REPORTE 1: Multiplicación de MAtrices

Daniela F. Milón Flores,Andre Ramos (*Autora)*

Universidad Católica San Pablo

	BlockSize	9	99	300	500	590
	MultMatriz	0.06	4.205.000	119.793.000	600.620.000	1.154.150.000
MultBlocked	3	0.06	7.005.000			
	9	0.06	5.256.000			
	3		5.660.000			
	33		7.947.000			
	99		4.557.000			
	30			115.506.000		
	100			97.783.000		
	300			91.527.000		
	50				486.786.000	
	100				454.370.000	
	500				419.279.000	
	59					762.954.000
	590					1.154.150.000

Se puede observar en el tablero que a medida que se incrementa la dimensión de las matrices el Algoritmo Blocked comienza a cobrar relevancia pues sus tiempos más cortos que los del algoritmo de 3 Bucles Anidados, que solo actúa de forma óptima con dimensiones más pequeñas.

Conclusiones de medida de Tiempos:

El Algoritmo de Blocked cumple con una mejor performance gracias a que carga en memoria caché un bloque del array original de la matriz. De esta forma a menos elementos en memoria con el mismo nivel de acceso a ellos, el algoritmo logra optimizar el cálculo de la multiplicación.

También es posible notar que la división de los bloques es sumamente relevante. Lográndose comprobar que a una división más pareja mejor resultado se obtiene. Aunque esto no siempre se cumple.

Ejecutando Valgrind

danielafe7@danielafe7:~/Escritorio$ g++ -o test -g 1.cpp
danielafe7@danielafe7:~/Escritorio$ valgrind --tool=cachegrind ./test 10000000
--9029-- warning: L3 cache found, using its data for the LL simulation.
Tu primera Matriz es:
[2] [2] [2]
[2] [2] [2]
[2] [2] [2]
Tu segunda Matriz es:
[3] [3] [3]
[3] [3] [3]
[3] [3] [3]

Multiplicación 3 Bucles Anidados
[18] [18] [18]
[18] [18] [18]
[18] [18] [18]

==9029==
==9029== I refs: 2,203,093
==9029== I1 misses: 1,670
==9029== LLi misses: 1,574
==9029== I1 miss rate: 0.08%
==9029== LLi miss rate: 0.07%
==9029==
==9029== D refs: 741,038 (543,859 rd + 197,179 wr)
==9029== D1 misses: 15,852 (13,663 rd + 2,189 wr)
==9029== LLd misses: 9,162 (7,776 rd + 1,386 wr)
==9029== D1 miss rate: 2.1% (2.5% + 1.1%)
==9029== LLd miss rate: 1.2% (1.4% + 0.7%)
==9029==
==9029== LL refs: 17,522 (15,333 rd + 2,189 wr)
==9029== LL misses: 10,736 (9,350 rd + 1,386 wr)
==9029== LL miss rate: 0.4% (0.3% + 0.7%)

Multiplicación Blocked Version
[18] [18] [18]
[18] [18] [18]
[18] [18] [18]

```
==12837==
==12837== I   refs:      2,203,015
==12837== I1  misses:        1,669
==12837== LLi misses:        1,573
==12837== I1  miss rate:      0.08%
==12837== LLi miss rate:      0.07%
==12837==
==12837== D   refs:        741,130  (543,951 rd   + 197,179 wr)
==12837== D1  misses:       15,847  ( 13,659 rd   +   2,188 wr)
==12837== LLd misses:        9,158  (  7,773 rd   +   1,385 wr)
==12837== D1  miss rate:       2.1% (    2.5%     +     1.1%  )
==12837== LLd miss rate:       1.2% (    1.4%     +     0.7%  )
==12837==
==12837== LL refs:          17,516  ( 15,328 rd   +   2,188 wr)
==12837== LL misses:        10,731  (  9,346 rd   +   1,385 wr)
==12837== LL miss rate:        0.4% (    0.3%     +     0.7%  )
```

Conclusiones al Ejecutar Vangrid

- Los resultados son mucho más bajos que los del Algoritmo de 3 Bucles anidados
- A pesar de que se fragmenta el array, parece no haber aumento de cache misses