Trabajo Algoritos Paralelos: Memoria Cache

Jesus Erick Vera Callme •jesus.vera@ucsp.edu.pe

1 Introduccion

En este trabajo se analizaran diversos algoritmos (bucles anidados) todos para hacer la multiplicacion de matrices. Se usó la herramienta "valgrind" y "cachegrind" para analizar el uso de la memoria cache.

```
for(int i=0; i < MAX; i++){
    for( int j=0; j < MAX; j++){
        y[i] += A[i][j] * x[j];
    }
}

for(int j=0; j < MAX; j++){
    for( int i=0; i < MAX; i++){
        y[i] += A[i][j] * x[j];
    }
}</pre>
```

Figure 1: Bucles 1 y 2

En la Figura 1. los bucles 1 y 2 nos muestran una distinto tipo de implementacion de una multiplicacion. En los 2 primeros 'for' podemos observar que la multiplicacion va a ser un uso optimo de la memoria cache en cuanto a que se aprovecha de la arquitectura para poder primero traer a memoria cache los elementos de la matriz que estan contiguos y asi solo ir saltando a los vecinos contiguos.

En la Figura 1 tambien para el segundo par de 'for', no hace un uso optimo de la memoria cache, y por tanto va a tener mas 'cache misses' que el primer bucle, esto debido a que el computador no ha alojado en la memoria cache los vecinos contiguos y el mismo algoritmo lo que sugiere es que no opere con vecinos sino con los elementos de abajo, los cuales no estan en la memoria rapida, sino que tiene que buscarlos en memoria para acceder al dato.

2 Tiempos entre primer bucle y segundo bucle

Tamano	Primer Bucle	Segundo Bucle
10	0.002	0.002
50	0.024	0.017
100	0.092	0.074
1000	5.848	8.919
10000	434.341	1462.012

Figure 2: Comparacion entre bucles

En la Figura 2. podemos identificar la comparacion de tiempo entre el Primer bucle y el Segundo Bucle. En los primeros tamaños (de 10 y 50) la diferencia no es notable debido a que el acceso se ase rapido. En cuanto el tamaño de dato crece (100, 1000, 10000) se ve una gran diferencia en cuanto a tiempo, el primer bucle al optimizar el acceso y aprovechar la memoria cache obtiene un tiempo menor al segundo bucle por casi 1/3 menos de tiempo.

3 Uso de cache en cuanto a primer bucle y segundo bucle

En la Figura 3. cuando ejecutamos el codigo del primer bucle con Valgrind y Cachegrind, podemos observar una cierta ventaja en cuanto al segundo bucle (Figura 4). El primer bucle en la instruccion (D1) o los 'cache misses de datos en el primer nivel' son superiormente mas bajos que el segundo bucle.

```
==8313==
==8313== I
             refs:
                         525,192,114
                               1,041
1,032
==8313== I1 misses:
==8313== LLi misses:
==8313== I1 miss rate:
                                0.00%
==8313== LLi miss rate:
                                0.00%
==8313==
==8313== D
             refs:
                         275,071,663
                                       (250,054,777 rd
                                                            25,016,886 wr)
                           6,258,719
3,129,106
==8313== D1
                                         6,258,128 rd
                                                                    591 Wr)
             misses:
                                                                    554 wr)
==8313== LLd misses:
                                          3,128,552 rd
==8313== D1 miss rate:
                                 2.3% (
                                                2.5%
                                                                   0.0%
==8313== LLd miss rate:
                                  1.1% (
                                                 1.3%
                                                                   0.0%
==8313==
==8313== LL refs:
                           6,259,760
                                          6,259,169 rd
                                                                    591 wr)
                                          3,129,584 rd
                                                                    554 Wr)
==8313== LL misses:
                           3,130,138
==8313== LL miss rate:
                                                0.4%
                                 0.4% (
                                                                    0.0%
chucho@ubuntu:~/Documents/Algoritmos Paralelos/Capitulo 2$
```

