Matrices: Desempeño en el caché

Jose David Mamani Vilca

I. Valgrind: Cachegrind

Comando: valgrind - -tool = cachegrind [Nombre del Archivo]
Ejemplos:

```
1000
==8989==
==8989== I   refs:      76,349,251
==8989== I1  misses:         1,739
==8989== LLi misses:         1,668
==8989== I1  miss rate:       0.00%
==8989== LLi miss rate:       0.00%
==8989==
==8989== D   refs:      56,390,753  (33,906,924 rd   + 22,483,829 wr)
==8989== D1  misses:       269,590  (   141,424 rd   +    128,166 wr)
==8989== LLd misses:       261,767  (   134,578 rd   +    127,189 wr)
==8989== D1  miss rate:        0.5% (       0.4%     +        0.6%  )
==8989== LLd miss rate:        0.5% (       0.4%     +        0.6%  )
==8989==
==8989== LL refs:          271,329  (   143,163 rd   +    128,166 wr)
==8989== LL misses:        263,435  (   136,246 rd   +    127,189 wr)
==8989== LL miss rate:         0.2% (       0.1%     +        0.6%  )
```

Figura 1. Ejemplo 1

```
1000
==9003==
==9003== I   refs:      76,349,264
==9003== I1  misses:         1,739
==9003== LLi misses:         1,668
==9003== I1  miss rate:       0.00%
==9003== LLi miss rate:       0.00%
==9003==
==9003== D   refs:      56,390,755  (33,906,925 rd   + 22,483,830 wr)
==9003== D1  misses:     1,519,087  ( 1,390,920 rd   +    128,167 wr)
==9003== LLd misses:       250,818  (   123,628 rd   +    127,190 wr)
==9003== D1  miss rate:        2.7% (       4.1%     +        0.6%  )
==9003== LLd miss rate:        0.4% (       0.4%     +        0.6%  )
==9003==
==9003== LL refs:        1,520,826  ( 1,392,659 rd   +    128,167 wr)
==9003== LL misses:        252,486  (   125,296 rd   +    127,190 wr)
==9003== LL miss rate:         0.2% (       0.1%     +        0.6%  )
```

Figura 2. Ejemplo 2

II. Propuesta 1: Lectura de matrices

Leer una matriz intercalando el orden de recorrido. Verificar su desempeño en la caché.

El primer algoritmo (A-1) figura un recorrido con los iteradores dispuestos normalmente. El segundo algoritmo (A-2) presenta los iteradores invertidos con el fin de ver su desarrollo en caché.

Porcentaje de fallas (%)		
Cant de datos	Algoritmo 1	Algoritmo 2
100	1.6	1.6
200	1.1	1.1
300	0.7	0.7
400	0.6	1.7
500	0.5	2.6
600	0.5	2.7
700	0.5	2.7
800	0.5	2.7
900	0.5	2.7
1000	0.5	2.7

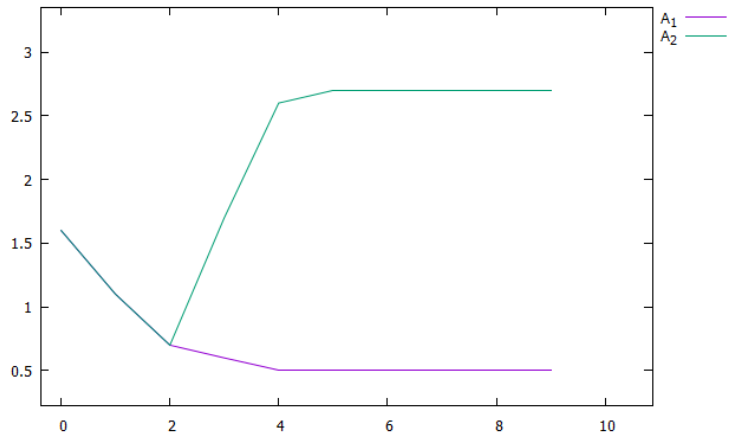

Figura 3. Recorrido normal(A1) y recorrido invertido (A2)

II-A. Gráfica

II-B. Conclusiones

El primer algoritmo resulta más eficiente en cuestiones de no faltas en la Caché. Sus datos se encuentran dispuestos de tal forma que las fallas en caché son pocas. En el caso del segundo algoritmo, los caché misses se elevan en una proporción no tan elevada pero que, sin embargo, demuestran que los iteradores invertidos recorren la estructura de una forma menos eficiente.

III. Prouesta 2: Multiplicación de Matrices

Multiplicar matrices de la forma tradicional y con el método de bucle anidado.

Porcentaje de fallas (%)		
Cant de datos	Normal	Bucle Anidado
10	1.9	1.9
20	1.0	1.1
30	0.4	0.6
40	0.2	0.3
50	0.1	0.2
100	0.1	0.1
150	0.1	0.0
200	0.1	0.0
250	0.1	0.1
300	0.2	0.1
400	0.2	0.1
500	0.4	0.1

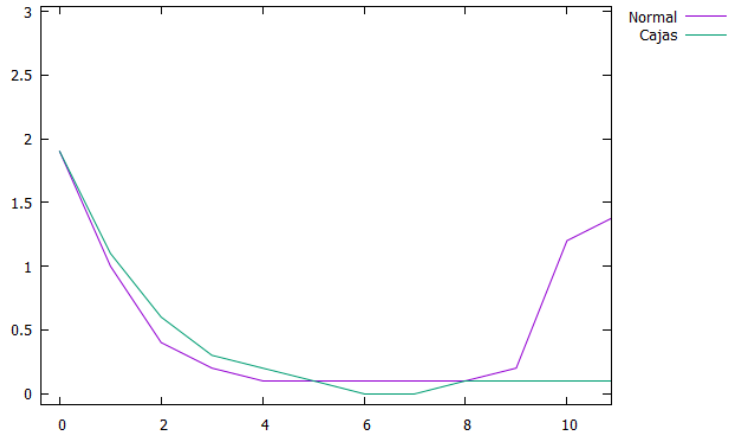

Figura 4. Multiplicacion normal y multiplicación anidada.

III-A. Gráfica

III-B. Conclusiones

El algoritmo basado en la multiplicación común de matrices presentó ciertas ventajas en las pruebas iniciales. Los pocos datos (o poquísimos en este caso) se desenvolvieron mejor y la ejecución en cuestiones de caché misses fueron ventajosas para el primer algoritmo. Sin embargo, conforme el número de datos incrementaba, el desempeño de la multiplicación por bucle anidados resulta ser mucho más ventajosas. La razón, posible, para esta ocurriencia se debe a que en la multiplicación normal nos vemos obligados a recorrer las matrices en su totalidad; en cambio en el método de bucle anidado tomamos pedazos de las matrices y los procesamos de manera individual.