,         INTENT(OUT)   :: zi (:,:)  ! mixed layer depth
REAL,         INTENT(OUT)   :: u (:,:,:) ! upper level u wind
REAL,         INTENT(OUT)   :: v (:,:,:) ! upper level v wind
REAL,         INTENT(OUT)   :: w (:,:,:) ! upper level w wind
REAL,         INTENT(OUT)   :: a (:,:,:) ! upper level temperature
REAL,         INTENT(OUT)   :: q (:,:,:) ! upper level moisture
REAL,         INTENT(OUT)   :: p (:,:,:) ! upper level pressure
REAL,         INTENT(OUT)   :: e (:,:,:) ! turbulent kinetic energy or v'2
REAL,         INTENT(OUT)   :: h (:,:,:) ! velocity variance u'2
REAL,         INTENT(OUT)   :: x (:,:,:) ! velocity variance w'2
```

Para determinar los datos que deben enviarse mediante *broadcast* fue necesario estudiar la lista de parámetros de `METINP`. Sólo interesaba enviar el contenido de aquellas variables cuya modificación tiene efecto más allá del ámbito de ejecución de la rutina, o lo que es lo mismo, las que se pasan como argumento de salida (`INTENT OUT`) o entrada/salida (`INTENT INOUT`). Por tanto, los parámetros de interés son: `mc, zt, ds, p0, u0, v0, t0, uf, vf, hf, rt, zi, u, v, w, a, q, p, e, h` y `x`.

Una vez determinados los argumentos de `METINP` a tener en cuenta en el *broadcast*, se estudiaron las llamadas a dicha rutina con el fin de identificar las variables correspondientes. `METINP` sólo es llamada desde la rutina `ADVPNT` un total de dos veces, cada una de las cuales se muestra a continuación:

```
!load meteorological data according to positioning
!reads data for NZS levels
CALL METINP(BACK,KG,KT1,KUNIT1,KREC1,LX1(KG),LY1(KG),NXS(KG),NYS(KG),     &
   NZS,FTIME(KG,K1),KEND1,FHOUR(K1,KG),ZT(:,:,KG),DS(:,:,K1,KG),          &
   PO(:,:,K1,KG),TO(:,:,K1,KG),UO(:,:,K1,KG),VO(:,:,K1,KG),               &
   UF(:,:,K1,KG),VF(:,:,K1,KG),HF(:,:,K1,KG),RT(:,:,K1,KG),ZI(:,:,K1,KG), &
   U(:,:,:,K1,KG),V(:,:,:,K1,KG),W(:,:,:,K1,KG),A(:,:,:,K1,KG),           &
   Q(:,:,:,K1,KG),P(:,:,:,K1,KG),E(:,:,:,K1,KG),H(:,:,:,K1,KG),           &
   X(:,:,:,K1,KG))

(code ...)

!load data for subgrid
CALL METINP(BACK,KG,KT2,KUNIT2,KREC2,LX1(KG),LY1(KG),NXS(KG),NYS(KG),     &
   NZS,FTIME(KG,K2),KEND2,FHOUR(K2,KG),ZT(:,:,KG),DS(:,:,K2,KG),          &
   PO(:,:,K2,KG),TO(:,:,K2,KG),UO(:,:,K2,KG),VO(:,:,K2,KG),               &
   UF(:,:,K2,KG),VF(:,:,K2,KG),HF(:,:,K2,KG),RT(:,:,K2,KG),ZI(:,:,K2,KG), &
   U(:,:,:,K2,KG),V(:,:,:,K2,KG),W(:,:,:,K2,KG),A(:,:,:,K2,KG),           &
   Q(:,:,:,K2,KG),P(:,:,:,K2,KG),E(:,:,:,K2,KG),H(:,:,:,K2,KG),           &
   X(:,:,:,K2,KG))
```

Se analizaron estas llamadas y en cada una de ellas se seleccionaron aquellas variables que son pasadas como argumento de salida o de entrada/salida (como se había establecido previamente). En ambos casos, las variables elegidas son de tipo *array* multidimensional: `FTIME`, `FHOUR`, `ZT`, `DS`, `PO`, `UO`, `VO`, `TO`, `UF`, `VF`, `HF`, `RT`, `ZI`, `U`, `V`, `W`, `A`, `Q`, `P`, `E`, `H` y `X`. Además, en estas dos llamadas se aprecia que las últimas dimensiones de estos *arrays* están fijadas por alguna de las variables: K1, K2 y KG. Por ello, en principio, se pensó que no era necesario enviar el contenido de todos los *arrays* multidimensionales (sino sólo el "trozo" especificado) al realizar un envío *broadcast*. No obstante, analizando el código fuente de la rutina `METINP` se descubrió que también se modifican otras regiones de dichos *arrays*, a partir de otros datos proporcionados como argumentos. Por tanto, esto hizo necesario que se tuviera que enviar el contenido de todos los *arrays* multidimensionales después de cada llamada a `METINP` para garantizar que todos los procesos dispusieran de los mismos valores tras la realización de la lectura. Las únicas excepciones fueron las variables `FTIME` y `FHOUR` donde no fue necesario enviar la totalidad del *array*.

Las definiciones de las variables `FHOUR` y `FTIME` se encuentran en la rutina `ADVPNT` (son variables locales), mientras que las del resto de variables se localizan en el fichero `METVAL.f`, ya que son globales. A continuación se listan las definiciones de todas ellas:

```
INTEGER, ALLOCATABLE :: fhour(:,:)     ! forecast hour associated with data
INTEGER, ALLOCATABLE :: ftime(:,:)     ! time of meteo data in array

REAL,    ALLOCATABLE :: u  (:,:,:,:,:) ! x wind (inp m/s; out grid/min)
REAL,    ALLOCATABLE :: v  (:,:,:,:,:) ! v wind (inp m/s; out grid/min)
REAL,    ALLOCATABLE :: w  (:,:,:,:,:) ! z wind (inp dp/dt; out sigma/min)
REAL,    ALLOCATABLE :: a  (:,:,:,:,:) ! ambient temp (deg-K)
REAL,    ALLOCATABLE :: q  (:,:,:,:,:) ! humid (inp rh or kg/kg; out rh/100)
REAL,    ALLOCATABLE :: p  (:,:,:,:,:) ! local air pressure (mb)
REAL,    ALLOCATABLE :: e  (:,:,:,:,:) ! input TKE (J/kg), output V'2 (m2/s2)
REAL,    ALLOCATABLE :: x  (:,:,:,:,:) ! vertical turbulence      W'2 (m2/s2)
REAL,    ALLOCATABLE :: h  (:,:,:,:,:) ! horizontal turbulence    U'2 (m2/s2)
REAL,    ALLOCATABLE :: p0 (:,:  ,:,:) ! model sfc press (mb)
REAL,    ALLOCATABLE :: rt (:,:  ,:,:) ! rainfall total (m)
REAL,    ALLOCATABLE :: u0 (:,:  ,:,:) ! low-level u horizontal wind (m/s)
REAL,    ALLOCATABLE :: v0 (:,:  ,:,:) ! low-level v horizontal wind (m/s)
REAL,    ALLOCATABLE :: t0 (:,:  ,:,:) ! low-level ambient temp (deg-K)
                                       ! u,v flux replaced by U* in prfcom
REAL,    ALLOCATABLE :: uf (:,:  ,:,:) ! u momentum flux (n/m2) or (kg/m2-s)
REAL,    ALLOCATABLE :: vf (:,:  ,:,:) ! v momentum flux (n/m2)
REAL,    ALLOCATABLE :: hf (:,:  ,:,:) ! sensible heat flux (w/m2)
REAL,    ALLOCATABLE :: zi (:,:  ,:,:) ! mixed layer depth  (m)
REAL,    ALLOCATABLE :: ds (:,:  ,:,:) ! downward shortwave flux (w/m2)
REAL,    ALLOCATABLE :: zt (:,:    ,:) ! terrain elevation (m)
```

Como se observa, todas estas variables son *arrays* dinámicos, es decir, se crean y destruyen en tiempo de ejecución y no se puede conocer su tamaño durante la compilación. Además son *arrays* multidimensionales (con 2, 3, 4 o incluso 5 dimensiones en este caso). Todos almacenan elementos de tipo `REAL`, excepto `fhour` y `ftime` cuyos elementos son tipo `INTEGER`.

Una vez localizadas las variables de interés en la lectura y los tipos de datos asociados, el siguiente paso consistió en estudiar las características del *broadcast* a implementar.

4.1.2. Selección del tipo de *broadcast*

Con el fin de determinar cuál es el tipo de *broadcast* [36], [50], [57] más adecuado para la implementación de la solución propuesta, se estudiaron los distintos mecanismos que provee MPI para agrupar datos en un único mensaje:

- *Parámetro contador*: permite agrupar datos que tengan el mismo tipo básico en un mismo mensaje. Para ello, dichos datos deben estar contiguos en memoria y el parámetro contador indica el número de elementos a enviar.
- *Tipos de datos derivados* (`MPI_Datatype`): se crean tipos de datos en tiempo de ejecución para permitir el envío, en un único mensaje, de varios datos con distintos tipos asociados y/o no contiguos en memoria.
- `MPI_Pack/MPI_Unpack`: `MPI_Pack` permite almacenar datos no contiguos en memoria en un *buffer* contiguo. `MPI_Unpack` copia los datos desde el *buffer* contiguo a zonas de memoria no contiguas.

Como prácticamente la totalidad de los datos a enviar están almacenados en zonas contiguas de memoria (*arrays*) y son del mismo tipo básico, lo más eficiente es usar el parámetro contador, ya que no incluye la sobrecarga adicional provocada por llamadas a constructores de tipos de datos derivados o llamadas a las rutinas `MPI_Pack/MPI_Unpack`. Por tanto, la llamada a la función MPI para envío de datos mediante *broadcast* que se eligió usar en la implementación es:

