Optimización de Algoritmos Secuenciales

Informe reentrega 1

Sistemas Distribuidos y Paralelos | Ingeniería en Computación

01515/4 Arreche, Cristian

01528/0 Borini, Ángel

06 de mayo 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
[...]

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, aprovechando al máximo los principios de localidad.

A su vez, en detectamos un importante overhead introducido por los reiterados llamados a funciones, debido a lo que hace la CPU al llamar a una función (respaldar registros, almacenamiento en stack, etc.), y en este caso, como como se llama N*N veces por cada matriz, se reduce notablemente la performance.

Por último, otra mejora realizada fue en el for más interno de la multiplicación, donde acumulamos el valor de cada operación A*B en una variable auxiliar interna con el modificador ‘register’, con el fin de indicarle al compilador una preferencia para que almacena la variable en los registros del procesador, si es posible, con el fin de optimizar su acceso. Y luego, fuera del for más interno, asignarlo a la matriz resultado C.

Los tiempos obtenidos para la versión original fueron:

TAMAÑO N		
N = 512	N = 1024	N = 2048
5.481538	60.370201	759.746711

Tiempo de ejecución versión no optimizada.

Mientras, que para la versión optimizada fueron:

TAMAÑO N		
N = 512	N = 1024	N = 2048
1.248065	10.147870	81.657253

Tiempo de ejecución versión optimizada.

Como podemos observar, existe una gran diferencia en tiempos de ejecución entre la versión original y la optimizada, obteniéndose mejoras más que positivas en rendimiento para esta última.

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).

M*L reducida

TAMAÑO N		
N = 512	N = 1024	N = 2048
0.811876	6.533626	53.310245

L reducida (sin almacenar los ceros)

M*L completa

TAMAÑO N		
N = 512	N = 1024	N = 2048
0.771941	6.219205	50.783960

L completa (almacenando los ceros)

La cantidad de multiplicaciones que se realizan es siempre la misma, lo que diferencia una versiòn de otra es que al no almacenar los ceros (matriz reducida), se gana en memoria almacenada y en accesos a cache, pero se pierde en tiempo de ejecución al tener que calcular el índice de recorrido de una manera bastante más compleja realizando una cantidad de operaciones mucho mayor

Podemos observar que si bien el tiempo de ejecución de la multiplicación M*L con la matriz triangular completa, es decir, almacenando los ceros de la misma, es menor no existe una gran diferencia temporal comparado con la multiplicación M*L siendo L reducida.

Pero creemos necesario tener en cuenta que a mayor N, dicha brecha temporal aumentará; otro factor a tener en cuenta es la mejora en almacenamiento en la versión reducida, la cual también será notoria para N mucho mayores. Por lo cual se puede concluir

que dependiendo cuales sean los requerimientos a optimizar, almacenamiento o tiempo de ejecución, se puede optar por una u otra solución.

De la misma manera, alternando el orden de las matrices, obtenemos:

L*M reducida

TAMAÑO N		
N = 512	N = 1024	N = 2048
0.759033	6.286901	50.348026

L reducida (sin almacenar los ceros)

L*M completa

TAMAÑO N		
N = 512	N = 1024	N = 2048
0.728134	5.990387	48.196133

L completa (almacenando los ceros)

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. Las conclusiones obtenidas son las mismas que se mencionaron en el caso anterior.