# Cache miss and Cache hit

Julio Quispe Q., julioqq29@gmail.com,

---

## 1. Problema

Analizar los siguientes algoritmos: Simple Matrix Multiplication y Tiled Matrix Multiplication, comparar accesos a memoria cache(cache misses and cache hits), y los tiempos relativos.

## 2. Experimento

Los algoritmos fueron implementados en C buscando controlar el tiempo de los mismos para cada caso de prueba, tengamos en cuenta que estos tiempos son referenciales, ya que dependen mucho de la maquina donde ellos se ejecuten. Se realizo comparaciones de acceso a memoria cache para cada caso de prueba, para esto se utilizó las herramientas Valgrind y Kcachegrind.

## 3. Anlisis y Resultados

Los casos de pruebas fueron hechos para matrices de 500x500, 600x600, 800x800, 1000x1000 y 1500x1500, los resultados obtenidos en la Tabla 1:

**Table 1** **Tiempos Simple Matrix Multiplication(SMM) y Tiled Matrix Multiplication(TMM).**

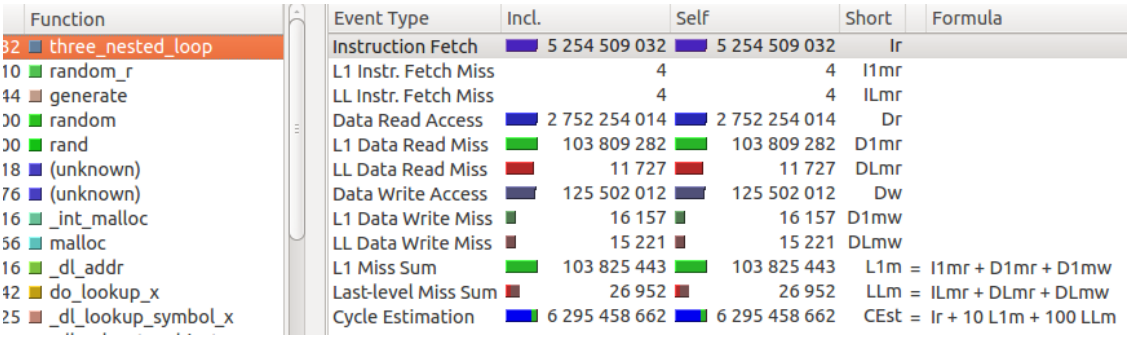
| Alg | 500x500 | 600x600 | 800x800 | 1000x1000 | 1500x1500 |
|---|---|---|---|---|---|
| SMM | 1.59 | 3.44 | 8.69 | 17.61 | 61.42 |
| TMM | 1.42 | 2.43 | 5.80 | 11.35 | 49.22 - 38.20 |

en el algoritmo de TMM se dividieron en 5, 10 y 20 bloques, obteniendo resultados similares para cada matriz, a excepcion de la matriz 1500x1500, que los tiempos disminuyeron

a medida que la cantidad de bloques aumentaba hasta un punto en el que se mantenia constante. En la comparacion de lecturas a cache(cache misses), se obtuvieron los siguientes resultados:

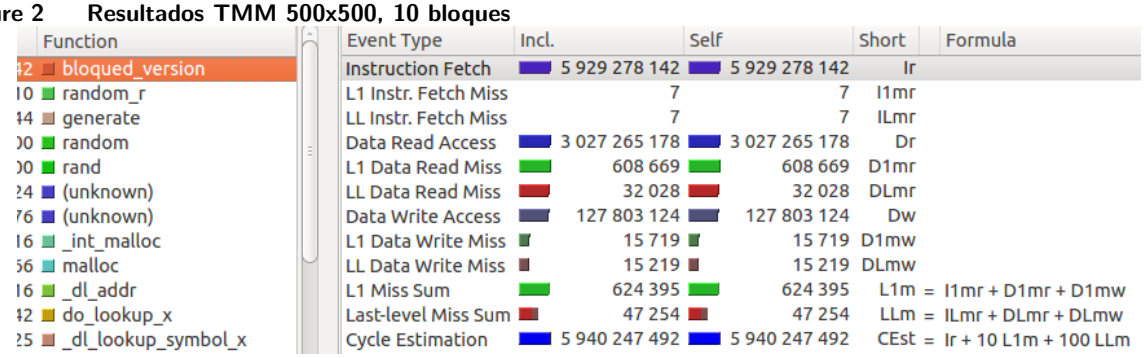
- Para la matriz 500x500, con 10 bloques