```
MPI_BROADCAST(source, counter, data_type, root, communicator, ierr)
```

donde `source` es un puntero al comienzo de la zona de memoria en la que se encuentran los datos a enviar, `counter` indica el número de elementos a enviar (parámetro contador), `root` representa el idenficador del proceso que envía los datos, `datatype` indica tipo de dato de los elementos a enviar, `communicator` es el comunicador y `ierr` el indicador de error de envío/recepción.

Tras elegir el tipo de *broadcast* más apropiado, se procedió a determinar el número de elementos que deben enviarse en cada llamada a `MPI_BROADCAST`.

4.1.3. Número y tipo de datos a enviar

Como se determinó en la sección 4.1.1, todas las variables que recibe `METINP` como argumento que son de interés para el envío *broadcast* son *arrays* dinámicos, así que el número de elementos que debe enviarse en cada caso viene determinado por otras variables del programa. Por ello, fue necesario analizar el código fuente con el fin de encontrar las dimensiones de dichos *arrays*. Inicialmente, se buscaron llamadas a la rutina `ALLOCATE`, ya que es la que permite reservar memoria para la creación de un *array* dinámico. Además, teniendo en cuenta que a `ALLOCATE` se le deben pasar como argumentos las dimensiones del *array* que se desea crear en tiempo de ejecución, esto permite conocer el número de elementos que se deben enviar.

Se encontraron varias llamadas a `ALLOCATE` desde `ADVPNT` y a partir de ellas, se pudieron determinar los valores y variables que definen las dimensiones de los *arrays*. Debido a que todos los elementos a enviar son, o bien del tipo básico `REAL` o `INTEGER`, el tipo de dato básico MPI asociado a cada envío fue `MPI_REAL` y `MPI_INTEGER`, respectivamente. En la Tabla 4.1 se presenta la relación entre variables, tipos de datos y valor del parámetro contador asociado. Las variables `NXT` y `NXY` indican el tamaño máximo de la sub-malla, mientras que `NGRD` indica el número de mallas meteorológicas y `NLVL` el número de niveles verticales.

Variables	**Tipo de dato MPI**	**Nº de elementos a enviar (parámetro contador)**
fhour,ftime	`MPI_INTEGER`	1
zt	`MPI_REAL`	NXT*NYT*NGRD
ds,p0,u0,v0,t0,uf,vf,hf,rt,zi	`MPI_REAL`	NXT*NYT*NGRD*2
u,v,w,a,q,p,e,h,x	`MPI_REAL`	NXT*NYT*NGRD*2*NLVL

Tabla 4.1: Relación de variables, tipo de datos y número de elementos para envío *broadcast*

Una vez conocidos en qué variables se almacenan los datos leídos, el número de elementos a enviar, a qué destino y de qué forma, tan sólo quedaba comenzar con la codificación de la solución a partir del código fuente disponible. En la siguiente sección se explican con detalle cada uno de los pasos seguidos.

4.1.4. Codificación

A la hora de realizar la implementación de la solución propuesta, se contaba con las siguientes restricciones:

- Implementar una nueva versión de `hymodelm` (denominada `hymodelmb`) que realice la lectura secuencial de los datos de entrada y los correspondientes envíos *broadcast*.
- Mantener en la medida de lo posible la estructura de librerías y módulos, así como la de los ficheros *makefiles* disponibles en la distribución del programa.

Debido a las restricciones anteriores, antes de comenzar con la codificación fue necesario estudiar el impacto que podían tener las modificaciones que debían llevarse a cabo en la rutina `ADVPNT` sobre el resto del código, especialmente en el momento de realizar la compilación. Se encontraron los siguientes problemas:

- Ninguna de las rutinas de HYSPLIT realiza llamadas a funciones de la librería MPI, sólo el programa principal.
- La rutina `ADVPNT` que se debe modificar para implementar la lectura secuencial y *broadcast* es externa y forma parte, junto con otras rutinas, de una librería dinámica (`libhysplit.a`) que se enlaza durante la compilación del programa principal. Por tanto, es necesario compilar previamente de dos formas distintas dicha rutina, una para el código original y otra para la implementación MPI de la solución, o bien "separarla" del resto rutinas.
- La librería `mpif.h` que se necesita incluir para realizar llamadas a rutinas MPI desde `ADVPNT`, no la encuentra el compilador desde el directorio donde se halla dicha rutina, con lo que se deben modificar convenientemente los ficheros *makefiles*.

El objetivo consistió en solventar todos los problemas listados anteriormente realizando las mínimas modificaciones posibles sobre la distribución del código de HYSPLIT. Para ello, se decidió mantener la librería de rutinas original y crear una nueva rutina denominada `ADVPNTBCAST` en el fichero donde se encuentra la implementación del *main* del modelo de concentraciones (`hymodelc.F`), justo a continuación de la última línea del *main*. La rutina `ADVPNTBCAST` es igual que la rutina `ADVPNT`, pero implementa la obtención de los datos de entrada realizando una lectura secuencial y envíos *broadcast*. Por otro lado, para poder crear una nueva versión de `hymodelm`, fue necesario definir una directiva para el preprocesador a la que se denominó `BCAST`. De esta forma, se puede llevar a cabo una compilación condicional del código fuente para la obtención de la nueva versión. El pseudocódigo asociado a las modificaciones que se realizaron sobre el fichero `hymodelc.F` se muestra a continuación:

```
--
| !Código Main...
|
|   !Asegura que el broadcast sólo es posible con la versión MPI del main
|   #ifdef BCAST
|   #ifndef MPI
|   #undef BCAST
|   #endif
|   #endif
|
|   !Para cada llamada a ADVPNT:
|
|   #ifdef BCAST
|     CALL ADVPNTBCAST
|   #else
|     CALL ADVPNT
|   #endif
|__


#ifdef BCAST
--
| !Código rutina ADVPNTBCAST...
|
|   !Para cada llamada a METINP:
|
|   !El proceso maestro lee los datos de entrada
|   IF (job_id .EQ. 0 )
|     CALL METINP
|   ENDIF
|
|   !El proceso maestro envía los datos/
|   !el resto de procesos recibe los datos
|   CALL MPI_BCAST
|    ...
|__
#endif
```

Tal como se aprecia, sólo se llama a la rutina `ADVPNTBCAST` si se define la constante `BCAST`, en caso contrario, el programa mantiene la implementación de la lectura de datos de entrada original (a través de `ADVPNT`).

Para la implementación del código asociado a la nueva rutina `ADVPNTBCAST`, se partió del código fuente de la rutina `ADVPNT` y se añadió un nuevo parámetro `job_id`, necesario para identificar el proceso que debe leer los datos de entrada (proceso maestro). Además, se definieron dos nuevas variables locales `SIZEA` y `IERR`, ambas de tipo `INTEGER`. `SIZEA` almacena el número de elementos a enviar en cada llamada a `MPI_BROADCAST`, mientras que `IERR` registra si un error ha ocurrido durante el envío. Asimismo, para la correcta compilación, fue necesario registrar el prototipo de la rutina `ADVPNTBCAST`, así como sus parámetros, en el fichero `DEFARG1.INC`.

Se colocó una sentencia condicional rodeando a cada llamada a `METINP` para permitir que sólo el proceso maestro lea los datos de entrada. A continuación de la llamada a `METINP`,

se implementó el envío *broadcast* de los datos leídos mediante sucesivas llamadas a la función `MPI_BCAST` disponible en la librería MPI para FORTRAN [58]. En el Apéndice C.4 se puede consultar el fragmento de código fuente de la rutina `ADVPNTBCAST` que implementa la lectura de datos secuencial y las correspondientes comunicaciones *broadcast*.

Finalmente, para realizar la compilación de `hymodelmb` se modificó el fichero *makefile* introduciendo las siguientes líneas:

```
all:
     (...)
     hymodelmb
     (...)

hymodelmb : $(SRC)/hymodelc.F  $(MPI) -o $@ $(PREP)-DMPI -DBCAST $(CFLAGS)
            $(SRC)/hymodelc.F $(MLIB) -L$(LIB) -lhysplit $(XLIB)
     (...)

clean:
      rm -f hymodelmb
     (...)
```

Como se observa, para la obtención de `hymodelmb` es necesario definir durante la compilación las constantes `MPI` y `BCAST`.

4.2. Validación

Una vez compilada la implementación de la solución propuesta, fue necesario comprobar la validez de la misma. Para ello se realizaron múltiples ejecuciones de `hymodelm` y `hymodelmb` con varios ficheros de entrada, diferente número de partículas y distinto número de procesadores, y se compararon las correspondientes salidas.

Para llevar a cabo las ejecuciones se utilizó el cluster *Tegasaste*. Este cluster consta de un total de 24 nodos (denominados *node1*, *node2*,... y *node24*, respectivamente) interconectados mediante una red Infiniband y ejecuta un sistema operativo de libre distribución (Linux). Los nodos utilizados para realizar las simulaciones fueron: *node17*, *node18*,... y *node24*. Cada uno de ellos cuenta con dos procesadores Intel Xeon de 3.20 GHz y 1 GB de memoria.

Como entrada se utilizaron varios ficheros con dominio meteorológico correspondiente a Alaska y Hawaii, obtenidos directamente desde el ftp de ARL [3]. La lista de los ficheros que se usaron se indica en la segunda columna de la Tabla 4.2. El fichero de configuración utilizado en las simulaciones se puede consultar en el Apéndice B.2 y el formato general de los ficheros de control asociados a los ficheros con datos meteorológicos de Alaska y Hawaii se muestra en el Apéndice A.4 y A.3, respectivamente.

La validación de la implementación de la solución propuesta consistió en realizar múltiples ejecuciones de `hymodelm` y `hymodelmb` con 1, 2, 4, 8 y 16 procesadores, comprobando que la salida de cada simulación realizada con `hymodelm` coincide con la correspondiente salida de `hymodelmb`. Para ello se utilizaron dos métodos:

- Comando `diff` [11]: compara dos ficheros e informa si existen diferencias entre ellos.

