Optimización de Algoritmos Secuenciales

Informe entrega 1

Sistemas Distribuidos y Paralelos | Ingeniería en Computación

01515/4 Arreche, Cristian

01528/0 Borini, Ángel

03 de abril de 2020

Facultad de Informática

Universidad Nacional de La Plata

Descripción de los procesadores utilizados

Obtenida a partir de visualizar el archivo virtual /proc/cpuinfo como:

```
cat /proc/cpuinfo
```

Cristian:

processor : 0
vendor_id : AuthenticAMD
cpu family : 16
model : 4
model name : AMD Phenom(tm) II X4 945 Processor
stepping : 3
microcode : 0x10000c8
cpu MHz : 3000.000
cache size : 512 KB
[...]
cpu cores : 4
[...]

Ángel:

processor : 0
vendor_id : AuthenticAMD
cpu family : 21
model : 16
model name : AMD A6-4400M APU with Radeon(tm) HD Graphics
stepping : 1
microcode : 0x6001119
cpu MHz : 1853.994
cache size : 1024 KB
[...]
cpu cores : 1
[...]

01_A. Matrices Cuadradas:

*Analizar el algoritmo matrices.c que resuelve la multiplicación de matrices cuadradas de N*N, ¿dónde cree se producen demoras? ¿cómo se podría optimizar el código? Optimizar el código y comparar los tiempos probando con diferentes tamaños de matrices.*

(Procesador utilizado para la resolución: Cristian)

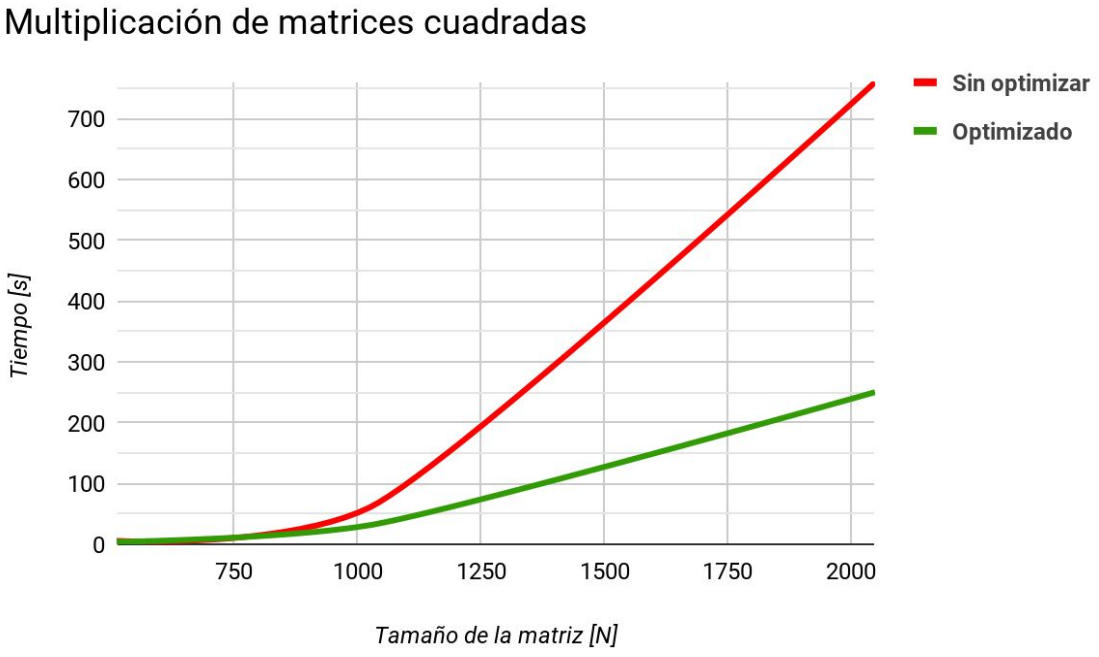
Cuando se trae un dato a memoria caché, también se traen los datos cercanos a este por el principio de localidad espacial. En este algoritmo, podemos ver que las demoras son producidas por los fallos de caché que se generan al ordenar en memoria todas las matrices por filas.

Debido a la naturaleza del producto matricial es comprobable que ordenar por columnas la matriz de la derecha en una multiplicación de matrices permite tener menos fallos de caché y así alcanzar mejoras de velocidad.

Siendo A, B y C matrices cuadradas de N*N, si C = A*B:

- C y A ordenadas en memoria por filas
- B ordenada en memoria por columnas

Esta ordenación permite aprovechar al máximo los principios de localidad.



Como podemos observar, la diferencia en tiempos de ejecución entre la versión original y la optimizada, comienza a incrementarse notablemente a medida que aumenta el tamaño N de las matrices.

