Universidad Nacional de La Plata
Facultad de Informática
Materia: Sistemas Paralelos

Práctica Nro. 1

Alumnos: Villalba Joaquín – Vizcaino Aldo

Contents

1 Algoritmo multBloques

El algortimo multBloques.c lo que hace es multiplicar 2 matrices (A y B) y guardar el resultado en una matriz C. A diferencia del ejercicio 1, utiliza multiplicación por bloques, lo que nos provee un aprovechamiento del uso de la cache, ya que usamos la matriz en tamaño de bloques, esos bloques que se necesitan para la operacion permanecen en cache hasta que se terminen de usar, liberandolos y pasando otros nuevos bloques a cache. De esta manera minimizamos la accesos a memoria principal y en consecuencia logrando una mejor eficiencia en la multiplicacion de matrices.
A diferencia del ejericio 1 sin modificar, en la mayoría de los casos los accesos a memoria son muchos menores, por lo que logra mejores tiempos en la matoría de los tamaños probados.
Si comparamos los tiempos con el ejercicio 1 modificado, no ha podido superar a éste. Ya que en el ejercicio 1 modificado se ha aprovechado los datos que se recorren en la matriz, que son almacenados en la cache y usados constantemente hasta terminar de utilizar la fila. Logrando hits de cache en múltiples ocasiones mejorando mucho el rendimiento del algoritmo. De todas maneras en el cuadro se pueden ver tiempos comparativos.
Los mejores tiempos de multBloques se diron cuando la cantidad y el tamaño de bloques era de manera proporcionada. Esto se puede deber al tamaño y scheduling de la cache, la manera en que trabaja logra mejores resultados con por ejemplo 32 bloques de tamaño 32, calcula una matriz de tamaño 1024 de manera más eficiente que si fuera:

```
Caso 1:
      1 bloque de 1024 : Tiempo en segundos: 46.941289
Caso 2:
      1024 bloques de 1: Tiempo en segundos: 117.409659
Caso 3:
      32 bloques de 32 : Tiempo en segundos: 7.829465
```

El primer caso no debe poder llevar todos los datos a cache por lo cuál tiene trabajo extra en buscar datos en memoria principal.
El segundo caso el costo de crear tantos bloques lo hacen imposible de operar.
El tercer caso resultó ser el mejor. Analogamente pasa lo mismo con matrices de tamaño 256 o 512, con los resultados descriptos a continuación.

Cabe recalcar que va a depender de la arquitectura en la cuál ejecutemos el algoritmo, ya que en otras arquitecturas se pueden lograr mejores tiempos de otra manera en cuanto a tamaño y cantidad de bloques.

Comentario adicional: En el ejercicio 1, luego vimos que si cambiabamos los indices en la parte donde se multiplica, dentro del for. Se podía lograr que se vaya recorriendo la matriz por columnas y una misma fila e ir obteniendo resultados parciales en la matriz C, haciendo la suma total a medida que se pasaba de fila (el for de más afuera). No logramos hacerlo andar por falta de tiempo ya que lo vimos muy al final del día. Pero esto nos hacía suponer funcionaba de tal manera que el algoritmo atraviesa la matriz B a lo largo de las columnas, beneficiandose así de la localidad espacial. Y también que atraviesa la matriz C varias veces teniendo beneficio así de la localidad temporal de la caché de 2do nivel probablemente. Es decir que no calcula de a un cuadro a la vez C, sino que va de a trozos. Mejoró en tiempo pero

nos daba error en el resultado. Por lo tanto no fue el definitivo, solo queríamos comentarlo.

2 Resultados:

N	Tiempo ej1	Tiempo ejOpt	Tiempo ej3	bloque 32x32	bloque 64x64	bloque 128x128
256	0.353138	0.097083		0.127543	0.151542	0.153500
512	3.221342	0.712713		0.973493	1.152079	1.164182
1024	56.334456	5.721960		7.810656	9.034560	9.464947

N	bloque 32x32	bloque 64x64	bloque 128x128
256	./multBloques 8 32 0	./multBloques 4 64 0	./multBloques 2 128 0
512	./multBloques 16 32 0	./multBloques 8 64 0	./multBloques 4 128 0
1024	./multBloques 32 32 0	./multBloques 16 64 0	./multBloques 8 128 0

3 Hardware:

Como dijimos anteriormente, estos resultados dependen de la arquitectura de la computadora en la cuál se estén evaluandolos. Estos datos fueron obtenidos en una máquina fisica con:

propiedad	valor
Processor:	AMD A4-5300 APU with Radeon(tm)
Architecture:	x86_64
Byte Order:	Little Endian
CPU(s):	2
On-line CPU(s) list:	0,1
Thread(s) per core:	2
Core(s) per socket:	1
Vendor ID:	AuthenticAMD
CPU MHz:	1400.000
L1d cache:	16K
L1i cache:	64K
L2 cache:	1024K