Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería Informática

Proyecto Final de Carrera

Evaluación de prestaciones de aplicaciones paralelas en diferentes configuraciones de memoria en arquitectura NUMA

Autor: Alejandro Lamas Daviña
Director: Federico Silla Jiménez
Codirector: Héctor Montaner Mas

Diciembre de 2010

Índice general

1. Introducción **5**

2. Hardware **9**
2.1. Arquitectura NUMA 9
2.2. Características 10

3. Software **13**
3.1. Sistema operativo 13
3.2. Búsqueda de software 13
3.2.1. Benchmarks para el software científico 13
3.3. Aplicaciones 14
3.3.1. STREAM 14
3.3.2. pChase 15
3.3.3. Gromacs 15
3.3.4. NAMD 16
3.3.5. ACML 17

4. Compilación e instalación **19**
4.1. STREAM 19
4.2. pChase 19
4.3. Gromacs 19
4.4. NAMD 20
4.5. ACML 21

5. Proceso de ejecución **23**
5.1. Escenarios 23
5.1.1. Sólo RAM en ejecución afín 23
5.1.2. Sólo RAM no afín 23
5.1.3. Ejecución en swap de disco 24
5.1.4. Ejecución en swap de ramdisk remoto 24
5.1.5. Ejecución en swap de ramdisk 25

6. Resultados **27**
6.1. Stream 27
6.1.1. Ejecución en RAM afín 27
6.1.2. Ejecución en RAM no afín 29
6.1.3. Ejecución con swap a disco 31
6.1.4. Ejecución con swap a ramdisk local 32
6.2. pChase 33

6.2.1. Ejecución en RAM afín . . . 33
6.2.2. Ejecución en RAM no afín . . . 36
6.2.3. Ejecución con swap a disco . . . 38
6.2.4. Ejecución con swap a ramdisk local . . . 42
6.3. Gromacs . . . 44
6.3.1. Ejecución en RAM afín . . . 45
6.3.2. Ejecución en RAM no afín . . . 47
6.3.3. Ejecución con swap a disco . . . 49
6.3.4. Ejecución con swap a ramdisk remoto . . . 52
6.3.5. Ejecución con swap a ramdisk local . . . 55
6.4. NAMD . . . 56
6.4.1. Ejecución en RAM afín . . . 57
6.4.2. Ejecución en RAM no afín . . . 59
6.4.3. Ejecución con swap a disco . . . 60
6.4.4. Ejecución con swap a ramdisk remoto . . . 62
6.4.5. Ejecución con swap a ramdisk local . . . 64
6.5. ACML . . . 66
6.5.1. Ejecución en RAM afín . . . 67
6.5.2. Ejecución en RAM no afín . . . 68
6.5.3. Ejecución con swap a disco . . . 69
6.5.4. Ejecución con swap a ramdisk remoto . . . 70
6.5.5. Ejecución con swap a ramdisk local . . . 71

7. Resumen de resultados **73**
7.1. stream . . . 73
7.2. pChase . . . 77
7.3. Gromacs . . . 81
7.4. NAMD . . . 83
7.5. ACML . . . 87

8. Conclusiones **91**

Referencias **93**

Capítulo 1

Introducción

El principal objetivo del presente proyecto ha sido medir la influencia del subsistema de memoria en el rendimiento de aplicaciones con gran huella de memoria en una arquitectura NUMA. La arquitectura NUMA usada ha sido la proporcionada por procesadores de tipo Opteron del fabricante AMD, interconectados por el protocolo HyperTransport. La idea final tras esta evaluación era estudiar el comportamiento de aplicaciones reales que requieren cantidades ingentes de memoria, pero que no disponen de ella a menos que se ejecuten en grandes computadores, provistos de varios TB de memoria RAM, pero que resultan tremendamente caros. La alternativa más inmediata al uso de estos grandes computadores es el uso de memoria swap en disco. Sin embargo, esta alternativa puede llegar a resultar inviable si el working set de la aplicación es mayor que la memoria física disponible, produciéndose en este caso el fenómeno denominado thrashing.

Para realizar este estudio se han realizado pruebas de rendimiento con aplicaciones sintéticas específicas para medir el ancho de banda y la latencia en el acceso a memoria con múltiples tamaños de la huella de memoria utilizada. Junto con éstas, también se han buscado y utilizado aplicaciones científicas con problemas reales preparados para benchmarks. Las pruebas se llevaron a cabo con el sistema operativo openSuSE basado en kernel linux en su versión 10.3.

Los diferentes escenarios en los que se han evaluado las prestaciones de estas aplicaciones son:

1. Sólo RAM en ejecución afín. El sistema se dotó de una gran cantidad de memoria RAM (128 GB) y se eliminó la memoria de intercambio (swap). Todas las aplicaciones se ejecutaron siempre con memoria libre suficiente y se comprobó que se usase la memoria RAM directamente conectada al procesador donde se estaba ejecutando cada uno de los procesos de los que se compone la aplicación (afinidad memoria-procesador). Nótese que éstos son los mejores tiempos que se pueden obtener con las diferentes configuraciones. Por tanto los tiempos obtenidos en estas pruebas fueron tomados como referencia para el resto.

2. Sólo RAM no afín. Se estableció la misma configuración que en el primer punto y se ejecutaron las aplicaciones de manera que la memoria que utilizaron fuese la asociada a un procesador diferente del que estaba eje-

cutando el proceso. El objetivo de esta prueba fue mostrar la diferencia de tiempo total en el acceso del procesador a memorias de otro socket.

3. RAM reducida y memoria de intercambio en disco local. En los sistemas operativos actuales se utiliza esta configuración de manera habitual. Cuando el sistema operativo agota la memoria física disponible consigue memoria adicional desalojando a disco algunas de las páginas en memoria RAM. Obviamente, el proceso de intercambio de páginas entre la memoria RAM y el disco duro tiene una sobrecarga no despreciable, que aumenta notablemente el tiempo de ejecución. Para llevar a cabo esta prueba se inició la máquina con una cantidad de memoria RAM suficiente para albergar el sistema operativo y dejar algo de memoria RAM adicional libre, y se activó un área de intercambio en una partición de un disco duro local de la máquina. Las pruebas realizadas con esta configuración tuvieron una huella de memoria considerablemente más pequeña para poder obtener los resultados en un tiempo medianamente razonable, pues el acceso a disco ralentiza mucho la aplicación.

4. RAM reducida y memoria de intercambio en RamDisk remoto. El uso de memoria RAM de otros nodos de un cluster como memoria de intercambio fue propuesta hace ya tiempo. La ventaja de esta técnica estriba en que recuperar las páginas de memoria remota es más rápido que recuperarlas del disco duro local, a pesar del sobrecoste inherente de las comunicaciones a través de la red local que interconecta las diferentes máquinas. En cualquier caso, esta técnica sigue penalizada por la sobrecarga introducida por el sistema operativo al realizar la paginación. Para llevar a cabo estas pruebas, la máquina en la que se ejecutaron las aplicaciones se inició con los parámetros adecuados para establecer un tamaño de memoria mínimo en el que tuviese cabida el sistema operativo y se utilizó una segunda máquina que exportó un RamDisk mediante el uso de NBD (network block device). La primera importó la memoria remota y la utilizó como área de intercambio. En esta configuración se incrementa la sobrecarga con el sistema de exportación de la memoria, la pila de protocolos TCP/IP sobre la que ésta se sustenta y la capa física de red que comunica ambas máquinas por ethernet mediante cable de cobre de par trenzado (en nuestro caso).

5. RAM reducida y memoria de intercambio en RamDisk local. Para esta configuración se inició la máquina con un tamaño de RamDisk suficientemente grande como para poder ocupar casi toda la memoria dejando la cantidad suficiente para albergar el sistema operativo y que quedase algo de memoria adicional libre. De esta manera las aplicaciones ejecutadas no tuvieron memoria libre suficiente y el sistema operativo se vió obligado a realizar intercambio de páginas entre la RAM y el espacio swap creado en RAM. Los tiempos obtenidos nos mostraron la sobrecarga introducida por el sistema operativo a la hora de realizar el swapping y gestionar el ramdisk.

Todas las aplicaciones usadas permiten ejecución en paralelo mediante uso de memoria compartida y se realizaron pruebas con cuatro configuraciones de paralelismo de uno, cuatro, ocho y dieciséis hilos respectivamente para todas[1] las

[1] En el segundo escenario (uso de memoria no afín) sólo se ejecutaron con 1 y 4 hilos.

configuraciones de memoria establecidas.

Capítulo 2

Hardware

Durante el desarrollo del trabajo se han utilizado diferentes máquinas. Todas de arquitectura NUMA y procesadores AMD Opteron. En la primera fase, (búsqueda de software) se utilizó un pequeño ordenador personal con un procesador dual-core y 1 gigabyte de memoria RAM. En este ordenador se realizó un primer entorno de pruebas en el que se instaló el mismo sistema operativo que tenía la máquina en la que posteriormente se realizarían las pruebas finales y en él se instalaron y probaron las aplicaciones durante el proceso de selección. Una vez elegidas las aplicaciones, las pruebas se realizaron en servidores supermicro H8QM8 con diferentes configuraciones de memoria para establecer cada uno de los escenarios de estudio.

2.1. Arquitectura NUMA

NUMA-(Non-Uniform Memory Access). La arquitectura NUMA define un acceso y organización de la memoria en máquinas multiprocesador con memoria compartida en las que todos los procesadores tienen un acceso a memoria común. La característica principal de esta arquitectura es (como su nombre indica) que no proporciona un acceso uniforme en tiempo a memoria. En esta arquitectura se tarda más en acceder a unas zonas de memoria que a otras ya que las diferentes zonas de memoria están en buses diferentes. Cada procesador tiene asociado un conjunto de memoria que se denomina memoria local a la CPU que le proporciona un tiempo de acceso mínimo. A la vez, todos los procesadores pueden acceder mediante buses de interconexión a las zonas de memoria locales de los otros con unos tiempos de acceso mayores debidos a la mayor distancia que los separa.

La arquitectura NUMA se diseñó para superar los límites de la arquitectura UMA (utilizada ampliamente por el fabricante Intel). En la arquitectura UMA todos los procesadores acceden con un tiempo igual a cualquier zona de la memoria compartida. Los accesos a memoria comparten el mismo bus y la competición por el acceso a éste provoca cuellos de botella cuando se escala el número de procesadores.

NUMA intenta solucionar el problema proporcionando memoria asociada a cada procesador (pero no esclusiva de éste) y que de esta forma no se dé el caso de que varios procesadores intenten acceder a la misma región de memoria a

la vez. Todos los procesadores pueden acceder a toda la memoria del sistema a través de interconexiones de buses.

El tiempo de respuesta de la memoria es muy importante para evitar tener el procesador ocioso ya que los procesadores actuales son mucho más rápidos que la memoria y es muy normal que el procesador tenga que esperar la respuesta de ésta.

2.2. Características

El modelo de servidor utilizado en la realización de las pruebas fue un supermicro H8QM8, con cuatro procesadores AMD Opteron 8350 de cuatro núcleos cada uno, dando un total de 16 núcleos de procesado. La mayor configuración de memoria con la que se trabajó fue con 16 módulos de 8 GB, que proporcionaron un total de 128 GB de memoria y permitieron maximizar en ancho de banda obtenido, al poblar todas las ranuras de la placa base y poder utilizar todos los buses del sistema simultáneamente.

La placa base está basada en el conjunto de chips nVidia MCP55 Pro y AMD 8132. El MCP55 Pro actúa como procesador de comunicaciones y el AMD-8132 como un tunel PCI-X. Los controladores de la memoria del sistema están integrados directamente en los procesadores Opteron.

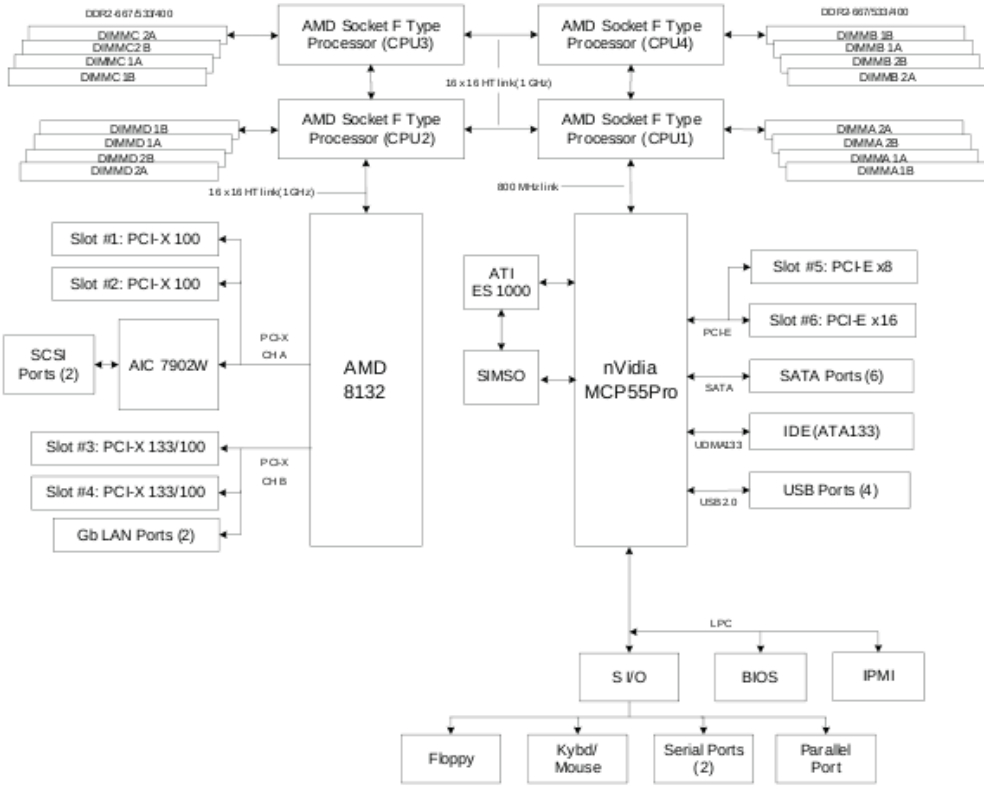
Procesador de comunicaciones MCP55 Pro: El MCP55 es un solo chip controlador de periféricos HyperTransport de alto rendimiento. Incluye un interfaz PCI-Express de 28 líneas, un interfaz de enlace HyperTransport AMD Opteron de 16 bits, un interfaz Serial-ATA de seis puertos a 3 Gb/s, un interfaz de dos puertos Ethernet Gb, un interfaz maestro de bus ATA133 y un interfaz USB 2.0. Este concentrador se conecta directamente a la CPU#1.

Tunel HyperTransport PCI-X AMD-8132: Este concentrador incluye tecnología específica de AMD que proporciona dos puentes PCI-X con bus de datos de 64 bits, modos de operación PCI-X separada y ratios de transferencia independientes. Cada puente soporta hasta cinco maestros PCI-X que incluyen el reloj y las señales de petición y respuesta. Este concentrador se conecta a los procesadores y al sistema de memoria. También se comunica directamente con los controladores Serial ATA y Ethernet.

Tecnología HyperTransport: La tecnología HyperTransport es un enlace punto a punto de baja latencia y alta velocidad que fue diseñado para incrementar la velocidad de comunicación entre los circuitos integrados. Esto se consigue parcialmente reduciendo el número de buses en el conjunto de chips (chipset) para reducir los cuellos de botella y habilitando un uso de la memoria más eficiente en los sistemas multiprocesador. El resultado obtenido es un notorio incremento en ancho de banda en el interior del chipset.

La conectividad de la máquina con el resto de máquinas la proporcionó el controlador Ethernet de dos puertos 1 Gb de los que se usó uno.

Figura 2.1: Esquema de la placa base.



En el esquema de la figura 2.1 se puede apreciar como cada procesador dispone de su propia memoria local y la conexión que tienen los procesadores entre si. Cada procesador está conectado con el resto por un enlace de tipo HyperTransport a una frecuencia de 1 GHz que utiliza para acceder a las zonas de memoria de los otros procesadores.

Componentes hardware:

- Placa base: Supermicro H8QM8-2
- CPU: 4x Quad-Core AMD Opteron(tm) Processor 8350
  - Caché L1: 4x 64 KB
  - Caché L2: 4x 512 KB
  - Caché L3: 4x 2048 KB
- RAM: 132352 MB (16 x DIMM 8192 MB DDR2 333 MHz (3.0 ns))
- Disco duro de 250 GB con interfaz SATA-II y 32 MB de caché.

Tabla 2.1: Características del disco duro.

Modelo	ST3250310NS
Revisión de firmware	SN04
Interfaz	SATA 3Gb/s
Caché	32 MB
Capacidad	250 GB
Velocidad de giro	7200 rpm
Ratio de transferencia interno (max)	1287 Mbps
Ratio de transferencia sostenido (max)	105 MB/s
Tiempo de búsqueda de pista en lectura	0,8 ms
Tiempo de búsqueda de pista en escritura	1,0 ms
Tiempo de búsqueda medio en lectura	8,5 ms
Tiempo de búsqueda medio en escritura	9,5 ms
Latencia media	4,16 ms
NCQ (Native Command Queueing)	Habilitado

Capítulo 3

Software

3.1. Sistema operativo

El sistema operativo utilizado en las pruebas fue openSUSE en su versión 10.3 compilado para la arquitectura x86_64. OpenSUSE es un sistema operativo basado en kernel linux derivado de la distribución SuSE de origen alemán. La versión del kernel de linux utilizado fue la 2.6.27.32.

3.2. Búsqueda de software

Durante la fase inicial de búsqueda se partió de la lista de software utilizado por ScaleMP en sus pruebas de rendimiento (http://www.scalemp.com/performance) para intentar obtener un conjunto de programas de diferente naturaleza que permitiesen realizar las mediciones. A excepción de los test sintéticos, el resto de programas objetivo eran todos específicos del área en la que se utilizan (bio-informática, dinámica de fluidos, dinámica molecular, predicciones meteorológicas, etc).

Como en la lista original no siempre se hacía referencia al nombre del distribuidor de software o a la web en la que se aloja, fue necesaria una pequeña labor de investigación en internet para ir encontrando los programas objetivo.

Una vez conocida la web donde se alojaba el software había que ver la disponibilidad de éste. La disponibilidad deseada era que se pudiese obtener el código fuente y que, tratándose de un proyecto académico, no hubiese que pagar por él. Aunque era bastante raro encontrar estas dos cualidades, no eran un requisito imprescindible y aunque finalmente no se realizó ninguna compra, en pocos casos se descartó un programa por su coste.

Los primeros datos anotados de cada programa eran su nombre, la web en la que se alojaba, la versión disponible, el tipo de licencia que tenían, su precio y en su caso alguna anotación pertinente del tipo "necesario registro"

3.2.1. Benchmarks para el software científico

De entre las aplicaciones científicas, y tras pasar las primeras selecciones, se eligieron aquellas que proporcionaban un conjunto de tests con el que medir y comparar resultados.

Los programas utilizados fueron:

- STREAM
- pChase
- Gromacs
- NAMD
- ACML

3.3. Aplicaciones

Para realizar las pruebas se usaron tests sintéticos y aplicaciones científicas reales. Los tests sintéticos permitieron adaptar las pruebas mediante la modificación de parámetros, de manera que la cantidad de memoria que necesitaban variaba, mientras que con las aplicaciones de uso científico se usaron varios ejemplos preestablecidos para cada aplicación con consumos diferentes de memoria.

3.3.1. STREAM

STREAM[7, 3] es un test de memoria que mide el ancho de banda sostenido de ésta y el ratio de cálculo de la CPU.

Este test está diseñado para trabajar con conjuntos de datos de mayor tamaño que la memoria caché del sistema para que ésta no desvirtúe los resultados y el rendimiento obtenido sea similar al que se obtendría con aplicaciones reales que hagan uso intensivo de la memoria mediante el uso de grandes vectores.

La regla general para el uso del test es que cada array debe tener por lo menos el tamaño de la suma de todos los últimos niveles de caché usados en la ejecución, o un millón de elementos (lo que sea mayor). En el caso de estudio, los tamaños se aumentaron considerablemente más para que el consumo de memoria fuese mayor, ya que el rendimiento obtenido no debe variar (al menos notablemente) una vez el array usado supera ampliamente el tamaño de la memoria caché.

En el caso de usar varios procesadores en paralelo, el tamaño del problema debe incrementarse acorde a la suma de las memorias caché, pero al igual que en ejecuciones secuenciales, la cantidad de RAM del sistema permitió que los tamaños usados fuesen lo suficientemente grandes.

El test se compone de cuatro operaciones que se realizan con los tres vectores de trabajo:

- **copy**: copia de los elementos de un vector sobre otro.
- **scale**: multiplicación de cada uno de los elementos de un vector por un escalar y almacenado el resultado en un segundo vector.
- **add**: suma de los elementos de dos vectores y almacenado el resultado en el tercer vector.
- **triad**: suma de los elementos de un vector a la multiplicación de los elementos de otro por un escalar y almacenado el resultado en el tercer vector.

Este software fue desde un principio uno de los elegidos. Para su utilización sólo hubo que modificar el tamaño del problema para que su ejecución utilizase la mayor parte de la memoria posible. Se crearon varias imágenes ejecutables con diferentes tamaños.

3.3.2. pChase

pChase[1] es un test de memoria que mide el ancho de banda y la latencia de diferentes patrones de acceso a distintos niveles de caché y memoria principal.

A diferencia de otros tests de memoria, el comportamiento por el cual ha de acceder al contenido del puntero en curso para conocer la siguiente dirección de memoria a acceder, hace que pueda conocer la latencia.

Este test se basa en que la memoria está dividida en jerarquías, y permite especificar el tamaño de cada nivel de la jerarquía. El test progresa seleccionando una página de memoria a la que referir. Una vez seleccionada la página, refiere todas las líneas de caché antes de seleccionar la siguiente página. Una iteración recorre todas las páginas de un mismo conjunto de memoria (en dominios NUMA). Una misma ejecución recorre el conjunto un número determinado de veces.

Las líneas de caché pueden seleccionarse de forma aleatoria o mediante avances constantes que pueden ser incrementando o decrementando las direcciones. Si se usa acceso aleatorio, la selección de páginas también es aleatoria y si se usa acceso de avance constante, se selecciona la página contigua siguiente según el avance.

En la ejecución se puede especificar el número de hilos que acceden a la memoria concurrentemente de forma que compiten en el acceso. En arquitecturas NUMA, el impacto debe ser mínimo ya que cada hilo accede únicamente a la memoria local al núcleo en el que se está ejecutando mientras que en arquitecturas UMA y multi-núcleo varios hilos pueden compartir la misma ruta a memoria provocando competición por el recurso.

