MÉTODOS DE LATTICE BOLTZMANN EN EQUIPOS MULTICORE

Miguel Montes [(1)] Carlos Sacco [(2)]
(1) Departamento Sistemas – Facultad de Ingeniería – Instituto Universitario Aeronáutico
Córdoba - <mmontes@iua.edu.ar>
(2) Departamento Mecánica Aeronáutica – Facultad de Ingeniería – Instituto Universitario Aeronáutico
Córdoba - <csacco@iua.edu.ar>

RESUMEN

Este trabajo reporta el resultado de una prueba de implementación paralela de lo que se conoce como Lattice Boltzmann Methods (LBM). Se trata de métodos relativamente nuevos que contrastan con el enfoque tradicional de la mecánica computacional de fluidos, describiendo al fluido a nivel molecular y proponiendo modelos para la colisión entre moléculas.

Se realizó en el marco del Intel(r) Manycore Testing Lab, una iniciativa de la empresa Intel que permite a universidades el acceso a computadoras de memoria compartida con 32 cores. Intel realizó un llamado para presentar proyectos, y nuestro grupo fue uno de los cinco seleccionados, junto con otro equipo de Argentina y tres de Rusia.

El objetivo fue paralelizar código secuencial preexistente (en lenguaje Fortran), y evaluar el comportamiento del código paralelizado con distinto número de procesadores. Para ello se trabajó con alumnos de la carrera Ingeniería en Informática que cursaban la asignatura Programación Concurrente, dirigidos por los autores. La paralelización se realizó utilizando OpenMP, tanto en el código original Fortran como en versiones en lenguaje C.

El equipo dispuso de cuatro semanas de acceso al Laboratorio Manycore, consistente en tres nodos de 32 procesadores. Este tiempo permitió testear distintas optimizaciones del código, tanto en Fortran como en C, y utilizando los compiladores GNU (gcc y gfortran) e Intel (icc e ifort). Se evaluaron la escalabilidad y la eficiencia del código para distintos tamaños de problema, para distinta cantidad de procesadores, con distintas estrategias de distribución del código entre procesadores, y las ventajas del uso o no de Hyperthreading.

Como resultado pudimos obtener datos numéricos que nos permiten evaluar la utilidad de distintas optimizaciones y la escalabilidad del código utilizando este modelo de programación, la diferencia de rendimiento entre el código generado por los distintos compiladores, y, en general, el rendimiento de equipos SMP con esta cantidad de procesadores. Desde el punto de vista docente, fue una experiencia valiosa para los alumnos, que pudieron acceder a equipos aún no disponibles comercialmente, interactuar con los mecanismos de gestión de clústeres utilizados, y experimentar de primera mano con conceptos tales como escalabilidad, speedup y eficiencia de sus programas.

Palabras clave: Procesamiento paralelo, Lattice-Boltzmann, OpenMP, multicore

1. INTRODUCCIÓN

Este trabajo reporta el resultado de un proyecto de paralelización de código que implementa el método de Lattice Boltzmann, realizado en el laboratorio Intel Manycore Testing Lab con alumnos de la carrera Ingeniería en Informática del Instituto Universitario Aeronáutico.
Este proyecto tuvo dos objetivos fundamentales: El primero fue utilizar el laboratorio como herramienta para la enseñanza de programación paralela, y evaluar su adecuación a este fin. El segundo objetivo fue obtener experiencia en la paralelización de un problema específico de mecánica computacional de fluidos, en el contexto de computadoras de memoria compartida con gran número de procesadores. En general, las experiencias anteriores sobre problemas de este tipo se centran en procesos secuenciales, o en utilización de computadoras de memoria distribuida ("clusters") con el paradigma de paso de mensajes (ej. MPI).

El Laboratorio Intel Manycore

En marzo de 2010 Intel anunció la disponibilidad del Intel Manycore Testing Lab [8], que brinda a miembros de la comunidad académica acceso a computadoras de memoria compartida con 32 procesadores físicos. La empresa realizó un llamado para presentación de proyectos para uso temprano de dicho laboratorio, y el proyecto presentado por los autores de este trabajo fue seleccionado como uno de los cinco finalistas. La propuesta presentada consistió en paralelizar código preexistente, escrito en Fortran, para testear su escalabilidad con distinto número de procesadores, con distintos compiladores y diversas formas de optimización.
El laboratorio consistía en un "cluster" de 3 equipos dotados de 4 procesadores Intel Xeon X7560 [9], con 32 procesadores físicos (64 hilos con HT habilitado). Uno de los equipos, acano01 funcionaba como nodo de login, y tenía HT habilitado, mientras que los otros dos equipos funcionaban como nodos de cálculo, y tenían HT deshabilitado.
El sistema operativo utilizado fue Red Hat Linux Enterprise, con PBS (Portable Batch System) como planificador de trabajos.
Las características del procesador pueden verse en la siguiente tabla:

Tabla 1: Detalles del procesador Xeon X7560

# de cores	8
# de threads	16
Frecuencia	2.266 Ghz
Cache L1	32 kb datos, 32 kb instrucciones, por core
Cache L2	256 kb por core
Cache L3	24 Mb accesibles por todos los cores
QPI	6.4GT/seg
Memoria	Quad-channel ECC registered DDR3-1066MHz

Conceptos básicos del Método de Lattice Boltzmann

Los Métodos de Lattice Boltzmann [1, 2, 3] son métodos relativamente nuevos que contrastan con el enfoque tradicional de la mecánica computacional de fluidos, describiendo al fluido a nivel molecular y proponiendo modelos para la colisión entre moléculas. No puede ser visto sólo como el sucesor del Lattice gas Automata (LGA), ya que las ecuaciones pueden ser derivadas en forma rigurosa a partir de modelos físicos y las ecuaciones de Boltzmann, además se puede demostrar que las ecuaciones de Navier Stokes son obtenidas en el límite macroscópico [2, 4]
La Ecuación de Boltzmann es una ecuación diferencial en derivadas parciales que describe la función distribución de una partícula f . Esta función de distribución es definida de forma que f (x, ξ, t) es la

probabilidad de que una partícula esté ubicada en un elemento de control de la fase del espacio $dx\_d\xi$, entre x y ξ en un instante t, donde x es la posición en el espacio, ξ es el vector velocidad de la partícula. Las cantidades macroscópicas como la densidad ρ y el momento ρu pueden ser obtenidas evaluando el primer momento de la función de distribución f

Despreciando las fuerzas externas la ecuación de transporte para $f(x, \xi, t)$ puede ser expresada mediante:

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi} \frac{\partial f}{\partial \mathbf{x}} = \mathcal{Q}(f, f)\,. \tag{1}$$

El término de colisión $Q(f, f)$ tiene expresión compleja. Una expresión muy utilizada para la integral de dicho término en condiciones cercanas al equilibrio y a bajo número de Mach es una aproximación mediante relajación del tiempo, esta aproximación es denominada modelo BGK (Bhatnagar-Gross-Krook) [5],

$$\mathcal{Q}(f, f) = -\frac{1}{\lambda}\left(f - f^{(0)}\right) \tag{2}$$

donde $f^{(0)}$ es la función distribución en equilibrio de Maxwell-Boltzmann y λ es el tiempo de relajación el cual controla el grado de la aproximación al equilibrio y representa la viscosidad del fluido. El modelo BGK el teorema H de Boltzmann y conserva en forma local la masa y el momento.