  ```
  diff -q SALIDA_hymodelm SALIDA_hymodelmb
  ```

  La opción `-q` informa tan sólo sobre si existen diferencias o no, sin mostrarlas.

- Comparación gráfica de las salidas: la versión para PC de HYSPLIT incopora una utilidad que permite convertir el fichero binario de salida a formato postscript a través de la opción *Concentration* → *Display* → *Concentration* → *Contours* [13]. Cada representación gráfica obtenida fue comparada visualmente.

Utilizando estos dos métodos, se logró comprobar que en todos los casos, las salidas de las simulaciones con `hymodelm` son iguales a las correspondientes con `hymodelmb`. Por tanto, se concluyó que la implementación de la solución propuesta es completamente válida. Una vez hecho esto, el último paso consistió en medir la mejora introducida por la nueva versión respecto a la original. Esto se explica detalladamente en la siguiente sección.

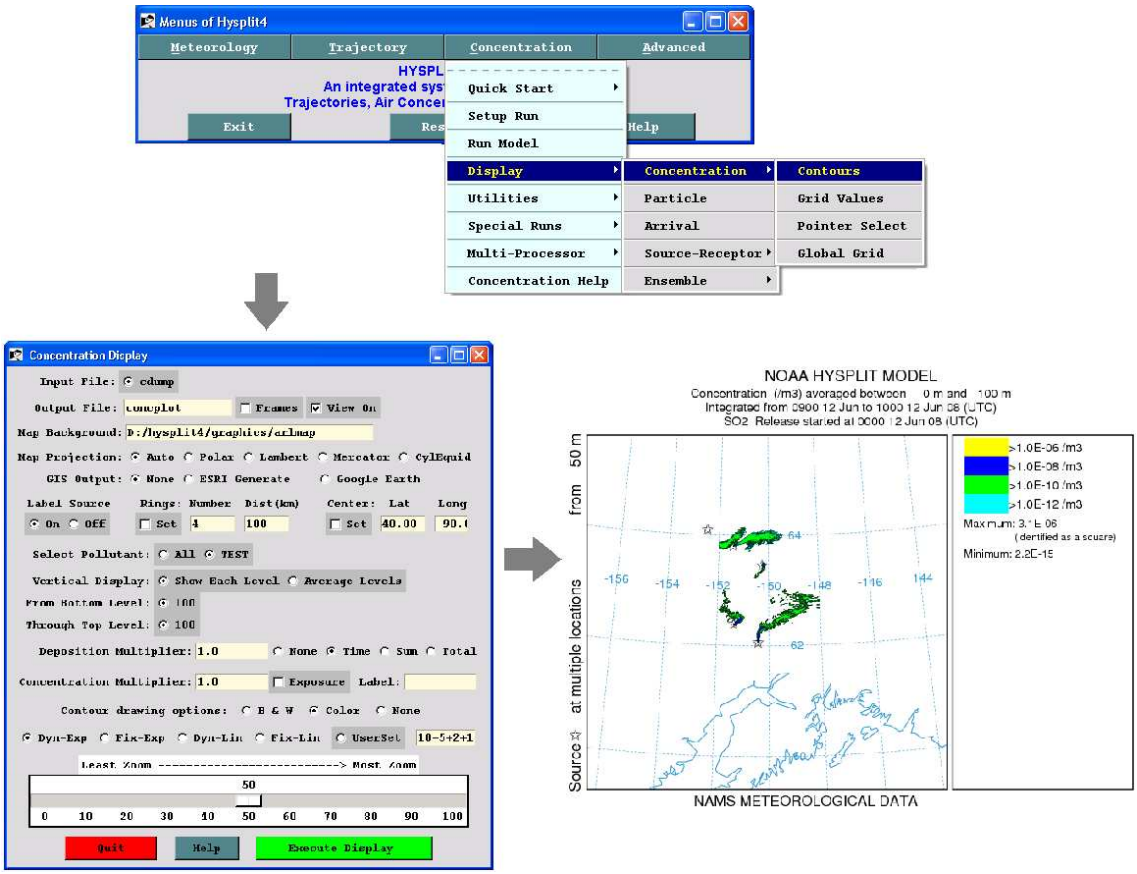

Figura 4.2: Conversión de la salida de una simulación en un fichero postscript

4.3. Estudio del rendimiento de `hymodelmb`

Con el fin de medir el rendimiento de la implementación de la solución propuesta (`hymodelmb`) se decidió determinar la aceleración (véase la Fórmula 1.1 en el capítulo 1), así como el porcentaje de tiempo reducido por `hymodelmb` respecto al tiempo de ejecución de `hymodelm`. Para ello, se añadió al código correspondiente una serie de temporizadores (usando la función `SYSTEM_CLOCK` de FORTRAN) para medir los tiempos requeridos.

El experimento consistió en la realización de un total de diez simulaciones usando distintos ficheros de entrada, de configuración y de control, y ejecutando con 1, 2 ,4, 8 y 16 procesadores. Para la ejecución se utilizaron los nodos: *node17*, *node18*,... y *node24* del cluster *Tegasaste*, cuyas características se especificaron ya en la sección anterior. Como entrada se utilizaron varios ficheros con dominio meteorológico de Alaska y Hawaii, obtenidos desde el ftp de ARL [3]. La lista de estos ficheros se muestra en la segunda columna de la Tabla 4.2. Como ficheros de configuración y control se usaron los indicados en la tercera y cuarta columna, respectivamente, de la tabla anteriormente citada.

Nótese que el fichero de configuración usado en cada simulación es prácticamente el mismo que se indica en los Apéndices A.3 y A.4, con la salvedad de que se cambió el valor de `numpar` al valor indicado en la columna "Número de partículas". De igual manera, el fichero de control utilizado en cada simulación es idéntico al de los Apéndices B.1 y B.2, con la excepción de que sólo varía el nombre de fichero de entrada por el de la columna "Fichero de entrada". La relación de ficheros de entrada, control y configuración asociada a cada simulación se especifican en la Tabla 4.2.

Para cada simulación x se adjuntan sus correspondientes gráficas Sx y Tx. En las gráficas Sx, con $x = 1, 2.,10$ se representa la aceleración asociada a las ejecuciones de `hymodelm` y de `hymodelmb`. En las gráficas Tx, con $x = 1, 2.,10$ se representa el porcentaje de tiempo reducido al ejecutar `hymodelmb` respecto al tiempo de ejecución de `hymodelm`.

En las Figuras 4.3 a la 4.5 se presentan las gráficas Sx y Tx con $x = 1, 2, 3$, asociadas a las simulaciones realizadas tomando como entrada ficheros con datos meteorológicos de Hawaii. Debido a que estos ficheros son de pequeño tamaño, el tiempo de lectura invertido es mucho menor que en los ficheros con meteorología de Alaska. Por ello, sólo se aprecia ligeramente la mejora de tiempos introducida con la solución propuesta, manifestándose

Número de simulación	**Fichero de entrada**	**Fichero de control**	**Fichero de configuración**	**Número de partículas**
1	20080711.namsa.HI	CtrlHI2	Config2	90000
2	20080612.namsa.HI	CtrlHI2	Config1	90000
3	20080711.namsa.HI	CtrlHI2	Config1	75000
4	20081212.namsa.AK	CtrlAK2	Config1	15000
5	20080711.namsa.AK	CtrlAK2	Config2	90000
6	20080612.namsa.AK	CtrlAK2	Config1	75000
7	20090101.namsa.AK	CtrlAK2	Config2	75000
8	20090101.namsa.AK	CtrlAK2	Config1	15000

Tabla 4.2: Relación de ficheros de entrada, configuración y control de cada simulación

sólo a partir de las ejecuciones con ocho o más procesadores. Esto se observa claramente en las gráficas $S1$, $S2$ y $S4$, donde la aceleración asociada a `hymodelmb` es superior a la de `hymodelm` (independientemente del fichero de entrada y de configuración) a partir de las ejecuciones con ocho procesadores. Además, en las gráficas $S2$ y $S3$ se aprecia que la aceleración asociada a `hymodelmb` presenta un crecimiento más rápido que la de `hymodelm` a medida que aumenta el número de procesadores.

En las gráficas $T1$, $T2$ y $T3$ se presenta el porcentaje de tiempo reducido por `hymodelmb` respecto al tiempo de ejecución de `hymodelm`. Se observa que se obtiene una mayor disminución cuando se realiza la simulación con ocho o más procesadores, llegando incluso a alcanzar un 7,10 % de reducción del tiempo de ejecución cuando se ejecuta sobre 16 procesadores, como se aprecia en la gráfica $T2$ de la Figura 4.4.

En las Figuras 4.6 a la 4.10 se indican las gráficas Sx y Tx con $x = 4, 5, \ldots, 8$ asociadas a las simulaciones realizadas con ficheros de entrada cuyo dominio meteorológico corresponde a Alaska. Estos ficheros son de mayor tamaño que los que contienen datos meteorológicos de Hawaii, por lo que el tiempo de lectura invertido en los mismos es superior. Por ello, son en estas ejecuciones donde se aprecia la considerable mejora introducida por la implementación de la solución propuesta. Observando las gráficas de la aceleración (veánse las gráficas $S4 - S8$) se puede ver que en todos los casos la aceleración asociada a `hymodelmb` es superior a la de `hymodelm`, especialmente en las gráficas $S4$, $S6$ y $S8$. Además, la aceleración de `hymodelmb` se incrementa con mayor rapidez desde las ejecuciones con dos procesadores y tiene tendencia a aumentar a medida que las ejecuciones se realizan sobre un mayor número de procesadores. Esta tendencia es deseable, ya que normalmente el número de procesadores involucrados en las simulaciones suele ser superior a 16.

En las gráficas $T4$-$T8$ el porcentaje de tiempo de ejecución reducido es, en todas las simulaciones realizadas con 16 procesadores, superior al 12 %, llegando a alcanzar incluso un valor máximo de 27.35 % en la gráfica $T8$. En la gráfica $T4$ de la Figura 4.6 se observa que el porcentaje de tiempo de ejecución reducido por `hymodelmb` respecto al obtenido con `hymodelm` se sitúa en torno al 20 % en todas las ejecuciones con dos o más procesadores y en general, aumentando a medida que crece el número de procesadores involucrados en la simulación. Lo mismo ocurre en la gráfica $T8$ de la Figura 4.10. Esto refleja un ahorro importante en tiempo de ejecución, que tiende a incrementarse con el número de procesadores.

En general, a partir de los resultados obtenidos, se observa que el rendimiento de `hymodelmb` es mejor que el de `hymodelm`, especialmente a medida que el número de procesadores aumenta, presentando una escalabilidad adecuada. Esto se aprecia en una amplia y diversa colección de simulaciones que emplean diferentes ficheros de entrada, configuración y control. Por otro lado, cabe destacar que normalmente se realizan simulaciones con dominios meteorológicos más amplios que el de Hawaii o Alaska, invirtiendo, por tanto, más tiempo en la lectura y empleando habitualmente un mayor número de procesadores. Teniendo en cuenta esto, se estima que la reducción del tiempo de ejecución proporcionada por `hymodelmb` es más apreciable y mucho mayor en dichos casos, solventando satisfactoriamente el problema de escalabilidad que presenta `hymodelm`.

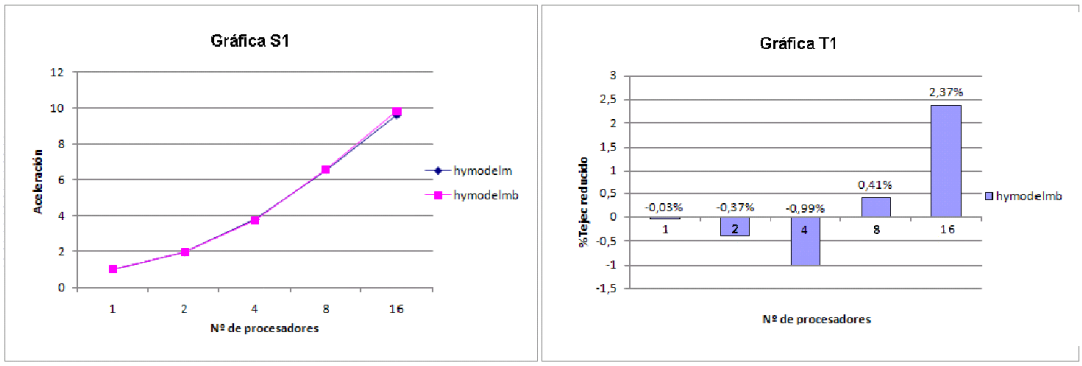

Figura 4.3: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 1)

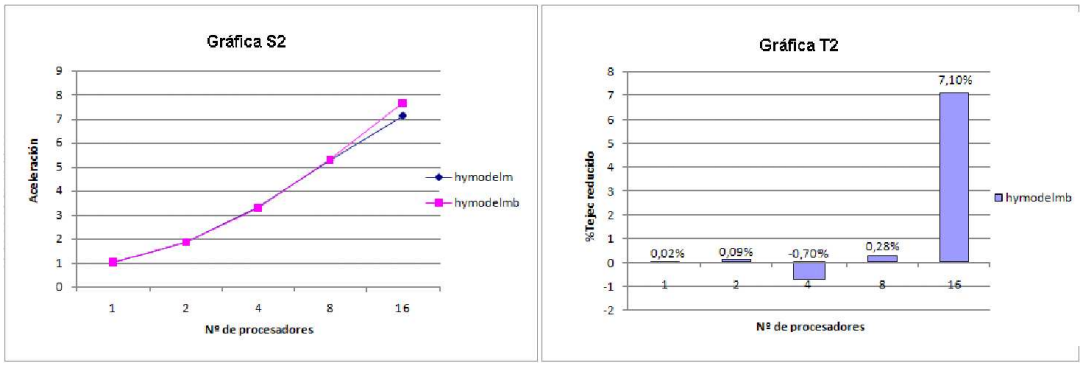

Figura 4.4: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 2)

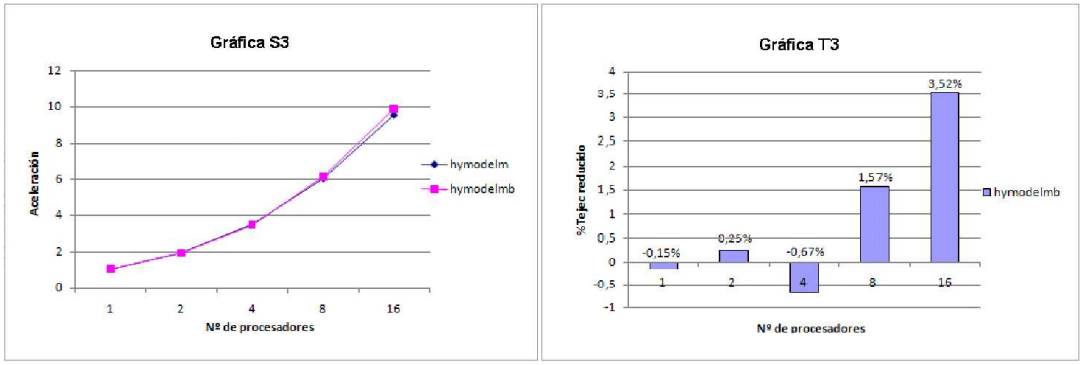

Figura 4.5: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 3)

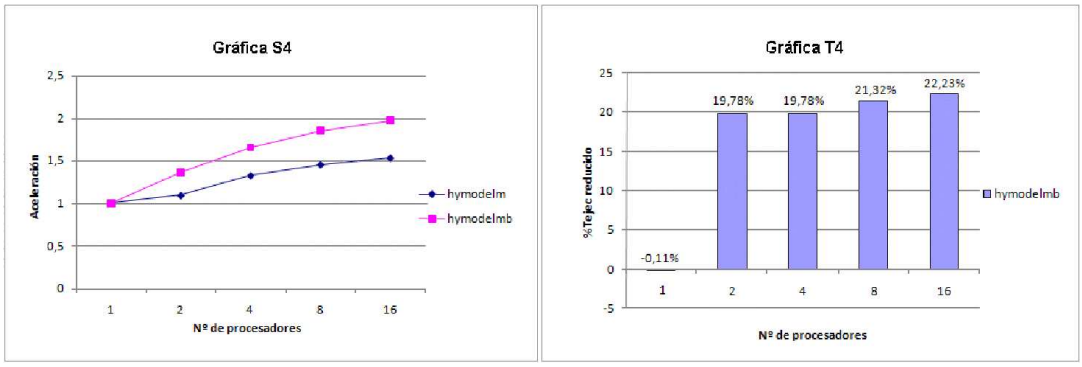

Figura 4.6: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 4)

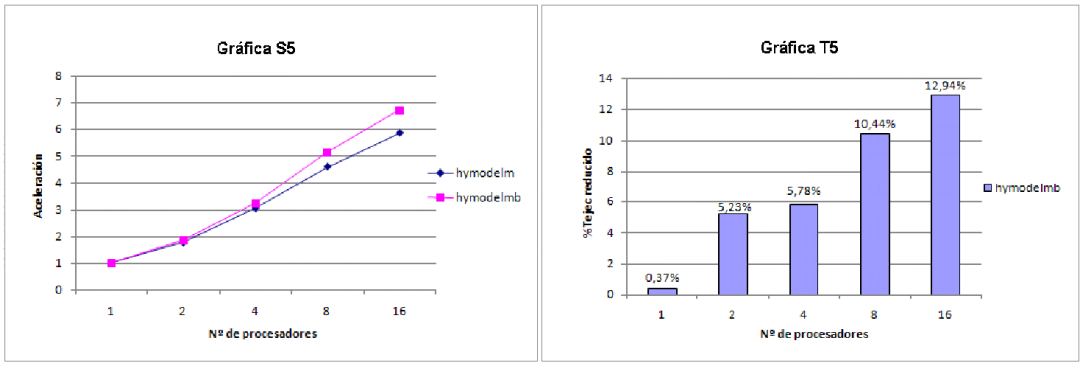

Figura 4.7: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 5)

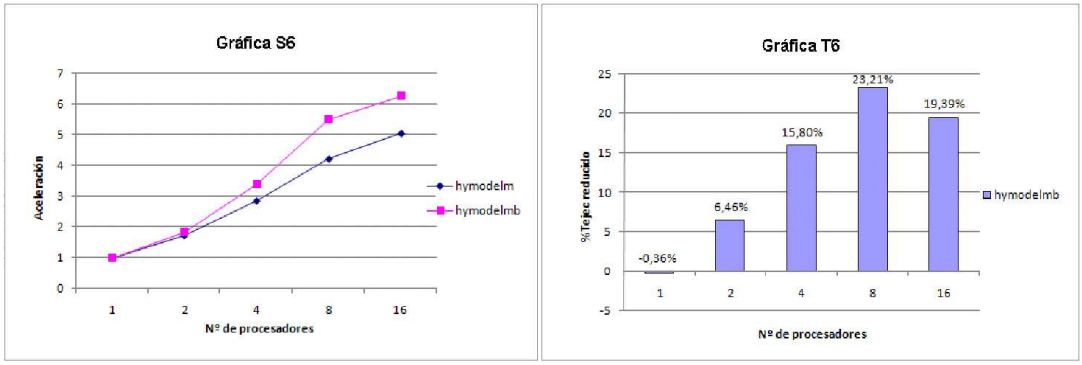

Figura 4.8: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 6)

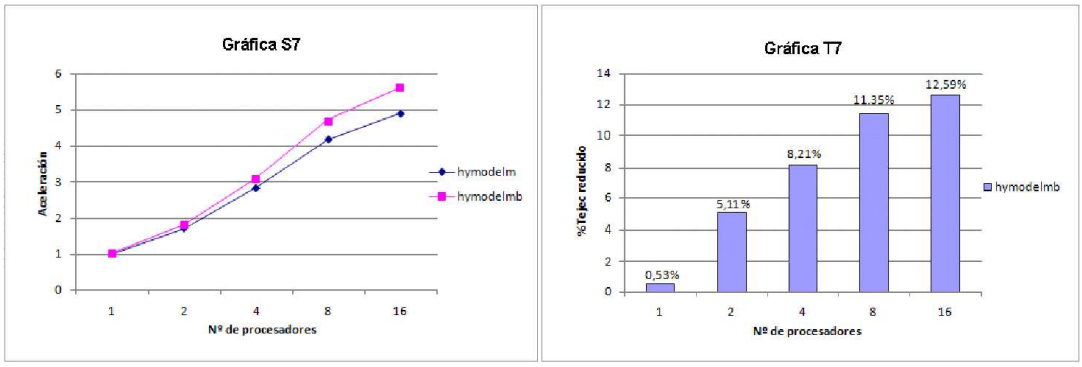

Figura 4.9: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 7)

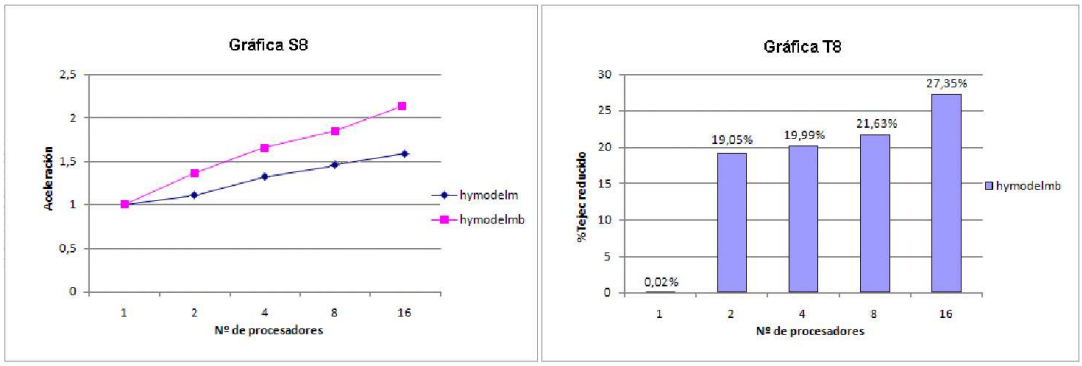

Figura 4.10: Gráficas del rendimiento de `hymodelm` y `hymodelmb` (simulación 8)

Capítulo 5

Conclusiones

HYSPLIT es un software que permite determinar la emisión, transporte, dispersión y deposición de partículas atmosféricas y contaminantes. Ha sido desarrollado por el ARL de la NOAA y la Agencia de Meteorología de Australia. Está escrito en FORTRAN y cuenta con versiones secuenciales y paralelas. Puede ejecutarse sobre distintas plataformas y tiene múltiples aplicaciones en el campo de la meteorología, entre las que destacan la predicción del transporte y dispersión de partículas contaminantes, cenizas volcánicas, incendios forestales, tormentas de polvo, polen, etc.

Las motivaciones para la realización de este proyecto surgieron a raíz del problema de escalabilidad que presenta la versión paralela del modelo de concentraciones (`hymodelm`) de HYSPLIT. Se observa que a medida que el número de procesadores con que se ejecuta dicho programa aumenta, el tiempo de ejecución se incrementa más de lo esperado. Debido a que las ejecuciones se realizan sobre un cluster con NFS, se sospechó que una de las principales causas del bajo rendimiento podía ser que la lectura paralela de los datos de entrada no está implementada de forma adecuada. Teniendo en cuenta el amplio número de aplicaciones que tiene HYSPLIT, principalmente en el campo meteorológico, como herramienta de pronóstico, se hizo realmente necesario la solución del problema citado. Por esta razón se plantearon los objetivos a alcanzar con el desarrollo del presente proyecto: estudiar el comportamiento de `hymodelm` con el fin de determinar cuáles son las causas por las que el programa no escala adecuadamente, proponer una solución al mismo y realizar la correspondiente implementación en código, creando una nueva versión del modelo.