En la ejecución también se puede especificar el número de referencias concurrentes que se permiten por cada hilo. Esto permite al test cargar rutas de memoria con referencias, mostrando más precisamente cual podría ser el rendimiento sostenido del sistema. Dos referencias por conjunto de memoria indica que tendrán lugar simultáneamente dos lecturas desde el mismo hilo. Esto es diferente a que las dos lecturas se realicen desde hilos diferentes, porque las rutas de acceso a memoria y el efecto en el uso de los recursos no es el mismo.

Este segundo benchmark sintético fue elegido candidato durante el proceso de documentación del proyecto pues no estaba en la lista inicial. No hizo falta hacer ninguna modificación al software pues es bastante completo y permite indicarle mediante parámetros todas las opciones de ejecución, lo que facilitó la tarea de realizar las mediciones.

3.3.3. Gromacs

Gromacs[6, 4] es un simulador de dinámica molecular escrito en C y publicado bajo la licencia libre GPL de GNU. Simula las ecuaciones del movimiento de Newton para sistemas de cientos o millones de partículas.

Está diseñado para trabajar con moléculas bioquímicas como proteínas, lípidos y ácidos nucleicos que tienen muchas interacciones fuertes complicadas, pero

como es muy rápido calculando interacciones débiles (que suelen predominar en las simulaciones) también se usa para investigación de sistemas no biológicos como polímeros.

Puede ser ejecutado en paralelo tanto mediante hilos como con MPI (Message Passing Interface).

Tests de rendimiento:

Para poder comparar el rendimiento del software en las diferentes plataformas de hardware existentes, los desarrolladores han preparado una serie de pruebas con unos pocos sistemas típicos. Todas las pruebas representan ejemplos reales (han sido seleccionados de proyectos de investigación en curso tanto de sus propios laboratorios como de diversos artículos científicos publicados).

Las pruebas están divididas en dos partes: la primera está orientada al rendimiento en máquinas individuales (sin paralelización en cluster, pero usando varios procesadores) y la segunda muestra la escalabilidad de grandes clusters al ejecutarse en paralelo sobre nodos comunicados por red.

El rendimiento es mostrado como pico-segundos de simulación por día para todos los tests y la escalabilidad en paralelo sobre N procesadores se define como:

$S = P_N/(N \cdot P_1)$ Siendo P_N el rendimiento en N procesadores.

Los cuatro tests disponibles fueron usados pues cada uno tiene un tamaño de problema diferente.

Este fue el primer candidato de software científico, visto con buenos ojos por su licencia GPL.

Su proceso de compilación e instalación cumple en estándar "`./configure; make; make install`". Para su compilación fue necesario instalar el paquete fftw "Fastest Fourier Transform" en su versión 3. Los benchmarks utilizados fueron los disponibles en: `ftp://ftp.gromacs.org/pub/benchmarks/gmxbench-3.0.tar.gz` que se componen de d.dppc, d.lzm, d.poly y d.villin.

El proyecto se empezó usando la última versión estable de Gromacs (4.0.7), pero durante el transcurso de este se publicó la versión 4.5.1 que permite el uso de hilos a nivel de sistema operativo para ejecución paralela (las versiones anteriores precisaban MPI) y se continuó el trabajo con esta nueva versión.

Las bibliotecas fftw que necesita Gromacs para la realización de transformadas de fourier se instalaron mediante el correspondiente paquete (rpm) de la distribución.

3.3.4. NAMD

NAMD[8, 5] es un simulador de dinámica molecular diseñado para simulaciones de alto rendimiento de sistemas biomoleculares grandes, escrito en C++.

Está diseñado con un sistema de datos distribuido para mejorar la escalabilidad en sistemas con gran número de procesadores.

Usa ficheros de bancos de datos de proteínas como entrada para describir la configuración de átomos de la simulación, pero también proporciona compatibilidad con ficheros de otras aplicaciones como Gromacs.

Está basado en "Charm++" un lenguaje de programación paralela orientado a objetos basado en C++ que posee una biblioteca (Charm kernel) para desarrollo de aplicaciones paralelas. A diferencia de los modelos tradicionales de

programación paralela basados en paso de mensajes o en variables compartidas, el modelo de programación de Charm++ está dirigido por mensajes y soporta diversos métodos de balanceo de carga dinámico mediante migración de objetos. Los cálculos son lanzados basándose en la llegada de mensajes asociados. Un planificador elige un mensaje de entre los disponibles y ejecuta los cálculos asociados con él.

El proceso de compilación de este software fue el más laborioso. En el capítulo 4 se describe este proceso.

3.3.5. ACML

ACML[2] (AMD Core Math Library) es una biblioteca desarrollada por AMD que proporciona un conjunto de funciones matemáticas optimizadas para sus procesadores. Está diseñada especialmente para permitir realizar cálculos multi-hilo y otras características incluidas en los procesadores AMD.

Entre sus componentes principales están:

- Implementación completa de BLAS (subrutinas de álgebra lineal básica) en los niveles 1 (operaciones vector-vector), 2 (operaciones matriz-vector) y 3 (operaciones matriz-matriz).
- Implementación completa de LAPACK (conjunto de rutinas de álgebra lineal).

Desarrollada para realización de cálculos científicos y uso en entornos de alto rendimiento es utilizada en sistemas de predicción meteorológica, análisis de elementos finitos, cálculo de dinámica de fluidos, análisis financieros, etc.

La biblioteca incluye un conjunto de ejemplos de uso que muestra como usarla con diferentes compiladores y como utilizar las funciones. Incluye también un conjunto de pruebas para medir el rendimiento de las funciones en un determinado sistema.

Del paquete de las bibliotecas ACML se utilizó uno de sus programas de muestra de rendimiento, compilado con soporte para ejecución en paralelo. El programa hace uso de la función DGETRF de LAPACK para realizar una factorización LU de una matriz N por N usando pivotación parcial con intercambio de filas. Este programa, escrito en fortran, se modificó para que mostrase en la salida el tiempo utilizado en los cálculos y, de manera similar a como se hizo con el test stream, se modificaron los tamaños de la matriz para ajustar el consumo de la memoria.

Para poder compilar este test fue necesario instalar el compilador de fortran "gfortran" en su versión 4.3. Como esta versión no estaba disponible para la distribución, fue necesario, a su vez, compilarla e instalarla. Para compilar gfortran-4.3 se utilizó el método estándar de creación de paquetes rpm. Una vez obtenidos los rpm correspondientes, se instalaron en la máquina sin eliminar ni afectar a la versión del compilador que incluye la distribución y que ya estaba instalado.

Capítulo 4

Compilación e instalación

En este capítulo se describe el proceso de compilación e instalación de las diferentes aplicaciones usadas a lo largo de este proyecto.

4.1. STREAM

Compilación:

```
$ gcc -O -fopenmp -D_OPENMP -mcmodel=medium stream.c -o stream
```

Instalación:

En el directorio de compilación.

4.2. pChase

Compilación:

```
$ make
```

Instalación:

En el directorio de compilación.

4.3. Gromacs

Instalación previa de los siguientes paquetes rpm del sistema:

- fftw3-devel (fftw3 ya estaba instalado)

Descompresión y desempaquetado:

```
$ tar zxf gromacs-4.5.1.tar.gz
$ cd gromacs-4.5.1
```

Configuración:

```
$ ./configure --enable-shared --prefix=~/soft/gromacs/gromacs-4.5.1-install
```

Compilación:

Para compilar se hizo uso de compilación en paralelo para agilizar el proceso.

```
$ gmake -j16
```

Instalación:

```
make install
```

4.4. NAMD

Compilar NAMD requiere compiladores de c y c++, charm++, tcl y fftw.

Los compiladores utilizados fueron los del sistema operativo, la versión de Charm++ fue la proporcionada en el tar de NAMD y las bibliotecas tcl y fftw utilizadas fueron las proporcionadas ya compiladas en http://www.ks.uiuc.edu/Research/namd/libraries/

Instalación previa de los siguientes paquetes rpm del sistema:

- gcc-c++
- gcc42-c++
- libstdc++42-devel

Descompresión y desempaquetado:

```
$ tar zxf NAMD_2.7b3_Source.tar.gz
$ cd NAMD_2.7b3_Source
$ tar xf charm-6.2.1.tar
$ cd charm-6.2.1
```

Compilación de Charm++:

```
$ ./build charm++ net-linux-amd64 smp
```

Nota: la opción "`--build-shared`" es añadida automáticamente al compilar Charm++ mediante el script "build", por lo que no es necesario especificarla.

Descargar e instalar las bibliotecas TCL y FFTW (en el directorio NAMD_2.7b3_Source).

```
$ wget http://www.ks.uiuc.edu/Research/namd/libraries/fftw-linux-x86_64.tar.gz
$ tar xzf fftw-linux-x86_64.tar.gz
$ mv linux-x86_64 fftw
$ wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl-linux-x86_64.tar.gz
$ tar xzf tcl-linux-x86_64.tar.gz
$ mv linux-x86_64 tcl
```

Configuración de NAMD:

```
$ ./config Linux-x86_64-g++ --charm-arch net-linux-amd64-smp
$ cd Linux-x86_64-g++
```

Compilación:

```
$ gmake -j16
```

Instalación:

En el directorio de compilación.

4.5. ACML

Compilación

```
gfortran-4.3 -c -fdefault-integer-8 time_dgetrf.f90 -o time_dgetrf.o
gfortran-4.3 -fopenmp time_dgetrf.o\
/usr/local/acml/acml4.4.0/gfortran64_mp_int64/lib/libacml_mp.a -lrt\
-o time_dgetrf-.exe
```

Instalación:

En el directorio de compilación.

Capítulo 5

Proceso de ejecución

Para la ejecución de los tests se crearon una serie de scripts de lanzamiento. Previo a la ejecución se detuvieron varios procesos del sistema (entre ellos crond) para evitar que se polucionasen los resultados con ejecuciones simultáneas de otras aplicaciones y se desmontaron los sistemas de ficheros por red (nfs).

5.1. Escenarios

5.1.1. Sólo RAM en ejecución afín

Este escenario sirvió para tomar tiempos y medidas de referencia siendo el escenario óptimo de ejecución. Todos los procesos dispusieron en todo momento de memoria libre. En este caso se midió la escalabilidad y grado de paralelización de las aplicaciones.

5.1.2. Sólo RAM no afín

NUMA es una arquitectura en la que los tiempos de acceso de un procesador a memoria para diferentes regiones de memoria varían acorde a la distancia del procesador a la región de memoria.

Cada región de memoria en la que los tiempos de acceso son los mismos para cada procesador, se le denomina nodo. En este tipo de arquitecturas el kernel debe intentar minimizar la comunicación entre nodos.

En el kernel de linux, la política de memoria determina de cual de los nodos de un sistema NUMA reservará memoria el kernel. Las políticas de memoria son un interfaz de programación de la que pueden hacer uso las aplicaciones. La política por defecto es usar memoria local al nodo en la que se ejecuta el proceso.

`numactl` y `numastat`:

numastat es una aplicación que muestra estadísticas sobre la asignación de memoria NUMA para cada nodo. De entre los datos que muestra, nos interesan los contadores local_node (se incrementa cuando a un proceso corriendo en el nodo se le asigna memoria de ese mismo nodo) y other_node (se incrementa cuando a un proceso corriendo en otro nodo se le asigna memoria de ese mismo nodo).

numactl es una aplicación que permite ejecutar procesos con una política de asignación de memoria o un planificador NUMA específicos. La política es establecida por el proceso del comando y heredada por todos sus descendientes.

Las distancias entre nodos conllevan tiempos de acceso diferente (tabla 5.1).

Tabla 5.1: Distancias entre nodos.

node	0	1	2	3
0	10	20	20	20
1	20	10	20	20
2	20	20	10	20
3	20	20	20	10

En este escenario se forzó la asociación de los procesos a un determinado procesador (con sus cuatro núcleos) y el uso de la memoria del resto de procesadores.

```
$ numactl --cpunodebind=0 --membind=1,2,3 /bin/bash
$ numactl --show
policy: bind
preferred node: 1
physcpubind: 0 1 2 3
cpubind: 0
nodebind: 0
membind: 1 2 3
```

5.1.3. Ejecución en swap de disco

En este escenario se inició la máquina con poca memoria para forzar el uso del área de intercambio de disco duro y se redujo el tamaño de las pruebas sintéticas pues los tiempos obtenidos crecieron considerablemente. En las aplicaciones que no se podía modificar el tamaño de las ejecuciones, se optó por ampliar la memoria disponible con el fin de que pudiesen terminar en plazos razonables. La norma general fue arrancar la máquina con 256 MB de memoria RAM (mediante el parámetro del kernel: mem=256M), pero en algunos casos hubo que reducirla a 128 MB y en otros, como ya se ha dicho, aumentarla.

5.1.4. Ejecución en swap de ramdisk remoto

En este escenario se utilizaron simultáneamente dos máquinas, conectadas por un enlace gigabit ethernet por medio de un switch. En una de ellas se creó un ramdisk y se exportó mediante el dispositivo de bloques por red NBD. En la otra, se arrancó el sistema limitando la memoria a 256 MB y se montó el dispositivo NBD como área de intercambio con mayor prioridad que el área de intercambio de disco duro local.

Los tests de memoria stream y pChase no pudieron obtener resultados con esta configuración ya que el NBD no soportó el estrés al que le sometieron y los procesos murieron en todas las ocasiones. El resto de aplicaciones, que no hacen un uso tan intensivo de la memoria, pudieron finalizar sus ejecuciones.

5.1.5. Ejecución en swap de ramdisk

En linux, el controlador de discos RAM (ramdisks) permite usar la memoria principal del sistema como un dispositivo de bloques. El ramdisk crece dinámicamente a medida que se necesita más espacio mediante el uso de la memoria RAM usada como caché. El controlador marca el espacio de almacenamiento temporal que está usando, como "sucio", para que el subsistema de memoria virtual no lo reclame más tarde (estas páginas no son liberadas nunca).

Este mecanismo de ramdisk crea un dispositivo de bloques sintético en un área de RAM. Este dispositivo de bloques es de tamaño fijo. Usar ramdisk implica tanto copiar memoria del falso dispositivo de bloques a páginas de caché (y volver a copiar los cambios de vuelta), como crear y eliminar *dentries* (entradas de directorio). Esto crea trabajo innecesario al procesador (y contamina su caché) pues gasta memoria y ancho de banda.

El sistema ramfs, es mucho más eficiente y simple que ramdisk, pero como no se puede limitar su tamaño, se puede escribir en él hasta ocupar toda la memoria. Este fue el motivo por el que se descartó este procedimiento.

Otro sistema de uso de memoria como espacio de ficheros que posee linux es tmpfs (un derivado de ramfs). Permite establecer límites en la memoria utilizada y que su contenido pase a espacio de intercambio. Debido a esta migración a espacio de intercambio, este dispositivo no se puede utilizar para establecer áreas de espacio de intercambio.

Para crear áreas de intercambio usando dispositivos ramdisk se puede optar por establecer un sistema de ficheros en el dispositivo, crear en él un fichero que ocupe todo el espacio disponible y usar éste como área de intercambio. Esta aproximación tiene la sobrecarga añadida del sistema de ficheros. Otra forma un poco más óptima es usar directamente el dispositivo ramdisk como área de intercambio (se puede hacer `mkswap /dev/ram0`). Para ello es necesario "tocar" todo el espacio del dispositivo previamente (`dd if=/dev/zero of=/dev/ram0`) con el fin de que el kernel marque la memoria como en uso.

Capítulo 6

Resultados

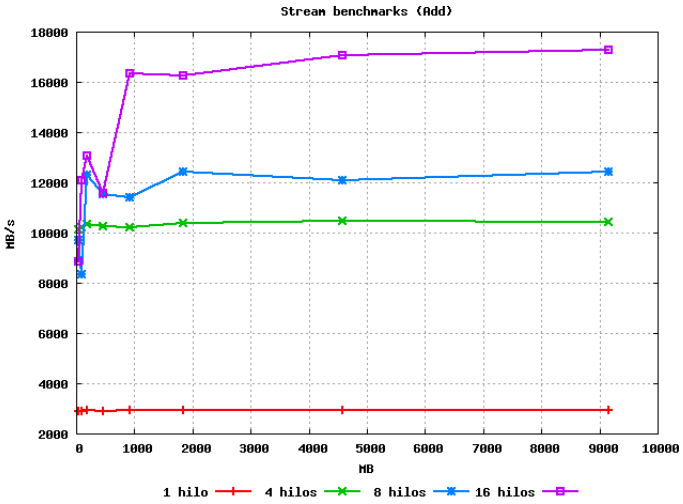
En este capítulo se muestran los resultados obtenidos tras la ejecución de las aplicaciones en los diversos escenarios.

6.1. Stream

A continuación se muestran los resultados de una de las cuatro operaciones que realiza stream y una figura con el resultado global de la ejecución. Los resultados nos permiten conocer el tiempo de ejecución y el ancho de banda obtenido.

6.1.1. Ejecución en RAM afín

Figura 6.1: Ancho de banda de la operación add por tamaño e hilos.



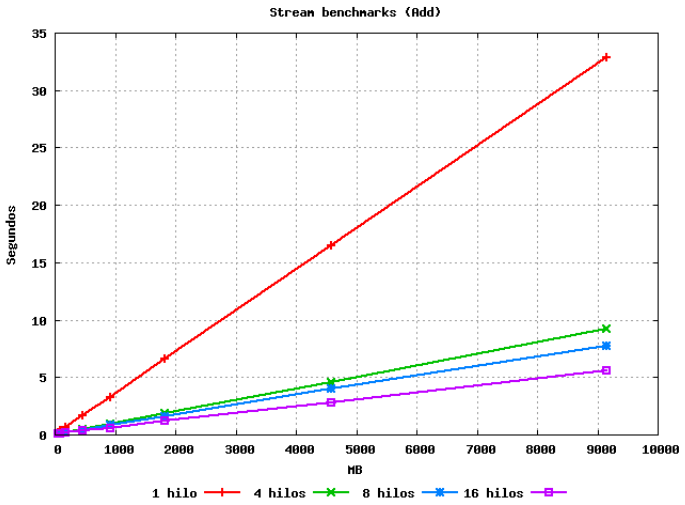
En la figura 6.1 se muestra el ancho de banda obtenido en la operación "add" para diferentes tamaños del vector con ejecuciones de 1, 4, 8 y 16 hilos. Una vez superado un determinado umbral del tamaño del problema, el ancho de banda se estabiliza en el máximo posible con ese grado de paralelización. Se puede apreciar que el máximo rendimiento de 17294 MB/s se obtiene con la ejecución de 16 hilos que es la que ocupa todos los núcleos de procesado de la máquina.

La tabla 6.1 resume el ancho de banda obtenido con el mayor tamaño del problema.

Tabla 6.1: Ancho de banda de la operación add con un tamaño de 9155,3 MB.

Hilos	1	4	8	16
MB/s	2927	10405	12410	17294

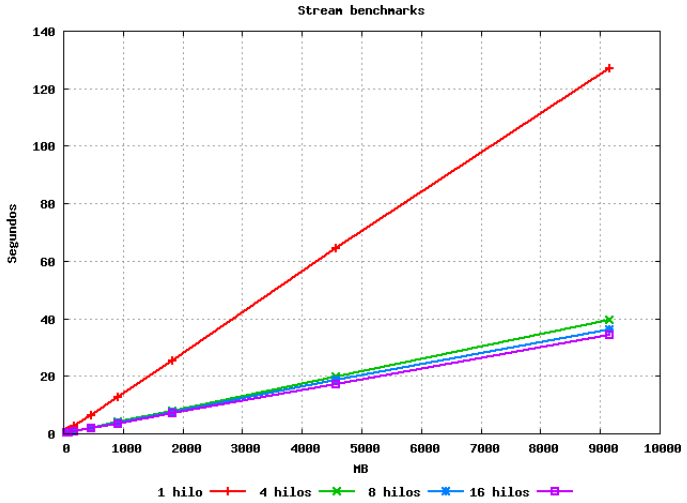
Figura 6.2: Tiempo de la operación add por tamaño e hilos.



En la figura de tiempos 6.2 se aprecia el alto grado de paralelización de la aplicación, pues simplemente se reparte el acceso al vector entre los diferentes hilos. La aceleración obtenida con cuatro hilos es $S_p = \frac{t_1}{t_4} = \frac{32{,}85}{9{,}234} = 3{,}56$ y la eficiencia $E_p = \frac{S_p}{p} = \frac{3{,}56}{4} = 0{,}89$.

Los resultados del resto de operaciones que se realizan durante la ejecución de stream siguen el mismo tipo de comportamiento que hemos visto.

Figura 6.3: Tiempo agregado de las 4 operaciones por tamaño e hilos.



La figura 6.3 muestra el tiempo total de la ejecución de stream para diferentes tamaños del vector e hilos de ejecución. El tiempo es el agregado de las cuatro operaciones que realiza. Se aprecia el gran salto existente entre los tiempos de la ejecución con un único hilo y las de múltiples hilos debido a la limitación intrínseca de tener un solo hilo de ejecución que impide aprovechar todos los recursos de la máquina. Por otro lado la sobrecarga en la gestión de múltiples hilos por parte del sistema operativo y los límites físicos de la memoria hacen que a partir de cuatro hilos, la mejora de rendimiento obtenida sea menor al aumentar el grado de paralelización. El reparto de la carga de trabajo a la hora de paralelizar que hace OpenMP hace que en el primer acceso por parte de un hilo a una determinada posición de memoria se produzca un fallo de caché, pero una vez está en caché, los demás hilos (que inmediatamente utilizarán las posiciones contiguas) siempre tienen aciertos y no se produce competición por la caché.

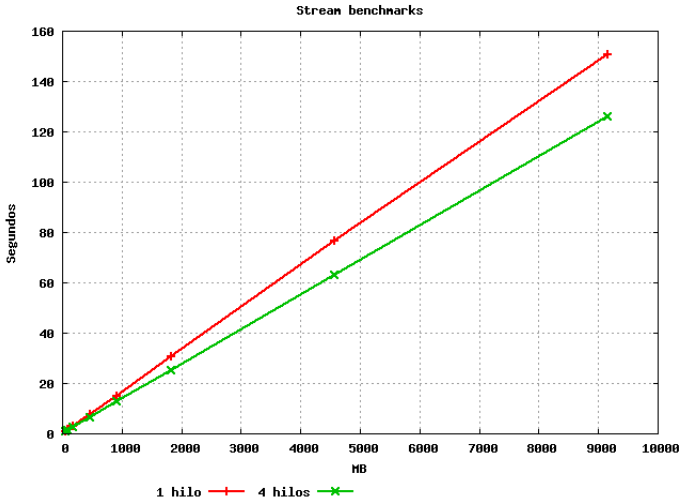
Tabla 6.2: Tiempo agregado de las 4 operaciones con un tamaño de 9155,3 MB.

Hilos	1	4	8	16
Segundos	127,1	39,48	36	34,09

Para el resto de escenarios se muestra la figura que resume la ejecución de stream.