Para resolver numéricamente la ecuación 13.1 se discretiza el espacio de las velocidades utilizando un número finito de vectores velocidad $\boldsymbol{e}_i$ (i=1,....,N). Esto lleva a un sistemas de ecuaciones de Boltzmann para velocidades discretas

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \frac{\partial f_i}{\partial \mathbf{x}} = -\frac{1}{\lambda}(f_i - f_i^{eq})\,, \qquad i = 0, \ldots, N \tag{3}$$

En esta ecuación $f_i(x, t)$ representa la función distribución para la componente i de velocidad $f(x, \boldsymbol{e}_i, t)$.

Para simular flujos bidimensionales existen diversos modelos, en este trabajo se utilizó el modelo de 9 velocidades, denominado D2Q9, en este caso $\boldsymbol{e}_i$ tiene las siguientes componentes:

$e_0 = (0,0)$;
$e_1 = (1,0)$; $e_2 = (0,1)$; $e_3 = (-1,0)$; $e_4 = (0,-1)$;
$e_5 = (1,1)$; $e_6 = (-1,1)$; $e_7 = (-1,-1)$; $e_8 = (1,-1)$

Para este modelo la función distribución en equilibrio f_i^{eq} tiene la siguiente expresión:

$$f_i^{eq} = \rho w_i \left[1 + \frac{3}{c^2}\mathbf{e}_i \cdot \mathbf{u} + \frac{9}{2c^4}(\mathbf{e}_i \cdot \mathbf{u})^2 - \frac{3}{2c^2}\mathbf{u} \cdot \mathbf{u}\right] \tag{4}$$

con $c = \Delta x/\Delta t$ y $\boldsymbol{e}_i$ es el vector de velocidad discreta. Los valores de w_i son factores de ponderación y dependen exclusivamente del modelo [6], en nuestro caso son:

$w_0 = 4 / 9$
$w_1 = w_2 = w_3 = w_4 = 1 / 9$
$w_5 = w_6 = w_7 = w_8 = 1 / 36$

Las funciones de equilibrio discretas f_i^{eq} fueron obtenidas de forma tal que el momento de la velocidad hasta el cuarto orden sean idénticas a los valores de la función distribución en equilibrio de Maxwell-Boltzman $f^{(0)}$. Las variables macroscópicas como la densidad ρ el momento ρu y el momento del tensor de flujos $\Pi_{\alpha\beta}$ pueden ser evaluados en forma discreta mediante:

$$\rho = \int_{-\infty}^{\infty} f\, d\boldsymbol{\xi} = \sum_{i=0}^{N} f_i = \sum_{i=0}^{N} f_i^{eq} \tag{5}$$

$$\rho\mathbf{u} = \int_{-\infty}^{\infty} \boldsymbol{\xi} f\, d\boldsymbol{\xi} = \sum_{i=0}^{N} \mathbf{e}_i f_i = \sum_{i=0}^{N} \mathbf{e}_i f_i^{eq} \tag{6}$$

$$\Pi_{\alpha\beta} = \int_{-\infty}^{\infty} \xi_\alpha \xi_\beta \, f \, d\xi = \sum_{i=0}^{N} \mathbf{e}_{i\alpha} \mathbf{e}_{i\beta} f_i \tag{7}$$

En este modelo la velocidad del sonido $c_s = c / \sqrt{3}$ y la presión se puede obtener con la ecuación de estado para gas ideal:

$$p = \rho c_s^2 \tag{8}$$

Para obtener la aproximación de Lattice Boltzmann la ecuación 1 es discretizada numéricamente mediante un esquema en diferencias finitas explícito, tanto para el espacio como para el tiempo. Escalando el espacio, el paso de tiempo y las velocidades discretas se llega a la siguiente ecuación explícita:

$$f_i(\mathrm{x} + \mathbf{e}_i \Delta t, t + \Delta t) - f_i(\mathrm{x}, t) = -\frac{1}{\tau}\left[f_i(\mathrm{x}, t) - f_i^{eq}(\mathrm{x}, t)\right] \tag{9}$$

donde $\tau = \lambda/\Delta t$ es el tiempo de relajación adimensional y **x** es el punto en el espacio discreto
El lado derecho de la ecuación anterior es denominado *collision step* (fase de colisión) mientras que el lado izquierdo de la misma se denomina *streaming step* (fase de propagación). Para el primer paso la función distribución en equilibrio debe ser calculada en cada celda para cada paso de tiempo a partir de los valores locales de ρ utilizando la ecuación 5 y de velocidad mediante la ecuación 6.
En general resulta ventajoso dividir la ecuación anterior en dos partes de forma tal que:

$$f_i^{out}(\mathrm{x}, t) = f_i^{in}(\mathrm{x}, t) - \frac{1}{\tau}\left[f_i^{in}(\mathrm{x}, t) - f_i^{eq}(\mathrm{x}, t)\right] \tag{10}$$

$$f_i^{in}(x + \mathbf{e_i}, t + \Delta t) = f_i^{out}(\mathrm{x}, t) \tag{11}$$

donde f_i^{out} representa el valor de la función distribución después de la fase de colisión, pero antes de la propagación, mientras que f_i^{in} representa el valor después de ambas operaciones (colisión y propagación). Este último es el valor pasado a las celdas vecinas para el siguiente paso de tiempo.

