Implementación de Algoritmos de Multiplicación de Matrices Densas utilizando hilos y memoria compartida

Germán Hüttemann
Universidad Nacional de Asunción, Facultad Politécnica,
Asunción, Paraguay
ghuttemann@gmail.com

y

Marcelo D. Rodas
Universidad Nacional de Asunción, Facultad Politécnica,
Asunción, Paraguay
rodas.marcelo@gmail.com

Abstract

The Matrix multiplication algorithms are one of most popular solution used to show the functionalities of Parallel Algorithm, so, it became an important tool to solve hard problems in a reasonable time. The essence in the solutions of this problem are mainly focus on the pattern found to distribute the data on the matrix, so the process of the basic operations can be execute as most independent possible. Specifically, this document presents two solutions to the problem mentioned, the 1-D algorithm and 2-D algorithm. For each presented solutions it will be specified the characteristics and the set of results obtain with the tests. This paper proves that given enough parallel support to the environment of the algorithms, it will be obtain efficient and satisfactory results.

Keywords: Matrix Multiplication algorithms, 1-D Algorithm, 2-D Algorithm, dense matrix.

Resumen

Los algoritmos de multiplicación de matrices densas son una de las soluciones más utilizadas para mostrar el funcionamiento de algoritmos paralelos, y por ende, se convierte en una herramienta para la solución de problemas difíciles en un tiempo razonable. Lo esencial en las soluciones de este problema, principalmente se enfocan en el patrón para distribuir los datos de una matriz, de tal forma que el cálculo de las operaciones básicas pueda ejecutarse lo más independientemente posible. Particularmente, este documento presenta dos soluciones al problema mencionado, el algoritmo 1-D y el algoritmo 2-D. Se presentarán las características de cada algoritmo, conjuntamente con los resultados obtenidos de las pruebas realizadas. Además, se podrá ver que la paralelización del problema presentado, solo será efectiva si se procesa en un ambiente con soporte para la paralelización, o sea, que haya tantos procesadores individuales para el nivel de paralelización esperado.

Palabras clave: Algoritmos de multiplicación de matrices, Algoritmo 1-D, Algoritmo 2-D, matrices densas.

1 Introducción

Este trabajo plantea la solución del problema de la multiplicación de matrices densas. Se presentan 2 soluciones, utilizando los algoritmos paralelos con particionamiento en una dimensión (algoritmo 1-D) y en dos dimensiones (algoritmo 2-D). Además, se presenta el algoritmo secuencial sin particionamiento para poder comparar con los algoritmos paralelos. Ambos algoritmos pueden ser considerados como una adaptación de los algoritmos 1-D y 2-D, para multiplicación de matriz-vector, presentados en [1].
El trabajo esta organizado de la siguiente manera: en la sección 2 se describe el problema a solucionar. En la sección 3 se presenta el modelo matemático para la multiplicación de matrices. En la sección 4, se presenta la descripción del algoritmo secuencial. Luego, en la sección 5, se presenta la descripción de los algoritmos paralelos. Finalmente, en la sección 6 se presentan los resultados obtenidos en las pruebas; en la sección 7, las conclusiones obtenidas y, en la sección 8, se proponen algunos trabajos futuros para mejorar el análisis del problema abordado.

2 Descripción del Problema

El problema consiste en realizar la multiplicación de matrices densas. Para ello se busca aprovechar las características de las operaciones básicas de multiplicación de matrices, que permiten la paralelización del proceso completo. La paralelización debe ser realizada con hilos de ejecución de un mismo programa, utilizando el mecanismo de comunicación de memoria compartida.

3 Modelo Matemático

La multiplicación de matrices tiene como entrada dos matrices A y B multiplicables, de dimensiones MxN y NxR, respectivamente. Como resultado, se obtiene otra matriz de tamaño MxR, normalmente llamada matriz C. En la Figura 1 puede apreciarse un ejemplo de multiplicación de matrices. Los elementos de las matrices pueden tomar valores de conjuntos de números enteros, reales, complejos, etc.

$A_{1,1}$	$A_{1,2}$	…	$A_{1,N}$
$A_{2,1}$	$A_{2,2}$	…	$A_{2,N}$
…	…	…	…
$A_{M,1}$	$A_{M,2}$	…	$A_{M,N}$

x

$B_{1,1}$	$B_{1,2}$	…	$B_{1,R}$
$B_{2,1}$	$B_{2,2}$	…	$B_{2,R}$
…	…	…	…
$B_{N,1}$	$B_{N,2}$	…	$B_{N,R}$

=

$C_{1,1}$	$C_{1,2}$	…	$C_{1,R}$
$C_{2,1}$	$C_{2,2}$	…	$C_{2,R}$
…	…	…	…
$C_{M,1}$	$C_{M,2}$	…	$C_{M,R}$

Figura 1. Ejemplo Gráfico de la multiplicación de matrices A x B = C

4 Descripción de la Implementación Secuencial

Para realizar la multiplicación de las matrices A y B, se creó la función genérica de multiplicación de la Figura 2. Esta función es capaz de recibir como argumentos (además de las matrices A, B y C) un rango de filas y columnas, que determina qué porción (elemento, fila, columna, bloque) de la matriz C se quiere calcular. De esta manera, resulta sencillo dividir la tarea de calcular la multiplicación de dos matrices cualesquiera, indistintamente de que ésta vaya a paralelizarse o no. Para la solución secuencial se utiliza esta función con un rango de filas y columnas que abarca toda la matriz C.

```
desde i = fila_inicial hasta (fila_inicial + cantidad_filas)
  desde j = columna_inicial hasta (columna_inicial + cantidad_columnas)
    C[i,j] = 0
    desde k = 1 hasta (cantidad_filas_de_A o cantidad_columnas_de_B)
      C[i,j] = C[i,j] + A[i,k] * B[k,j]
    fin-desde
  fin-desde
fin-desde
```

Figura 2. Función Principal para la multiplicación de matrices.