6.1.2. Ejecución en RAM no afín

Figura 6.4: Tiempo agregado de las 4 operaciones por tamaño e hilos.



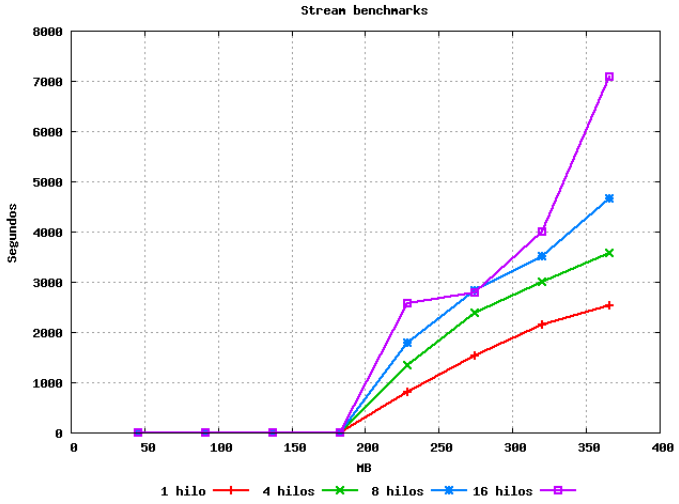
En la figura 6.4 se ve como al usar memoria de otros procesadores, se incrementan los tiempos de acceso a la memoria y el tiempo total utilizado aumenta. Con un hilo, empeora un 18 % respecto a la configuración base y es notable la pérdida de rendimiento con la ejecución de cuatro hilos pues es un 220 % más lento que la configuración base y se pierde la gran diferencia entre uno y cuatro hilos.

Tabla 6.3: Tiempo agregado de las 4 operaciones con un tamaño de 9155,3 MB.

Hilos	1	4	8	16
Segundos	150,54	126,16	-	-

6.1.3. Ejecución con swap a disco

Figura 6.5: Tiempo agregado de las 4 operaciones por tamaño e hilos.



En la figura 6.5 se observa como las ejecuciones de menor tamaño han corrido en RAM, y como, a medida que aumenta el tamaño del problema y se hace necesario utilizar mayor cantidad del área de intercambio, el tiempo aumenta. Siendo las ejecuciones con mayor número de hilos las que más tardan, al tener que competir los hilos entre sí por las peticiones a disco y el reducido espacio en RAM que van sobreescribiendo continuamente, interrumpiendo las ejecuciones de los demás hilos.

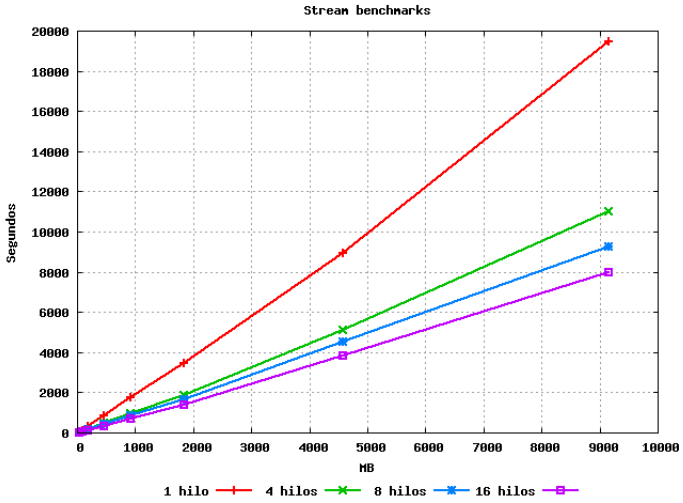
Tabla 6.4: Tiempo agregado de las 4 operaciones para un tamaño de 366 MB.

Hilos	1	4	8	16
Segundos	2538	3584,4	4660	7094

Comparando los tiempos de la tabla 6.4 con los de la ejecución del escenario base para un tamaño de 457,8 MB, se puede ver que el rendimiento empeora más de un 39743 % para la ejecución secuencial (que es con la que se obtiene el mejor tiempo), y más de un 178226 %, 243879 % y 363694 % para las de 4, 8 y 16 hilos. Esto significa una diferencia de tres órdenes de magnitud entre el escenario base y el uso de swap a disco.

6.1.4. Ejecución con swap a ramdisk local

Figura 6.6: Tiempo agregado de las 4 operaciones por tamaño e hilos.



La ejecución en ramdisk, mostrada en la figura 6.5, recupera la pauta de ejecución en RAM, en la que a mayor grado de paralelismo se mejora el rendimiento, pero con unos tiempos mayores debidos a la gestión del área de intercambio.

Tabla 6.5: Tiempo agregado de las 4 operaciones con un tamaño de 9155,3 MB.

Hilos	1	4	8	16
Segundos	19449	11029	9235	7976

El tiempo obtenido (tabla 6.5) empeora un 15202 %, 27835 %, 25552 % y 23296 % para las ejecuciones de 1, 4, 8 y 16 hilos respecto al escenario base (una diferencia de dos órdenes de magnitud). Sin embargo, estos resultados no pueden compararse directamente con los del apartado anterior (tabla 6.4) por tratarse de tamaños de problema muy diferentes.

Tabla 6.6: Tiempo agregado de las 4 operaciones con un tamaño de 457,8 MB.

Hilos	1	4	8	16
Segundos	864,94	454,23	409,43	338,8

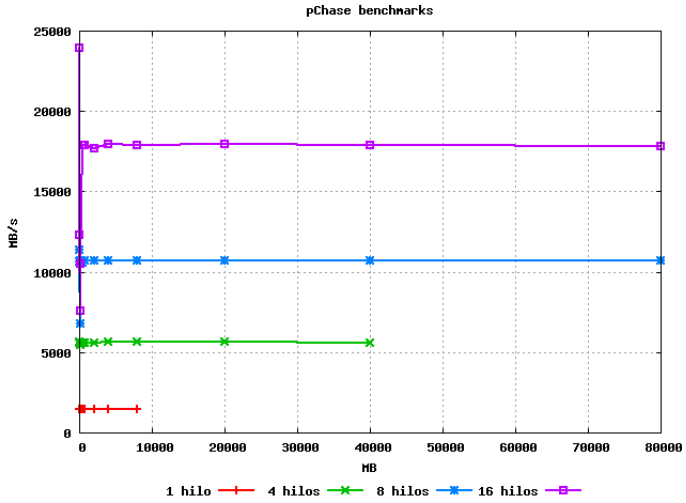
La tabla 6.6 contiene los tiempos empleados para un tamaño de 457,8 MB. Este tamaño es el siguiente más cercano al tamaño más grande utilizado en la ejecución con swap a disco, ya que los tamaños inferiores que se ejecutaron (en común) en ambas pruebas, corrieron en RAM y no se pueden utilizar. A pesar de que la ejecución con swap a ramdisk es bastante más lenta que la del escenario base, supera ampliamente a la ejecución con swap a disco, que es un 193 %, 689 %, 1038 % y 1993 % más lenta (para 1, 4, 8 y 16 hilos). La ejecución con swap a ramdisk es un orden de magnitud más rápida que con swap a disco para los tamaños comparados. Sin embargo para tamaños mayores la diferencia será mucho mayor, pues el rendimiento de la ejecución a ramdisk es casi lineal y el de la ejecución con swap a disco se incrementa en mayor proporción.

6.2. pChase

Se muestran a continuación los resultados de pChase, indicando el ancho de banda, el tiempo de latencia en el acceso a memoria y el tiempo total de las ejecuciones para cada uno de los escenarios.

6.2.1. Ejecución en RAM afín

Figura 6.7: Ancho de banda por tamaño e hilos.

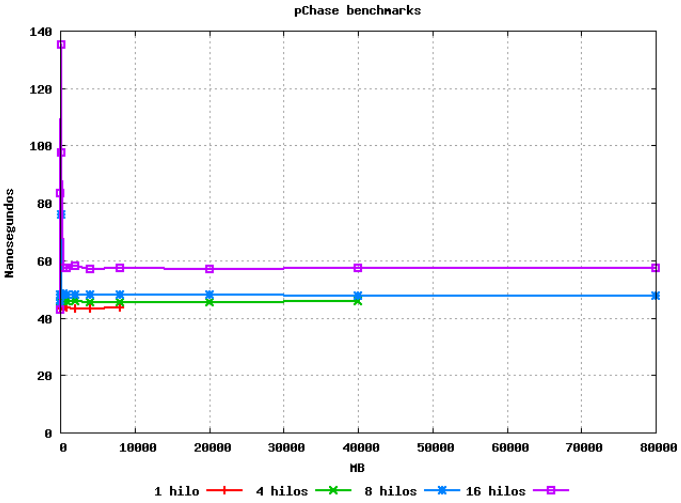


La figura 6.7 nos muestra el ancho de banda obtenido por pChase según el número de hilos de la ejecución y el tamaño del problema. Se aprecia que en los tamaños pequeños del problema el ancho de banda es variable y que a partir de los 8 GB de tamaño, el valor se estabiliza. Aunque las ejecuciones se hicieron con tamaños mucho mayores, el valor con el que trabajaremos es el de 8 GB por ser el máximo para el que terminó correctamente el programa (para la ejecución con 1 y 4 hilos, los tamaños superiores a 8 GB murieron, de ahí que no se muestren datos en la gráfica).

Tabla 6.7: Ancho de banda con un tamaño de 8 GB.

Hilos	1	4	8	16
MB/s	1470	5628	10672	17867

Figura 6.8: Latencia de memoria por tamaño e hilos.

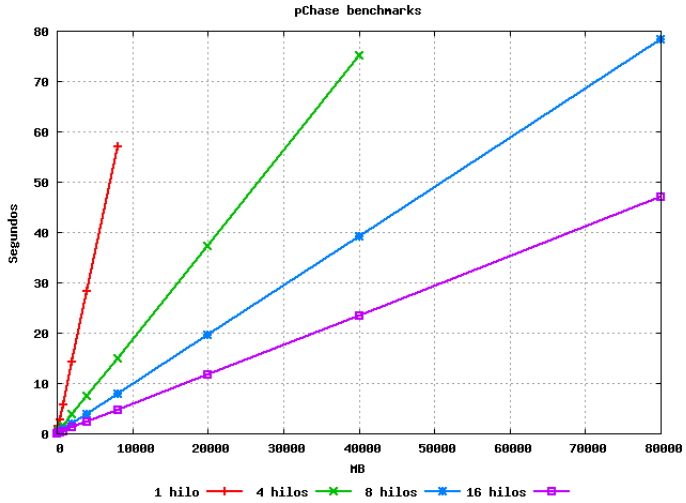


En la figura 6.8, que nos muestra el tiempo de acceso a memoria, observamos que el mínimo tiempo se obtiene en la ejecución secuencial (1 hilo) pues no hay ningún tipo de comptetición en el acceso. Para las ejecuciones paralelas, la latencia aumenta al incrementar el número de hilos, pues a mayor paralelismo, mayor competición por el acceso a la memoria. Al igual que en los resultados de ancho de banda, los datos fluctuan para los valores pequeños del problema.

Tabla 6.8: Latencia de memoria con un tamaño de 8 GB.

Hilos	1	4	8	16
nanosegundos	43,53	45,49	47,98	57,31

Figura 6.9: Tiempo por tamaño e hilos.



En la figura de tiempos 6.9 se aprecian curvas rectas para cada grado de paralelización, dadas por la estabilización del ancho de banda que se ha mencionado. Se pueden ver las diferentes pendientes que se obtienen según se ejecute con un número determinado de hilos.

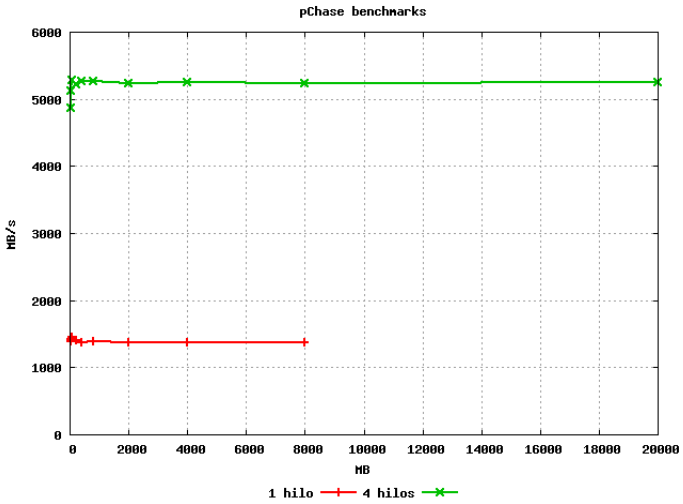
Tabla 6.9: Tiempo con un tamaño de 8 GB.

Hilos	1	4	8	16
Segundos	57,1	14,9	7,9	4,7

El tiempo se divide casi exactamente entre el número de hilos hasta llegar a dieciséis, valor para el cual baja la mejoría. La pérdida de mejoría es debida a la competición entre los diferentes hilos por el acceso a memoria y a los límites físicos de esta.

6.2.2. Ejecución en RAM no afín

Figura 6.10: Ancho de banda por tamaño e hilos.



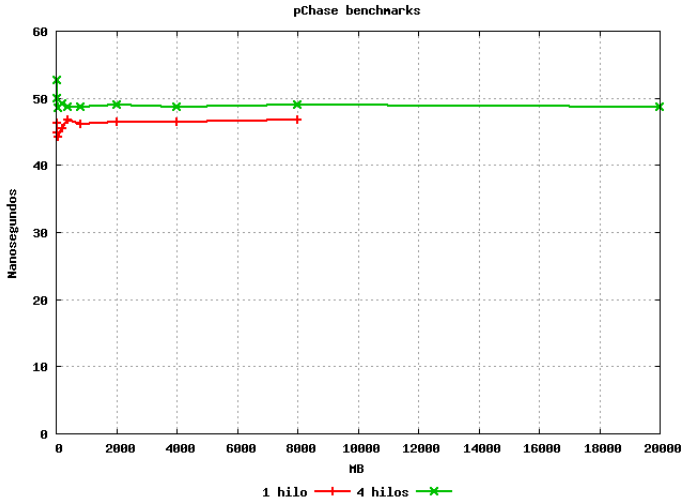
La figura 6.10 muestra que la proporción entre la ejecución con un hilo y la de cuatro es prácticamente la misma que en la ejecución en memoria RAM afín, sin embargo los valores absolutos en esta configuración son menores por el uso de memoria de otros procesadores.

Tabla 6.10: Ancho de banda con un tamaño de 8 GB.

Hilos	1	4	8	16
MB/s	1369,7	5229,8	-	-

El rendimiento obtenido con la ejecución secuencial es un 93,1 % del obtenido en la ejecución en el escenario base. Y el que hizo uso de cuatro hilos tuvo un rendimiento del 92,9 % respecto al escenario base. La pérdida es del 7,3 % y 7,6 % respectivamente.

Figura 6.11: Latencia de memoria por tamaño e hilos.



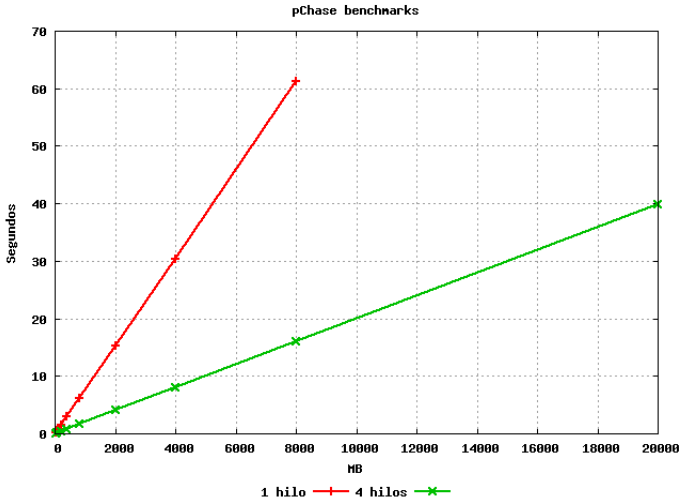
Al igual que con el ancho de banda, en la figura 6.11 se ve como la proporción entre la latencia de acceso a memoria obtenida con uno y cuatro hilos es la misma que en la ejecución del escenario base, pero las cifras absolutas han aumentado debido a la mayor distancia entre el procesador que ejecuta el proceso y la memoria que está utilizando.

Tabla 6.11: Latencia de memoria con un tamaño de 8 GB.

Hilos	1	4	8	16
nanosegundos	46,72	48,95	-	-

Tanto la ejecución secuencial como la que usó cuatro hilos tuvieron una pérdida de rendimiento en la latencia del 7 % (7,3 % y 7,6 % respectivamente) que coincide con los datos del ancho de banda.

Figura 6.12: Tiempo por tamaño e hilos.



Las rectas que muestra la figura 6.12 representan el ancho de banda constante obtenido, que hace que sea lineal la evolución del tiempo respecto al tamaño del problema.

Tabla 6.12: Tiempo con un tamaño de 8 GB.

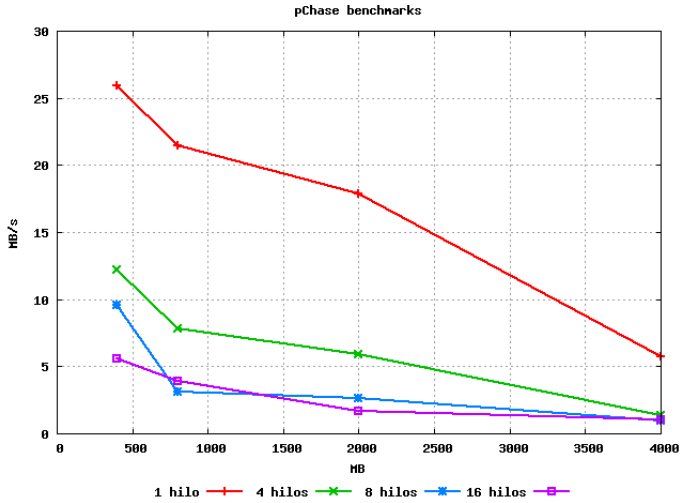
Hilos	1	4	8	16
segundos	61,2	16,0	-	-

De la misma forma que en los resultados de la ejecución del escenario base, puede verse en la tabla 6.12 que el tiempo obtenido se divide casi a la par con el número de hilos utilizados. Constatándose que la diferencia en el rendimiento viene toda de la mayor distancia a la que se encuentra la memoria utilizada. El empeoramiento de tiempos con uso de memoria de otros procesadores es de un 7 % (7,3 % y 7,6 % con uno y cuatro hilos respectivamente).

6.2.3. Ejecución con swap a disco

En la ejecución con swap a disco se utilizaron valores del problema de menor tamaño ya que el rendimiento obtenido con este tipo de memoria es mucho menor, y los tiempos de ejecución son muy elevados. Además, aunque también se ejecutaron pruebas de menor tamaño al que aparece en las gráficas, los resultados no se muestran por haberse ejecutado estas íntegramente en RAM y desvirtuar las gráficas con sus valores.

Figura 6.13: Ancho de banda por tamaño e hilos.



Como puede verse en la figura 6.13, a diferencia de la ejecución en RAM en la que el ancho de banda se mantiene estable, en este caso el ancho de banda va disminuyendo a medida que se incrementa el tamaño del problema. Aunque las especificaciones del disco muestran valores máximos (tabla 2.1) y ya se esperaba un valor real menor (en torno al 70 %), las cifras obtenidas distan mucho de este 70 %. No se puede achacar al disco tan bajo rendimiento.

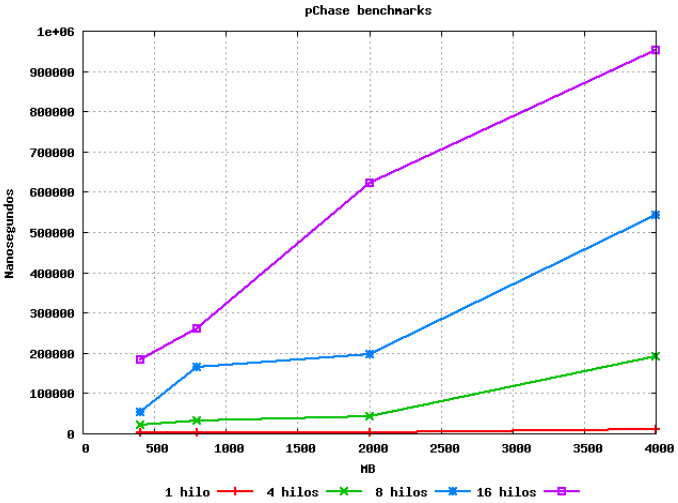
Otro punto a destacar es que en el escenario base el ancho de banda se incrementaba con el número de hilos y ahora se decrementa al aumentarlos.

Tabla 6.13: Ancho de banda con un tamaño de 4 GB.

Hilos	1	4	8	16
MB/s	5,7	1,3	0,9	1,0

En la tabla 6.13 se ve como en la ejecución secuencial (la más favorable) se consigue un 0,39 % del ancho de banda del escenario base, llegando a caer hasta 0,02 %, 0,009 % y 0,006 % con 4, 8 y 16 hilos respectivamente. La diferencia de resultados es de tres órdenes de magnitud respecto al escenario base para las ejecuciones con 1 y 4 hilos y de cuatro órdenes de magnitud para las de 8 y 16 debido a la tendencia a la baja del rendimiento al aumentar el número de hilos en este escenario.

Figura 6.14: Latencia de memoria por tamaño e hilos.



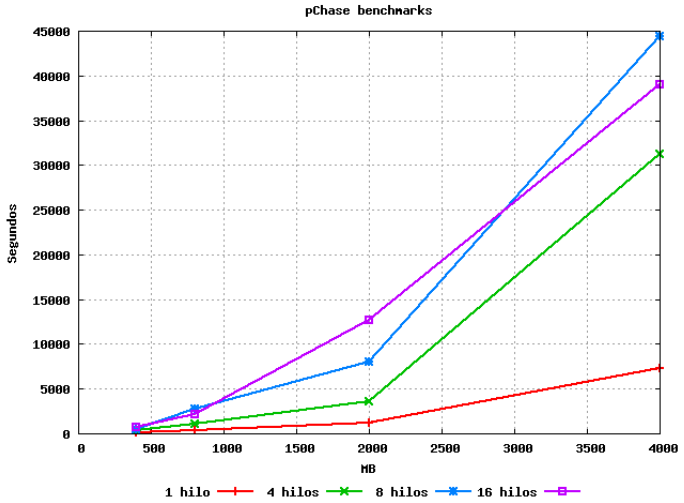
En la figura 6.14 se aprecia como a la par que el ancho de banda disminuye al aumentar el tamaño y número de hilos, la latencia aumenta. La sobrecarga en esta configuración viene dada por la gestión de la swap por parte del kernel y el acceso a disco. Incluso con el reordenamiento de las peticiones que hace el disco duro autónomamente, se incrementan los tiempos de acceso al aumentar el número de peticiones.