01_E. Matrices Triangulares:

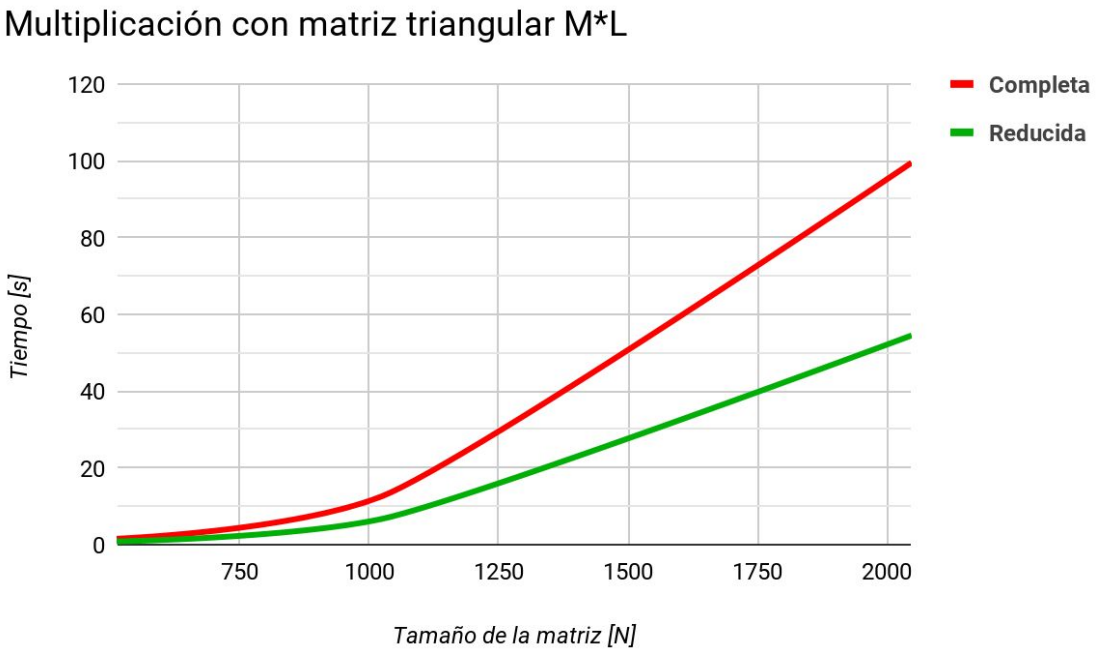
*Implementar las soluciones para la multiplicación de matrices M*L y L*M. Donde M es una matriz de N*N y L es una matriz triangular inferior.*

Analizar los tiempos de ejecución para la solución que almacena los ceros de las triangulares respecto a la que no los almacena.

(Procesador utilizado para la resolución: Cristian)

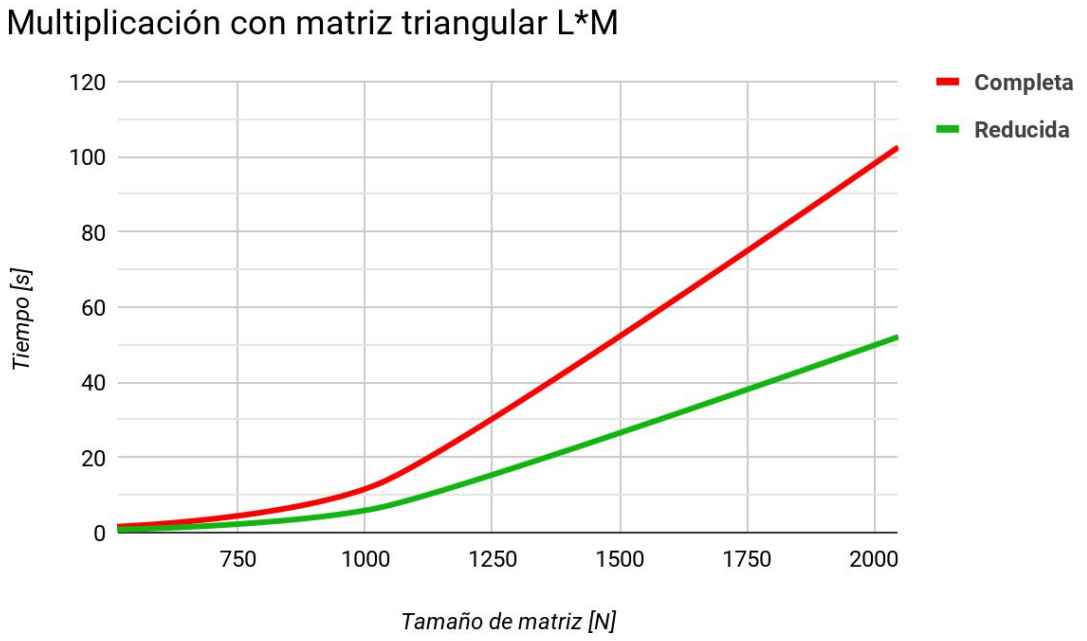
Una matriz triangular es una matriz cuadrada, donde los elementos por debajo o por encima de la diagonal, son cero. El objetivo es minimizar el espacio de almacenamiento , evitando guardar los elementos de la parte nula de la matriz (siempre y cuando no sean necesarios, por ejemplo como en una multiplicación de matrices).