4.1 Costo Asintótico

Es importante mencionar que para facilitar el calculo de los costos, se asumen los cálculos sobre matrices cuadradas de tamaño *L*. El costo asintótico de la multiplicación de matrices con el algoritmo propuesto, considerando a la combinación de sumas y multiplicaciones como operaciones elementales, es del orden de $O(L^3)$.

5 Descripción de la Implementación Paralela

Para paralelizar la tarea de multiplicación de las dos matrices A y B, se utiliza un particionamiento de datos de salida de la matriz C. Aquí también se utiliza la función de la Figura 2, con un rango de filas y columnas de tal forma que cada hilo solo calculo la sub-matriz que le corresponda.

5.1 Algoritmo 1-D

Este algoritmo realiza un particionamiento en una dimensión, dividiendo las filas de la matriz C entre los hilos disponibles. Para calcular una fila de la matriz C, se necesita la fila correspondiente de la matriz A y toda la matriz B. De esta forma, cada hilo procesará un subconjunto de filas de la matriz A y por ende, calculará la misma cantidad de filas de la matriz C (Figura 3).
En la Figura 4 se presenta el algoritmo de particionamiento 1-D. Si la matriz C tiene M filas y están disponibles P hilos, h_1, h_2, ..., h_p, a cada uno de los hilos se les asignará M / P filas, de tal manera que las primeras M / P filas serán asignadas al hilo h_1, las siguientes M / P filas al hilo h_2, y así sucesivamente.
En el caso de que $\lfloor M / P \rfloor < M / P$, existirán $M - \lfloor M / P \rfloor * P$ filas restantes. Estas filas restantes serán asignadas al hilo h_p (Figura 5). Cabe destacar que esto podría generar un gran desbalanceo de la carga. Para reducir este desbalanceo, lo que podría realizarse es asignar cada una de las filas restantes a los hilos h_1, h_2, ..., h_S, donde $S = M - \lfloor M / P \rfloor * P$.

$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$		$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$		P_1	P_1	P_1	P_1
$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	$A_{2,4}$		$B_{2,1}$	$B_{2,2}$	$B_{2,3}$	$B_{2,4}$		P_2	P_2	P_2	P_2
$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,4}$	x	$B_{3,1}$	$B_{3,2}$	$B_{3,3}$	$B_{3,4}$	=	P_3	P_3	P_3	P_3
$A_{4,1}$	$A_{4,2}$	$A_{4,3}$	$A_{4,4}$		$B_{4,1}$	$B_{4,2}$	$B_{4,3}$	$B_{4,4}$		P_4	P_4	P_4	P_4

Figura 3. Ejemplo del proceso realizado para el Algoritmo 1-D con 4 hilos (1 por fila).

```
desde i = 1 hasta cantidad_hilos
  asignar matrices al hilo[i]
  asignar filas al hilo[i]
fin-desde

si hay filas restantes entonces
  asignar filas restantes a último hilo
fin-si
```

Figura 4. Algoritmo de Particionamiento 1-D.

$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$		$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$		P_1	P_1	P_1	P_1
$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	$A_{2,4}$		$B_{2,1}$	$B_{2,2}$	$B_{2,3}$	$B_{2,4}$		P_2	P_2	P_2	P_2
$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,4}$	x	$B_{3,1}$	$B_{3,2}$	$B_{3,3}$	$B_{3,4}$	=	P_3	P_3	P_3	P_3
$A_{4,1}$	$A_{4,2}$	$A_{4,3}$	$A_{4,4}$		$B_{4,1}$	$B_{4,2}$	$B_{4,3}$	$B_{4,4}$		P_4	P_4	P_4	P_4
$A_{5,1}$	$A_{5,2}$	$A_{5,3}$	$A_{5,4}$							P_4	P_4	P_4	P_4

Figura 5. Ejemplo del proceso realizado para el Algoritmo 1-D con 4 hilos y con 1 fila restante.

5.2 Algoritmo 2-D

Este algoritmo realiza un particionamiento en dos dimensiones. En este caso, se trata de subdividir mejor el conjunto de datos, aprovechando que para calcular el primer elemento ($C_{1,1}$) solo de necesita la primera fila de A ($A_{1,*}$) y la primera columna de B ($B_{*,1}$). De esta forma cada hilo puede calcular un solo elemento de la matriz C, o idealmente, una sub-matriz de la matriz C.

Para el particionamiento en dos dimensiones, se considera que los *P* hilos disponibles conforman una malla cuadrada de $\sqrt{P} * \sqrt{P}$ nodos, tal que la primera fila contendrá a los hilos h_1, h_2, ..., h_t, la segunda fila a los hilos h_{t+1}, h_{t+2}, ..., h_{2t}, y así sucesivamente, tal que $t = \sqrt{P}$ (Figura 6). Para que esto sea posible, la cantidad *P* de hilos debe ser un cuadrado perfecto.