Tabla 6.14: Latencia de memoria con un tamaño de 4 GB.

Hilos	1	4	8	16
nanosegundos	11159,66	191004,70	542157,06	951474,49

La tabla 6.14 contiene la latencia de las ejecuciones con un tamaño de 4 GB. Sólo en el caso más favorable ya se empeora el rendimiento más de un 25000 %, siendo la diferencia de tres órdenes de magnitud y llegando a cuatro en el resto de casos.

Figura 6.15: Tiempo por tamaño e hilos.



En la figura 6.15 se puede ver como el tiempo obtenido aumenta con el tamaño y el número de hilos utilizados (pues disminuye el ancho de banda y aumenta la latencia). Si bien habría que tener más valores intermedios entre los 2 y 4 GB para entender mejor el comportamiento de la ejecución de 8 hilos, se establece una pauta clara de empeoramiento con tamaños grandes. Recordar que el tamaño de la RAM total era de 256 MB de los que unos 90 MB estaban ocupados por el sistema operativo. Con tamaños muy grandes del problema hay que estar constantemente intercambiando páginas entre memoria y la swap en disco que son devueltas a disco antes de que se terminen de utilizar, y esto provoca una caida del rendimiento muy acentuada.

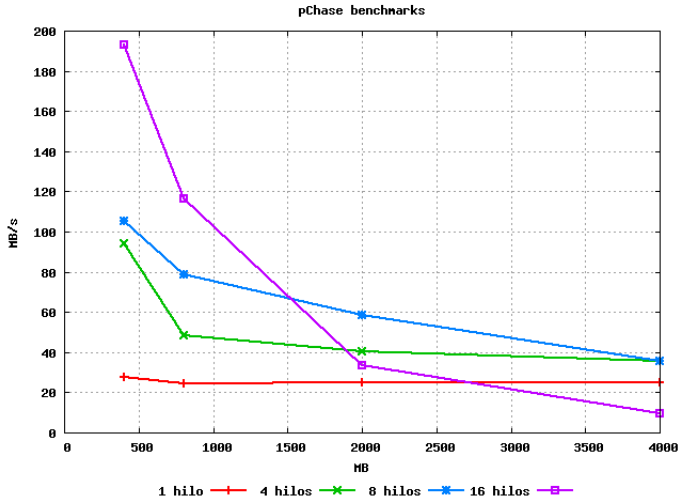
Tabla 6.15: Tiempo con un tamaño de 4 GB.

Hilos	1	4	8	16
segundos	7313,6	31294,2	44413,5	38972,4

La ralentización de tiempo con uso de swap a disco duro que refleja la tabla 6.15 supera el 25000 % en la ejecución secuencial y 400000 %, 1000000 %, 1600000 % para el resto. Equivale a una diferencia de dos órdenes de magnitud con respecto al escenario base para el caso más favorable (el secuencial) y de cuatro órdenes de magnitud para el resto, ya que al incrementar el número de hilos de ejecución simultanea en el escenario base se obtiene mejor rendimiento y en este se empeora. Estos tiempos tan elevados se deben al fenómeno de thrashing que se produce con tamaños grandes.

6.2.4. Ejecución con swap a ramdisk local

Figura 6.16: Ancho de banda por tamaño e hilos.



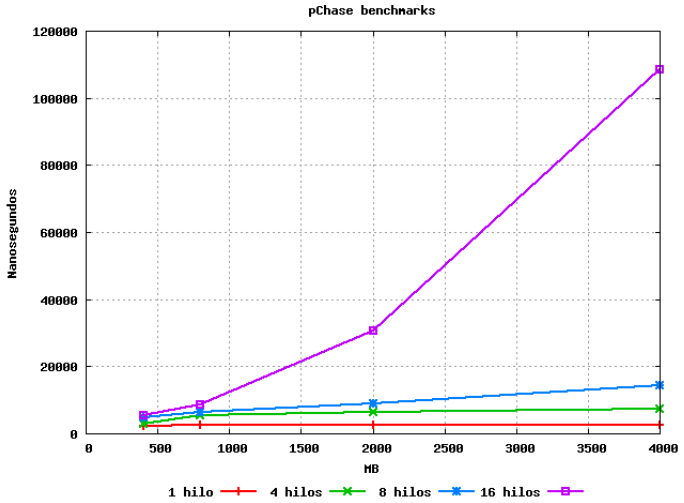
En la figura 6.16 se muestra el ancho de banda obtenido en la ejecución con swap a ramdisk. En ella se puede observar que el único caso en el que el ancho de banda se mantiene constante como en el escenario base, es el de un único hilo. En tamaños pequeños del problema el rendimiento aumenta con el número de hilos de igual manera que sucece en el escenario base, pero a medida que se incrementa el tamaño, el rendimiento decrece más acentuadamente cuanto mayor es el número de hilos en ejecución paralela.

Tabla 6.16: Ancho de banda con un tamaño de 4 GB.

Hilos	1	4	8	16
MB/s	24,878	35,568	35,507	9,427

La tabla 6.16 muestra el ancho de banda obtenidos para un tamaño de 4 GB. La ejecución secuencial obtiene un 1,7 % del ancho de banda obtenido con el escenario base y las paralelas de 4, 8 y 16 hilos obtienen un 0,6 %, 0,3 % y 0,05 % respectivamente. Aunque a la hora de comparar con el escenario base tomásemos los anchos de banda para un tamaño de 800 MB, el tanto por ciento obtenido no variaría mucho. Al tener la ejecución secuencial un ancho de banda sostenido, se puede asignar a la gestión de la swap y del ramdisk toda la sobrecarga observada.

Figura 6.17: Latencia de memoria por tamaño e hilos.



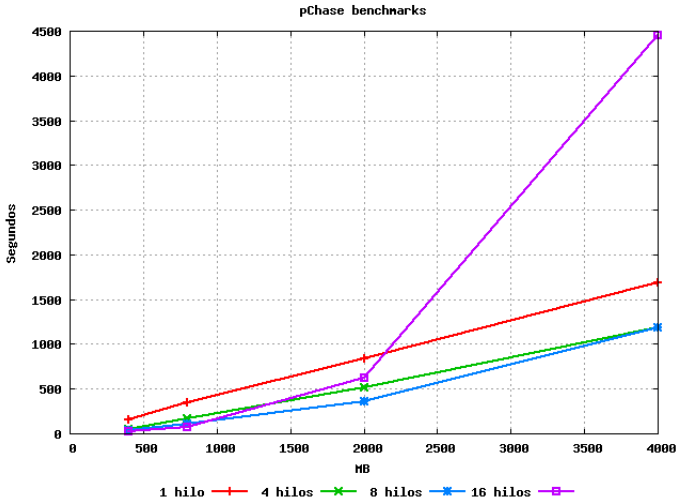
Los tiempos de latencia mostrados en la figura 6.17 se incrementan con el número de hilos de la ejecución, de manera similar a como ocurre en el escenario base. Pero en este escenario también aumenta el tiempo a la vez que aumenta el tamaño del problema, de forma más acentuada cuanto mayor es el nivel de paralelización.

Tabla 6.17: Latencia de memoria con un tamaño de 4 GB.

Hilos	1	4	8	16
nanosegundos	2572,60	7197,52	14419,53	108627,11

La tabla 6.17 muestra los tiempos de latencia para un tamaño de 4 GB. La pérdida de rendimiento en comparación con el escenario base es del 5838 %, 15725 %, 30009 % y 190406 % para las ejecuciones con 1, 4, 8 y 16 hilos respectivamente. La diferencia de rendimiento va desde dos órdenes de magnitud con 1 y 4 hilos hasta tres y cuatro órdenes de magnitud para 8 y 16 hilos.

Figura 6.18: Tiempo por tamaño e hilos.



En la figura 6.18 aparecen los tiempos de ejecución para distinto número de hilos y tamaño del problema. Se aprecia el incremento lineal de los tiempos de la ejecución secuencial y la casi lineal en el caso de paralelización con 4 hilos. El tiempo consumido aumenta en mayor proporción que el tamaño al incrementarse el número de hilos usados. Esta sobrecarga se debe a la competición de los diferentes hilos de ejecución por los recursos de la máquina.

Tabla 6.18: Tiempo con un tamaño de 4 GB.

Hilos	1	4	8	16
segundos	1686,0	1179,2	1181,2	4449,4

El tiempo de ejecución que aparece en la tabla 6.18 para un tamaño de 4 GB muestra unos porcentajes de rendimiento, respecto al escenario base, casi iguales a los obtenidos en el tiempo de latencia. La diferencia de rendimiento respecto al escenario base es de dos órdenes de magnitud en el caso de la ejecución secuencial y de tres órdenes de magnitud en las ejecuciones paralelas.

6.3. Gromacs

Se muestran a continuación los tiempos de los resultados de los cuatro benchmarks ejecutados con Gromacs. Excepto en el primer escenario, en el que la aplicación disponía de toda la memoria, en el resto se inició la máquina con 128 MB de RAM para forzar el uso del área de intercambio.

6.3.1. Ejecución en RAM afín

De mayor a menor, vemos en las figuras 6.19 y 6.20 como todos los benchmarks paralelizan bien.

Figura 6.19: Tiempo de ejecución de d.dppc y d.lzm.

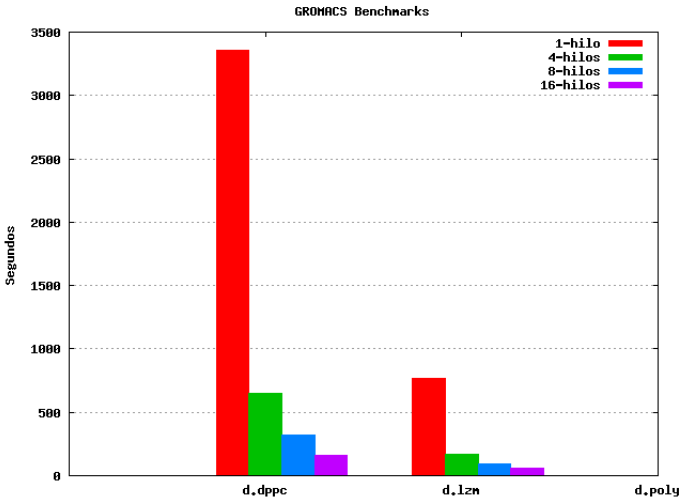


Tabla 6.19: Tiempo de ejecución de d.dppc y d.lzm.

Hilos	1	4	8	16
segundos d.dppc	3356,0	645,4	323,6	164,0
segundos d.lzm	770,0	168,8	91,6	59,9

Estudiando los tiempos de la tabla 6.19 se observa que, entre el tiempo de la ejecución secuencial y las paralelas, el decremento no es lineal con el número de hilos utilizados, sinó que es mayor. Si calculamos las aceleraciones que se producen al ejecutar d.dppc:

$$S_4 = \frac{t_1}{t_4} = \frac{3356,0}{645,4} = 5,2 > 4$$

$$S_8 = \frac{t_1}{t_8} = \frac{3356,0}{323,6} = 10,4 > 8$$

$$S_{16} = \frac{t_1}{t_{16}} = \frac{3356,0}{164,0} = 20,5 > 16$$

Se puede ver claramente que se trata de una aceleración superlineal posiblemente causada por la influencia de la memoria caché. En el caso de d.lzm se produce el mismo efecto $S_4 = 4,6$, $S_8 = 8,4$ y $S_{16} = 12$, pero al trabajar con un tamaño menor de memoria el efecto es menor y en la ejecución con 16 hilos la aceleración ya es normal (sublineal).

Figura 6.20: Tiempo de ejecución de d.poly y d.villin.

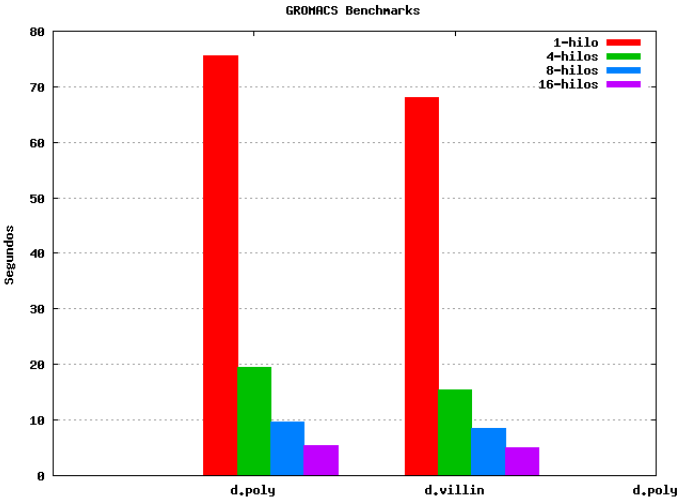


Tabla 6.20: Tiempo de ejecución de d.poly y d.villin.

Hilos	1	4	8	16
segundos d.poly	75,5	19,4	9,6	5,5
segundos d.villin	68,1	15,3	8,5	5,0

La tabla 6.20 recoge el tiempo de ejecución de d.poly y d.villin. En el caso de d.poly la aceleración para 4 y 8 hilos roza el máximo y con 16 ya baja a un valor de 13,8, pero con d.villin se vuelve a producir un efecto de aceleración superlineal con las ejecuciones de 4 y 8 hilos, que se normaliza en la de 16 hilos.

6.3.2. Ejecución en RAM no afín

Figura 6.21: Tiempo de ejecución de d.dppc y d.lzm.

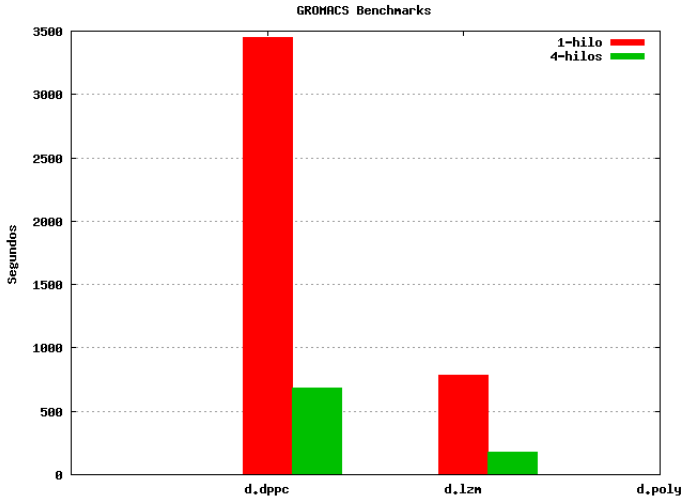


Tabla 6.21: Tiempo de ejecución de d.dppc y d.lzm.

Hilos	1	4	8	16
segundos d.dppc	3446,4	682,3	-	-
segundos d.lzm	783,0	180,8	-	-

En la tabla 6.21 se muestran los tiempos de d.dppc y d.lzm al usar memoria no afín. El tiempo de ejecución secuencial de d.dppc usando memorias de otros procesadores es un 2.7 % mayor que la ejecucicón del escenario base. Con la ejecución paralela que usa 4 hilos, el rendimiento baja más, llegando a ser un 5,7 % más lenta que en la ejecución del escenario base.

Similares resultados se obtienen con d.lzm, siendo un 1,6 % y un 7,1 % más lentas en este escenario que en el escenario base las ejecuciones de 1 y 4 hilos.

Figura 6.22: Tiempo de ejecución de d.poly y d.villin.

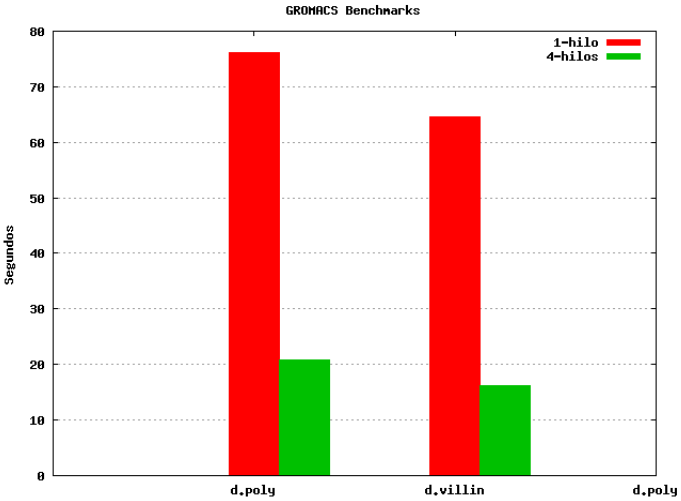


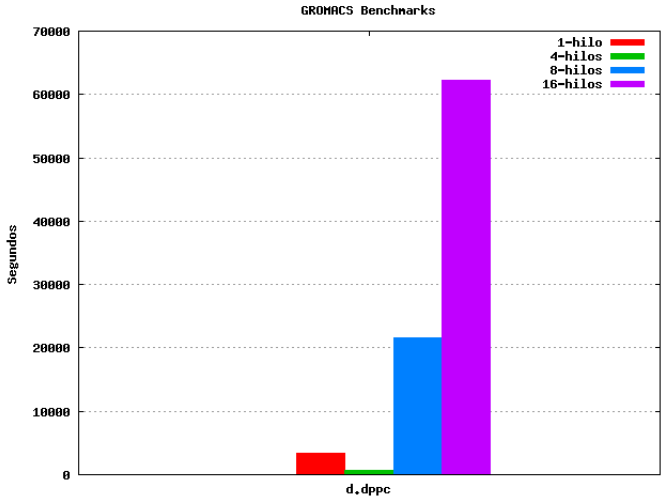
Tabla 6.22: Tiempo de ejecución de d.poly y d.villin.

Hilos	1	4	8	16
segundos d.poly	76,2	20,9	-	-
segundos d.villin	64,5	16,1	-	-

En la tabla 6.22 se recogen los tiempos representados en la figura 6.22. Analizándolos, se observa que la prueba d.poly ha incrementado sus tiempos de ejecución un 1,0 % y un 7,5 % con 1 y 4 hilos respectivamente, comparados con los del escenario base. Los tiempos obtenidos en las ejecuciones de d.villin son mayores en la ejecución secuencial y menores en la ejecución paralela que los respectivos del escenario base. Tras comparar los tiempos de esta prueba (d.villin) en todos los escenarios, se comprobó que, debido a su reducido tamaño, se ha ejecutado en memoria RAM y no ha llegado a usar el área de intercambio en ninguna de las ocasiones por lo que no se ha usado para más comparaciones.

6.3.3. Ejecución con swap a disco

Figura 6.23: Tiempo de ejecución de d.dppc.



En la figura 6.23 se muestra el comportamiento del benchmark d.dppc utilizando swap a disco duro, en ella se comprueba como hasta con 4 hilos de ejecución paralela la aplicación mejora el rendimiento, pero a partir de esa cifra, la competición por los recursos entre los diferentes hilos hace que el tiempo aumente exponencialmente (efecto thrashing).

Tabla 6.23: Tiempo de ejecución de d.dppc.

Hilos	1	4	8	16
segundos	3418,0	665,7	21575,6	62244,0

Los tiempos recogidos en la tabla 6.23 indican que el tiempo de ejecución al utilizar disco ha aumentado un 1,8 % y un 3,2 % en las ejecuciones de 1 y 4 hilos lo que hace suponer que el uso de la swap ha sido mínimo. Las ejecuciones de 8 y 16 hilos han hecho un uso mucho mayor del área de intercambio y sus tiempos han aumentado en dos órdenes de magnitud respecto a los obtenidos en el escenario base.

Figura 6.24: Tiempo de ejecución de d.lzm.

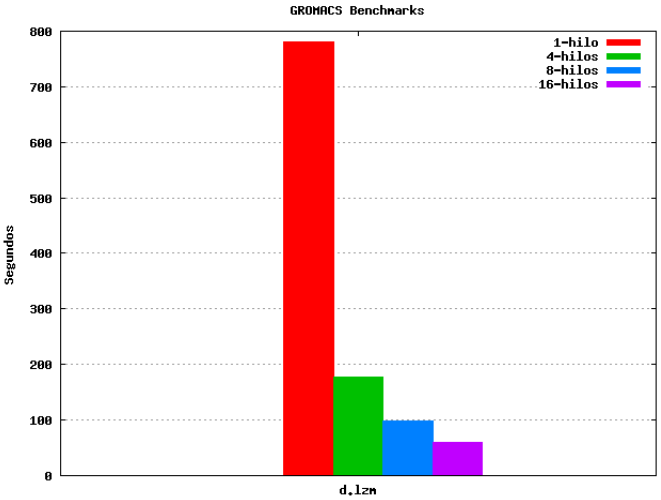


Tabla 6.24: Tiempo de ejecución de lzm.

Hilos	1	4	8	16
segundos	779,9	176,7	98,8	60,5

El tiempo de ejecución de la prueba d.lzm, representado en la figura 6.24 y mostrado en la tabla 6.24, indica un leve descenso del rendimiento de 1,2 %, 4,6 %, 7,8 y 1,0 % para las ejecuciones de 1, 4, 8 y 16 hilos con respecto a la ejecución del escenario base y hacen pensar que no han llegado a utilizar el área de intercambio o si lo han hecho ha sido de manera marginal.

Figura 6.25: Tiempo de ejecución de d.poly y d.villin.

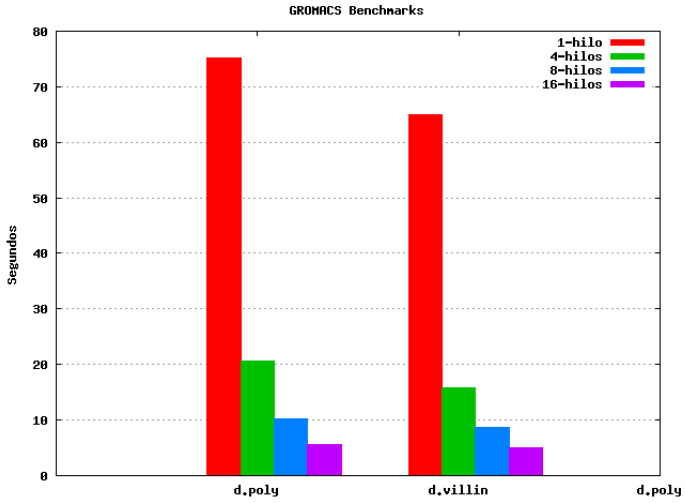


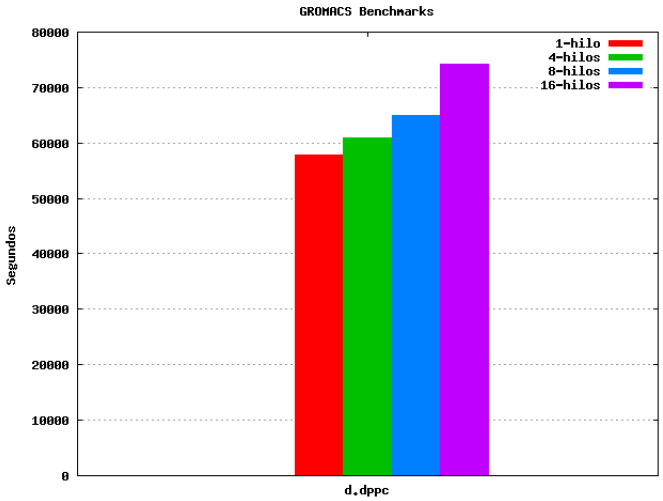
Tabla 6.25: Tiempo de ejecución de d.poly y d.villin.