**Figure 1** **Resultados SMM 500x500**



| Function | Event Type | Incl. | Self | Short | Formula |
|---|---|---|---|---|---|
| 32 three_nested_loop | Instruction Fetch | 5 254 509 032 | 5 254 509 032 | Ir | |
| 10 random_r | L1 Instr. Fetch Miss | 4 | 4 | I1mr | |
| 44 generate | LL Instr. Fetch Miss | 4 | 4 | ILmr | |
| 00 random | Data Read Access | 2 752 254 014 | 2 752 254 014 | Dr | |
| 00 rand | L1 Data Read Miss | 103 809 282 | 103 809 282 | D1mr | |
| 18 (unknown) | LL Data Read Miss | 11 727 | 11 727 | DLmr | |
| 76 (unknown) | Data Write Access | 125 502 012 | 125 502 012 | Dw | |
| 16 _int_malloc | L1 Data Write Miss | 16 157 | 16 157 | D1mw | |
| 56 malloc | LL Data Write Miss | 15 221 | 15 221 | DLmw | |
| 16 _dl_addr | L1 Miss Sum | 103 825 443 | 103 825 443 | L1m | = I1mr + D1mr + D1mw |
| 42 do_lookup_x | Last-level Miss Sum | 26 952 | 26 952 | LLm | = ILmr + DLmr + DLmw |
| 25 _dl_lookup_symbol_x | Cycle Estimation | 6 295 458 662 | 6 295 458 662 | CEst | = Ir + 10 L1m + 100 LLm |

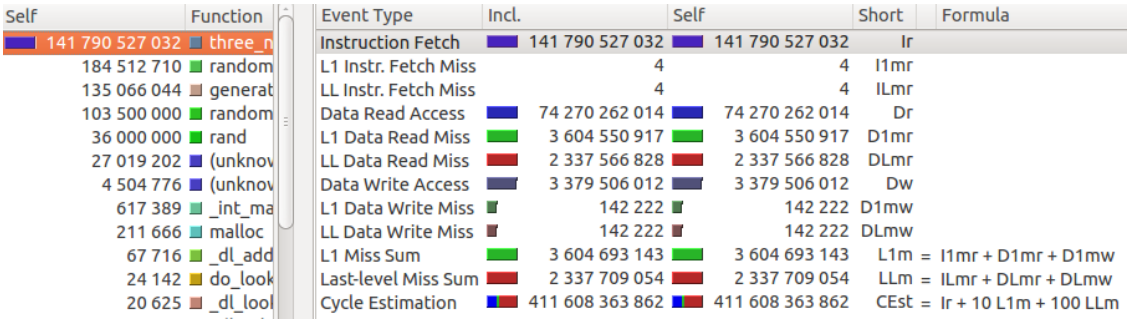
**Figure 2** **Resultados TMM 500x500, 10 bloques**



| Function | Event Type | Incl. | Self | Short | Formula |
|---|---|---|---|---|---|
| 42 bloqued_version | Instruction Fetch | 5 929 278 142 | 5 929 278 142 | Ir | |
| 10 random_r | L1 Instr. Fetch Miss | 7 | 7 | I1mr | |
| 44 generate | LL Instr. Fetch Miss | 7 | 7 | ILmr | |
| 00 random | Data Read Access | 3 027 265 178 | 3 027 265 178 | Dr | |
| 00 rand | L1 Data Read Miss | 608 669 | 608 669 | D1mr | |
| 24 (unknown) | LL Data Read Miss | 32 028 | 32 028 | DLmr | |
| 76 (unknown) | Data Write Access | 127 803 124 | 127 803 124 | Dw | |
| 16 _int_malloc | L1 Data Write Miss | 15 719 | 15 719 | D1mw | |
| 56 malloc | LL Data Write Miss | 15 219 | 15 219 | DLmw | |
| 16 _dl_addr | L1 Miss Sum | 624 395 | 624 395 | L1m | = I1mr + D1mr + D1mw |
| 42 do_lookup_x | Last-level Miss Sum | 47 254 | 47 254 | LLm | = ILmr + DLmr + DLmw |
| 25 _dl_lookup_symbol_x | Cycle Estimation | 5 940 247 492 | 5 940 247 492 | CEst | = Ir + 10 L1m + 100 LLm |