En la figura 7 se presenta el algoritmo de particionamiento 2-D. Si la matriz C tiene M filas y N columnas, cada hilo, h_1, h_2, ..., h_p, de la malla, tendrá asignado un bloque de la matriz, conformado por $M / \sqrt{P}$ filas y $N / \sqrt{P}$ columnas. Esta distribución ocurrirá de tal manera que el bloque con las primeras $M / \sqrt{P}$ filas y las primeras $N / \sqrt{P}$ columnas, será asignado al hilo h_1. El bloque, de mismo tamaño, inmediatamente contiguo hacia la derecha, será asignado al hilo h_2, y así sucesivamente, hasta llegar al hilo h_t. Con los hilos de las siguientes filas de la malla se procederá análogamente.

En el caso de que $\lfloor M / \sqrt{P} \rfloor < M / \sqrt{P}$ y/o $\lfloor N / \sqrt{P} \rfloor < N / \sqrt{P}$, existirán $M - \lfloor M / \sqrt{P} \rfloor * \sqrt{P}$ filas restantes y/o $N - \lfloor N / \sqrt{P} \rfloor * \sqrt{P}$ columnas restantes. Estas filas restantes y/o columnas restantes serán asignadas de la siguiente manera: para cada fila de hilos de la malla, las columnas restantes serán asignadas al hilo h_{jt}, donde $j = 1,2,...,\sqrt{P}$. Además, para cada columna de hilos de la malla, las filas restantes serán asignadas al hilo h_i, donde $i = P - \sqrt{P} + 1, P - \sqrt{P} + 2, ..., P$ (Figura 8).

$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$
$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	$A_{2,4}$
$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,4}$
$A_{4,1}$	$A_{4,2}$	$A_{4,3}$	$A_{4,4}$

x

$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$
$B_{2,1}$	$B_{2,2}$	$B_{2,3}$	$B_{2,4}$
$B_{3,1}$	$B_{3,2}$	$B_{3,3}$	$B_{3,4}$
$B_{4,1}$	$B_{4,2}$	$B_{4,3}$	$B_{4,4}$

=

P_1	P_1	P_2	P_2
P_2	P_2	P_2	P_2
P_3	P_3	P_4	P_4
P_3	P_3	P_4	P_4

Figura 6. Ejemplo del proceso realizado para el Algoritmo 2-D con 4 hilos (2 filas y 2 columnas cada uno).

```
k = 1
desde i = 1 hasta raiz_cuadrada_cantidad_hilos
  desde j = 1 hasta raiz_cuadrada_cantidad_hilos
    asignar matrices al hilo[k]
    asignar filas al hilo[k]
    asignar columnas al hilo[k]

    k = k + 1
  fin-desde

  si hay columnas restantes entonces
    asignar columnas restantes al hilo[k]
  fin-si
fin-desde

si hay filas restantes entonces
  k = cantidad_hilos - raiz_cuadrada_cantidad_hilos + 1
  mientras k <= cantidad_hilos hacer
    asignar filas restantes a hilo[k]
    k = k + 1
  fin-mientras
fin-si
```

Figura 7. Algoritmo de Particionamiento 2-D.

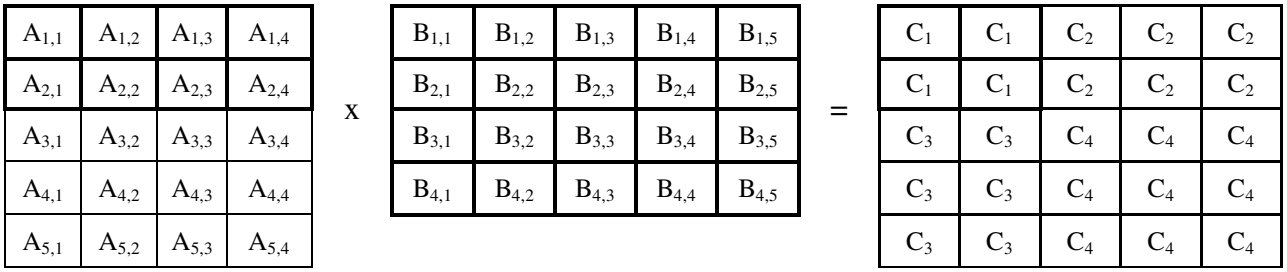

Figura 8. Ejemplo del proceso realizado para el Algoritmo 2-D con 4 hilos y 1 fila y 1 columna restantes.

5.3 Ajuste de la Cantidad de Hilos

La cantidad de hilos disponibles para la multiplicación determina el grado de división de trabajo que habrá. Mientras más hilos estén disponibles, mayor será el grado de división del trabajo.
Cómo podrá notarse, el grado de división tiene un límite. El particionamiento en una dimensión no tendrá sentido si $M < \mathrm{P}$. Análogamente, el particionamiento en dos dimensiones no tendrá sentido si $M < \sqrt{P}$ o $N < \sqrt{P}$.
Por lo visto anteriormente, es necesario ajustar la cantidad de hilos disponibles cuando se cumplen las condiciones mencionadas. Para el caso de particionamiento en una dimensión, el valor al que se ajustará la cantidad de hilos es $\lfloor \sqrt{M} \rfloor$. Para el caso de particionamiento en dos dimensiones, el valor al que se ajustará $\max\left(\lfloor \sqrt{M} \rfloor, \lfloor \sqrt{N} \rfloor\right)$.