Para iniciar el desarrollo del proyecto, se contó con la versión 4.8 del código fuente de HYSPLIT y como única documentación, la Guía de Usuario de HYSPLIT. Se comenzó por analizar el código fuente, con el fin de seleccionar las zonas de código asociadas a `hymodelm`. Una vez hecho esto, se consiguió determinar el diagrama de flujo del programa, así como dónde y de qué forma se realiza la lectura de los datos de entrada. Se descubrió que la lectura se hace bajo demanda y que todos los procesos leen los mismos datos de entrada y de forma simultánea. Esta fase de análisis fue especialmente difícil debido a la falta de documentación y al gran tamaño y complejidad del código, teniendo que hacer uso de algunas herramientas como `gprof` para la realización de un perfil de ejecución. Este *profile* sirvió además, para estimar el porcentaje de tiempo que ocupa la rutina de lectura respecto

al tiempo de ejecución total en la ejecución secuencial del modelo de concentraciones, que no resultó ser un valor tan alto como se pensaba. Por ello, se decidió estudiar aquellos factores que pueden influir en el aumento del tiempo de lectura. Se descubrió que la variable `mgmin` es la que más influye y se aconsejó cambiar su valor con el fin de reproducir las circunstancias en las normalmente se ejecuta el programa. De esta forma se hizo un experimento para determinar si el número de partículas emitidas influye también en el incremento del tiempo de lectura, tomando como entrada diversos ficheros de meteorología y realizando ejecuciones en secuencial. Se obtuvo que el tiempo de lectura no aumentaba significativamente.

Posteriormente, se decidió estudiar el comportamiento de la lectura en paralelo realizando un programa en FORTRAN 90 que reprodujera la obtención de datos de entrada que se realizaba en `hymodelm`. Se realizaron varios experimentos, variando el número de procesadores y tomando como entrada ficheros de diversos tamaños. Con los resultados obtenidos, se concluyó que la lectura en paralelo de un fichero es realmente ineficiente, especialmente cuando se realiza con un número de procesadores mayor o igual a cuatro, llegando a ralentizarse la lectura hasta veinte veces más que la realizada en secuencial. La razón de esto es que los procesos compiten por el acceso a un mismo recurso, lo que hace que a medida que aumente el número de procesadores implicados, la lectura tiende a secuencializarse, con el correspondiente incremento del tiempo empleado en ello. Una vez determinada la causa del problema, se pensó que una solución al mismo era la lectura del fichero de entrada por un único proceso, y el posterior envío de los datos leídos al resto de procesadores (*broadcast*). Para comprobar la validez de esta propuesta, se creó un nuevo programa en FORTRAN 90 que lo implementara. Con él se realizaron varios experimentos ejecutando con 1, 2, 4, 8 y 16 procesadores, utilizando diferentes tamaños de *buffers* de envío y tomando como entrada ficheros de diversos tamaños. En todos los casos se obtuvieron resultados análogos y muy favorables que permitían realizar la lectura en un tiempo razonable, aún ejecutando sobre un número elevado de procesadores. Por ello, se propuso como solución al problema la obtención de datos de entrada mediante la lectura secuencial y *broadcast*.

La implementación de la solución propuesta requirió seguir una serie de pasos. Primero se determinaron las variables que almacenaban los datos leídos. En base al número y tipo de los datos a enviar, se eligió el tipo de *broadcast* más adecuado y se comenzó con la codificación. Se encontraron algunas dificultades a la hora de incorporar la nueva versión (denominada `hymodelmb`) en HYSPLIT, que se solventaron creando una nueva rutina `ADVPNTBCAST` en el fichero `hymodelc.F`, a continuación del *main*. Esta rutina es la misma que `ADVPNT` salvo que la obtención de los datos de entrada se implementa mediante una lectura en secuencial y *broadcast*. Para la compilación de `hymodelmb` se definió en tiempo de compilación una nueva constante para obtener el correspondiente ejecutable, modificando el fichero *makefile*. Una vez finalizada la implementación de la solución, se procedió a validarla. Se realizaron varias simulaciones ejecutando `hymodelm` y `hymodelmb` y se compararon sus respectivas salidas, que resultaron ser idénticas en todos los casos. Con esto se comprobó que la implementación de `hymodelmb` es correcta.

El siguiente paso consistió en comparar el rendimiento de ambos programas. Se decidió determinar la aceleración, así como el porcentaje de tiempo reducido por `hymodelmb`

respecto al tiempo de ejecución de `hymodelm`. Para ello, se añadieron al código una serie de temporizadores con el fin de medir los tiempos requeridos. Se realizaron una serie de simulaciones usando distintos ficheros de entrada, de configuración y de control, y ejecutando sobre diferente número de procesadores. Los resultados obtenidos en la mayoría de los casos fueron altamente favorables. En las representaciones gráficas realizadas de la aceleración se muestra que `hymodelmb` presenta un mejor rendimiento que `hymodelm` en prácticamente todos los casos, especialmente a medida que aumenta el número de procesadores con que se ejecuta. Asimismo, el porcentaje de tiempo de ejecución reducido por `hymodelmb` respecto al tiempo de ejecución de `hymodelm` tiende a aumentar con el número de procesadores involucrados en la simulación, llegando a alcanzar incluso un 27,35 %, como se aprecia en la gráfica $T10$ de la Figura **??**.

Por tanto, se ha logrado desarrollar de forma exitosa una nueva versión de `hymodelm` que permite obtener los mismos resultados en un tiempo menor de ejecución y presentando una mejor escalabilidad, solventando así el problema que motivó el desarrollo del presente proyecto. A modo de resumen, se citan a continuación las principales conclusiones obtenidas en este trabajo:

- La versión paralela del modelo de concentraciones (`hymodelm`) presenta problemas de escalabilidad cuando se ejecuta sobre clusters con NFS, derivados de la inadecuada gestión de la lectura de datos de entrada.
- Los datos meteorológicos que recibe como entrada `hymodelm` son solicitados bajo demanda durante la ejecución en paralelo y son leídos de forma simultánea por todos los procesos.
- Cuando todos los procesos intentan obtener los datos de entrada del fichero, entran en competición por dicho recurso, aumentando así, el tiempo invertido en la lectura. Este tiempo aumenta con el número de procesadores involucrados en la ejecución, provocando que `hymodelm` tienda a secuencializarse.
- Como todos los procesos necesitan leer exactamente los mismos datos de entrada, basta con que sea un único proceso el encargado de realizar la lectura, enviando posteriormente la información al resto de procesos (*broadcast*).
- La implementación de la obtención de los datos de entrada mediante una lectura secuencial y *broadcast* permite que se disminuya considerablemente el tiempo de ejecución con respecto a la lectura paralela, especialmente en las ejecuciones con ocho o más procesadores.

Cabe destacar que la reducción del tiempo de ejecución que ofrece `hymodelmb` respecto a `hymodelm` es apreciable cuanto mayor sea la cantidad de tiempo invertido en la lectura de los datos de entrada respecto al de ejecución, lo que puede variar en función del contenido del fichero de entrada y de su tamaño, así como de los ficheros de configuración y de control utilizado en la simulación. Por otro lado, también es importante hacer especial hincapié en que normalmente se realizan simulaciones con dominios meteorológicos más amplios que los aquí presentados (como por ejemplo Estados Unidos o incluso dominios

globales) tomando como entrada ficheros con datos meteorológicos de mayor tamaño, con lo que el tiempo invertido en la lectura de los mismos es mayor. Además, el número de procesadores involucrados en dichas simulaciones supera la treintena (en el caso de la NOAA). Por tanto, tras el estudio realizado se estima que la reducción del tiempo de ejecución proporcionada por `hymodelmb` será más apreciable y mucho mayor en dichos casos, pues la mejora es más visible a medida que el fichero de entrada y el número de procesadores es mayor. No cabe duda de que esto proporcionará un gran ahorro de tiempo de ejecución en las simulaciones del modelo de concentraciones en paralelo, con las consecuentes ventajas que esto conllevará para todos aquellos centros meteorológicos, organismos y universidades que actualmente utilizan el modelo para la realización de pronósticos e investigaciones.

Como posibles trabajos futuros, podrían realizarse ejecuciones de `hymodelmb` con mayor número de procesadores y dominios meteorológicos más amplios que los utilizados en este proyecto, con el fin de estudiar su rendimiento en dichos casos (el cual se estima bastante favorable). Asimismo, la solución propuesta en este trabajo para la versión paralela del modelo de concentraciones de HYSPLIT podría aplicarse a la versión paralela del modelo de trayectorias, pues ambas hacen uso de la misma rutina para la lectura de los datos de entrada. Tan sólo se necesita incluir la rutina `ADVPNTBCAST` en el fichero `hymodelt.F` (fichero *main* del modelo de trayectorias) y colocar las directivas del preprocesador correspondientes para la compilación de una nueva versión, al igual que se hizo con el modelo de concentraciones. De esta forma, las conclusiones obtenidas en el presente proyecto acerca de la lectura de los datos de entrada sirven como base para futuras optimizaciones de HYSPLIT aplicables a otros modelos que implementa.

Agradecimientos

Quisiera agradecer a todos los organismos e instituciones que han hecho posible el presente proyecto:

Al Centro de Investigación Atmosférica de Izaña (CIAI) de la Agencia Estatal de Meteorología (AEMET) y a la Universidad de La Laguna, en España, por el convenio de colaboración establecido entre ambos, a través del cual se llevó a cabo este proyecto. Asimismo, agradecer los recursos proporcionados por dichos organismos para el desarrollo del trabajo.

Al ARL (*Air Resources Laboratory*) de la NOAA, en Estados Unidos, por la propuesta de este trabajo a petición del CIAI-AEMET, así como por el código fuente de HYSPLIT cedido para el estudio.

Al Gobierno de Canarias, por la financiación de la beca de colaboración y a Fundación Empresa Universidad (FEU) por la gestión de la misma.

A Ariel Stein, investigador *senior* asociado al ARL de la NOAA y a Carlos Marrero, meteorólogo del CIAI-AEMET, por su colaboración en el presente proyecto.

Apéndice A

Ficheros de control

A.1. Fichero de control CtrlHI1 (Hawaii)