Hilos	1	4	8	16
segundos d.poly	75,1	20,6	10,2	5,7
segundos d.villin	65,0	15,8	8,7	4,9

Los resultados de las ejecuciones de d.poly y d.villin mostrados en la figura 6.25 y recogidos en la tabla 6.25 indican que no han utilizado el área de intercambio en disco.

6.3.4. Ejecución con swap a ramdisk remoto

Figura 6.26: Tiempo de ejecución de d.dppc.



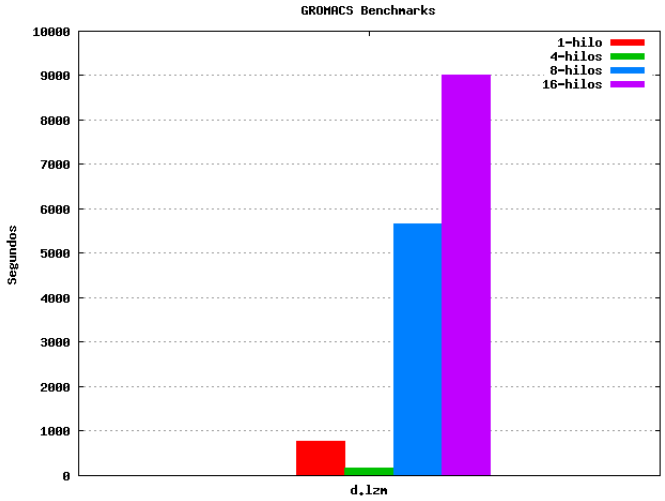
En la figura 6.26 se puede observar el cambio de comportamiento de la aplicación al usar el ramdisk remoto como área de intercambio. No sólo se incrementan abundantemente los tiempos de ejecución sino que las ejecuciones paralelas, en vez de mejorar los tiempos, empeoran respecto a la secuencial. El NBD serializa las peticiones, lo que no sólo elimina la posibilidad de mejora mediante paralelización, sino que provoca que el usar varios hilos frene la ejecución.

Tabla 6.26: Tiempo de ejecución de d.dppc.

Hilos	1	4	8	16
segundos	57746,2	60985,2	64962,5	74299,5

Los tiempos de la tabla 6.26 muestran la pérdida de rendimiento obtenida. La ejecución secuencial tarda un 1620 % más que la del escenario base (un órden de magnitud más). Como las ejecuciones paralelas empeoraron respecto a la secuencial, la perdida de rendimiento en ellas respecto al escenario base es mucho mayor. La ejecución con 4 hilos es un 9349 % más lenta, la de 8 hilos un 20074 % y la de 16 hilos llega al 45299 %, todas con dos órdenes de magnitud de diferencia respecto al escenario base.

Figura 6.27: Tiempo de ejecución de lzm.



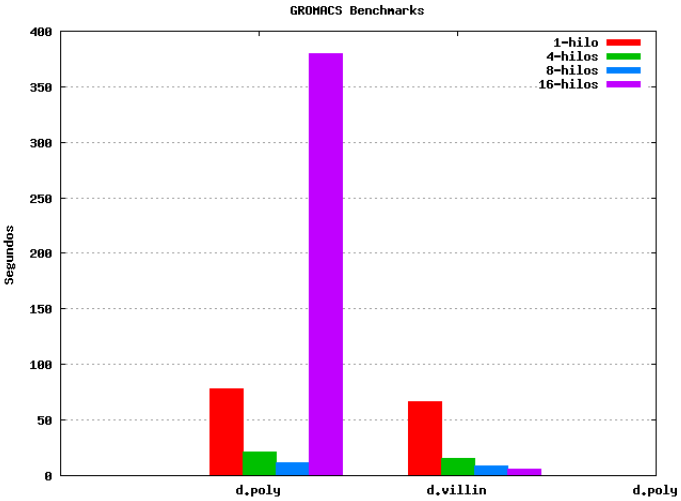
En la figura 6.27 se muestra el comportamiento de la aplicación al ejecutar d.lzm. Las ejecuciones con 1 y 4 hilos parecen ejecutarse en RAM, mientras que las de 8 y 16 (que tienen mayor consumo de memoria) hacen uso de la swap remota e incrementan sus tiempos considerablemente. Notar que en el escenario que usa disco como swap, representado en la figura 6.24, no se llegó a utilizar la swap y en este escenario sí. Esta diferencia del comportamiento de la misma prueba en similares condiciones puede deberse al orden de ejecución de las aplicaciones desde el arranque del sistema, pues la primera aplicación que necesite más memoria que la disponible en el sistema provoca la migración de páginas de otros programas en ejecución al área de swap y la siguiente aplicación que se ejecute ya tendrá esa memoria disponible sin tener que esperar a la migración. Si la prueba consumiese grandes cantidades de memoria, el uso del área de intercambio se daría en todas las ocasiones, pero si las necesidades de memoria no son muy grandes, pueden darse situaciones como la comentada.

Tabla 6.27: Tiempo de ejecución de d.lzm.

Hilos	1	4	8	16
segundos	780,9	173,0	5666,7	9006,0

Aunque los tiempos de ejecución (tabla 6.27) son mayores que los del escenario base para cualquier nivel de paralelismo utilizado, el aumento de tiempo con 1 y 4 hilos es mínimo, y si se ha llegado a usar el área de intercambio, su uso ha sido marginal. Con 8 y 16 hilos sí se aprecia la influencia del swap remoto alcanzando tiempos de ejecución de dos órdenes de magnitud mayores.

Figura 6.28: Tiempo de ejecución de d.poly y d.villin.



La figura 6.28 muestra que en el caso de d.poly y d.villin sólo una de las ejecuciones de d.poly hizo uso del área de intercambio. El resto de ejecuciones no hace uso suficiente de la memoria como para llegar a utilizar la swap.

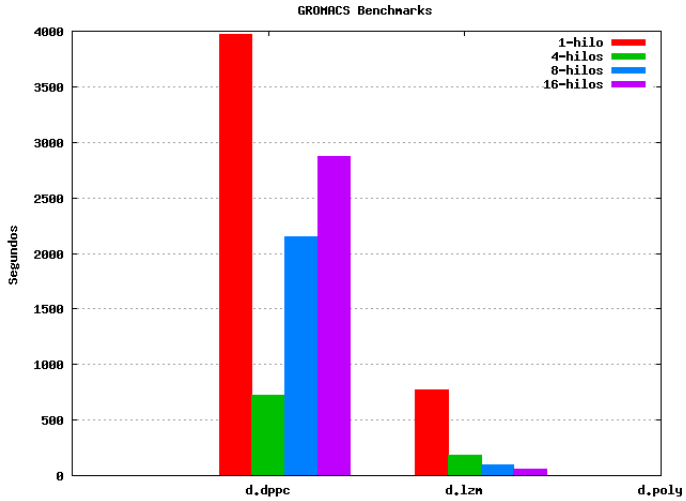
Tabla 6.28: Tiempo de ejecución de d.poly.

Hilos	1	4	8	16
segundos	78,0	21,1	11,2	380,1

Viendo los tiempos de la tabla 6.28, se aprecian incrementos respecto al escenario base para todas las ejecuciones, pero el tamaño de memoria que utilizan las tres primeras es pequeño, y no se puede asociar la diferencia de tiempo al uso del área de intercambio. Al usar 16 hilos de ejecución sí se utiliza el ramdisk remoto y el tiempo obtenido es dos órdenes de magnitud mayor.

6.3.5. Ejecución con swap a ramdisk local

Figura 6.29: Tiempo de ejecución de d.dppc y d.lzm.



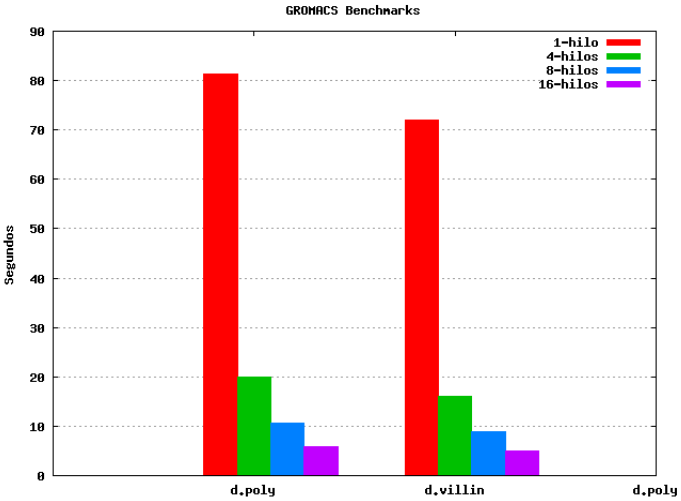
Se puede ver en la figura 6.29, como las ejecuciones con mayor número de hilos de d.dppc hacen un uso claro del área de intercambio, incrementando el tiempo de ejecución. El resto sigue un patrón de ejecución en RAM con tiempos menores al aumentar el número de hilos.

Tabla 6.29: Tiempo de ejecución de d.dppc y d.lzm.

Hilos	1	4	8	16
segundos d.dppc	3971,0	724,5	2148,7	2875,7
segundos d.lzm	767,1	179,3	100,5	61,8

Al analizar un poco más el escenario, y comparar los tiempos obtenidos (tabla 6.29) con los del escenario base, se comprueba que todas las ejecuciones de d.dppc han aumentado el tiempo. Las ejecuciones de 1 y 4 hilos, utilizando minimamente el ramdisk, han tardado un 18 % y un 12 % más. Las de 8 y 16 hilos ya han hecho un uso más intenso del área de intercambio y sus tiempos de ejecución han sido un orden de magnitud mayores que en el escenario base. Los tiempos de d.lzm son de ejecución en RAM.

Figura 6.30: Tiempo de ejecución de d.poly y d.villin.



Según se aprecia en la figura 6.30 y más tarde se verifica en la tabla de tiempos 6.30, ambas pruebas se han ejecutado en RAM por sus reducidos tamaños.

Tabla 6.30: Tiempo de ejecución de d.poly y d.villin.

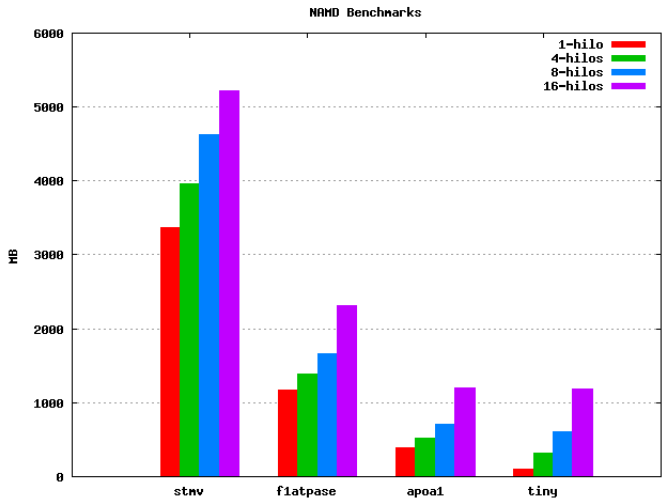
Hilos	1	4	8	16
segundos d.poly	81,2	19,9	10,6	5,9
segundos d.villin	71,9	16,1	8,8	5,0

6.4. NAMD

Se muestran a continuación los resultados de los benchmarks de NAMD. Al igual que en el caso de Gromacs, se inició la máquina con menos memoria para forzar el uso del área de intercambio. Pero en este caso, se varió el tamaño de memoria disponible dependiendo del test a realizar, para disminuir la cantidad de swap que iba a usar y así evitar tiempos de ejecución demasiado grandes. Debido a estas variaciones en la cantidad de memoria, nos podemos fijar en el comportamiento general, pero no establecer comparativas directas entre las pruebas.

6.4.1. Ejecución en RAM afín

Figura 6.31: Memoria utilizada.



Al igual que se ha visto anteriormente con Gromacs, la figura 6.31 muestra la pauta clara de mayor consumo de memoria al aumentar el nivel de paralelización de las aplicaciones. La información de consumo de memoria es la proporcionada por el propio programa.

Figura 6.32: Tiempo de ejecución de stmv y f1atpase.

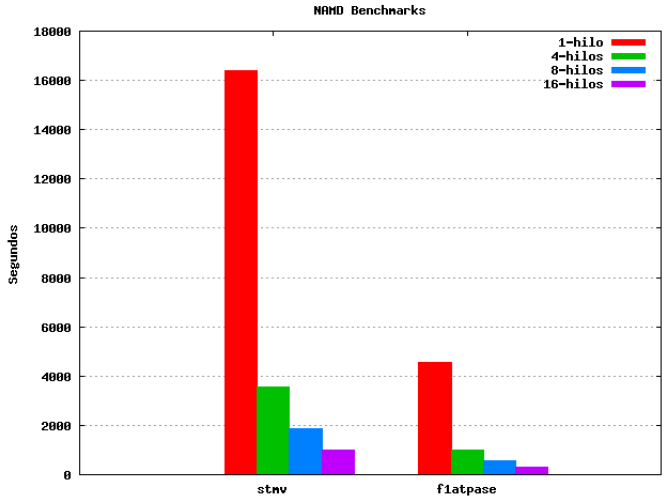


Tabla 6.31: Tiempo de ejecución de stmv y f1atpase.

Hilos	1	4	8	16
segundos stmv	16387,8	3541,2	1847,8	979,7
segundos f1atpase	4533,8	999,6	545,1	297,1

Analizando los tiempos de la tabla 6.31 se observa de nuevo el mismo comportamiento de aceleración superlineal entre la ejecución secuencial y las paralelas de ambas pruebas, y se afianza el motivo de la influencia de la memoria caché.

Figura 6.33: Tiempo de ejecución de apoa1.

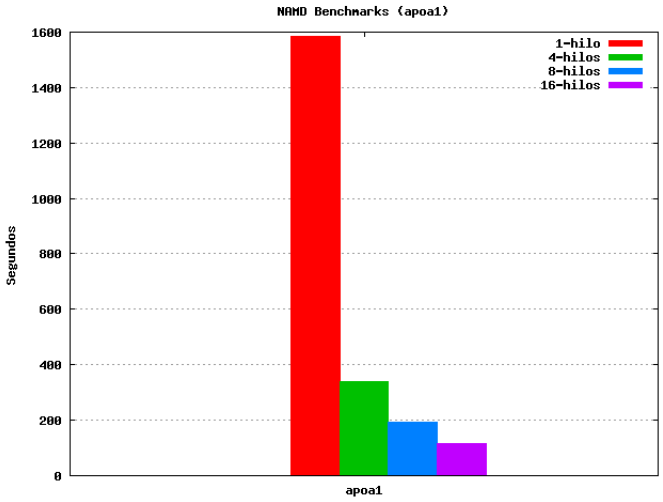


Tabla 6.32: Tiempo de ejecución de apoa1.

Hilos	1	4	8	16
segundos apoa1	1583,6	339,2	191,6	116,7
segundos tiny	15,2	11,8	13,5	11,1

Los tiempos mostrados en la tabla 6.32 que se encuentran representados en la figura 6.33, indican que la aceleración de apoa1 al paralelizar con 4 y 8 hilos es superlineal, y desciende a límites sublineales al usar 16 hilos. Con la prueba tiny, en cambio, no se aprecia mejora a la hora de paralelizar. Dos factores influyen en este resultado: el poco tiempo de cómputo que utiliza, y el gran aumento de memoria que necesita al incrementar el número de hilos.

6.4.2. Ejecución en RAM no afín

Figura 6.34: Tiempo de ejecución de stmv y f1atpase.

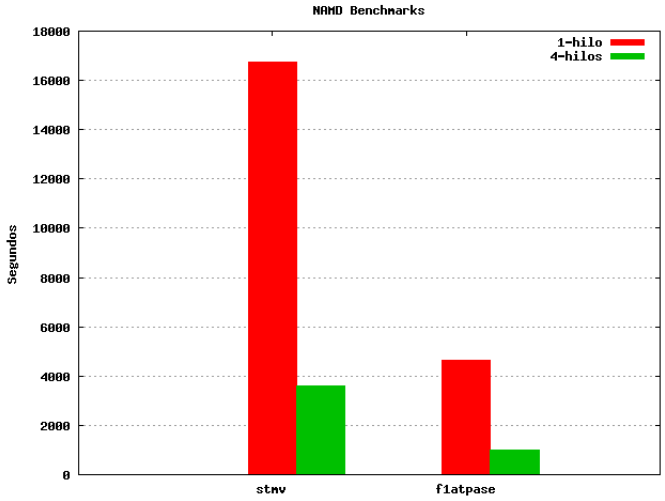


Tabla 6.33: Tiempo de ejecución de stmv y f1atpase.

Hilos	1	4	8	16
segundos stmv	16737,7	3600,5	-	-
segundos f1atpase	4649,2	1014,2	-	-

Los tiempos mostrados en la tabla 6.33, indican un ralentizamiento de la ejecución de stmv al usar memoria de otros procesadores en un 2,1 % y un 1,6 % para 1 y 4 hilos respecto a la ejecución del escenario base. La ejecución de f1atpase se ralentiza un 2,5 % y 1,5 % para 1 y 4 hilos respectivamente.

Figura 6.35: Tiempo de ejecución de apoa1.

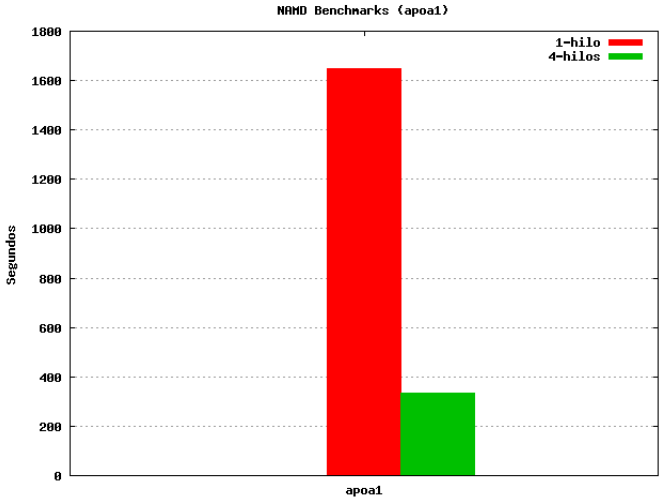


Tabla 6.34: Tiempo de ejecución de apoa1.

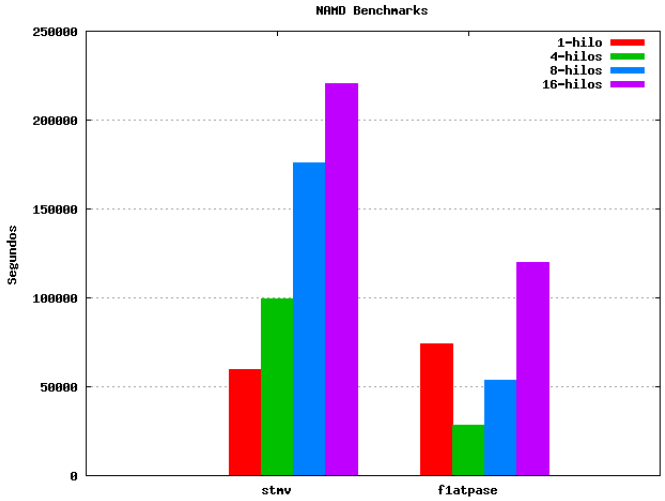
Hilos	1	4	8	16
segundos apoa1	1647,1	333,6	-	-

La figura 6.35 representa los tiempos de ejecución de la prueba apoa1 mostrados en la tabla 6.34. En la ejecución secuencial, la prueba se ralentizó un 4,0 % respecto al escenario base. En la ejecución paralela, el tiempo obtenido fue menor que en el escenario base. Esto puede haber sido debido a que algún hilo se ejecutase en el procesador incorrecto, o que la ejecución del escenario base resultase ralentizada por algún motivo externo.

6.4.3. Ejecución con swap a disco

Para la ejecución con swap a disco se varió la cantidad de memoria de la máquina según la prueba a ejecutar. Para stmv se usaron 2,5 GB de memoria RAM, para f1atpase, 768 MB y para apoa1, 256 MB.

Figura 6.36: Tiempo de ejecución de stmv y f1atpase.



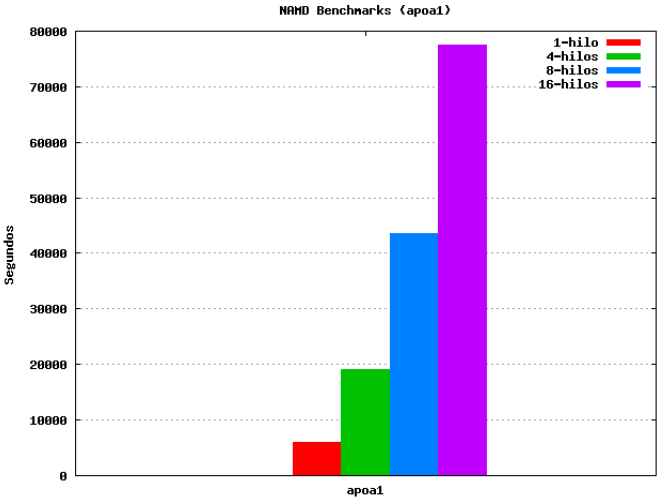
Como puede verse en la figura 6.36, el tiempo de ejecución al usar disco como área de intercambio aumenta considerablemente en todas las ejecuciones, y mientras con stmv se incrementa siempre con el número de hilos usados, en f1atpase se puede observar la mejoría de la paralelización en el caso de 4 y 8 hilos.

Tabla 6.35: Tiempo de ejecución de stmv y f1atpase.

Hilos	1	4	8	16
segundos stmv	59377,1	99553,6	175949,3	220359,5
segundos f1atpase	74139,6	28235,9	53613,6	119971,2

La tabla 6.35 muestra el tiempo de cada una de las ejecuciones de las pruebas stmv y f1atpase, con disco como área de intercambio. stmv tarda un 362 % más que en el escenario base en su ejecución secuencial y en las ejecuciones paralelas, la diferencia de tiempos es de uno, dos y tres órdenes de magnitud respectivamente. f1atpase aumenta el tiempo de su ejecución secuencial un 1635 % respecto al escenario base (un órden de magnitud de diferencia) y aunque las ejecuciones paralelas de 4 y 8 hilos mejoren el tiempo obtenido por la secuencial, marcan una diferencia de dos órdenes de magnitud respecto a las del escenario base. La ejecución de 16 hilos establece un tiempo de ejecución de tres órdenes de magnitud más que en el escenario base.