- Para la matriz 1500x1500, con 10 bloques

**Figure 3** **Resultados SMM 1500x1500**



| Self | Function | Event Type | Incl. | Self | Short | Formula |
|---|---|---|---|---|---|---|
| 141 790 527 032 | three_n | Instruction Fetch | 141 790 527 032 | 141 790 527 032 | Ir | |
| 184 512 710 | random | L1 Instr. Fetch Miss | 4 | 4 | I1mr | |
| 135 066 044 | generat | LL Instr. Fetch Miss | 4 | 4 | ILmr | |
| 103 500 000 | random | Data Read Access | 74 270 262 014 | 74 270 262 014 | Dr | |
| 36 000 000 | rand | L1 Data Read Miss | 3 604 550 917 | 3 604 550 917 | D1mr | |
| 27 019 202 | (unknov | LL Data Read Miss | 2 337 566 828 | 2 337 566 828 | DLmr | |
| 4 504 776 | (unknov | Data Write Access | 3 379 506 012 | 3 379 506 012 | Dw | |
| 617 389 | _int_ma | L1 Data Write Miss | 142 222 | 142 222 | D1mw | |
| 211 666 | malloc | LL Data Write Miss | 142 222 | 142 222 | DLmw | |
| 67 716 | _dl_add | L1 Miss Sum | 3 604 693 143 | 3 604 693 143 | L1m | = I1mr + D1mr + D1mw |
| 24 142 | do_look | Last-level Miss Sum | 2 337 709 054 | 2 337 709 054 | LLm | = ILmr + DLmr + DLmw |
| 20 625 | _dl_lool | Cycle Estimation | 411 608 363 862 | 411 608 363 862 | CEst | = Ir + 10 L1m + 100 LLm |

**Figure 4** **Resultados TMM 1500x1500, 10 bloques**

| Self | Function |
|---|---|
| 159 107 300 142 | bloque |
| 184 512 710 | random |
| 135 066 044 | generat |
| 103 500 000 | random |
| 36 000 000 | rand |
| 27 019 206 | (unknov |
| 4 504 776 | (unknov |
| 617 389 | _int_ma |
| 211 666 | malloc |
| 67 716 | _dl_add |
| 24 142 | do_look |
| 20 625 | _dl_lool |

| Event Type | Incl. | Self | Short | Formula |
|---|---|---|---|---|
| Instruction Fetch | 159 107 300 142 | 159 107 300 142 | Ir | |
| L1 Instr. Fetch Miss | 7 | 7 | I1mr | |
| LL Instr. Fetch Miss | 7 | 7 | ILmr | |
| Data Read Access | 81 242 276 178 | 81 242 276 178 | Dr | |
| L1 Data Read Miss | 444 808 828 | 444 808 828 | D1mr | |
| LL Data Read Miss | 2 426 861 | 2 426 861 | DLmr | |
| Data Write Access | 3 399 907 124 | 3 399 907 124 | Dw | |
| L1 Data Write Miss | 140 913 | 140 913 | D1mw | |
| LL Data Write Miss | 140 913 | 140 913 | DLmw | |
| L1 Miss Sum | 444 949 748 | 444 949 748 | L1m | = I1mr + D1mr + D1mw |
| Last-level Miss Sum | 2 567 781 | 2 567 781 | LLm | = ILmr + DLmr + DLmw |
| Cycle Estimation | 163 813 575 722 | 163 813 575 722 | CEst | = Ir + 10 L1m + 100 LLm |

## 4. Conclusiones

- EL algoritmo TMM en comparacion con el de SMM es mucho mas eficiente en tiempos, esto debido a que la lectura a disco es menor.
- El algoritmo TMM tiene mucho menos perdidas de lectura a cache(cache misses), tras las pruebas el algoritmo SMM es casi un 70 porciento mas ineficiente.