```
00 00 00 00
5
18.453373 -159.263757    50.0 267.0
20.381941 -158.356905    50.0 491.0
19.085742 -157.492970    50.0 126.0
18.042345 -159.411533    50.0 491.0
17.804243 -158.437750    50.0 68.0
24
0
10000.0
1
../working/
20080312_hysplit.t00z.namsa.HI
1
SO2
1.0
24.0
00 00 00 00 00
1
18.0 -158.0
0.005 0.005
4.0 4.0
./
5Sources_em_curr
1
100
00 00 00 00 00
00 00 00 00 00
00 1 00
1
0.0 0.0 0.0
0.003 0.0 0.0 0.0 0.0
0.0 0.0 0.0
0.0
0.0
```

A.2. Fichero de control CtrlAK1 (Alaska)

```
00 00 00 00
5
63.453373 -150.263757    50.0 267.0
62.381941 -151.356905    50.0 491.0
64.085742 -152.492970    50.0 126.0
62.042345 -150.411533    50.0 491.0
63.804243 -151.437750    50.0 68.0
24
0
10000.0
1
../working/
20080312_hysplit.t00z.namsa.AK
1
SO2
1.0
24.0
00 00 00 00 00
1
63.0 -150.0
0.005 0.005
4.0 4.0
./
5Sources_em_curr
1
100
00 00 00 00 00
00 00 00 00 00
00 1 00
1
0.0 0.0 0.0
0.003 0.0 0.0 0.0 0.0
0.0 0.0 0.0
0.0
0.0
```

A.3. Fichero de control CtrlHI2 (Hawaii)