5.4 Costo Asintótico

Para facilitar el cálculo de los costos, se utilizan matrices cuadradas de tamaño *L*. Para el caso concurrente, se agrega un sobrecosto relativo al particionamiento de la matriz C entre los hilos que realizarán la multiplicación. Para el particionamiento en una dimensión, se incurre siempre a un sobrecosto $O(P)$, ya que se debe asignar la partición a cada hilo. Para el particionamiento en dos dimensiones, el sobrecosto en el caso mejor es de $O(P)$, al asignar la partición a cada hilo y, en el caso peor, es de $O\left(P + \sqrt{P}\right)$, ya que si hubo filas restantes, se deberá iterar sobre la última fila de hilos de la malla para asignarles las filas restantes.
Básicamente, el sobrecosto de particionamiento es una función lineal de la cantidad de hilos. Considerando que la cantidad de hilos disponibles no es de gran magnitud, el sobrecosto agregado es despreciable cuando el valor de *L* es lo suficientemente grande. De esta manera, podríamos omitir el sobrecosto y calcular el costo asintótico para la multiplicación concurrente solo en función a la cantidad de operaciones elementales realizadas por cada hilo.
Continuando la asunción de las matrices cuadradas y suponiendo que las divisiones serán siempre exactas, realizamos el cálculo del costo asintótico para las versiones concurrentes, en una y dos dimensiones. Para el caso de una dimensión, cada hilo tendrá asignado L/P filas y *L* columnas de la matriz C, teniendo así L^2/P elementos que calcular. Para calcular cada uno de estos elementos, cada hilo necesitará realizar *L* operaciones elementales. Por tanto, el costo asintótico total será $O(L^3/P)$. Para el caso de dos dimensiones, cada hilo tendrá asignado $L/\sqrt{P}$ filas y $L/\sqrt{P}$ columnas de la matriz C, teniendo así L^2/P elementos que calcular. Para calcular cada uno de estos elementos, cada hilo necesitará realizar *L* operaciones elementales. Por tanto, el costo asintótico total será $O(L^3/P)$, al igual que el particionamiento en una dimensión.
Es importante destacar que como no existe interdependencia entre tareas, el grado máximo de concurrencia siempre será igual a *P*, que es la ganancia que se obtiene al paralelizar el problema entre *P* hilos. Sin embargo, la máxima ganancia obtenida está acotada por *F*, la cantidad de procesadores o núcleos disponibles en la computadora en la que se ejecuta el programa. De esta manera, se obtendrán ganancias significativas cuando $1 < P \leq F$. Cuando $P > F$, la ganancia no será mayor a la obtenida con $P = F$, sin embargo, la ganancia seguirá siendo mayor que la ganancia obtenida con $P < F$, hasta que los sobrecostos relacionados al paralelismo hagan decaer el rendimiento. En este trabajo no se realizaron pruebas para comprobar el valor hasta el cual exista ganancia cuando $P > F$.

6 Resultados

6.1 Ambiente de Prueba

Las pruebas se realizaron en un maquina con las siguientes características:

- Procesador: Intel Xeon, con 8 procesadores x86_64 del tipo E5310 de 1.60GHz y con 4MB de cachê.
- Memoria: 4GB.
- Sistema Operativo: Linux Fedora 8 (kernel 2.6.23.1-42.fc8).
- Compilador: GCC 4.1.2 20070925.

Los porcentajes de utilización de procesador y memoria, en el momento de las pruebas, fueron de aproximadamente 2% y 15%, respectivamente. Es decir, el sistema estaba prácticamente sin carga.

6.2 Métricas

Las métricas medidas para el análisis de las distintas implementaciones son las siguientes:

1) Tiempo de Ejecución.
2) Sobrecosto de Concurrencia.
    a. Tiempo Total de Particionamiento.
    b. Tiempo Total de Creación de hilos.
    c. Tiempo Promedio de Creación de hilos.
3) Aceleración (tiempo secuencial / tiempo concurrente).

Cabe destacar que todos los tiempos fueron medidos en milisegundos. Además, como ya se mencionó más arriba, el sobrecosto de particionamiento, al ser una función lineal de la cantidad de hilos, es ínfimo. Se omiten los resultados sobre la métrica de particionamiento debido a que como máximo fue de 2 milisegundos. Tampoco se probó con valores de P mayores a 16, por lo que no se puede analizar su impacto sobre esta métrica.

6.3 Gráficos Resultantes

6.3.1. Comparación Algoritmo Secuencial con Algoritmos Concurrentes

Se empieza mostrando la relación existente entre los tiempos de ejecución de la multiplicación del algoritmo secuencial y el algoritmo 1-D con sus distintas cantidades de hilos: con 4 hilos (Figura 9.a.), con 9 hilos (Figura 9.b.) y con 16 hilos (Figura 9.c.).
De forma paralela a las figuras mencionadas se presenta la comparación entre los tiempos de ejecución de la multiplicación de algoritmos secuenciales y el algoritmo 2-D con sus distintas cantidades de hilos: con 4 hilos (Figura 10.a.), con 9 hilos (Figura 10.b.) y con 16 hilos (Figura 10.c.).
Cabe destacar que entre las versiones 1-D y 2-D no existen prácticamente diferencias, lo cual es de esperarse, ya que los costos asintóticos son los mismos. Sin embargo, se podrían analizar las diferencias a un más bajo nivel, si se varían los tamaños de la matrices de manera a forzar el desbalanceo de carga, lo cual se da cuando el particionamiento no es exacto. Creemos que eligiendo más delicadamente los tamaños de matrices, podría analizarse el comportamiento de cada algoritmo respecto al particionamiento y su efecto en el tiempo de ejecución.