Las ecuaciones de Navier Stokes (hasta segundo orden en espacio y tiempo) pueden ser derivadas formalmente a partir de las ecuaciones de Lattice Boltzmann mediante la expansión de Chapman-Enskog [2, 7, 8].
La relación entre el tiempo de relajación τ y la viscosidad cinemática ν se puede obtener a partir de los resultados de la expansión de Chapman-Enskog. Como el error de discretización es conocido, el mismo puede ser corregido, de esta forma los resultados del LBM no presentan difusión numérica como ocurre con la mayoría de los métodos basados en diferencias finitas. La relación para la viscosidad cinética queda:

$$\nu = (\tau - 1/2)\, c_s^2 \Delta t \tag{12}$$

Como el LBM es un método basado en la cinética, las condiciones de contorno macroscópicas no tienen una equivalencia directa y deben ser remplazadas por una regla apropiada que conduzca a un comportamiento macroscópico correcto. La forma mas simple para introducir una pared sólida es utilizar la regla denominada *bounce back* en los nodos de la pared

$$f_i^{out}(\mathrm{x}, t) = f_{\bar{i}}^{in}(\mathrm{x}, t) \tag{13}$$

con $\boldsymbol{e}_i = -\boldsymbol{e}_{\bar{i}}$ y $f_i(\mathbf{x},t) = f_i(\mathbf{x},\mathbf{e},t) = f_{\bar{i}}(\mathbf{x},-\mathbf{e},t)$ de esta forma la función de distribución es rotada en la pared y vuelve al fluido con un momento opuesto en el paso de tiempo siguiente, de esta forma la velocidad en la pares es nula y asegura que el flujo a través de la misma es nulo. Esto es equivalente a la condición de no deslizamiento desde el punto de vista macroscópico.

2. PROCEDIMIENTO Y DESARROLLO

El proyecto se llevó a cabo con los alumnos cursantes de la materia Programación Concurrente, de la carrera Ingeniería en Informática. Se partió de código preexistente, escrito en Fortran, que analiza un flujo en 2D con un obstáculo. El programa parte de ciertas condiciones iniciales, y genera archivos de salida aptos para ser postprocesados con el programa GiD [10]. Como resultado de este postprocesamiento pueden obtenerse videos que muestran la evolución del fluido en el tiempo, o imágenes que representan un instante dado, como la que se aprecia en la Figura 1.

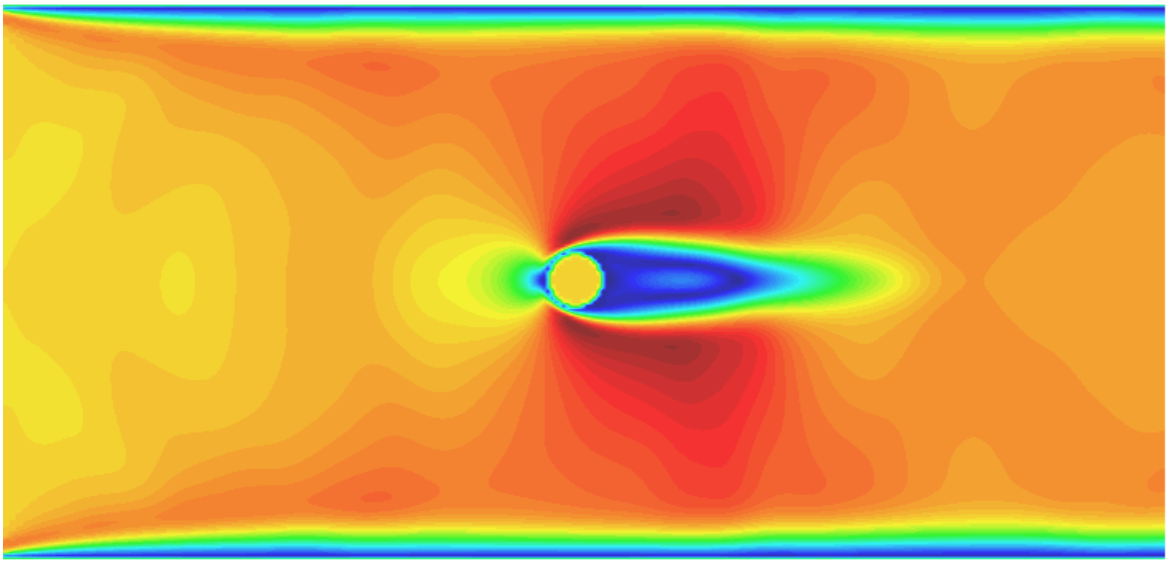

Figura 1: Fluido en 2D con un obstáculo circular

Se proveyó a los alumnos del programa en Fortran y de una traducción trivial a C. La consigna consistió en paralelizar uno o ambos programas utilizando OpenMP [11], y testear los resultados en el laboratorio Manycore. La tarea fue realizada por los alumnos en forma individual, con presentaciones en clase de los avances parciales, y con un informe final escrito.

El acceso remoto al laboratorio se realizó mediante ssh, con una conexión previa a una red privada virtual (VPN), utilizando el protocolo VPN de Cisco. Cada uno de los miembros del equipo (docentes y alumnos) disponía de una cuenta de usuario propia.
Las tareas de compilación se realizaban en el nodo de login, y los ejecutables resultantes podían correrse en el mismo nodo, o enviarse como trabajo a uno de los nodos de cálculo mediante el planificador de trabajos PBS. La ejecución en el nodo de login era útil para probar rápidamente el código recién compilado. Sin embargo, para realizar benchmarks era necesario enviar el trabajo a los nodos de cálculo. Esto se debe a que en el nodo de login puede haber múltiples usuarios corriendo procesos que compiten por los recursos del nodo. En cambio, al enviar el trabajo a un nodo de cálculo, el planificador garantiza que el proceso dispondrá de los recursos solicitados (cantidad de procesadores y memoria), por el tiempo requerido. Por ejemplo, el siguiente script requiere el uso de 32 procesadores y 256 MB de memoria durante 45 minutos, y ejecuta el programa glb_1d_omp con 16 y con 32 procesadores:

```
#!/bin/sh
#PBS -N omp_lb
#PBS -l walltime=0:45:00,ncpus=32,mem=256MB
cd $HOME/lb_omp
export OMP_NUM_THREADS=16
```

```
./glb_1d_omp
export OMP_NUM_THREADS=32
./glb_1d_omp
```

Para compilar se contó con los siguientes compiladores:

- GNU C Compiler 4.4.3 (gcc)
- Intel C Compiler 11.1 (icc)
- GNU Fortran Compiler 4.4.3 (gfortran)
- Intel Fortran Compiler 11.1 (ifort)

3. RESULTADOS

Uso del laboratorio como herramienta para la enseñanza de programación paralela

El laboratorio tuvo excelentes resultados como herramienta educativa. Los alumnos tuvieron oportunidad de experimentar con equipo habitualmente no disponible, y comprobar la eficiencia y escalabilidad de su código en equipos con gran número de procesadores.
En general la participación de los alumnos fue intensa, con acceso frecuente al laboratorio fuera de los horarios de clase, incluyendo horarios nocturnos y fines de semana.
El desarrollo se orientó mayormente a C. Todos los alumnos tenían alguna experiencia en Fortran, lenguaje utilizado en la asignatura "Métodos numéricos en computadoras", pero se sentían más confortables programando en C. Las primeras pruebas no mostraron diferencias significativas de rendimiento entre ambos lenguajes, con una leve ventaja en el rendimiento de las versiones en C que puede ser atribuida a la mayor experiencia de los alumnos con dicho lenguaje.