```
00 00 00 00
5
18.453373 -159.263757    50.0 267.0
18.501941 -157.906905    50.0 491.0
19.085742 -157.492970    50.0 126.0
18.042345 -159.411533    50.0 491.0
17.804243 -158.437750    50.0 68.0
24
0
10000.0
1
../working/
20080711_hysplit.t00z.namsa.HI
1
SO2
1.0
24.0
00 00 00 00 00
1
18.0 -158.0
0.005 0.005
4.0 4.0
./
5Sources_em_curr
1
100
00 00 00 00 00
00 00 00 00 00
00 1 00
1
0.0 0.0 0.0
0.003 0.0 0.0 0.0 0.0
0.0 0.0 0.0
0.0
0.0
```

A.4. Fichero de control CtrlAK2 (Alaska)

```
00 00 00 00
5
63.453373 -150.263757    50.0 267.0
62.381941 -151.356905    50.0 491.0
63.105742 -149.122970    50.0 126.0
62.042345 -150.411533    50.0 491.0
63.804243 -151.437750    50.0 68.0
24
0
10000.0
1
../working/
20090101_hysplit.t00z.namsa.AK
1
SO2
1.0
24.0
00 00 00 00 00
1
63.0 -150.0
0.005 0.005
4.0 4.0
./
5Sources_em_curr
1
100
00 00 00 00 00
00 00 00 00 00
00 1 00
1
0.0 0.0 0.0
0.003 0.0 0.0 0.0 0.0
0.0 0.0 0.0
0.0
0.0
```

Apéndice B

Ficheros de configuración

B.1. Fichero de configuración Config1

```
&SETUP
 delt = 12.0,
 initd = 4,
 kpuff = 0,
 khmax = 240,
 frmr = 0.01,
 numpar =75000,
 qcycle = 1.0,
 efile = '',
 isot = 0,
 ninit = 1,
 ndump = 1,
 ncycl = 1,
 pinpf = 'PARINIT',
 poutf = 'PARDUMP',
 mgmin = 10000,
 kmsl = 0,
 maxpar = 30000000,
 cpack = 1,
 cmass = 0,
 dxf = 1.0,
 dyf = 1.0,
 dzf = 0.005,
 ichem = 3,
 p10f = 1.0,
 /
```

B.2. Fichero de configuración Config2

```
&SETUP
tratio = 0.75,
initd = 0,
kpuff = 0,
khmax = 9999,
numpar = 10000,
qcycle = 0.0,
efile = '',
isot = 0,
tkerd = 0.18,
tkern = 0.18,
ninit = 1,
ndump = 12,
ncycl = 0,
pinpf = 'PARINIT',
poutf = 'PARDUMP',
mgmin = 1000,
kmsl = 0,
maxpar = 1000000,
cpack = 1,
cmass = 0,
dxf = 1.0,
dyf = 1.0,
dzf = 0.01,
ichem = 0,
/
```

B.3. Fichero de configuración Config3

```
&SETUP
tratio = 0.75,
initd = 0,
kpuff = 0,
khmax = 9999,
numpar = 10000,
qcycle = 0.0,
efile = '',
isot = 0,
tkerd = 0.18,
tkern = 0.18,
ninit = 1,
ndump = 12,
ncycl = 0,
pinpf = 'PARINIT',
poutf = 'PARDUMP',
mgmin = 10,
kmsl = 0,
maxpar = 1000000,
cpack = 1,
cmass = 0,
dxf = 1.0,
dyf = 1.0,
dzf = 0.01,
ichem = 0,
/
```

Apéndice C

Código fuente

C.1. Programa de lectura paralela

```
!Lectura paralela de un fichero por N procesadores
PROGRAM lecturaParalela
    IMPLICIT NONE
    INCLUDE 'mpif.h'
    CHARACTER :: dato
    INTEGER ::count0,count_rate,count_max,job_id,num_job,ierr
    REAL*8 :: t1,t2,t3,t4,t5,t6,taux

    !Momento inicial de ejecución
    CALL system_clock(count0,count_rate,count_max)
    t1 = dble(count0)

    !Inicializar MPI
    CALL MPI_INIT(ierr)
    CALL MPI_COMM_SIZE (MPI_COMM_WORLD,num_job,ierr)
    CALL MPI_COMM_RANK (MPI_COMM_WORLD,job_id,ierr)

    !Tiempo de comienzo de lectura del proceso N
    CALL system_clock(count0,count_rate,count_max)
    t4 = dble(count0)

    !Lectura del fichero por cada proceso
    OPEN (UNIT=10, FILE='entrada.dat',STATUS="OLD")
    finFich = EOF(10)
    DO WHILE (.NOT.EOF(10))
      READ(10,*) dato
    END DO
    CLOSE(10)

    !Tiempo de finalización de lectura del proceso
    CALL system_clock(count0,count_rate,count_max)
    t5 = dble(count0)
    taux =(t5/dble(count_rate))
    write (*,*)  'Proceso ',job_id,' acaba en t=',taux
    t6 = ((t5-t4)/dble(count_rate))
```

```
write (*,*) 'TIEMPO TOTAL DE LECTURA (Proceso: ',job_id,'): ',t6

!Finalizar MPI
CALL MPI_FINALIZE(ierr)

!Tiempo de ejecución total
CALL system_clock(count0,count_rate,count_max)
t2 = dble(count0)
taux =(t2/dble(count_rate))
write (*,*)  'Proceso ',job_id,' termina en t=',taux
t3 = ((t2-t1)/dble(count_rate))
write (*,*)  'TIEMPO TOTAL DE EJECUCIÓN (Proceso: ',job_id,')=',t3
STOP
END PROGRAM
```

C.2. Programa de lectura secuencial y *broadcast*

```
!Lectura secuencial de un fichero y broadcast
PROGRAM broadcast
   IMPLICIT NONE
   INCLUDE "mpif.h"
   CHARACTER*32 dato
   INTEGER ::i,count0,count_rate,count_max,job_id,num_job,ierr
   REAL*8 :: t1,t2,t3,t4,t5,t6,taux
   LOGICAL :: finFich
   CHARACTER*65536 buff
   INTEGER :: tamBuff,numBytes
   REAL*8 :: totalBytes,bytesLeidos
   PARAMETER (tamBuff=65536)
   PARAMETER (totalBytes=2303090817.0)

   bytesLeidos = 0
   numBytes = 1

   !Momento inicial de ejecución
   CALL system_clock(count0,count_rate,count_max)
   t1 = dble(count0)

   !Inicializar MPI
   CALL MPI_INIT(ierr)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD,job_id,ierr)
   CALL system_clock(count0,count_rate,count_max)
   t4 = dble(count0)

   !Proceso maestro lee el fichero y envía los datos a los demás procesos
   IF (job_id == 0) THEN
     OPEN (UNIT=10, FILE='entrada.txt',STATUS="OLD")
     finFich = EOF(10)
     DO WHILE (.NOT.finFich)
       IF (numBytes .LT. tamBuff) THEN
         READ(10,'(A32)'), dato
         buff(numBytes:numBytes+32) = dato
         numBytes = numBytes + 32
         bytesLeidos = bytesLeidos + 32
         finFich = EOF(10)
       ELSE
         CALL MPI_BCAST(buff,tamBuff,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)
         numBytes = 1
       END IF
     END DO
     IF (numBytes .GT. 1) THEN
       CALL MPI_BCAST(buff,tamBuff,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)
       bytesLeidos = bytesLeidos + numBytes
     END IF
     CLOSE(10)

   !No es el proceso maestro (recibe datos leídos por el proceso maestro)
   ELSE
     DO WHILE (bytesLeidos .LT. totalBytes)
```

```
        bytesLeidos = bytesLeidos + tamBuff
        CALL MPI_BCAST(buff,tamBuff,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)
      END DO
    ENDIF

    !Tiempo de finalización de lectura del proceso N
    CALL system_clock(count0,count_rate,count_max)
    t5 = dble(count0)
    taux =(t5/dble(count_rate))
    write (*,*)  'Proceso ',job_id,' acaba en t=',taux
    t6 = ((t5-t4)/dble(count_rate))
    write (*,*) 'TIEMPO TOTAL DE LECTURA (Proceso: ',job_id,'): ',t6

    !Finalizar MPI
    CALL MPI_FINALIZE(ierr)