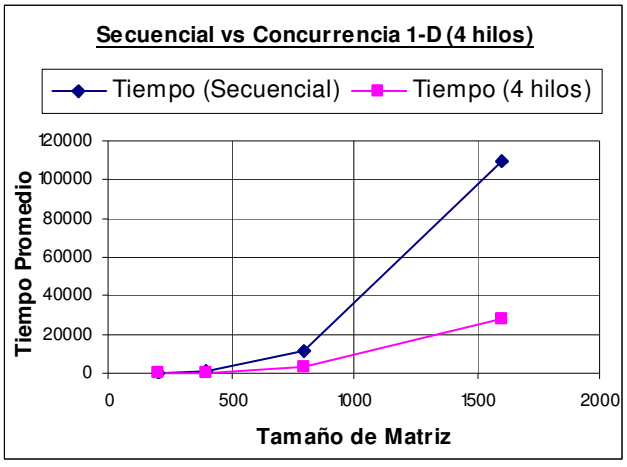

Figura 9.a. Comparación de Tiempos (sec. vs. 4 hilos).

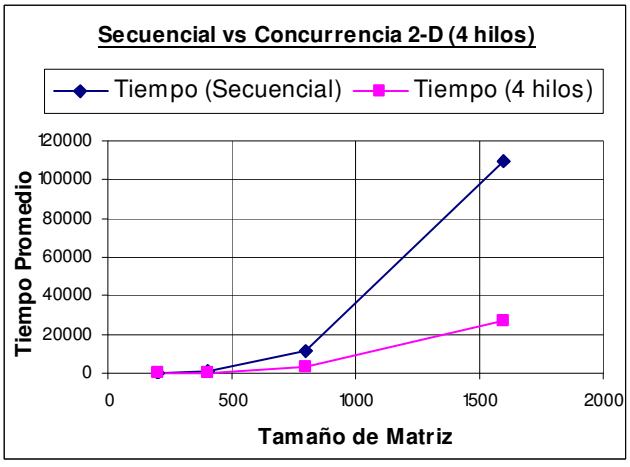

Figura 10.a. Comparación de Tiempos (sec. vs. 4 hilos).

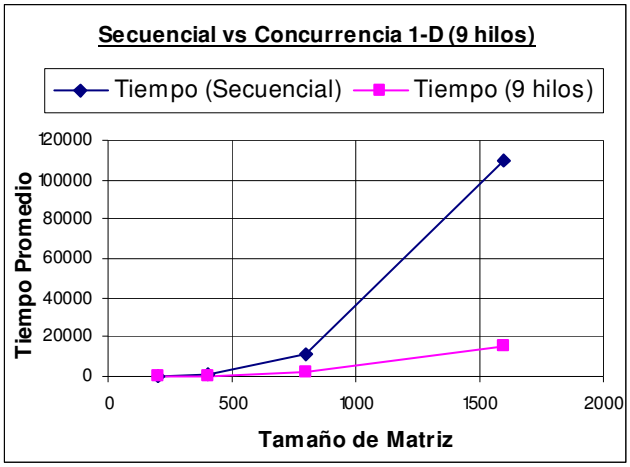

Figura 9.b. Comparación de Tiempos (sec. vs. 16 hilos).

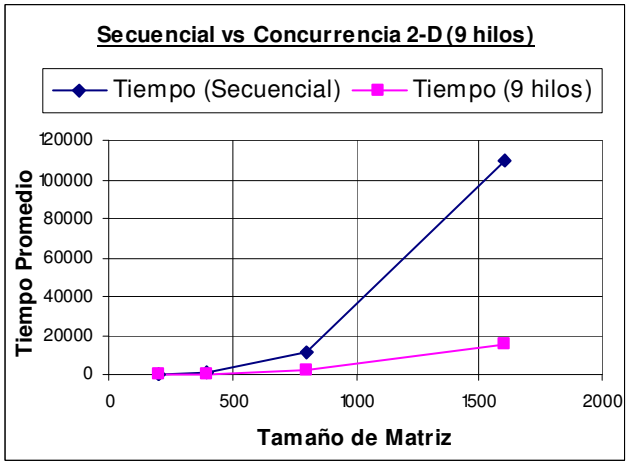

Figura 10.b. Comparación de Tiempos (sec. vs. 9 hilos).

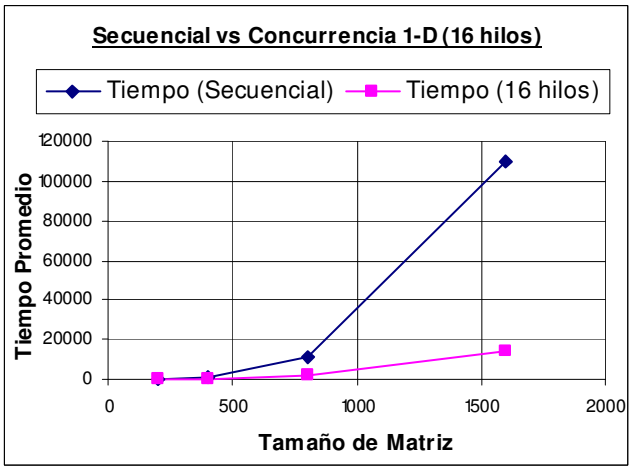

Figura 9.c. Comparación de Tiempos (sec. vs. 16 hilos).