Figura 6.37: Tiempo de ejecución de apoa1.



La ejecución de la prueba apoa1 con swap a disco, representada en la figura 6.37, muestra la inversión del comportamiento obtenido en el escenario base al utilizar paralelización.

Tabla 6.36: Tiempo de ejecución de apoa1.

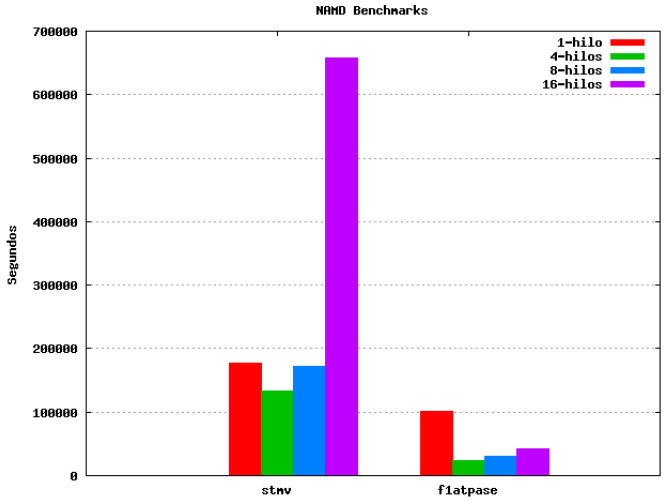
Hilos	1	4	8	16
segundos apoa1	5918,5	19096,5	43524,0	77570,7

En la tabla 6.36 aparecen los tiempos de ejecución de apoa1 utilizando swap a disco. En la ejecución secuencial, el incremento de tiempo es de un 373,7 % respecto a la realizada con el escenario base. El resto de ejecuciones paralelas incrementan los tiempos del escenario base en dos órdenes de magnitud. Siendo mayor el incremento cuanto mayor es la paralelización.

6.4.4. Ejecución con swap a ramdisk remoto

Las ejecuciones utilizando un ramdisk de una máquina remota como área de intercambio se realizaron todas con 256 MB de memoria RAM en la máquina en la que corrieron las pruebas.

Figura 6.38: Tiempo de ejecución de stmv y f1atpase.



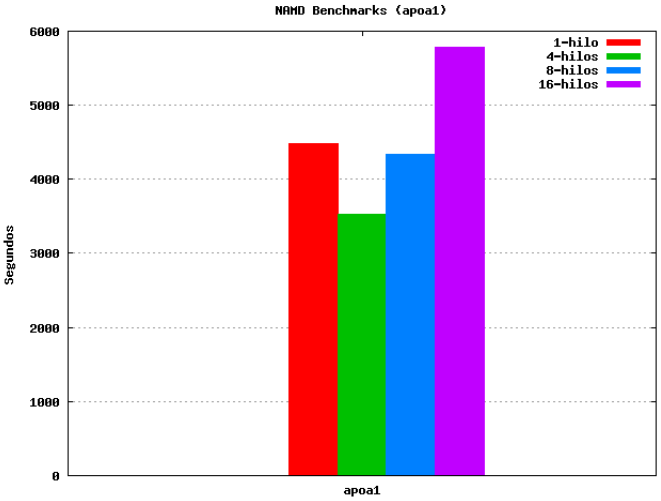
La figura 6.38 representa los tiempos de ejecución de las pruebas stmv y f1atpase cuando utilizan un ramdisk de una máquina remota para establecer el área de intercambio. Se aprecia una reducción del tiempo al utilizar varios hilos de ejecución respecto a la ejecución secuencial. A excepción de la prueba de stmv con mayor consumo de RAM (la de 16 hilos), que incrementa el tiempo casi en un factor cuatro, el resto de ejecuciones paralelas necesitan menos tiempo para completarse que la secuencial.

Tabla 6.37: Tiempo de ejecución de stvm y f1atpase.

Hilos	1	4	8	16
segundos stmv	177073,0	133283,2	172170,4	657076,4
segundos f1atpase	100377,0	23544,3	30606,5	42809,4

Los tiempos que aparecen en la tabla 6.37 indican uno, dos, dos y tres órdenes de magnitud de diferencia para las ejecuciones de stmv respecto al escenario base. La prueba f1atpase mantiene un incremento de tiempo de dos órdenes de magnitud en todas sus ejecuciones.

Figura 6.39: Tiempo de ejecución de apoa1.



En la figura 6.39 vemos que, con un comportamiento similar a stmv, apoa1 reduce el tiempo de ejecución al utilizar paralelización y pasa a incrementarlo cuando el número de hilos (y por lo tanto la memoria) aumenta.

Tabla 6.38: Tiempo de ejecución de apoa1.

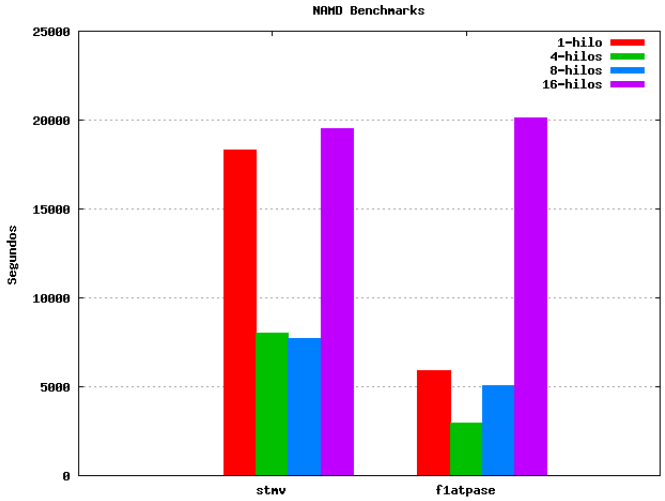
Hilos	1	4	8	16
segundos apoa1	4484,0	3529,2	4342,2	5781,5

En la tabla 6.38 se pueden ver los tiempos de ejecución de la prueba apoa1 al utilizar ramdisk remoto como área de intercambio. En la ejecución secuencial, el tiempo se incrementa un 283,1 % respecto al escenario base y en las ejecuciones paralelas, el incremento es de un orden de magnitud en todas ellas, aumentando la diferencia al aumentar el número de hilos.

6.4.5. Ejecución con swap a ramdisk local

Igual que en el escenario de swap a disco duro, en la ejecución con swap a ramdisk local se varió la cantidad de memoria de la máquina según la prueba a ejecutar. Se usaron los mismos valores de 2,5 GB de memoria RAM para stmv, 768 MB para f1atpase, y 256 MB para apoa1.

Figura 6.40: Tiempo de ejecución de stmv y f1atpase.



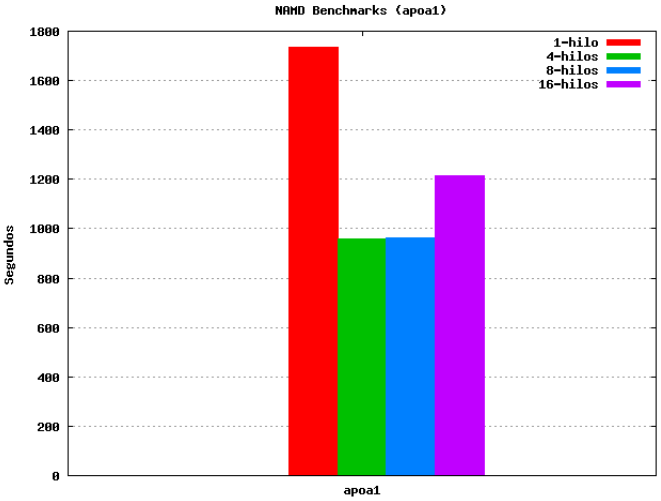
Aunque a diferentes escalas, los escenarios mostrados en la figura 6.40 para las pruebas stvm y f1atpase en su ejecución con swap a ramdisk local son similares, pues en ambos casos se aprecia una reducción del tiempo de ejecución al ejecutar la aplicación de manera paralela y en ambos casos se dispara el tiempo al llegar a 16 hilos en la paralelización.

Tabla 6.39: Tiempo de ejecución de stmv y f1atpase.

Hilos	1	4	8	16
segundos stmv	18339,4	8015,4	7716,0	19531,7
segundos f1atpase	5913,9	2981,7	5047,5	20107,2

La tabla de tiempos 6.39 muestra las ejecuciones de stmv y f1atpase al utilizar ramdisk local como área de intercambio. Las ejecuciones de stmv incrementan sus tiempos en un 11,9 %, 126 %, 317 %, y 1993 % (dos órdenes de magnitud de diferencia). Los resultados de f1atpase se incrementaron en un 30 % en la ejecución secuencial, un orden de magnitud para las ejecuciones con 4 y 8 hilos y dos órdenes de magnitud para la de 16 hilos.

Figura 6.41: Tiempo de ejecución de apoa1.



La misma pauta que se encuentra en stmv y f1atpase puede verse en la figura 6.41 para la prueba apoa1. Se decrementan los tiempos al ejecutar la aplicación en paralelo, pero a mayor número de hilos, mayor tiempo requerido.

Tabla 6.40: Tiempo de ejecución de apoa1.

Hilos	1	4	8	16
segundos apoa1	1734,2	956,9	962,1	1216,5

En la tabla 6.40 se recogen los tiempos de la ejecución de NAMD con la prueba apoa1. La ejecución secuencial sufre un incremento de tiempo de un 9,5 % respecto al escenario base. Con 4 hilos de paralelización, el tiempo se incrementa un 182 % y con 8 hilos un 402 %. Al utilizar 16 hilos, el tiempo de ejecución alcanza un orden de magnitud más respecto al escenario base (un 942 %).

6.5. ACML

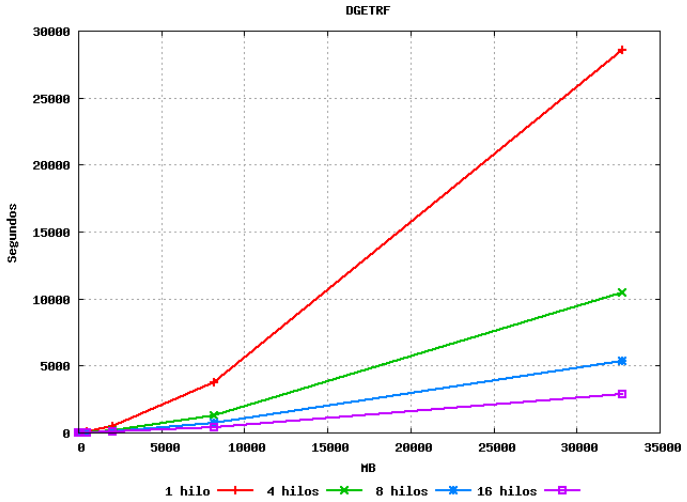
En este apartado se recogen los resultados de las ejecuciones del test que utiliza la función dgetrf de las bibliotecas ACML para cada uno de los escenarios estudiados. Hay que destacar la claridad de lectura de los resultados de esta prueba, que no presentó problemas y resume fielmente el comportamiento de los diferentes escenarios.

Como en los diferentes escenarios se utilizaron tamaños de problema distintos, se utilizarán tres de estos tamaños para hacer las comparativas entre ellos. Con el tamaño más pequeño se pueden comparar todos los escenarios y con los

tamaños intermedios, se pueden establecer comparaciones entre determinados escenarios.

6.5.1. Ejecución en RAM afín

Figura 6.42: Tiempo de ejecución de dgetrf.



En la figura 6.42 se representan las curvas de tiempos obtenidos para distintos tamaños del problema y número de hilos de ejecución. Se observa que el incremento de tiempo de ejecución no es lineal con el incremento del tamaño del problema y que la aplicación paraleliza bien al disminuir el tiempo con el aumento del número de hilos.

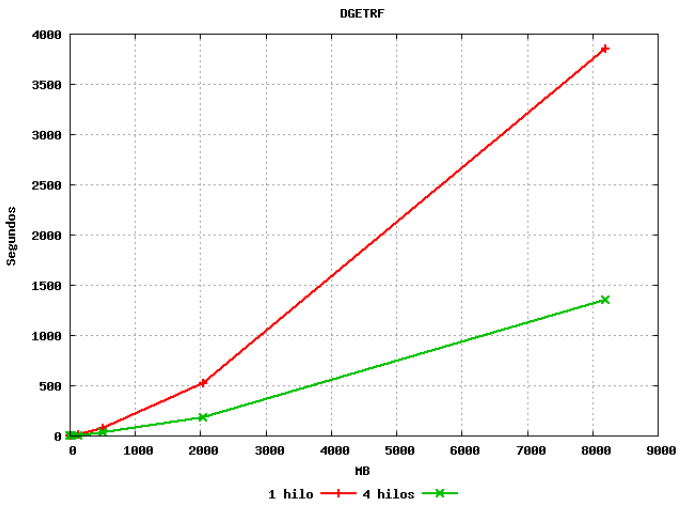
Tabla 6.41: Tiempo de ejecución de dgetrf según tamaño.

Hilos	1	4	8	16
segundos 512 MB	72,5	24,8	16,4	18,4
segundos 2048 MB	503,3	172,3	99,3	72,3
segundos 8192 MB	3712,4	1314,2	695,2	421,1

La tabla 6.41, recoge los tiempos de dgetrf para varios tamaños del problema. Se puede comprobar que con tamaños pequeños, la aceleración obtenida es poca y al ir aumentando el tamaño del problema, mejor aceleración se obtiene, pero sin llegar a una aceleración superlineal como en las otras aplicaciones.

6.5.2. Ejecución en RAM no afín

Figura 6.43: Tiempo de ejecución de dgetrf.



La figura 6.43, que representa el tiempo al usar memoria de otros procesadores, muestra el mismo comportamiento que la ejecución del escenario base.

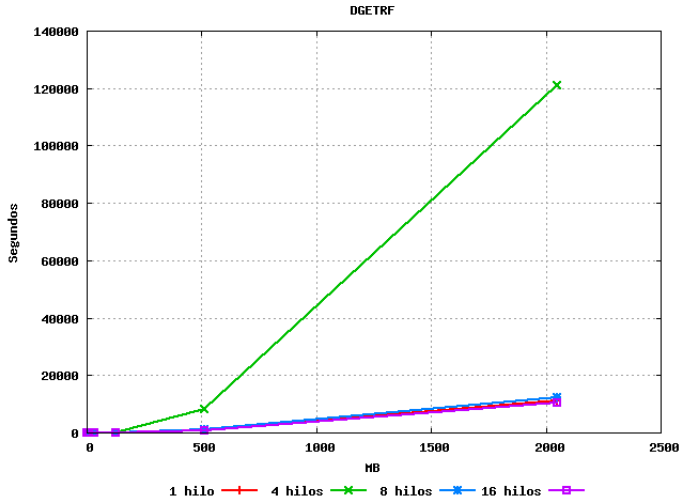
Tabla 6.42: Tiempo de ejecución de dgetrf según tamaño.

Hilos	1	4	8	16
segundos 512 MB	76,7	27,1	-	-
segundos 2048 MB	526,4	184,4	-	-
segundos 8192 MB	3854,0	1355,5	-	-

Todos los tamaños mostrados en la tabla 6.42 presentan una aceleración de 2,8 para la ejecución paralela respecto a la secuencial. El incremento de tiempos en la ejecución secuencial es de 5,7 %, 4,7 % y 3,8 % para los diferentes tamaños. En la ejecución paralela es de un 9 %, 6,7 % y 3,1 % respectivamente.

6.5.3. Ejecución con swap a disco

Figura 6.44: Tiempo de ejecución de dgetrf.



En la figura 6.44, que representa los tiempos de ejecución de dgetrf al utilizar disco como área de intercambio se observa un comportamiento muy irregular en el caso de la ejecución con 4 hilos. Estos tiempos de ejecución tan elevados pudieran ser resultado de un marcado efecto thrashing.

Tabla 6.43: Tiempo de ejecución de dgetrf según tamaño.

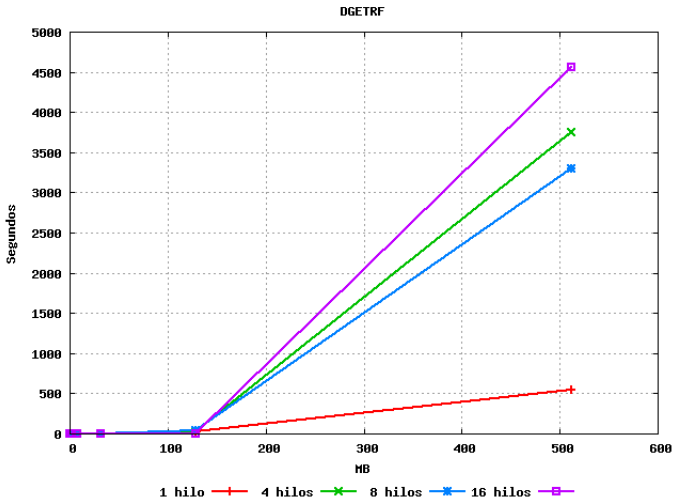
Hilos	1	4	8	16
segundos 512 MB	850,5	8346,3	1077,9	830,8
segundos 2048 MB	11220,0	120896,2	12435,0	10552,9
segundos 8192 MB	-	-	-	-

Los tiempos de ejecución recogidos en la tabla 6.43 muestran que no existe mejora al utilizar paralelismo en este escenario. La aceleración obtenida es negativa o mínima (para el caso de 16 hilos).

La ejecución secuencial tiene unos tiempos de uno y dos órdenes de magnitud más que la misma en el escenario base. Las ejecuciones con 4 y 8 hilos aumentan los tiempos en dos y tres órdenes de magnitud y la de 16 hilos lo hace en uno y tres órdenes de magnitud respecto al escenario base.

6.5.4. Ejecución con swap a ramdisk remoto

Figura 6.45: Tiempo de ejecución de dgetrf.



Los tiempos de dgetrf al utilizar un ramdisk remoto como área de intercambio están representados en la figura 6.45. De la misma forma que ocurrió con las otras aplicaciones, no se obtiene mejora al paralelizar las ejecuciones en este escenario. Los tiempos se incrementan considerablemente al aumentar el número de hilos. Si bien el tiempo de la ejecución de 8 hilos es menor que el de la de 4, no se puede decir que sea gracias a un mayor nivel de paralelismo. Es más probable que se deba al mismo efecto thrashing que se vió en el escenario anterior, que se agudiza al utilizar 4 hilos.

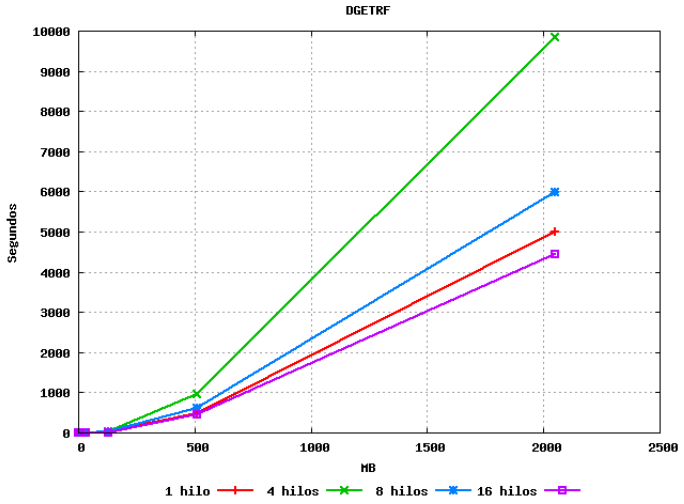
Tabla 6.44: Tiempo de ejecución de dgetrf según tamaño.

Hilos	1	4	8	16
segundos 512 MB	542,7	3743,9	3297,0	4566,6
segundos 2048 MB	-	-	-	-
segundos 8192 MB	-	-	-	-

Los tiempos de la tabla 6.44, representan un incremento de un orden de magnitud para la ejecución secuencial y de dos órdenes de magnitud para las ejecuciones paralelas respecto a las mismas en el escenario base.

6.5.5. Ejecución con swap a ramdisk local

Figura 6.46: Tiempo de ejecución de dgetrf.



Una vez más, se puede apreciar en la figura 6.46 como la ejecución paralela con 4 hilos incrementa su tiempo considerablemente más que el resto y que sólo al usar 16 hilos se mejora el tiempo de la ejecución secuencial.

Tabla 6.45: Tiempo de ejecución de dgetrf según tamaño.

Hilos	1	4	8	16
segundos 512 MB	475,2	951,1	599,4	456,4
segundos 2048 MB	4989,6	9847,2	5982,9	4447,5
segundos 8192 MB	-	-	-	-

La tabla de tiempos 6.45 muestra que al utilizar ramdisk como área de intercambio se ha perdido toda la aceleración que se obtiene en el escenario base. La diferencia de tiempos para las ejecuciones de 1 y 4 hilos son de un orden de magnitud respecto al escenario base y las de 8 y 16 hilos incrementan el tiempo en uno y dos órdenes de magnitud según se incrementa el tamaño del problema.

Capítulo 7

Resumen de resultados

Para tratar de clarificar los resultados obtenidos, se han agrupado estos por aplicación y nivel de paralelización, de forma que se aprecien mejor las diferencias de rendimiento entre los distintos escenarios utilizados. En las figuras, los escenarios están nombrados por su orden de presentación:

Ejecución en RAM afín: conf-1

Ejecución en RAM no afín: conf-2

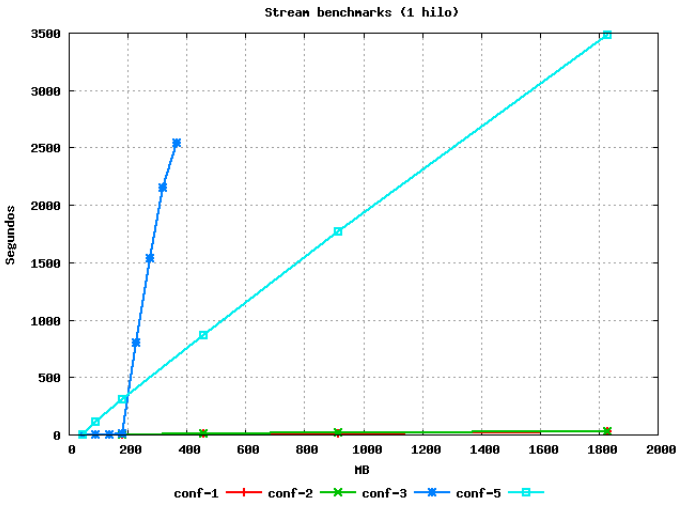
Ejecución con swap a disco: conf-3

Ejecución con swap a ramdisk remoto: conf-4

Ejecución con swap a ramdisk local: conf-5

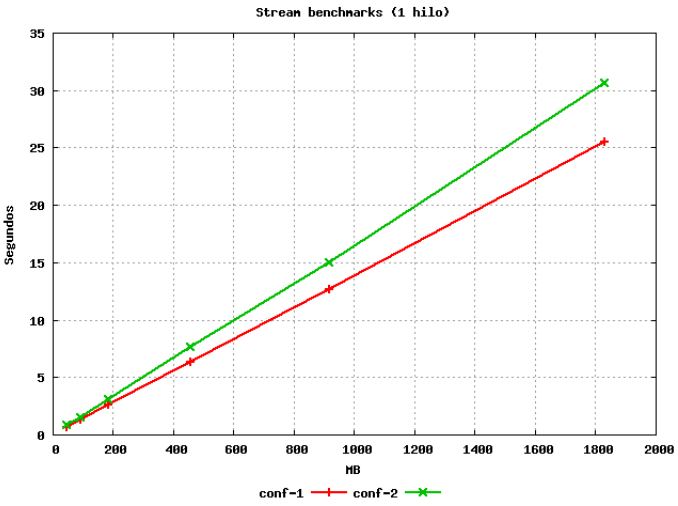
7.1. stream

Figura 7.1: Tiempo de stream con 1 hilo.