Experiencia en paralelización del Método de Lattice Boltzmann

El código utilizado es fácilmente paralelizable con OpenMP. Consiste en una serie de lazos que pueden ser paralelizados con las directivas OpenMP correspondientes (ej. #pragma omp parallel for).
Ejemplo:

```
#pragma omp for private (ix, jy)
for (j = 0; j < nbo; ++j) {
        ix = nbox[j];
        jy = nboy[j];
        f[ix][jy][3] = f[ix][jy][1] - rho[ix][jy] * 2.0 / 3.0 * ux[ix][jy];
        f[ix][jy][7] = f[ix][jy][5] + (f[ix][jy][2] - f[ix][jy][4]) * .5
                        - rho[ix][jy] * .5f * uy[ix][jy] - rho[ix][jy] * ux[ix][jy]
                        / 6.0;
        f[ix][jy][6] = f[ix][jy][8] + (f[ix][jy][4] - f[ix][jy][2]) * .5
                        + rho[ix][jy] * .5f * uy[ix][jy] - rho[ix][jy] * ux[ix][jy]
                        / 6.0;
}
```

Sin embargo, es necesario tener en cuenta múltiples factores para generar código eficiente, y las optimizaciones que funcionan en el código secuencial no siempre tienen el mismo efecto en el código paralelo. A continuación se presentan los resultados en una serie de aspectos. Los gráficos presentados se extraen de los informes de distintos alumnos, por lo que el estilo del gráfico y los valores obtenidos pueden diferir.

Compilador

Los programas fueron compilados con el compilador GNU y el compilador de Intel. En general este último logra un mejor nivel de optimización, y los tiempos de ejecución son consistentemente menores. Las opciones de compilación más comunes se muestran en la Tabla 2.

Tabla 2: Opciones de compilador (C) habituales

Compilador	Opciones
gcc	-fopenmp -O3
icc	-openmp -fast -xSSE4.2

En la figura siguiente, puede apreciarse el tiempo para un tamaño de problema específico, con distinta cantidad de procesos, y para código compilado con ambos compiladores:

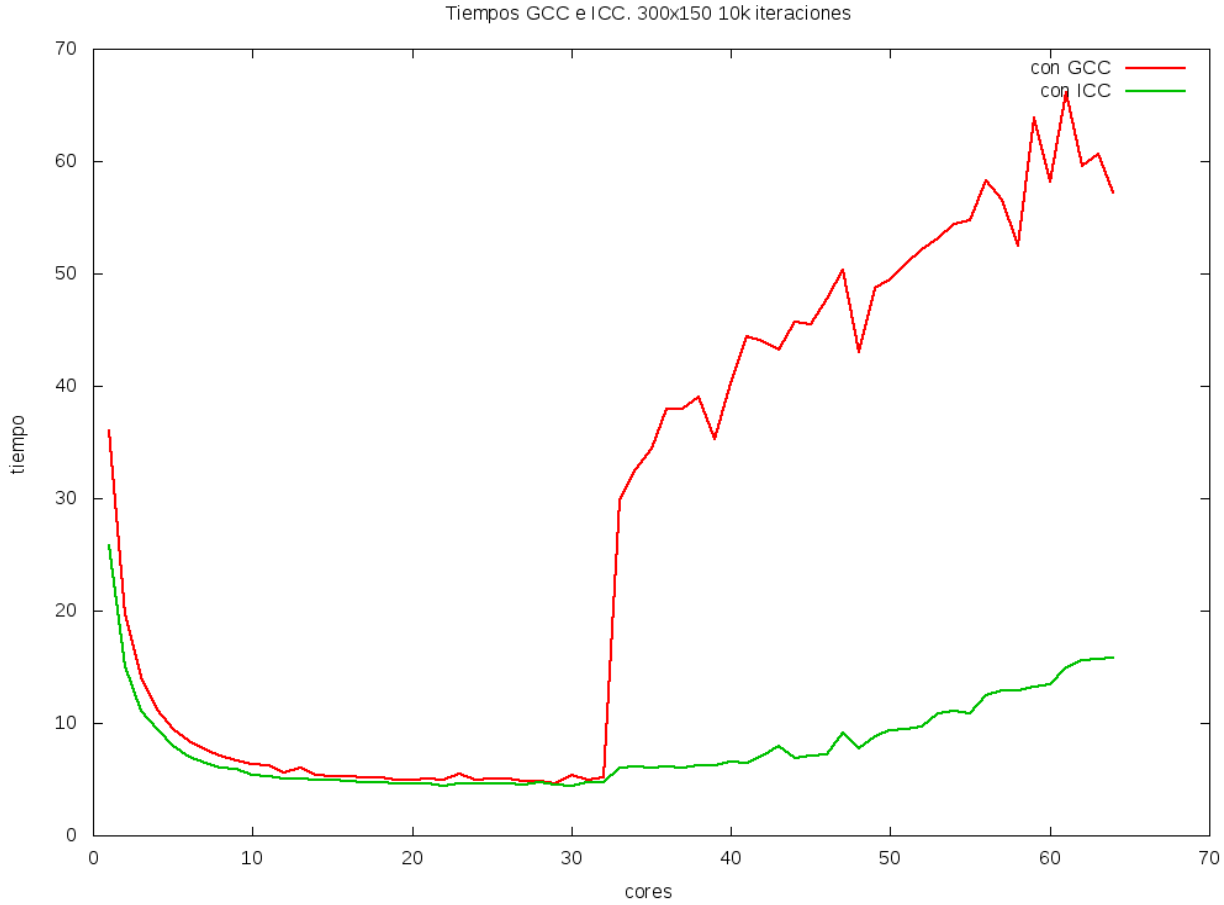

Figura 2: Tiempos de ejecución de un problema específico(datos de Ivan Slobodiuk)

Puede apreciarse que, para el mismo código, los tiempos de ejecución son menores en el programa compilado con icc. La diferencia es particularmente notable para números de procesos superiores a la cantidad de procesadores físicos, en los cuales el rendimiento de gcc (en este caso particular), cae sensiblemente.
En el gráfico siguiente se representa el speedup correspondiente al mismo caso.