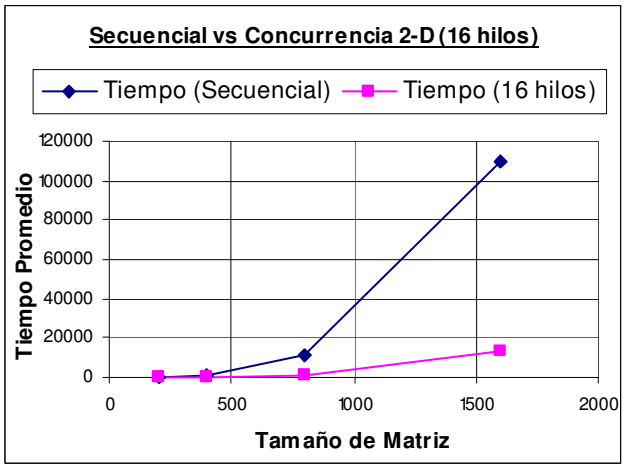

Figura 10.c. Comparación de Tiempos (sec. vs. 16 hilos).

6.3.2. Comparación de distintas cantidades de hilos para algoritmo 1-D

La Figura 11 muestra cómo se va reduciendo el tiempo de ejecución del algoritmo 1-D a medida que va aumentando la cantidad de hilos. Cabe notar que la reducción de tiempo es proporcional al incremento de la cantidad de hilos. Cada vez que la cantidad de hilos se duplica, el tiempo se reduce aproximadamente a la mitad, hasta llegar al límite de paralelización soportado por hardware, lo cual puede notarse en la curva correspondiente a 9 hilos.

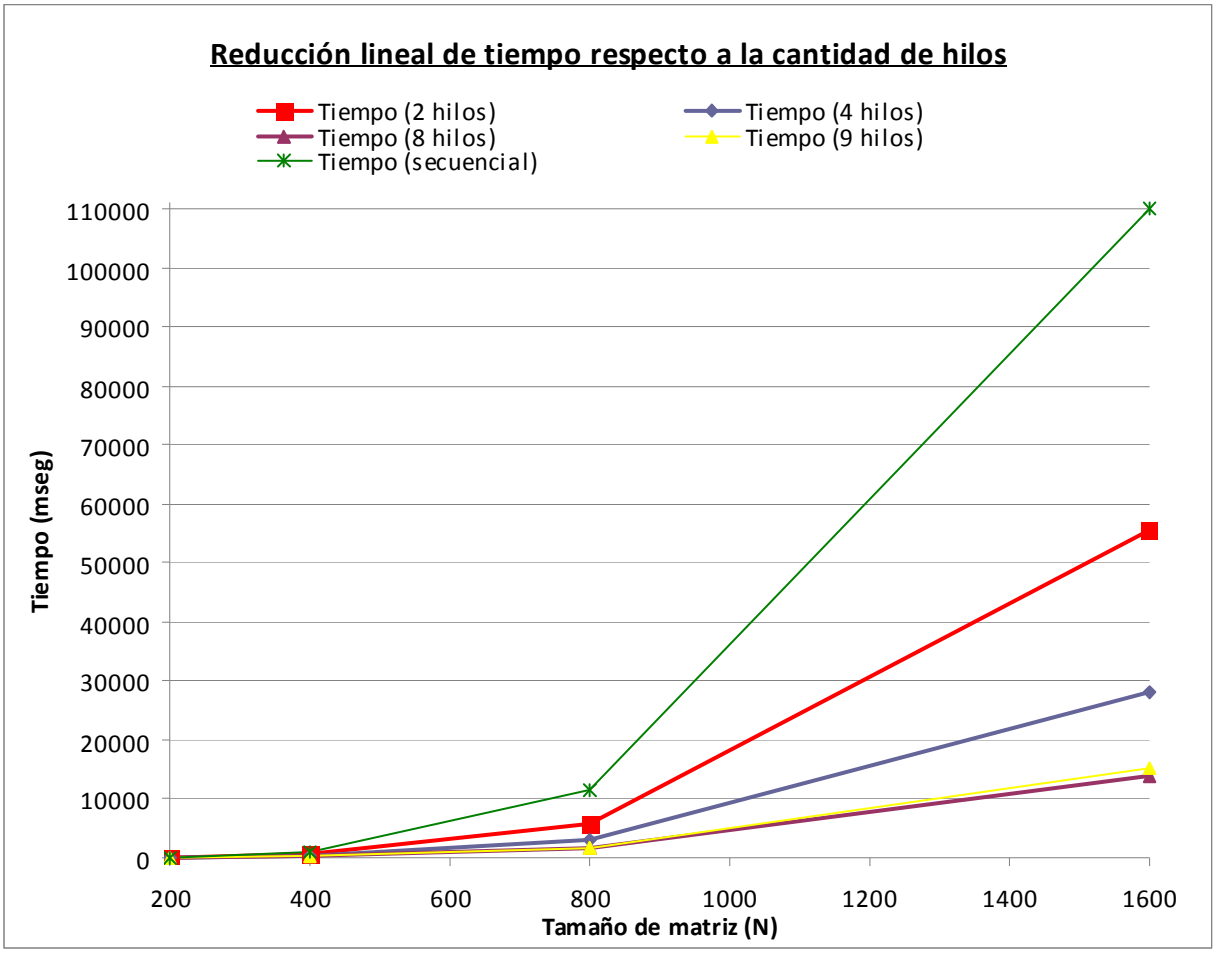

Figura 11. Reducción lineal del tiempo de ejecución del algoritmo 1-D respecto a la cantidad de hilos.