Figure 3: Valgrind con primer bucle

```
==8336==
==8336== I
                         525,192,128
             refs:
==8336== I1
            misses:
                               1,041
                               1,032
==8336== LLi misses:
==8336== I1 miss rate:
                                0.00%
==8336== LLi miss rate:
                                0.00%
==8336==
==8336== D
                                      (250,054,780 rd
             refs:
                         275,071,669
                                                         + 25,016,889 wr)
==8336== D1 misses:
                          28,133,715
                                      ( 28,133,124 rd
                                                                  591 Wr)
==8336== LLd misses:
                          3,134,106
                                         3,133,552 rd
                                                                  554 wr)
==8336== D1 miss rate:
                                10.2%
                                              11.3%
                                                                  0.0%
==8336== LLd miss rate:
                                 1.1% (
                                                1.3%
                                                                  0.0%
==8336==
                          28,134,756
==8336== LL refs:
                                      ( 28,134,165 rd
                                                                   591 wr)
                                 138 (
0.4% (
==8336== LL misses:
                          3,135,138
                                         3,134,584 rd
                                                                   554 wr)
==8336== LL miss rate:
                                               0.4%
                                                                   0.0%
chucho@ubuntu:~/Documents/Algoritmos Paralelos/Capitulo 2$
```

Figure 4: Valgrind con segundo bucle

4 Comparacion de los Algoritms "nested-loops" para multiplicacion de matrices

Tamano	3 nested-loop	6 nested-loop
10	0.000012	0.000022
50	0.001278	0.001144
100	0.007503	0.008179
500	0.928656	0.803104
1000	21.356322	6.683981

Figure 5: Comparacion de Tiempo entre 3 nested-loops y 6 nested-loops

Ahora se procede al analisis en cuanto a tiempo de los algoritmos 3 nested-loops y 6 nested-loops. En la Figura 5 se hace la comparacion de tiempos de los algoritmos. En los primeros tamaños (10 y 50) no se observa la diferencia, pero a medida que el tamaño crece el algoritmo de 6 nested-loops se da en menor tiempo que el 3 nested-loops. Esa diferencia debido a la construccion del algoritmo 6 nested-loops, debido a que no opera en un bloque gigante, sino se va a subdividir en bloques mas pequeños para asi subdividir el trabajo total.

5 Comparacion a nivel Cache de los Algoritms "nested-loops" para multiplicacion de matrices

En la Figura 6, se observa que con la herramienta Valgrind y Cachegrind se procedio al analisis total del algoritmo 3 nested-loop. Se tomo en cuenta tanto instrucciones leidas (I refs), cache misses en 1er (I1)y ultimo nivel(L1), lectura de datos, rates de misses y misses en el ultimo nivel, pero se tuvo consideracion en D1 (cache misses de datos en primer nivel), los cuales son superiores a comparacion del algoritmo 6 nested-loops.

En la Figura 7, del analisis del algoritmo 6 nested-loops, se observo que los 'cache misses de datos en primer nivel' son menores que en 3 nested-loops.

Simbologia	3 nested: 10	3 nested: 50	3 nested: 100	3 nested: 500
I refs	223,492	6,578,313	50,765,516	6,263,983,810
11	1,162	1,155	1,150	1,163
LLi	1,141	1,135	1,129	1,149
Il rate	52%	0.02%	0.00%	0.00%
LLi rate	0.51%	0.02%	0.00%	0.00%
D refs	81,482	3,014,512	23,375,931	2,882,668,904
D1	3,085	3,890	72,001	136,342,406
LLd	2,494	2,973	4,452	50,293
D1 misses	3.80%	0.10%	0.30%	4.70%
LLd misses	3.10%	0.10%	0.00%	0.00%
LL refs	4,247	5,045	73,151	136,343,569
LL misses	3,635	4,108	5,581	51,442
LL miss rate	1.20%	0.00%	0.00%	0.00%
	I refs I1 LLi I1 rate LLi rate D refs D1 LLd D1 misses LLd misses LL refs LL misses	I refs 223,492 II 1,162 LLi 1,141 II rate 52% LLi rate 0.51% D refs 81,482 DI 3,085 LLd 2,494 DI misses 3.80% LLd misses 3.10% LL refs 4,247 LL misses 3,635	I refs 223,492 6,578,313 II 1,162 1,155 LLi 1,141 1,135 II rate 52% 0.02% LLi rate 0.51% 0.02% D refs 81,482 3,014,512 DI 3,085 3,890 LLd 2,494 2,973 DI misses 3.80% 0.10% LLd misses 3.10% 0.10% LL refs 4,247 5,045 LL misses 3,635 4,108	I refs 223,492 6,578,313 50,765,516 II 1,162 1,155 1,150 LLi 1,141 1,135 1,129 II rate 52% 0.02% 0.00% LLi rate 0.51% 0.02% 0.00% D refs 81,482 3,014,512 23,375,931 D1 3,085 3,890 72,001 LLd 2,494 2,973 4,452 D1 misses 3.80% 0.10% 0.30% LLd misses 3.10% 0.10% 0.00% LL refs 4,247 5,045 73,151 LL misses 3,635 4,108 5,581