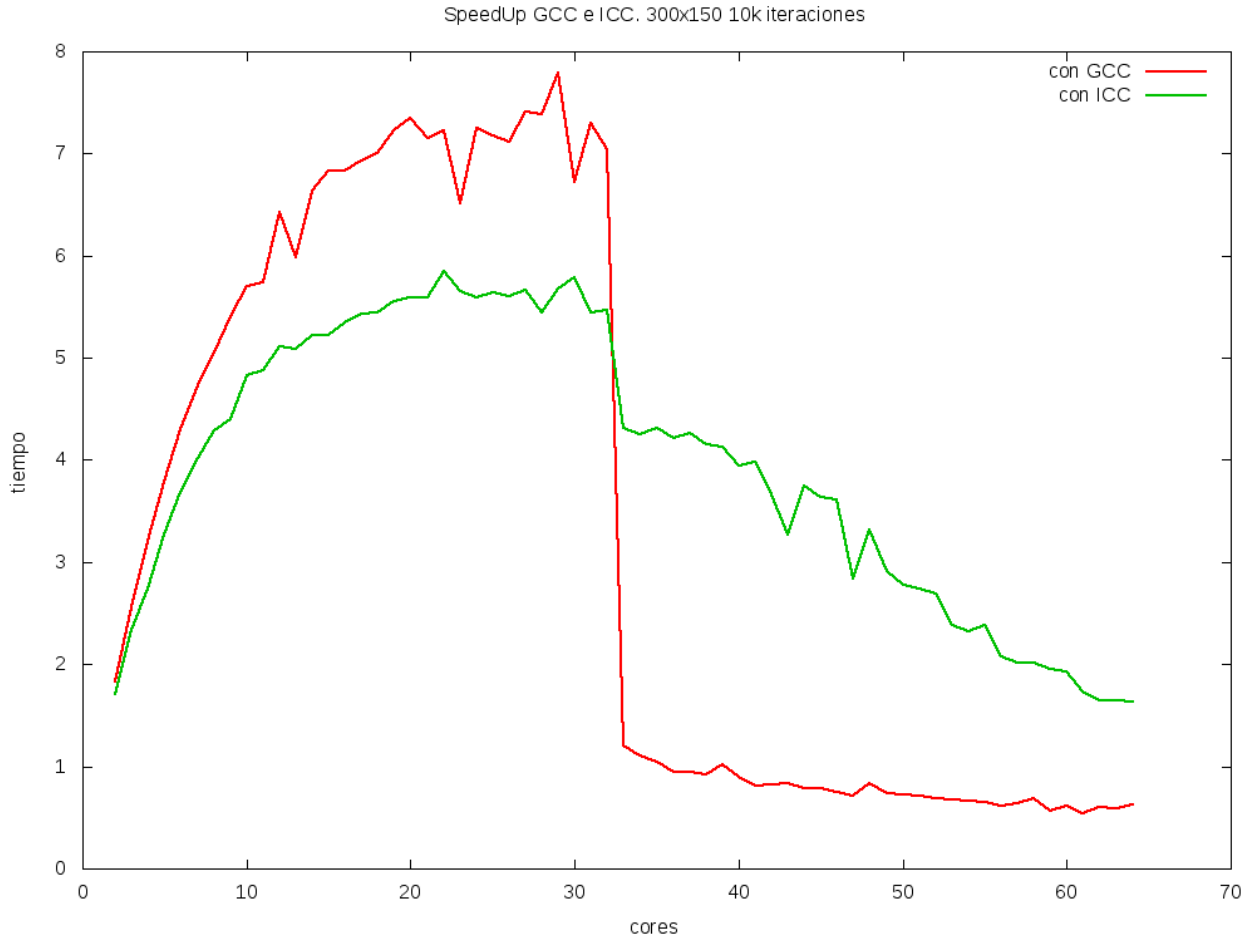

Figura 3: Speedup para el mismo caso de la figura anterior (datos de Ivan Slobodiuk)

En este caso, el speedup es mayor en gcc, pero esto ocurre porque el tiempo de referencia (con un sólo hilo), es mayor en el caso de gcc. Esto puede interpretarse cómo que la diferencia entre ambos compiladores es mayor en el caso del código secuencial. También puede observarse que el speedup máximo es bajo, en relación con el número de procesadores, por lo que la eficiencia del código también es baja.

El gráfico siguiente, obtenido por otro alumno, muestra valores consistentes con el anterior:

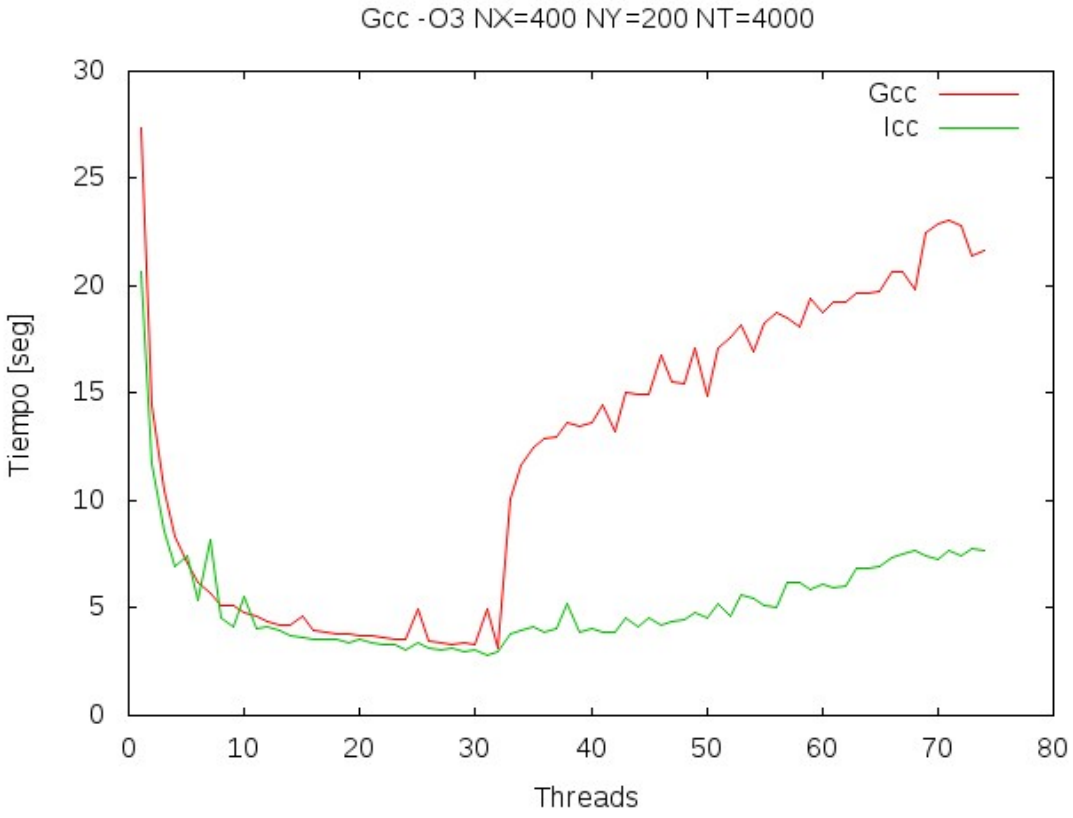

Figura 4: Speedup en función del número de hilos (datos de Emiliano Castro)

Disposición de los datos en memoria (memory layout)

Este tipo de problemas es altamente sensible a la disposición de los datos en memoria. Como C y Fortran utilizan distintas distribuciones de matrices multidimensionales en memoria, fue necesario invertir el orden de las matrices al traducir los programas de un lenguaje a otro.

Optimizaciones manuales y optimizaciones del compilador

En una primera etapa, se realizaron múltiples optimizaciones sobre el código:

- Unroll de loops
- Eliminación de indirecciones (uso de constantes, reemplazo de arreglos pequeños por variables individuales)
- Extracción de invariantes en los lazos
- Etc.

En general, la mayor parte de estas optimizaciones se mostraron innecesarias. Obtenían mejoras de rendimiento cuando se compilaba sin optimizaciones (-O0), pero las ganancias disminuían al compilar con optimizaciones (-O3, -fast). Esto significa que buena parte de esas optimizaciones son realizadas de todas formas por el compilador, y la mejor estrategia es brindarle al compilador la información necesaria para que tome buenas decisiones.
También resultó útil la generación de información de profiling, que permite al compilador usar información recopilada en tiempo de ejecución (-fprofile-generate y -fprofile-use).

Cantidad de hilos

La cantidad de hilos en un programa OpenMP puede controlarse externamente mediante la variable de entorno OMP_NUM_THREADS. Se realizaron pruebas para distinto numero de hilos, tanto menores a la cantidad de procesadores, como mayores (oversubscribing). En el caso de este problema particular (LBM), la sobresuscripción resulta perjudicial, y los tiempos de ejecución comienzan a aumentar una vez que la

cantidad de hilos supera la cantidad de procesadores disponibles. De hecho, el máximo speedup parece lograrse utilizando una cantidad de hilos ligeramente inferior a la cantidad de procesadores (ver figura 3).

El mismo efecto puede apreciarse en el gráfico siguiente, que mide la eficiencia en función del número de hilos:

$$Eficiencia = \frac{Tiempo\ 1\ hilo}{Tiempo\ n\ hilos \times n}$$

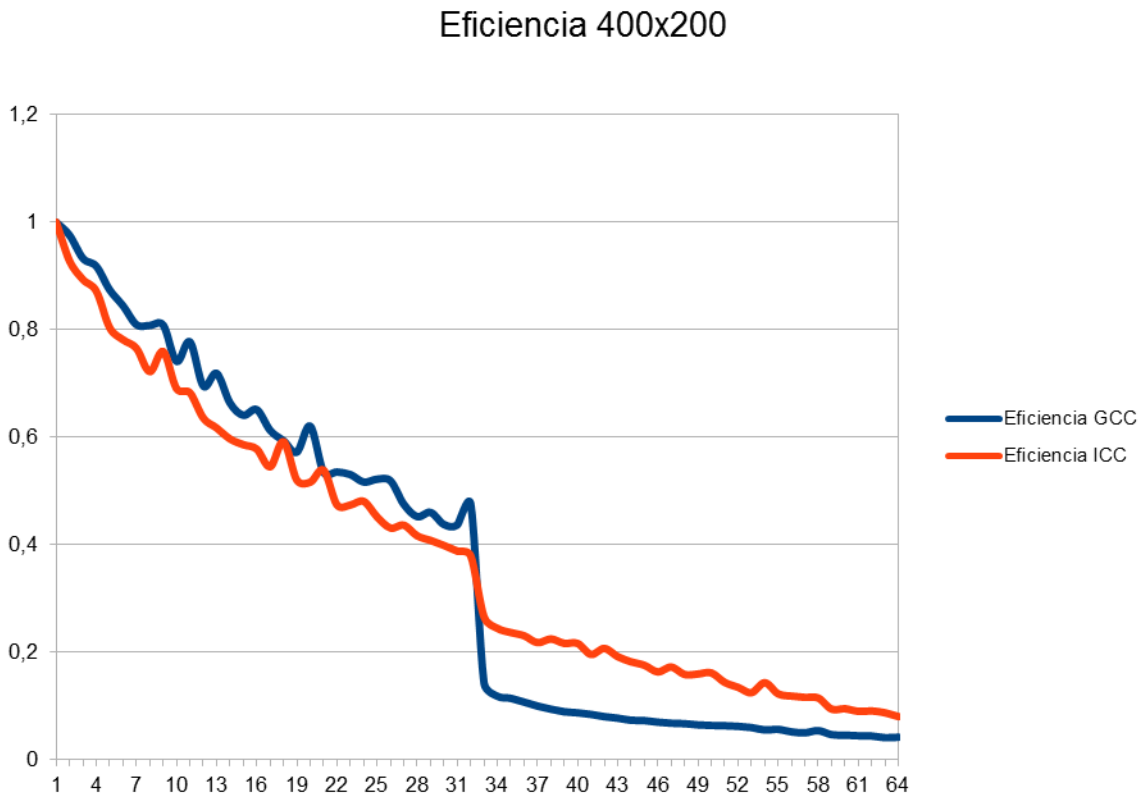

Figura 5: Eficiencia en función del número de hilos (datos de Fernando Stefanutti)

Uso de Hyperthreading

El uso de Hyperthreading (HT) no se mostró beneficioso en este caso. Si bien los nodos de cálculo tenían HT deshabilitado, fue posible hacer pruebas en el nodo login.