*6.3.3. **Tiempo de Creación de hilos (sobrecosto)***

Como es de esperar, el tiempo de creación de hilos va aumentando a medida que aumenta la cantidad de hilos. Algo llamativo que puede notarse en las Figuras 12.a y 12.b es que el incremento en el tiempo de creación de hilos no es lineal. Por el contrario, a medida que va aumentando la cantidad de hilos el tiempo el incremento en el tiempo se reduce. Esta reducción podría deberse a que de alguna manera se optimizan las llamadas a la primitiva de creación de hilos tras varias llamadas realizadas.

Por otra parte, como puede verse en las Figuras 12.c y 12.d, el tiempo de creación promedio de cada hilo también va aumentando a medida que se crea una mayor cantidad de hilos. Esto podría deberse a que a medida que los hilos van creándose ya van iniciando su ejecución, por lo que reduce la disponibilidad de procesador. Este inconveniente puede solucionarse utilizando una barrera (pthread_barrier) para sincronizar el inicio de ejecución de todos los hilos. La excepción a esto son las ejecuciones con 16 hilos. Pruebas con mayor cantidad de hilos no fueron realizadas por lo que no pudo analizarse si esto es una tendencia o si fue simplemente una excepción.

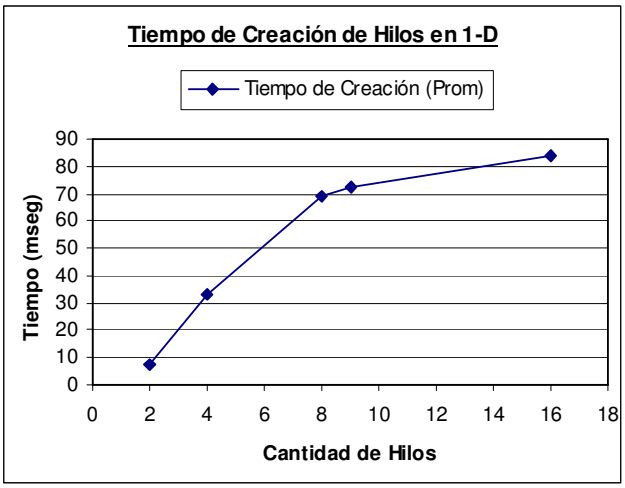

Figura 12.a. Tiempo de Creación de Hilos (Prom) en 1-D.

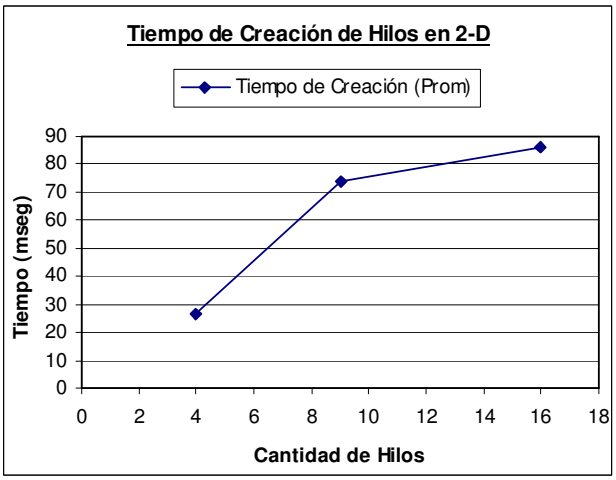

Figura 12.b. Tiempo de Creación de Hilos (Prom) en 2-D.

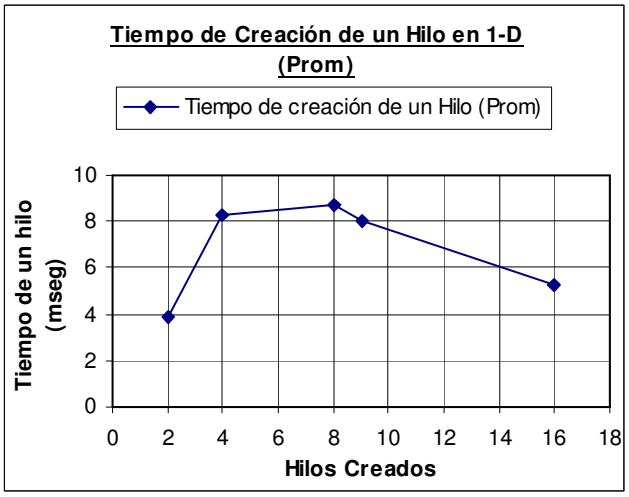

Figura 12.c. Tiempo de Creación de un solo Hilo (Prom) en 1-D.