A su vez, para este ejercicio, se mantuvo la optimización lograda en el punto anterior donde verificamos que se alcanzan mejoras de velocidad cuando la matriz de la derecha es ordenada en memoria por columnas.



Siendo M una matriz cuadrada de N*N ordenada en memoria por fila y L una matriz triangular inferior ordenada por columnas, podemos observar que el tiempo de ejecución de la multiplicación con la matriz triangular completa, es decir, almacenando los ceros de la misma, es mucho mayor.

Esto es debido a que el consumo de memoria de las matrices completas es de aproximadamente el doble que las almacenadas de manera reducida, en consecuencia, el tiempo de procesamiento aumenta considerablemente ya que en la versión reducida solo recorremos y procesamos las posiciones de la parte no nula de la matriz.



Cambiando el orden de los operandos en la multiplicación, y ahora ordenando la matriz triangular L en memoria por filas y la matriz M por columnas por ser la matriz de la derecha, no encontramos diferencias significativas en los tiempos de procesamiento, ya que la cantidad de cálculos a realizar es prácticamente la misma.

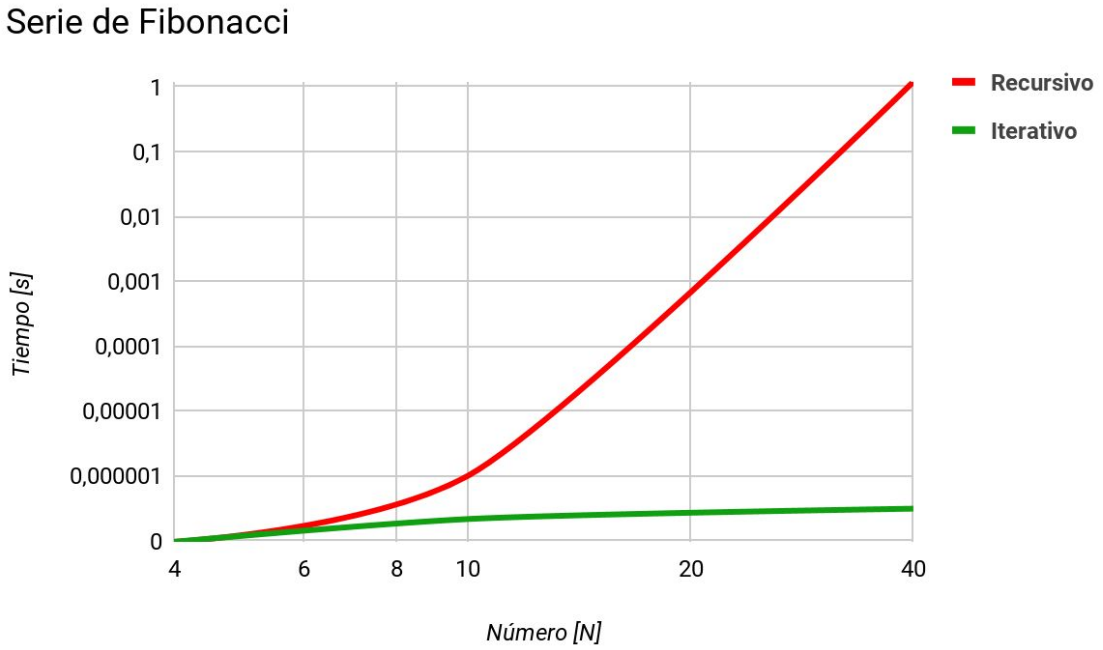
02. Serie de Fibonacci:

El algoritmo fib.c resuelve la serie de Fibonacci para un número N dado utilizando dos métodos, el método recursivo y el método iterativo. Analizar los tiempos de ambos métodos ¿Cuál es más rápido? ¿Por qué?

(Procesador utilizado para la resolución: Ángel)

Generalmente, un algoritmo iterativo trabaja continuamente sobre un mismo stack frame; este hecho, en cambio, no se da en una versión recursiva ya que en cada llamado a una nueva subrutina es necesario reservar espacio de almacenamiento adicional para las variables futuras, lo cual claramente conlleva un mayor consumo de memoria y de tiempo.

Más precisamente, en el algoritmo recursivo para resolver la serie de Fibonacci, cada vez que se ejecuta la función, realizará dos nuevas llamadas a sí misma, luego cada una de estas hará a la vez dos llamadas más y así sucesivamente hasta que terminen en 0 o en 1. Este proceso se denomina "recursión de árbol" y sus requisitos de tiempo crecen de forma exponencial



Como mencionamos, se observa claramente que a medida que aumentamos el número N a calcular, el tiempo de ejecución de la versión recursiva es considerablemente mayor que el de la versión iterativa (más de 10^6 veces mayor).

En cuanto a localidad de caché, en las versiones iterativas se logra optimizar dicho principio, ya que como anteriormente mencionamos opera siempre sobre las mismas variables, en cambio, para la versión recursiva en cada llamado a una nueva subrutina provocará nuevos fallos de caché.