    !Tiempo de ejecución total
    CALL system_clock(count0,count_rate,count_max)
    t2 = dble(count0)
    t3 = ((t2-t1)/dble(count_rate))
    write (*,*)  'TIEMPO TOTAL DE EJECUCION: ',t3
    STOP
END
```

C.3. Programa de lectura secuencial y *broadcast* con acceso a datos

```
!Lectura secuencial y broadcast, realizando
!una operación de acceso a los datos leídos
PROGRAM broadcastOp
   IMPLICIT NONE
   INCLUDE "mpif.h"
   CHARACTER*32 dato
   INTEGER ::i,count0,count_rate,count_max,job_id,num_job,ierr
   REAL*8 :: t1,t2,t3,t4,t5,t6,taux
   LOGICAL :: finFich
   CHARACTER*65536 buff
   CHARACTER*2 :: dat
   INTEGER :: tamBuff,numBytes
   REAL*8 :: totalBytes,bytesLeidos
   PARAMETER (tamBuff=65536)
   PARAMETER (totalBytes=2303090817.0)

   bytesLeidos = 0
   i = 1
   numBytes = 1

   !Momento inicial de ejecución
   CALL system_clock(count0,count_rate,count_max)
   t1 = dble(count0)

   !Inicializar MPI
   CALL MPI_INIT(ierr)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, job_id, ierr)
   CALL system_clock(count0,count_rate,count_max)
   t4 = dble(count0)

   !Proceso maestro lee el fichero y envía los datos a los demás procesos
   IF (job_id == 0) THEN
     OPEN (UNIT=10, FILE='entrada.txt',STATUS="OLD")
     finFich = EOF(10)
     DO WHILE (.NOT.finFich)
       IF (numBytes .LT. tamBuff) THEN
         READ(10,'(A32)'), dato
         buff(numBytes:numBytes+32) = dato
         !Acceso al dato por el proceso maestro
         dato = buff(numBytes:numBytes+32)
         numBytes = numBytes + 32
         bytesLeidos = bytesLeidos + 32
         finFich = EOF(10)
       ELSE
         CALL MPI_BCAST(buff,tamBuff,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)
         numBytes = 1
       END IF
     END DO
     IF (numBytes .GT. 1) THEN
       CALL MPI_BCAST(buff,tamBuff,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)
```

```
          !Acceso al dato por el proceso maestro
          dato = buff(numBytes:numBytes+32)
          bytesLeidos = bytesLeidos + numBytes
        END IF
        CLOSE(10)

      !No es el proceso maestro (recibe datos leídos por el proceso maestro)
      ELSE
        DO WHILE (bytesLeidos .LT. totalBytes)
          bytesLeidos = bytesLeidos + tamBuff
          CALL MPI_BCAST(buff,tamBuff,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)
          !Acceso a los datos recibidos
          DO WHILE (i .LT. tamBuff)
            dat = buff(i:i+1)
            i = i + 1
          END DO
          i = 1
        END DO
      ENDIF

      !Tiempo de finalización de lectura del proceso N
      CALL system_clock(count0,count_rate,count_max)
      t5 = dble(count0)
      taux =(t5/dble(count_rate))
      write (*,*)  'Proceso ',job_id,' acaba en t=',taux
      t6 = ((t5-t4)/dble(count_rate))
      write (*,*) 'TIEMPO TOTAL DE LECTURA (Proceso: ',job_id,'): ',t6

      !Finalizar MPI
      CALL MPI_FINALIZE(ierr)

      !Tiempo de ejecución total
      CALL system_clock(count0,count_rate,count_max)
      t2 = dble(count0)
      t3 = ((t2-t1)/dble(count_rate))
      write (*,*)  'TIEMPO TOTAL DE EJECUCION: ',t3
      STOP
  END
```

C.4. Lectura de datos secuencial y *broadcast* (rutina ADVPNTBCAST)

```
 (...)
!------------------------------------------------------------------------------
! test if new data required at the last time (k1)
!------------------------------------------------------------------------------
  IF(NEW1)THEN

!    load meteorological data according to positioning
!    reads data for NZS levels

!  Root job reads input data
   IF (job_id == 0) THEN
     CALL METINP(BACK,KG,KT1,KUNIT1,KREC1,LX1(KG),LY1(KG),NXS(KG),NYS(KG),     &
        NZS,FTIME(KG,K1),KEND1,FHOUR(K1,KG),ZT(:,:,KG),DS(:,:,K1,KG),          &
        PO(:,:,K1,KG),TO(:,:,K1,KG),UO(:,:,K1,KG),VO(:,:,K1,KG),               &
        UF(:,:,K1,KG),VF(:,:,K1,KG),HF(:,:,K1,KG),RT(:,:,K1,KG),ZI(:,:,K1,KG), &
        U(:,:,:,K1,KG),V(:,:,:,K1,KG),W(:,:,:,K1,KG),A(:,:,:,K1,KG),           &
        Q(:,:,:,K1,KG),P(:,:,:,K1,KG),E(:,:,:,K1,KG),H(:,:,:,K1,KG),           &
   ENDIF

!     Root job sends input data / jobs 1..num_job-1 receive input data
!     Send/receive argument mc
      CALL MPI_BCAST(FTIME(KG,K1),1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

!     Send/receive argument ifhr
      CALL MPI_BCAST(FHOUR(K1,KG),1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

!     Send/receive input data

!     Size of array variable zt
      SIZEA = NXT*NYT*NGRD
      CALL MPI_BCAST(ZT(:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)

!     Size of array variables: ds,p0,u0,v0,t0,uf,vf,hf,rt,zi
      SIZEA = SIZEA*2
      CALL MPI_BCAST(DS(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(PO(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(UO(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(VO(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(TO(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(UF(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(VF(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(HF(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(RT(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(ZI(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)

!     Size of array variables: u,v,w,a,q,p,e,h,x
      SIZEA = SIZEA*NLVL
      CALL MPI_BCAST(U(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(V(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(W(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(A(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(Q(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
```

```
      CALL MPI_BCAST(P(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(E(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(H(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(X(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)

!     vertical interpolation to terrain following coordinates
!     data from NZS levels processed to NLVL of internal grid
      CALL PRFCOM(TKERD,TKERN,KG,KT1,KSFC,GX(:,:,KG),GY(:,:,KG),Z0(:,:,KG),     &
         ZT(:,:,KG),NXS(KG),NYS(KG),NZS,ZMDL,ZSG,NLVL,VMIX,ZI(:,:,K1,KG),      &
         P0(:,:,K1,KG),T0(:,:,K1,KG),U0(:,:,K1,KG),V0(:,:,K1,KG),UF(:,:,K1,KG), &
         VF(:,:,K1,KG),HF(:,:,K1,KG),SF(:,:,K1,KG),SS(:,:,K1,KG),U(:,:,:,K1,KG),&
         V(:,:,:,K1,KG),W(:,:,:,K1,KG),A(:,:,:,K1,KG),T(:,:,:,K1,KG),           &
         Q(:,:,:,K1,KG),P(:,:,:,K1,KG),E(:,:,:,K1,KG),H(:,:,:,K1,KG),           &
         X(:,:,:,K1,KG))

!     reset meteo time position if data are missing
      IF(FTIME(KG,K1).NE.MTIME(1))THEN
         WRITE(KF21,*)'WARNING advpnt: Time 1 input does not match request!'
         WRITE(KF21,*)' Internal time: ',JET
         WRITE(KF21,*)'  Time of data: ',FTIME(KG,K1)
         WRITE(KF21,*)'  Request time: ',MTIME(1)
         MTIME(1)=FTIME(KG,K1)
      END IF
   END IF

!-------------------------------------------------------------------------------
! test if new data required at the next time (k2)
!-------------------------------------------------------------------------------

   IF(NEW2)THEN

!  load data for subgrid
!  Root job reads input data
    IF (job_id == 0) THEN
      CALL METINP(BACK,KG,KT2,KUNIT2,KREC2,LX1(KG),LY1(KG),NXS(KG),NYS(KG),     &
         NZS,FTIME(KG,K2),KEND2,FHOUR(K2,KG),ZT(:,:,KG),DS(:,:,K2,KG),          &
         P0(:,:,K2,KG),T0(:,:,K2,KG),U0(:,:,K2,KG),V0(:,:,K2,KG),               &
         UF(:,:,K2,KG),VF(:,:,K2,KG),HF(:,:,K2,KG),RT(:,:,K2,KG),ZI(:,:,K2,KG), &
         U(:,:,:,K2,KG),V(:,:,:,K2,KG),W(:,:,:,K2,KG),A(:,:,:,K2,KG),           &
         Q(:,:,:,K2,KG),P(:,:,:,K2,KG),E(:,:,:,K2,KG),H(:,:,:,K2,KG),           &
         X(:,:,:,K2,KG))
    ENDIF

!     Root job sends input data / jobs 1..num_job-1 receive input data
!     Send/receive argument mc
      CALL MPI_BCAST(FTIME(KG,K2),1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

!     Send/receive argument ifhr
      CALL MPI_BCAST(FHOUR(K2,KG),1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

!     Send/receive input data

!     Size of array variable zt
      SIZEA = NXT*NYT*NGRD
```

```
      CALL MPI_BCAST(ZT(:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)

!     Size of array variables: ds,p0,u0,v0,t0,uf,vf,hf,rt,zi
      SIZEA = SIZEA*2
      CALL MPI_BCAST(DS(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(P0(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(U0(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(V0(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(T0(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(UF(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(VF(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(HF(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(RT(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(ZI(:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)

!     Size of array variables: u,v,w,a,q,p,e,h,x
      SIZEA = SIZEA*NLVL
      CALL MPI_BCAST(U(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(V(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(W(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(A(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(Q(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(P(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(E(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(H(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(X(:,:,:,:,:),SIZEA,MPI_REAL,0,MPI_COMM_WORLD,ierr)

!    vertical interpolation of profile
     CALL PRFCOM(TKERD,TKERN,KG,KT2,KSFC,GX(:,:,KG),GY(:,:,KG),Z0(:,:,KG),     &
        ZT(:,:,KG),NXS(KG),NYS(KG),NZS,ZMDL,ZSG,NLVL,VMIX,ZI(:,:,K2,KG),       &
        P0(:,:,K2,KG),T0(:,:,K2,KG),U0(:,:,K2,KG),V0(:,:,K2,KG),UF(:,:,K2,KG), &
        VF(:,:,K2,KG),HF(:,:,K2,KG),SF(:,:,K2,KG),SS(:,:,K2,KG),U(:,:,:,K2,KG),&
        V(:,:,:,K2,KG),W(:,:,:,K2,KG),A(:,:,:,K2,KG),T(:,:,:,K2,KG),           &
        Q(:,:,:,K2,KG),P(:,:,:,K2,KG),E(:,:,:,K2,KG),H(:,:,:,K2,KG),           &
        X(:,:,:,K2,KG))