Tal como se describe en el apartado anterior, no es beneficiosa la sobresuscripción. Sin embargo, y en este problema específico, se detectó un comportamiento anómalo en el código compilado con gcc cuando se ejecutaba con cantidad de hilos próximas a la suma de procesadores físicos y lógicos (en este caso, 64). Los datos se presentan en la siguiente tabla, y no en un gráfico, debido a que las diferencias de escala dificultan la elaboración de un gráfico significativo. Puede observarse que en el caso de gcc, el tiempo de ejecución con más de 60 hilos es decenas de veces superior al tiempo secuencial. Esto no ocurre con el código compilado con icc. Esto pareciera indicar la existencia de algún problema en el algoritmo de planificación de hilos cuando la cantidad de estos es próxima a la cantidad total de procesadores físicos y lógicos. Si bien no se aprecia en la tabla, se comprobó que este comportamiento sólo se produce en la vecindad de ese número, y una vez sobrepasado, el tiempo de ejecución vuelve a disminuir.

Este comportamiento requiere más estudio, y no fue posible reproducirlo con otros programas (por ejemplo, multiplicación de matrices).

Se concluye que, al menos para este problema, el uso de HT no representa una ventaja, y es conveniente la asignación de los procesos a procesadores físicos utilizando afinidad de procesador (por ejemplo, con el comando taskset, en Linux, o utilizando las variables de entorno adecuadas, tal como KMP_AFFINITY)

Tabla 3: Tiempos de ejecución, HT habilitado

Hilos	Tiempo gcc	Tiempo icc
1	6,16	5,40
2	3,57	3,03
4	2,11	1,72
8	1,12	0,95
16	0,76	0,65
25	0,64	0,54
32	0,83	0,59
48	0,85	0,53
60	23,13	0,43
61	61,42	0,96
62	187,25	0,96
63	84,95	1,01
64	399,25	1,18

Tamaño del problema

El tamaño del problema, considerado como el tamaño de la grilla utilizada, influye fuertemente en el speedup y eficiencia del algoritmo. Obviamente, la cantidad de iteraciones afecta también el tiempo de ejecución, pero en forma lineal. El tamaño de la grilla, en cambio, se relaciona fuertemente con el uso de cache (localidad temporal y espacial). Los gráficos siguientes presentan tiempo de ejecución, speedup y eficiencia para distintos tamaños de grilla.

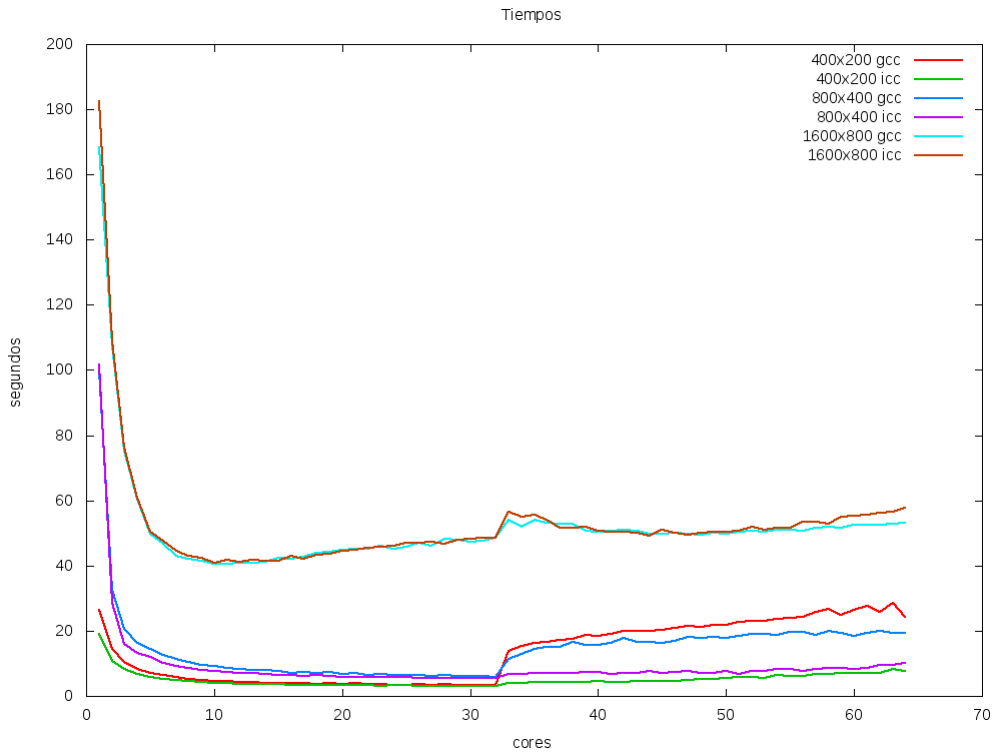

Figura 6: Tiempos de ejecución para distintos tamaños de problema (I. Slobodiuk)

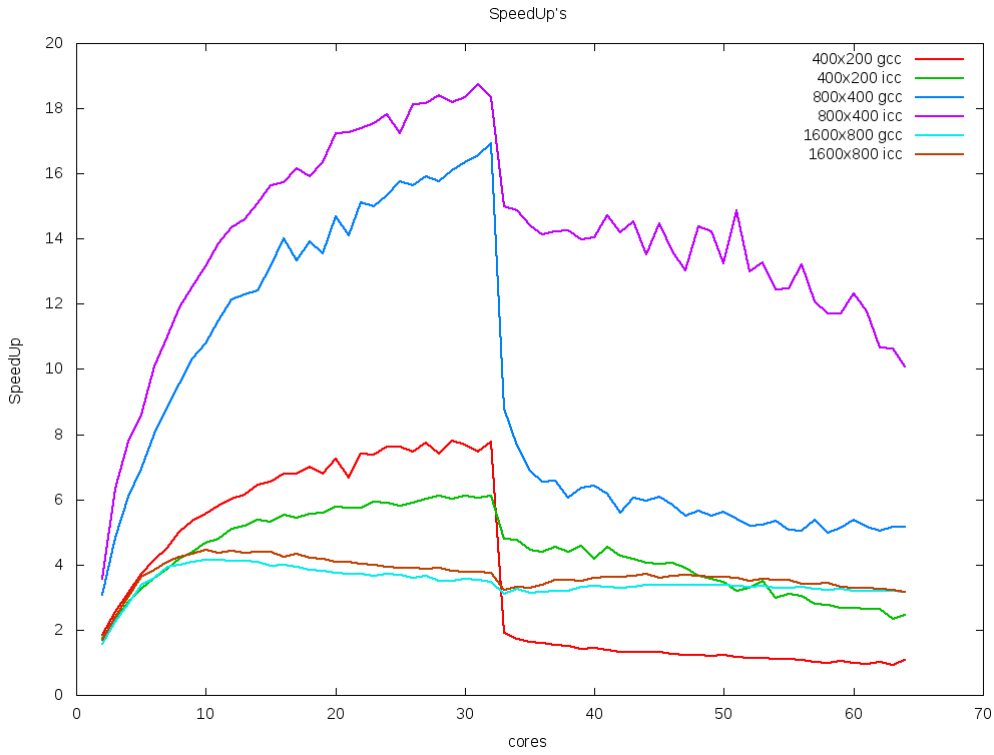

Figura 7: Speedup para distintos tamaños de problema (I. Slobodiuk)