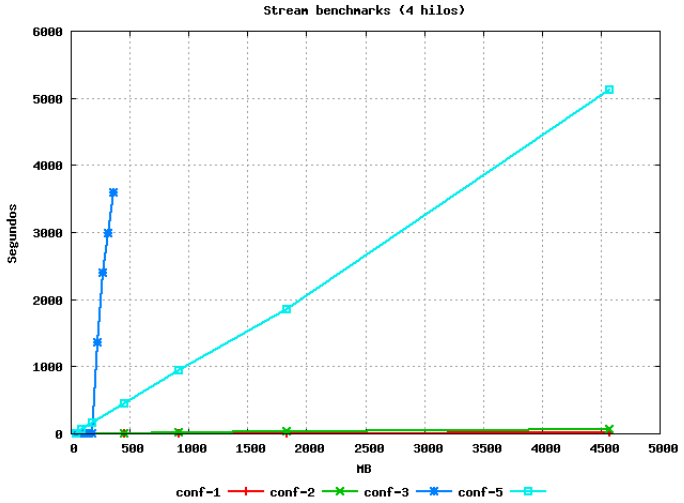
La figura 7.1 resume la ejecución secuencial de stream en los cuatro escenarios en los que se corrió. Dada la escala de la gráfica, la diferencia de rendimiento entre la ejecución en la RAM asociada al procesador y la ejecución en RAM de procesadores diferentes al que corre el proceso, casi no se aprecia. Podría decirse que esos dos escenarios obtienen el mismo rendimiento. En el uso de ramdisk local como área de intercambio la diferencia de tiempos es de dos órdenes de magnitud y en la ejecución con uso de disco el tiempo necesario aumenta hasta en tres órdenes de magnitud respecto al escenario base.

Figura 7.2: Tiempo de stream con 1 hilo.



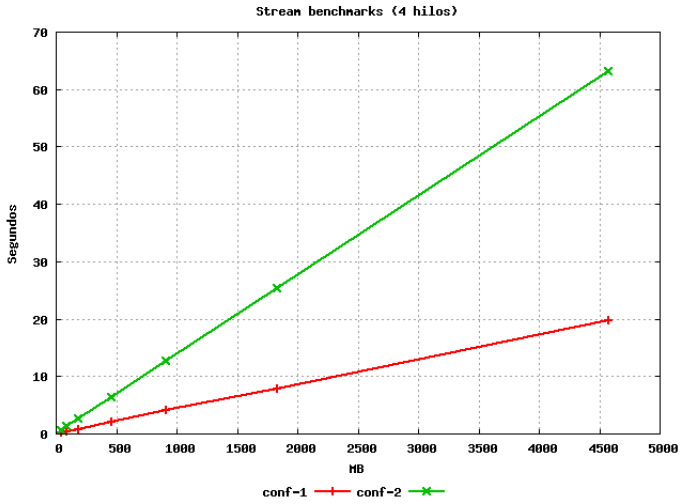
La diferencia de comportamiento entre los dos primeros escenarios, que no se aprecia bien en la figura 7.1, puede verse aumentado en la figura 7.2. El tiempo obtenido al utilizar memoria de otros procesadores aumentó aproximadamente un 18 %.

Figura 7.3: Tiempo de stream con 4 hilos.



La figura 7.3 recoge las ejecuciones con 4 hilos. El comportamiento general de los escenarios no varía. Sólo en los mayores tamaños se distingue levemente una diferencia de tiempo entre los dos primeros escenarios. El uso de ramdisk marca una diferencia de entre dos y tres órdenes de magnitud. Y el uso de disco produce un incremento de tiempo de entre tres y cuatro órdenes de magnitud.

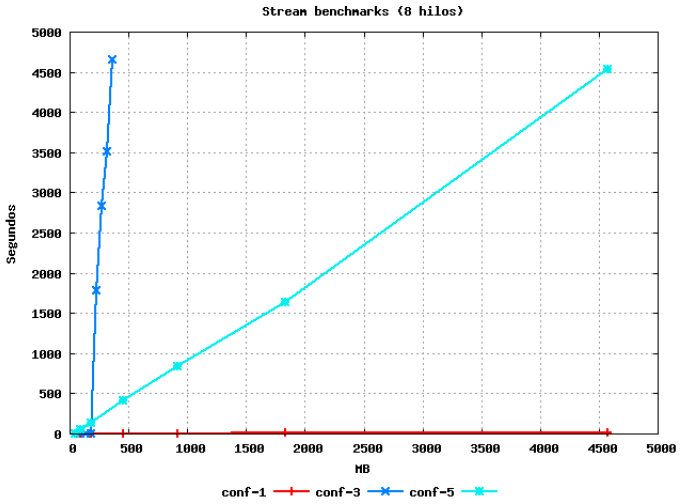
Figura 7.4: Tiempo de stream con 4 hilos.



En la figura 7.4 aparecen representados los tiempos de los dos primeros escenarios en la ejecución paralela con 4 hilos. Se aprecia una mayor diferencia

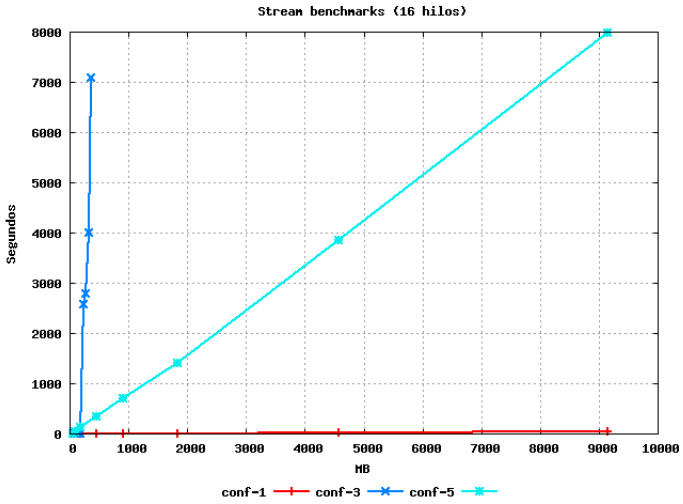
de tiempos entre ambos escenarios en en la ejecución secuencial. En este caso, el incremento producido por el uso de memoria no afín al procesador que ejecuta el programa es de un 220 %.

Figura 7.5: Tiempo de stream con 8 hilos.



La figura 7.5 representa las ejecuciones paralelas con 8 hilos en los distintos escenarios. Se aprecia un ligero incremento de la pendiente de las curvas respecto a las ejecuciones con menor número de hilos. En este caso, la diferencia entre el escenario base y el uso de un ramdisk local como swap, es de dos órdenes de magnitud y el uso de disco conlleva un incremento del tiempo de tres órdenes de magnitud.

Figura 7.6: Tiempo de stream con 16 hilos.

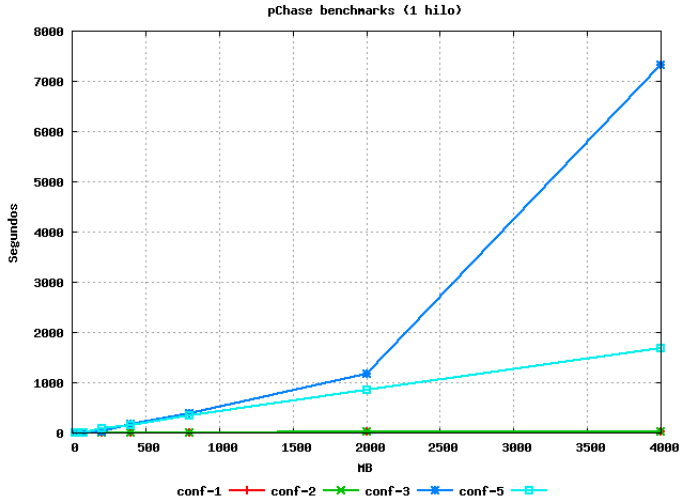


El mismo comportamiento puede observarse en la figura 7.6, que vuelve a incrementar un poco más la pendiente de las ejecuciones con swap respecto a las que utilizaban menos hilos. El uso de ramdisk aumenta el tiempo entre dos y tres órdenes de magnitud y el uso de disco como swap lo incrementa entre tres y cuatro órdenes de magnitud. Las diferencias de tiempo entre los escenarios son claras.

7.2. pChase

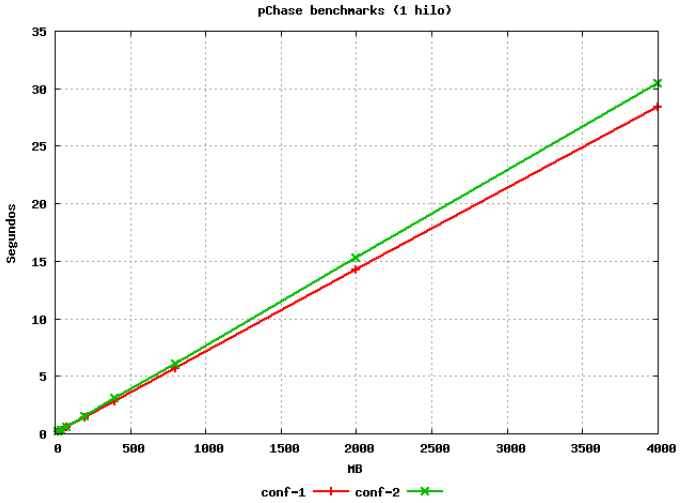
La representación de los escenarios con el test pChase se ha hecho de la misma forma que con stream. Se representan las ejecuciones en los diferentes escenarios y en los casos en los que no se aprecian las diferencias en las gráficas, se muestran los detalles en una figura aparte.

Figura 7.7: Tiempo de pChase con 1 hilo.



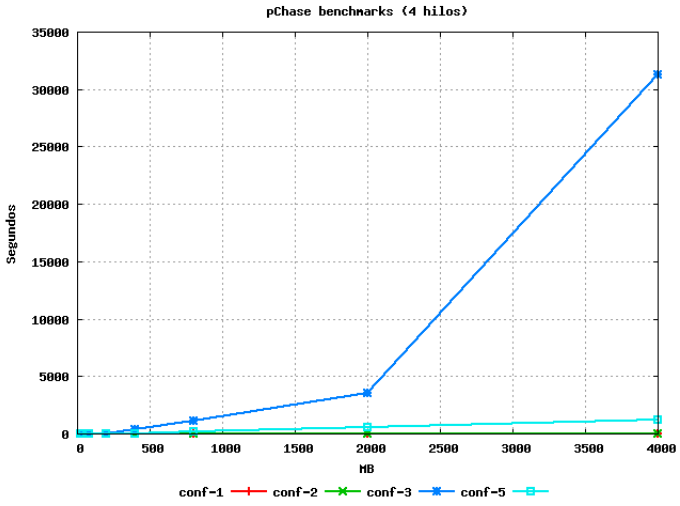
En la figura 7.7 aparecen las ejecuciones secuenciales de pChase en los cuatro escenarios en los que se ejecutó. Los tiempos de la ejecución con swap a disco son tan elevados en comparación con los de los dos primeros escenarios que impiden apreciar estos últimos. El uso de ramdisk incrementa los tiempos en dos órdenes de magnitud respecto a la ejecución íntegra en memoria RAM. El uso de disco como área de intercambio resulta ser el acceso más lento. Tiene un comportamiento similar al del ramdisk para tamaños no muy elevados, pero una vez se alcanza un determinado tamaño, el tiempo necesario para la ejecución con swap a disco aumenta exponencialmente.

Figura 7.8: Tiempo de pChase con 1 hilo.



La figura 7.10 muestra la diferencia de tiempos en la ejecución secuencial para los dos primeros escenarios. Aunque el incremento de tiempo al utilizar memoria de otros procesadores varía un poco en las pruebas con menor tamaño, al final se estabiliza en torno a un 7 %.

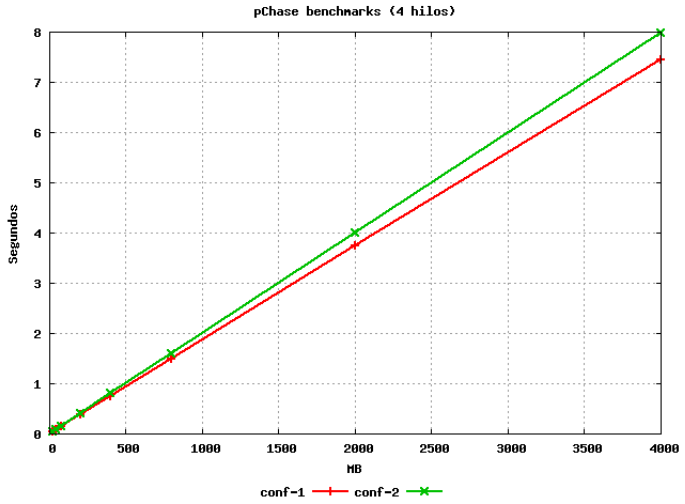
Figura 7.9: Tiempo de pChase con 4 hilos.



Los tiempos de la ejecución con 4 hilos mostrados en la figura 7.9 tienen el mismo patrón por escenarios que la ejecución secuencial. Si bien, con este nivel de paralelismo, aumentan mucho las diferencias de tiempo entre los escenarios al

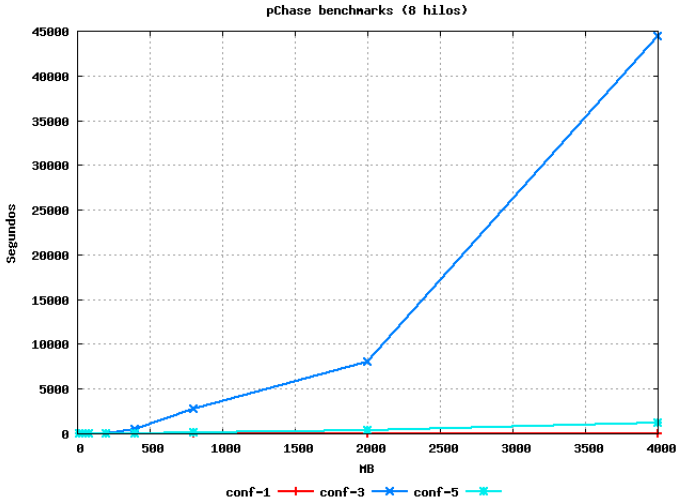
disminuir el tiempo de las ejecuciones que usan RAM, RAM no afín y ramdisk, y al aumentar el tiempo de la ejecución que usa disco como swap. El incremento de tiempo al utilizar un ramdisk local como área de intercambio es de dos órdenes de magnitud en los primeros tamaños, llegando a tres en el mayor tamaño del problema. El uso de disco como swap tiene un incremento de tiempo de tres órdenes de magnitud que llega hasta cuatro órdenes de magnitud en el mayor de los tamaños.

Figura 7.10: Tiempo de pChase con 4 hilos.



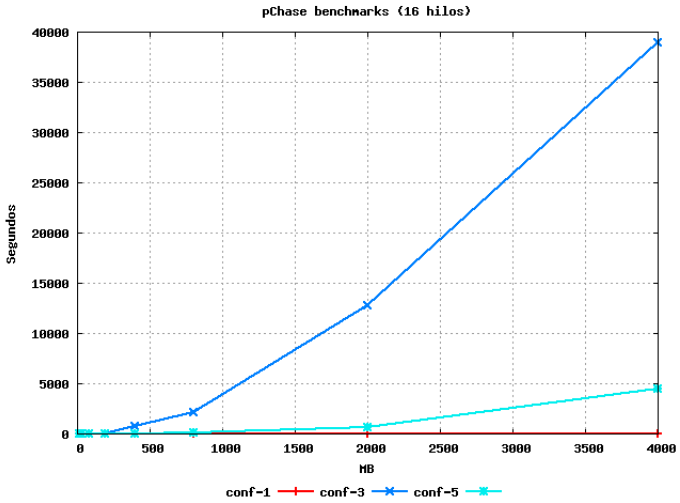
La figura 7.10 representa la diferencia de tiempos en la ejecución paralela con 4 hilos entre el escenario base y el uso de RAM de otros procesadores. Con variaciones de rendimiento en las ejecuciones con tamaños pequeños, se estabiliza con los tamaños mayores (igual que en la ejecución secuencial) en torno al 7 % más de tiempo al usar memoria de otros procesadores.

Figura 7.11: Tiempo de pChase con 8 hilos.



La ejecuciones paralelas con 8 hilos que se representan en la figura 7.11 marcan todavía más la diferencia de tiempo entre los escenarios. El tiempo del escenario de swap a disco distorsiona el resto de resultados. El incremento de tiempo con el uso de ramdisk local respecto al escenario base está entre dos y tres órdenes de magnitud. Para el caso de swap a disco, la diferencia de tiempo se incrementa considerablemente pasando a oscilar entre tres y cuatro órdenes de magnitud más.

Figura 7.12: Tiempo de pChase con 16 hilos.



Se muestran en la figura 7.12 las ejecuciones paralelas con 16 hilos. Una vez más, el tiempo de la ejecución con swap a disco acapara la atención. Con este nivel de paralelismo, el uso de ramdisk tiene unos tiempos de dos órdenes de magnitud más que el escenario base, con los primeros tamaños del problema. Al aumentar el tamaño acaba llegando a tener una diferencia de tres órdenes de magnitud. En el escenario de disco duro como área de intercambio, el tiempo respecto al escenario base se incrementa en cuatro órdenes de magnitud hasta los tamaños estudiados y muestra tendencia de crecimiento exponencial.

7.3. Gromacs

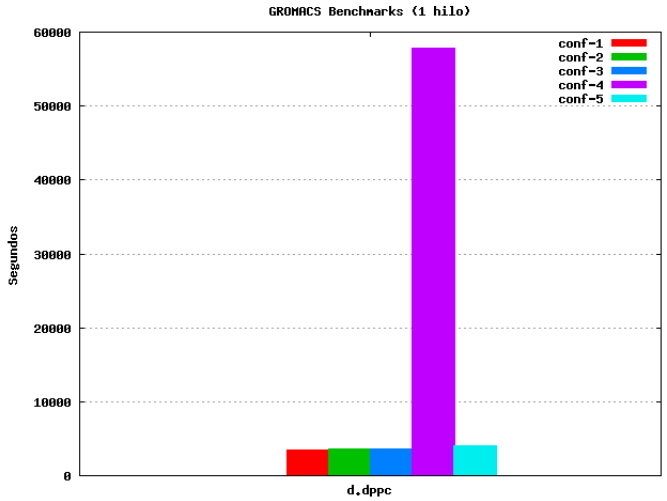
De entre las pruebas que se realizaron con Gromacs, se muestran a continuación los resultados de la prueba d.dppc. Algo característico de esta aplicación, en contraste con el resto, es que el escenario que más penalizó las ejecuciones fue el de swap en ramdisk remoto en vez de ser el que establecía el área de swap en el disco duro local de la máquina.

Recordar que en las configuraciones con área de intercambio, el tamaño de RAM de la máquina se estableció en 128 MB. Los tamaños de memoria usados por la aplicación con la prueba d.dppc fueron los mostrados en la tabla 7.1

Tabla 7.1: Memoria usada por Gromacs con d.dppc.

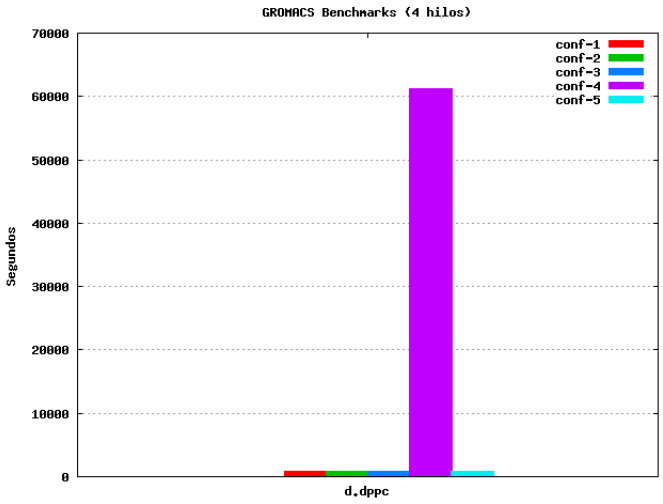
Hilos	1	4	8	16
Memoria residente (KB)	62348	90752	112736	149888
Memoria virtual (KB)	115456	314452	601248	1188256

Figura 7.13: Tiempo de d.dppc con 1 hilo.



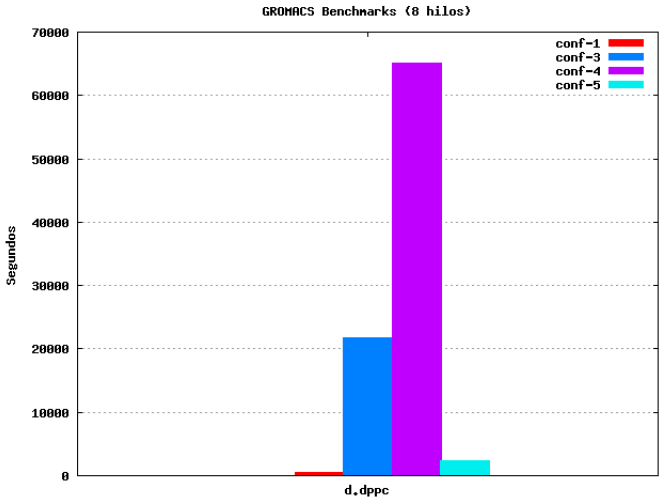
La figura 7.13 resume la ejecución secuencial en los distintos escenarios utilizados. El gran valor de tiempo en la ejecución con swap en ramdisk remoto distorsiona el resto de resultados. En cualquier caso, al ser esta ejecución la que menos memoria consumió, sus resultados han sido muy variables, pues el tiempo medido al usar swap a disco fue menor que en los escenarios dos y cinco.