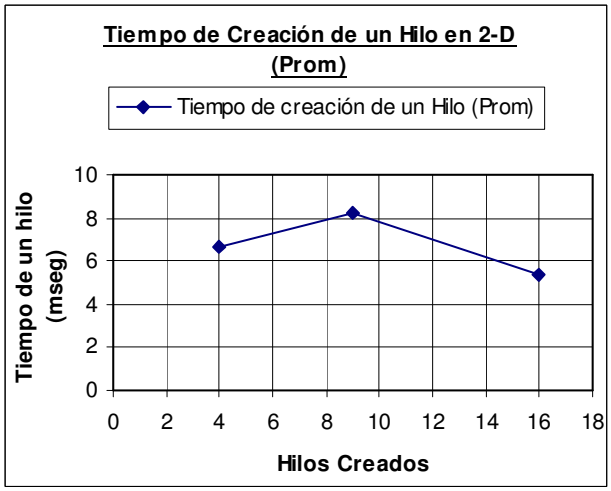

Figura 12.d. Tiempo de Creación de un solo Hilo (Prom) en 2-D.

6.3.4. Aceleración de los algoritmos concurrentes respecto al algoritmo secuencial

Debido a que el costo asintótico de ambos algoritmos, 1-D y 2-D, es el mismo, la aceleración obtenida en cada caso también es prácticamente la misma. Esto podría cambiar si se tuviese en cuenta lo expuesto en la sección 6.3.1.

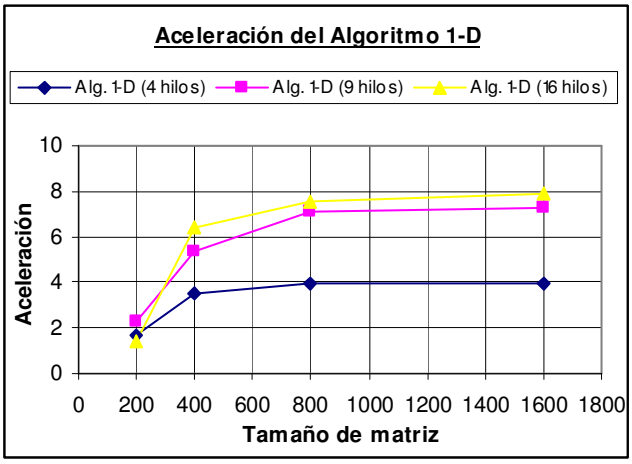

Figura 13.a. Aceleración del algoritmo 1-D.

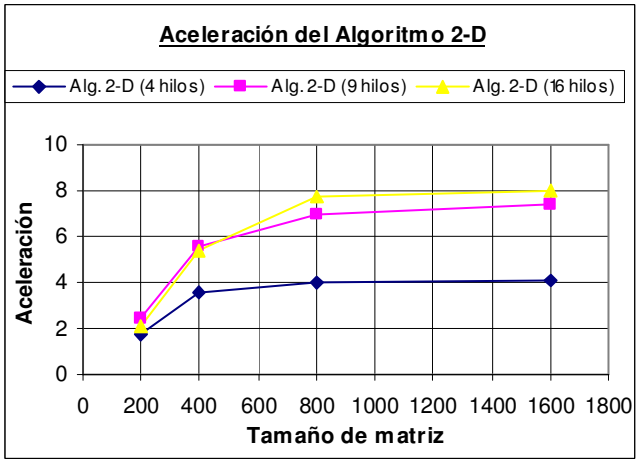

Figura 13.b. Aceleración del algoritmo 2-D.

7 Conclusiones

Luego de analizar los distintos resultados obtenidos, la principal conclusión obtenida es que el nivel de concurrencia está fuertemente ligado a que el hardware pueda proveer la concurrencia esperada, ya que si no se provee el soporte necesario, se crearían hilos de ejecución que finalmente se ejecutarían de forma secuencial. Además, con la creación de cada hilo se genera un sobrecosto, que no son significativos en los resultados porque el número de hilos es pequeño con relación al tamaño de las matrices.
Otro aspecto importante son las aceleraciones de los algoritmos concurrentes, al mostrar como con el aumento de los hilos aumenta la aceleración aproximadamente en la misma proporción. Esta relación solo se mantiene mientras exista todavía soporte para la paralelización, o sea, el número de hilos menor o igual al número de procesadores. Luego, para mayores cantidades de hilos se sigue mejorando respecto al secuencial pero no en la misma proporción.

8 Trabajos Futuros

En vista que mencionamos que con el aumento de hilos se logra una mejora dependiendo del grado de paralelismo soportado por el hardware, sería un buen experimento encontrar hasta que punto el número de hilos creados sigue sin afectar significativamente al tiempo de ejecución del algoritmo.
Además, se podría plantear un método más eficiente para la asignación de filas y columnas sobrantes, para evitar en mayor medida el desbalanceo de carga, ya que la asignación de tareas es en forma estática, dejándole las columnas o filas restantes a los últimos hilos.
Otro punto a considerar sería la selección más adecuada de tamaños de matrices de manera a que pueda ser evaluado el comportamiento de los algoritmos en función al particionamiento que realizan.
Finalmente, algo muy complejo de solucionar sería el problema de la memoria disponible para la carga de las matrices en el espacio de direccionamiento del programa, ya que para matrices muy grandes el rendimiento podría verse muy afectado por el paginamiento de porciones de las matrices.

Referencias

[1] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Introduction to Parallel Computing, Segunda Edición. Enero 2003.