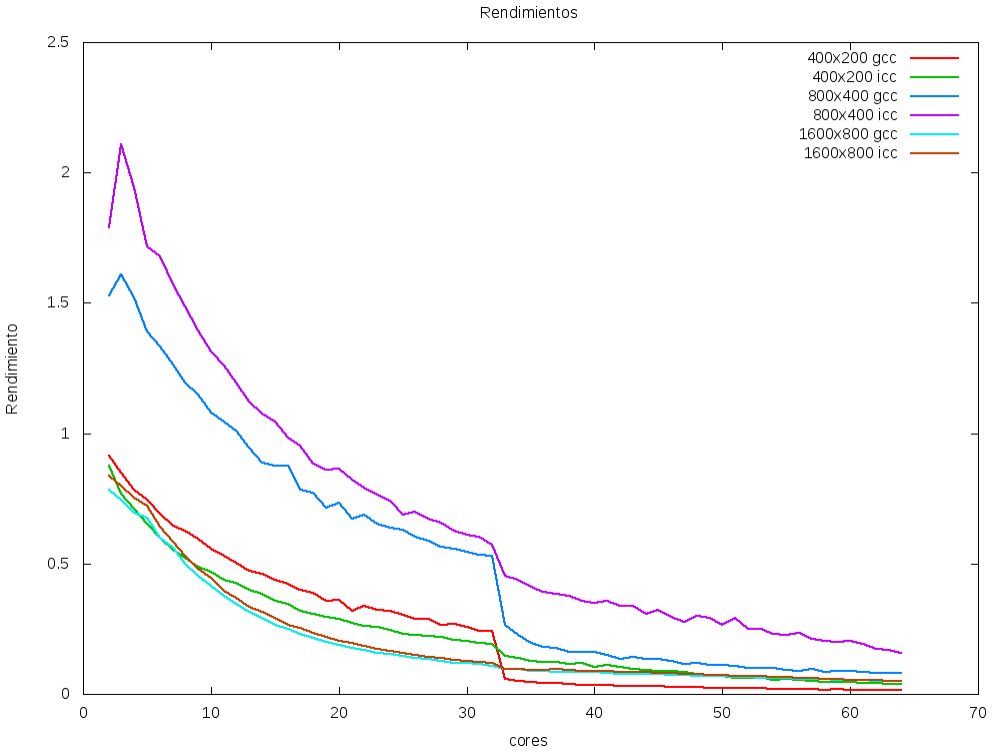

Figura 8: Eficiencia para distintos tamaños de problema (I. Slobodiuk)

Puede observarse que, para los tamaños analizados (400x200, 800x400 y 1600x800), se obtuvo el mejor rendimiento en el caso intermedio. Esto es consistente con el hecho de que para pequeños tamaños de problema, la sobrecarga de creación y destrucción de hilos y la sobrecarga de sincronización (barreras implícitas en cada bucle for) son significativas, mientras que para tamaños grandes esas sobrecargas son bajas, pero comienza a tener relevancia el uso de caché y de memoria (localidad espacial y temporal).
El caso intermedio logra un adecuado balance, y de hecho se observa que la eficiencia en algunos casos es superior a 1. Esto indica que la ganancia por la paralelización es mayor que la que se obtiene por agregar más procesadores. Este comportamiento puede estar causado por un uso más eficiente del caché, y requiere más investigación.

4. CONCLUSIONES

La experiencia de uso del laboratorio puede considerarse exitosa, tanto en relación con el proceso de aprendizaje de los alumnos, como del aprendizaje concreto sobre el uso de computadoras de memoria compartida con gran número de procesadores. Es intención de los autores repetir la experiencia, y avanzar sobre los temas no resueltos. El laboratorio se brinda por tiempos limitados, por lo cual es necesario planificar adecuadamente su uso para obtener el mayor rendimiento posible.
También resultó positiva la vinculación interdisciplinaria entre las carreras de Ingeniería Aeronáutica e Ingeniería en Informática, que permitió que alumnos de esta última aplicaran sus conocimientos de programación en el ámbito disciplinario de la primera.
Surgen asimismo distintos temas que ameritan mayor investigación, tales como el comportamiento de gcc para determinadas cantidades de hilos, o potencialidades de la plataforma que no pudieron ser utilizadas a fondo: uso de afinidad de procesador, explotación de los distintos niveles de caché, uso de herramientas de profiling (p. ej. Vtune).

REFERENCIAS

1. S. Succi. The Lattice Boltzmann Equation – For Fluid Dynamics and Beyond. Clarendon Press, 2001.
2. D. A. Wolf-Gladrow. Lattice Gas Cellular Automata and Lattice Boltzmann Models, volume 1725 of Lecture Notes in Mathematics. Springer, Berlin, 2000.
3. S. Chen and G. D. Doolen. Lattice Boltzmann Method for Fluid Flows. Annu. Rev. Fluid Mech., 30:329–364, 1998.
4. X. He and L.-S. Luo. A Priori Derivation of the Lattice Boltzmann Equation. Phys. Rev. E, 55(6):R6333–R6336, 1997.
5. P. Bhatnagar, E. P. Gross, and M. K. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. Phys. Rev., 94(3):511–525, 1954.
6. Y. H. Qian, D. d'Humieres, and P. Lallemand. Lattice BGK Models for Navier-Stokes Equation. Europhys. Lett., 17(6):479–484, 1992.
7. U. Frisch, D. d'Humieres, B. Hasslacher, P. Lallemand, Y. Pomeau, and J.-P. Rivet. Lattice Gas Hydrodynamics in Two and Three Dimensions. Complex Systems, 1:649–707, 1987.
8. http://software.intel.com/en-us/articles/intel-many-core-testing-lab/
9. http://ark.intel.com/Product.aspx?id=46499
10. http://gid.cimne.upc.es/
11. http://www.openmp.org/
12. Francisco Almeida, Domingo Giménez, José Miguel Mantas y Antonio Vidal – Introducción a la Programación Paralela – Paraninfo, 2008.