Figura 7.14: Tiempo de d.dppc con 4 hilos.



En la ejecución paralela con 4 hilos resumida en la figura 7.14 se observa el mismo comportamiento del caso secuencial. Aunque dados los tamaños de RAM consumida y disponibles es obligatorio el uso del área de intercambio por esta aplicación, los resultados no coinciden con lo esperado.

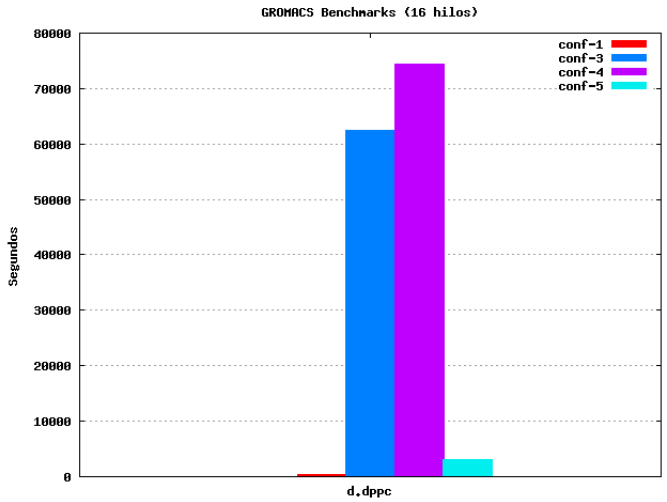
Figura 7.15: Tiempo de d.dppc con 8 hilos.



La figura 7.15 muestra las ejecuciones con un paralelismo de 8 hilos. El aumento del consumo de memoria al aumentar el número de hilos hace que las pautas de comportamiento se acentuen y con este nivel de paralelismo ya se aprecien resultados más acorde a lo esperado.

El uso de ramdisk local como área de intercambio incrementa el tiempo de ejecución en un orden de magnitud respecto al escenario base. El disco duro local aumenta la diferencia hasta llegar a los dos ordenes de magnitud, incremento que comparte con el uso de un ramdisk remoto. Pero en este último escenario, la aplicación necesita tres veces el tiempo que usa con swap a disco duro.

Figura 7.16: Tiempo de d.dppc con 16 hilos.



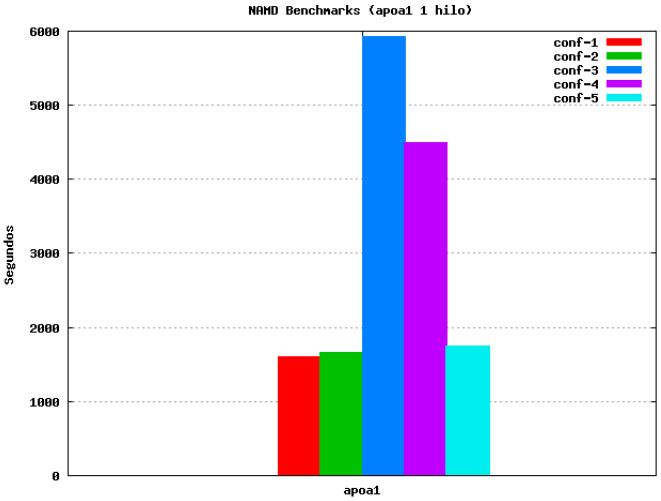
La ejecución paralela con 16 hilos vuelve a incrementar el consumo de memoria. La figura 7.16 representa los resultados en los diferentes escenarios.

Se observa el mismo comportamiento global entre escenarios que se ha visto con la ejecución de 8 hilos. El uso de ramdisk local aumenta el tiempo de ejecución en un orden de magnitud y, tanto el uso de disco duro local como el uso de ramdisk remoto como área de intercambio incrementan los tiempos en dos órdenes de magnitud respecto al escenario base. Si bien, con el incremento de memoria, la diferencia entre el escenario 2 y 4 se reduce considerablemente.

7.4. NAMD

Como durante las ejecuciones de NAMD se varió el tamaño de la memoria RAM de la máquina con algunas de las pruebas, se presentan a modo de resumen los resultados de la prueba apoa1, ya que esta se ejecutó en todos los escenarios con la misma cantidad de memoria disponible y se pueden establecer comparaciones directas.

Figura 7.17: Tiempo de apoa1 con 1 hilo.

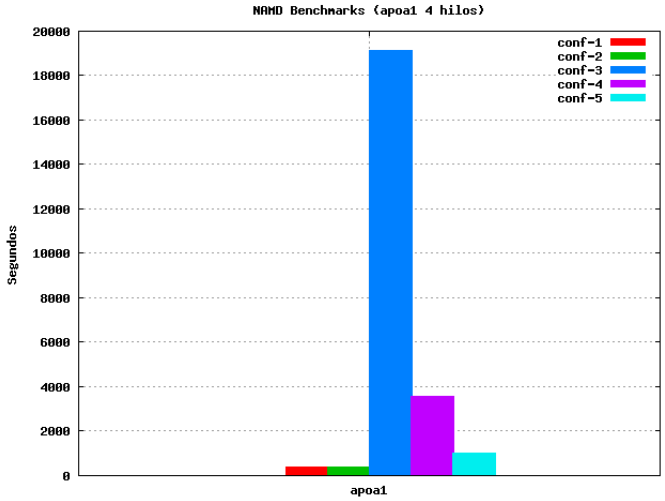


En la figura 7.17 se muestran los tiempos obtenidos en la ejecución secuencial de NAMD con la prueba apoa1 en las distintas configuraciones de memoria estudiadas. Se observa a simple vista la diferencia entre escenarios.

El incremento de tiempo por utilizar memoria de otros procesadores es de un 4 % respecto al escenario base. El uso de ramdisk local para establecer el área de intercambio sube al 9,5 %. El ramdisk en una máquina remota incrementa considerablemente el tiempo, hasta llegar a un 183 % respecto a la ejecución en RAM. Y la ejecución más lenta se produce al usar el disco duro local de la máquina como swap, que incrementa el tiempo base en un 273 %.

El consumo de memoria en la ejecución secuencial fue aproximadamente de 370 MB.

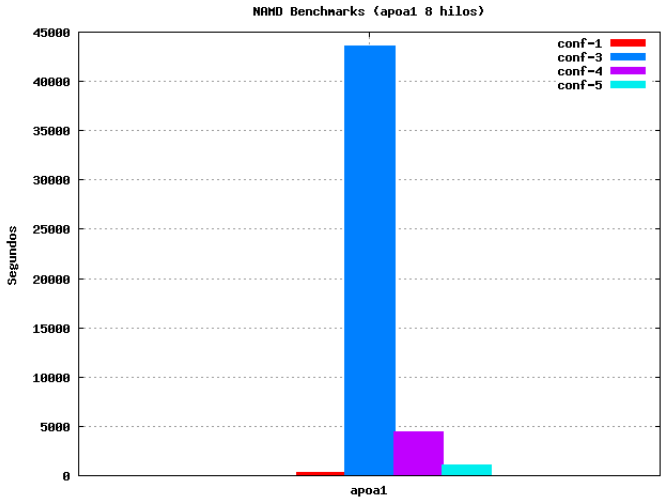
Figura 7.18: Tiempo de apoa1 con 4 hilos.



La ejecución paralela con 4 hilos representada en la figura 7.18 marca todavía más las diferencias entre escenarios observadas en la ejecución secuencial. El estar las diferencias más acentuadas, se debe al mayor consumo de memoria por parte de la aplicación al incrementar el número de hilos con los que se ejecuta. El consumo con 4 hilos fue aproximadamente de 550 MB.

Los tiempos entre los escenarios 1 y 2 no son concluyentes. La configuración con ramdisk como área de intercambio incrementa el tiempo en un 182 % respecto al tiempo base. El ramdisk remoto lo aumenta en un 940 % (un orden de magnitud) y el disco duro local llega a un incremento de dos órdenes de magnitud con un 5529 %.

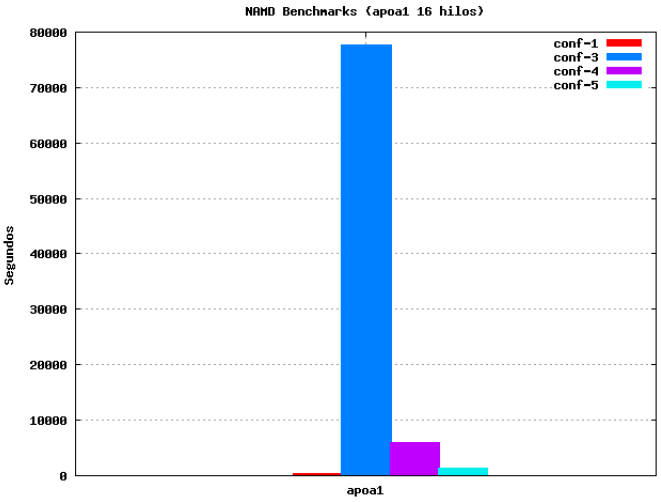
Figura 7.19: Tiempo de apoa1 con 8 hilos.



Se muestran en la figura 7.19 los tiempos de las ejecuciones paralelas con 8 hilos en los distintos escenarios. En este caso, el consumo de memoria aumentó hasta aproximadamente unos 670 MB.

Al utilizar un ramdisk local como swap, el tiempo se incrementó en un 402 % respecto al escenario base. Utilizando un ramdisk en una máquina remota, el incremento de tiempo llegó a ser de un 2166 %, lo que representa un orden de magnitud más. El mayor de los tiempos se obtuvo una vez más con el disco duro local al tener un incremento de dos órdenes de magnitud (un 22618 %).

Figura 7.20: Tiempo de apoa1 con 16 hilos.



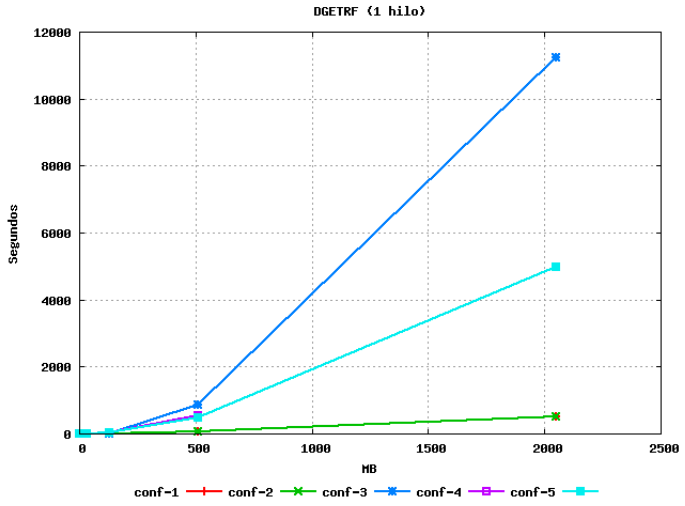
Las ejecuciones paralelas de 16 hilos de NAMD con apoa1, representadas en la figura 7.20 tuvieron un consumo de aproximadamente 1200 MB de memoria RAM. Este aumento de la memoria utilizada incrementó los tiempos de ejecución en los escenarios con swap.

En el uso de ramdisk (tanto local como remoto) el tiempo aumentó hasta llegar a un incremento de un orden de magnitud respecto al escenario base siendo de 942 % y 4853 % para local y remoto respectivamente.

El escenario con swap a disco duro sufrió una vez más el mayor de los aumentos, al terminar con una diferencia de dos órdenes de magnitud respecto al escenario base.

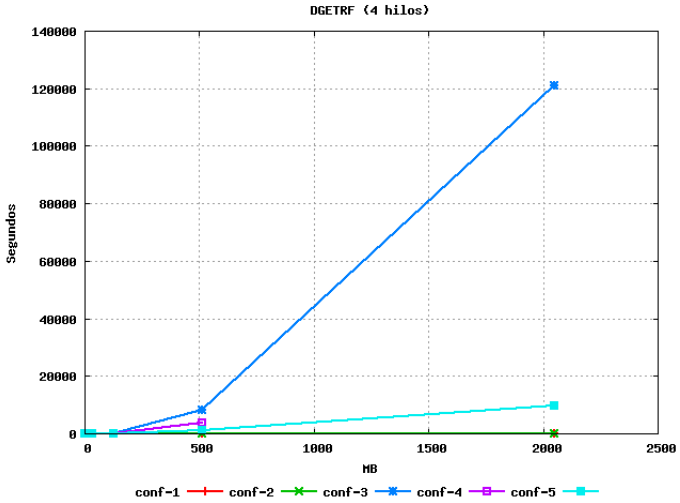
7.5. ACML

Figura 7.21: Tiempo de ACML con 1 hilo.



En la figura 7.21 aparecen los resultados de la ejecución secuencial de ACML para los distintos escenarios. La diferencia de tiempo al utilizar memoria de otros procesadores es aproximadamente un 5 % y baja a 4 % y 3 % con los mayores tamaños utilizados. Con el uso de ramdisk local se incrementan los tiempos en un orden de magnitud. En la gráfica casi no se aprecia el ramdisk remoto, ya que los tiempos obtenidos con este escenario tienen el mismo orden que el ramdisk local, pero son mayores en un 7 % y un 14 %. Aunque con los datos actuales no se puede verificar, parece lógico esperar que continue la tendencia ascendente y se incremente diferencia entre ramdisk local y remoto al aumentar el tamaño del problema. El uso de disco alcanza un incremento de tiempo de dos órdenes de magnitud respecto al escenario base.

Figura 7.22: Tiempo de ACML con 4 hilos.

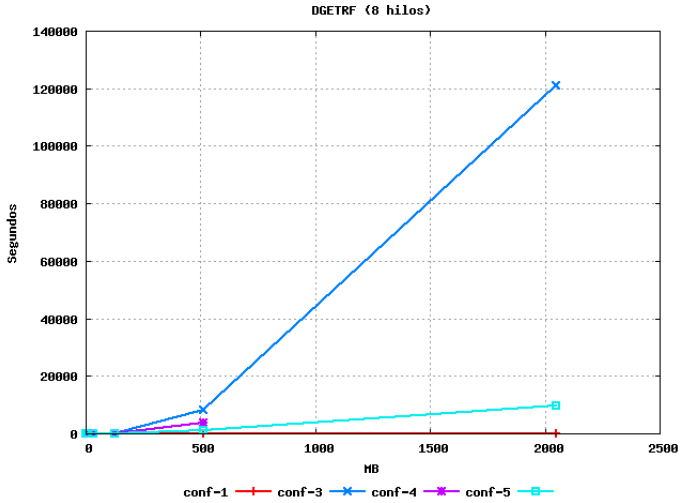


Con la ejecución de 4 hilos, representada en la figura 7.22, el incremento de tiempo al utilizar memoria de otros procesadores empieza siendo de hasta un 16 % respecto al uso de RAM local al procesador. A medida que aumenta el tamaño del problema, esta diferencia va disminuyendo pasando por un 9 % y 6 % hasta terminar siendo de sólo un 3 % en el mayor de los tamaños.

El uso de un ramdisk local para establecer el área de intercambio aumenta el tiempo en un orden de magnitud respecto al escenario base. De la misma manera que en la ejecución secuencial, al usar ramdisk remoto la diferencia de tiempos empieza siendo de un orden de magnitud, pero a medida que se incrementan los tamaños, pasa a ser de dos órdenes de magnitud. Estos resultados coinciden con la suposición del apartado anterior para este tipo de área de intercambio.

El incremento de tiempo más importante se vuelve a producir en el uso de swap a disco duro. Para tamaños pequeños, el tiempo aumenta en dos órdenes de magnitud y pasa tener una diferencia de tres órdenes de magnitud al aumentar el tamaño del problema.

Figura 7.23: Tiempo de ACML con 8 hilos.

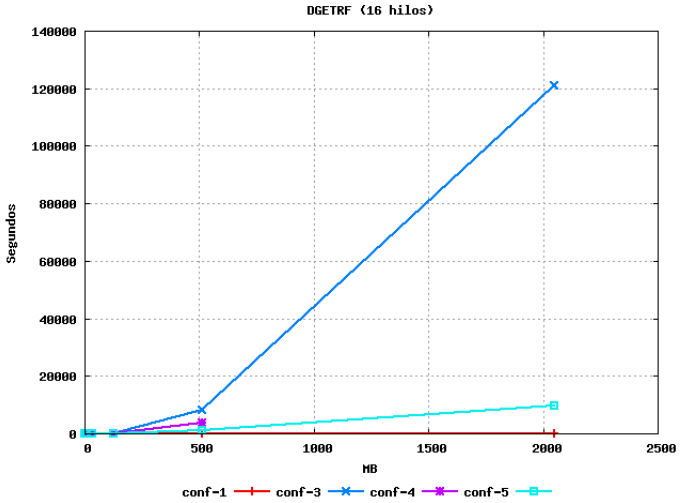


A medida que se aumenta el nivel de paralelismo de la aplicación, se incrementan las diferencias de tiempo entre los escenarios. La figura 7.23 recoge los resultados de la ejecución paralela con 8 hilos.

El menor de los incrementos lo marca el uso de ramdisk local y aún así, la diferencia comienza en un orden de magnitud y llega hasta dos al aumentar el tamaño del problema. El escenario con ramdisk remoto sigue la misma pauta que el local pero tiende a aumentar la diferencia en mayor proporción a medida que se aumenta el tamaño del problema.

Los resultados del uso de disco como swap no varían respecto a las anteriores ejecuciones. Empieza con incrementos de dos órdenes de magnitud y pasa a tres al aumentar el tamaño del problema.

Figura 7.24: Tiempo de ACML con 16 hilos.



Se muestran en la figura 7.24 los resultados de la ejecución paralela con 16 hilos. Siguiendo la pauta de las otras ejecuciones, la menor diferencia con el escenario base se consigue con el uso de ramdisk local, con un incremento de tiempo de un orden de magnitud en tamaños pequeños, que pasa a ser de dos al incrementar el tamaño. El ramdisk remoto tiene el mismo orden de diferencia con unos incrementos superiores.

Con el uso de disco para establecer el área de intercambio el tiempo se incrementa en tres órdenes de magnitud respecto a su ejecución en RAM.

Capítulo 8

Conclusiones

En este proyecto hemos analizado el comportamiento de varias aplicaciones de memoria compartida en una serie de escenarios con distintas configuraciones de memoria. Con los tests de memoria STREAM y pChase hemos podido medir el rendimiento que se obtiene en el acceso a memoria en cada uno de estos escenarios. Por otra parte, las aplicaciones NAMD, Gromacs y la función dgetrf de las librerías ACML nos han mostrado el comportamiento de un software real en estos entornos.

Las variaciones en el tiempo de ejecución producidas por las distintas características del sistema de memoria en los diferentes escenarios han dependido de la aplicación en particular con que se ha probado el sistema. Así pues, los tests de memoria son los que más han sobrecargado el sistema y por consiguiente los que han presentado mayores variaciones en el tiempo de ejecución al testar los distintos escenarios. El resto de aplicaciones, que hace un uso más normal de los recursos, no experimenta un impacto tan acusado al moverse a sistemas de memoria de menor rendimiento como ocurre con el caso de STREAM y pChase.

Dentro de los diferentes escenarios, el uso de *ramdisk* generó de media un incremento del tiempo de ejecución de un orden de magnitud. Este incremento ha de asociarse a la gestión del *ramdisk* y al área de intercambio por parte del sistema operativo. El *ramdisk* en máquina remota proporcionó unos tiempos mayores que el *ramdisk* en la máquina local, como era de esperar, pero dentro del mismo orden de magnitud al menos para los tamaños estudiados. El aumento de tiempo ha de asociarse a la sobrecarga debida a la gestión del protocolo de red, al sistema de bloques utilizado y al hecho de que los datos han de viajar de un computador a otro a la hora de intercambiar páginas de memoria.

El disco duro proporcionó un incremento en el tiempo de ejecución de entre dos y tres órdenes de magnitud debido a su limitado rendimiento, especialmente a su alta latencia y a sus pobres prestaciones en cuanto a acceso no secuencial de la información.

El uso de hilos en la ejecución de un programa permite su paralelización y por tanto una reducción en su tiempo de ejecución (al menos cuando no existen cuellos de botella en el sistema). Este aumento en el número de flujos de ejecución concurrentes no implica un aumento significativo en la huella de memoria del programa: aunque las estructuras particulares de cada hilo sí que se verán replicadas, la información que maneja la aplicación (*data set*) permanece constante. No obstante, la cantidad de datos "vivos" sí que aumentará al aumentar

el número de hilos. Estos datos "vivos" o conjunto de trabajo (*working set*) son el conjunto de datos que está manejando la aplicación durante una cierta ventana de tiempo y marcan el ratio de información transvasada entre los distintos niveles en la jerarquía de memoria.

Lo que en ejecución en RAM se traduce en una mejora al paralelizar las aplicaciones, cuando se utiliza algún tipo de memoria más lenta como área de intercambio, se traduce en peores prestaciones cuanto mayor es el nivel de paralelismo, pues además de aumentar la sobrecarga por la gestión de los hilos y el coste de sincronización de estos, también aumentan los recursos que el programa necesita en un determinado momento (*working set*), y la proporción de uso RAM-swap se desequilibra hacia un mayor uso del área de intercambio.

Se ha podido comprobar que cuando se usa algún tipo de área de intercambio, es muy importante la proporción de memoria que se encontrará en RAM durante la ejecución, como acabamos de decir, pues por muy rápido que sea el acceso a la memoria remota, éste siempre será más lento (tanto en ancho de banda como en latencia) que el acceso directo a RAM, de la misma forma que sucede en las capas inferiores (relación caché-RAM). Así pues, si la mayor parte del conjunto de trabajo de la aplicación se encuentra en RAM, sí será factible el utilizar otras memorias como espacio de intercambio. A partir de una determinada proporción entre el uso de RAM y el del área de swap, el rendimiento decrece notablemente. Cuánto más cercano esté el rendimiento del área de intercambio al de la RAM, más uso le podrá dar la aplicación sin que se aprecie demasiado la ralentización que provoca. En cualquier caso, debido a que el espacio de intercambio se sitúa a un nivel distinto al de la memoria principal, el sistema tendrá la sobrecarga de la gestión del área de intercambio y de la capa de software y/o hardware que dé acceso a esa memoria.

Referencias

[1] *http://pchase.org.*

[2] *http://www.amd.com/acml.*

[3] *http://www.cs.virginia.edu/stream.*

[4] *http://www.gromacs.org.*

[5] *http://www.ks.uiuc.edu/Research/namd.*

[6] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. *GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation.*

[7] John D McCalpin. Memory bandwidth and machine balance in current high performance computer. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, December 1995.

[8] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of Computational Chemistry*, 2005.