Figure 6: Comparacion a nivel cache 3 nested-loops

Simbologia	6 nested: 10	6 nested: 50	6 nested: 100	6 nested: 500
I refs	229,728	7,339,923	56,726,917	6,953,456,244
11	1,163	1,157	1,151	1,164
LLi	1,143	1,138	1,131	1,150
Il rate	0.51%	0.02%	0.00%	0.00%
LLi rate	0.50%	0.02%	0.00%	0.00%
D refs	84,054	3,337,016	25,894,292	3,167,868,631
D1	3,086	3,916	10,405	796,125
LLd	2,494	2,973	4,452	50,316
D1 misses	3.70%	0.10%	0.00%	0.00%
LLd misses	3.00%	0.10%	0.00%	0.00%
LL refs	4,249	5,073	11,556	797,289
LL misses	3,637	4,111	5,583	51,466
LL miss rate	1.20%	0.00%	0.00%	0.00%
	I refs I1 LLi I1 rate LLi rate D refs D1 LLd D1 misses LLd misses LL refs LL misses	I refs 229,728 II 1,163 LLi 1,143 II rate 0.51% LLi rate 0.50% D refs 84,054 DI 3,086 LLd 2,494 DI misses 3.70% LLd misses 3.00% LL refs 4,249 LL misses 3,637	I refs 229,728 7,339,923 II 1,163 1,157 LLi 1,143 1,138 II rate 0.51% 0.02% LLi rate 0.50% 0.02% D refs 84,054 3,337,016 DI 3,086 3,916 LLd 2,494 2,973 DI misses 3.70% 0.10% LLd misses 3.00% 0.10% LL refs 4,249 5,073 LL misses 3,637 4,111	I refs 229,728 7,339,923 56,726,917 II 1,163 1,157 1,151 LLi 1,143 1,138 1,131 II rate 0.51% 0.02% 0.00% LLi rate 0.50% 0.02% 0.00% D refs 84,054 3,337,016 25,894,292 D1 3,086 3,916 10,405 LLd 2,494 2,973 4,452 D1 misses 3.70% 0.10% 0.00% LLd misses 3.00% 0.10% 0.00% LL refs 4,249 5,073 11,556 LL misses 3,637 4,111 5,583

Figure 7: Comparacion a nivel cache 6 nested-loops)

6 Conclusiones

Al analizar los algoritmos "nested-loops" en cuanto a tiempo y accesos a memoria cache, tenemos las siguientes conclusiones:

- Cuando se analizo el algoritmo 3 nested-loops $(3n^3)$ es mas lento que el 6 nested-loops cuantos mayores son los tamaños (10, 50, 100...).
- El algoritmo 6 nested-loops resulto ser mejor en cuanto a tiempo, debido a que la multiplicacion de matrices no se hace comun y ordenamente, el algoritmo 6 nested-loops mejora en cuanto a que las operaciones buscara hacerlas a nivel de subloques dentro del bloque de tamano total (ejemplo si el bloque es de 1000 los subbloques pueden ser de tamano 10) por lo que las operaciones son mas rapidas.
- El tamaño de los subbloques en el algoritmo 6 nested-loops es contradictorio debido a que si el subbloque es muy pequeño es lo mismo que hacer operaciones con 3 nested-loops, pero si es muy grande tambien se asimilara a 3 nested-loops debido a que solo operara en un bloque entero y los "for" que operan seran usados en vano.
- El algoritmo 6 nested-loops buscara dividir el gran bloque para asi poder ponerlos en la memoria cache para que de esa manera eviten hacer muchas lecturas a memoria principal.
- Para probar los algoritmos se uso valgrind y cachegrind como una forma de medir el uso de memoria
 cache en sus distintos niveles, el algoritmo con 6 loops presenta un menor rate en cuanto a cache misses,
 con lo que podemos decir que se accede muy pocas veces a memoria princioal, por lo anterior tambien
 podemos agregar que se mejora la localidad temporal(dato usado muchas veces) y espacial(conjunto de
 datos que se asumen que seran usados al estar contiguos al dato con localidad temporal).