!    missing data metpos will wait longer before reading
     IF(FTIME(KG,K2).NE.MTIME(2))THEN
        WRITE(KF21,*)'WARNING advpnt: Time 2 input does not match request!'
        WRITE(KF21,*)' Internal time: ',JET
        WRITE(KF21,*)'  Time of data: ',FTIME(KG,K2)
        WRITE(KF21,*)'  Request time: ',MTIME(2)
        MTIME(2)=FTIME(KG,K2)
     END IF
  END IF

 (...)
```

Bibliografía

[1] Air Resources Laboratory (ARL). *http://www.arl.noaa.gov.*

[2] ARL archived data . *http://www.arl.noaa.gov/ready.php.*

[3] ARL ftp . *ftp://gus.arlhq.noaa.gov/pub/archives/nams.*

[4] Bureau of Meteorology of Australia. *http://www.bom.gov.au.*

[5] Centro de Investigación Atmosférica de Izaña-Agencia Estatal de Meteorología (CIAI-AEMET). *http://www.aemet.es.*

[6] CIECEM (Universidad de Huelva). *http://www.ciecem.uhu.es/hysplit.htm.*

[7] El modelo de pronóstico numérico WRF. *http://www.contingencias.mendoza.gov.ar/wrf.php.*

[8] El modelo MM5. *http://galileo.imta.mx/chiapas/Modelo_MM5/inicio.php.*

[9] FT-MPI. *http://icl.cs.utk.edu/ftmpi.*

[10] *GNU gprof. http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html.*

[11] *GNU/Linux: Comando diff. http://enavas.blogspot.com/2008/03/el-shell-de-linux-comando-diff.html.*

[12] HYSPLIT (Hybrid Single Particle Lagrangian Integrated Trajectory) Model. *http://www.arl.noaa.gov/HYSPLIT_info.php.*

[13] HYSPLIT PC Training Seminar. Technical report, Air Resources Laboratory. *http://www.arl.noaa.gov/documents/workshop/hysplit1/english/agenda.pdf.*

[14] LA-MPI. *http://public.lanl.gov/lampi.*

[15] LAM/MPI. *http://www.lam-mpi.org.*

[16] Message Passing Interface Forum. *http://www.mpi-forum.org.*

[17] MM5. *http://www.mmm.ucar.edu/mm5.*

[18] MPICH. *http://www.mcs.anl.gov/research/projects/mpi/mpich1.*

[19] National Center for Atmospheric Research (NCAR). *http://www.ncar.ucar.edu*.

[20] National Centers for Environmental Prediction (NCEP). *http://www.ncep.noaa.gov*.

[21] National Oceanic anf Athmospheric Administration (NOAA). *http://www.noaa.gov*.

[22] OpenMP. *http://openmp.org/wp*.

[23] OpenMPI. *http://www.open-mpi.org*.

[24] PVM (Parallel Virtual Machine). *http://www.csm.ornl.gov/pvm*.

[25] READY (Real-time Environmental Applications and Display sYstem). *http://www.arl.noaa.gov/ready.php*.

[26] Research Papers Referencing HYSPLIT. *http://www.arl.noaa.gov/HYSPLIT_refs.php*.

[27] The Message Passing Interface (MPI) standard. *http://www-unix.mcs.anl.gov/mpi*.

[28] The UNIX System. *http://www.unix.org*.

[29] The Weather Research and Forecasting Model. *http://www.wrf-model.org/index.php*.

[30] Web Pages Referencing HYSPLIT. *http://www.arl.noaa.gov/HYSPLIT_links.php*.

[31] El reto de pronosticar el tiempo. Periodismo de Ciencia y Tecnología, June 1999. *http://www.invdes.com.mx/anteriores/Junio1999/htm/imta.html*.

[32] Earth-Sun System Applied Sciences Program Homeland Security Program Element FY 2005-2009 Plan. Technical report, NASA Science Mission Directorate, Earth-Sun System Division, March 2005.

[33] Modelos meteorológicos utilizados por Windguru, 2009. *http://www.windguru.com/es/help_index.php?sec=models*.

[34] Selim G. Akl. *Parallel Computation Models and Methods*. Prentice Hall, 1997.

[35] Francisco Almeida, Domingo Giménez, José Miguel Mantas, and Antonio M. Vidal. *Introducción a la Programación Paralela*. Paraninfo, 1st edition, 2008.

[36] Yukiya Aoyama and Jun Nakano. RS/6000 SP: Practical MPI Programming. Technical report, International Technical Support Organization, August 1999.

[37] Andrew Bachler and Jenwei Hsieh. Maximizing NFS Scalability on Dell Servers and Storage in High-Performance Computing Environments, October 2004. High-Performance Computing.

[38] Annette Baerman, Steven Businger, Josh Porter, Duane Stevens, and Roland Draxler. Concentration and Dispersion Modeling of the Kilauea Plume. Technical report, University of Hawaii, NOAA. *http://www.soest.hawaii.edu/MET/Faculty/businger/poster/vog/*.

[39] Bernal C. Iván and Mejía N. David and Fernández A. Diego. Desarrollo de Aplicaciones Paralelas para Clusters utilizando MPI (Message Passing Interface) . XIX Jornadas en Ingeniería Eléctrica y Electrónica, Escuela Politécnica Nacional Quito-Ecuador, November 2005.

[40] Rohit Chandra, Leonardo Dagum, Dave Khor, Dror Maydan, Jef McDonald, and Ramesh Menon. *Parallel Programming in OpenMP.* Academic Press, 2001.

[41] Stephen J. Chapman. *FORTRAN 90/95 for Scientists and Engineers.* Mc Graw Hill, 2nd edition, 2004.

[42] Richard D. Clark, Cheol-Heon Jeong, and C. Russell Phibrick. The Influence of Canadian Wildfires on Air Quality in Philadelpia PA during NE-OPS-DEP. Technical report, Millersville University. *http://ams.confex.com/ams/pdfpapers/55639.pdf.*

[43] Kevin Dowd and Charles Severance. *Programming Models.* O'reilly, 2nd edition, July 1998.

[44] Roland Draxler, Barbara Stunder, Glenn Rolph, Ariel Stein, and Albion Taylor. HYSPLIT4 User´s Guide. Technical report, Air Resources Laboratory, January 2009. *http://www.arl.noaa.gov/documents/reports/hysplit_user_guide.pdf.*

[45] Roland R. Draxler and G. D. Hess. Description of the HYSPLIT 4 Modeling System. Technical report, Air Resources Laboratory, December 1997. Last Revision: 2004, January.

[46] David DuBois. Measurement of Aerosol Optical Thickness. Technical report, University of Nevada, Reno, September 1998. *http://wolfweb.unr.edu/homepage/daved/aotpaper.pdf.*

[47] Ian Foster. *Designing and Building Parallel Programs Concepts and Tools for Parallel Software Engineering.* Addison Wesley, 2nd edition, 1995.

[48] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos. PVM and MPI: A comparison of features. *Calculateurs Paralleles*, 8:137–150, 1996.

[49] William Gopp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Janusz Kowalik, 2nd edition, 1999.

[50] Sergei Gorlatch, Christoph Wedler, and Christian Lengauer. Optimization Rules for Programming with Collective Operations. In *IPPS/SPDP'99. 13th Int. Parallel Processing Symp.10th Symp. on Parallel and Distributed Processing*, pages 492–499. Society Press, 1999.

[51] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing.* Addison Wesley, 2nd edition, 2003.

[52] Instituto Nacional de Ecología. *Introducción a la Evaluación de los Impactos de las Termoeléctricas de México*, chapter 3: Modelación de la Contaminación Atmosférica. México, 1st edition, 2006.

[53] Aaron M. Kinser. Simulating Wet Deposition of Radiocesium from the Chernobyl Accident. Technical report, Air Force Inst of Tech Wright-Patterson AFB OH School of Engineering and Management. *http://www.stormingmedia.us/43/4352/A435293.html*.

[54] Rob Latham, Rob Ross, and Rajeev Thakur. The Impact of File Systems on MPI-IO Scalability, 2004.

[55] Rebecca Lindsey. 15 December 2003 - Blowing Dust/Sand in New Mexico and Texas. Technical report, NASA Earth Observatory Feature, January 2004. http://cimss.ssec.wisc.edu/goes/misc/031215/031215.html.

[56] Rebecca Lindsey. Smoke's Surprising Secret. Technical report, NASA Earth Observatory Feature, January 2004. *http://earthobservatory.nasa.gov/Features/SmokeSecret*.

[57] Glenn R. Luecke, Bruno Raffin, and James J. Coyle. The Performance of the MPI Collective Communication Routines. In *The Cray T3E600, the Cray Origin 2000, and the IBM SP. The Journal of*, page Iowa State Unversity, August 1999.

[58] Peter S. Pacheco and Woo Chat Ming. Message Passing Programing: MPI User's Guide in FORTRAN, March 2006.

[59] Jose T. Palma, M. Carmen Garrido, Fernando Sánchez, and Alexis Quesada. *Programación Concurrente*, chapter 7: Paso de Mensajes. Thomson, 2nd edition, 2003.

[60] Gregory F. Pfister. *In Search of Clusters: The Comming Battle in Lowly Parallels Computing*, chapter 8: Basic Programming Models and Issues. Prentice Hall, 1995.

[61] Michael J. Quinn. *Parallel Computing: Theory and Practice*, chapter 4: Parallel Programming Languages. Mc Graw Hill, 1994.

[62] Jamie R. Rhome, Dutta S. Niyogi, and Sethu Raman. Assessing Seasonal Transport and Deposition of Agricultural Emissions in Eastern North Carolina, U.S.A. Technical report, North Carolina State University, 2003. *http://www.nc-climate.ncsu.edu/sraman/publications/J163.pdf*.

[63] Amina Saify, Garima Kochhar, and Jenwei Hsieh. Enhancing High-Performance Computing Clusters with Parallel File Systems, May 2005. High-Performance Computing.

[64] William C. Skamarock, Joseph B. Klemp, Jimmy Dudhia, David o.Gill, Dale M. Barker, Michael G. Duda, Xiang-Yu Huang, Wei Wang, and Jordan G. Powers. *A Description of the Advanced Research WRF Version 3*, June 2008.

[65] Walter F. Tichy and Peter J. Denning. Highly Parallel Computation. *Science*, 250:1217–1222, August 1990.

[66] David Walker and Jack Dongarra. volume 1. MIT Press, 2nd edition, September 1998.