

Multiplicación de Matrices: Clasica vs Bloques

Blas David Cruz Huaman
Escuela de Ciencia de la Computación
Universidad Catolica San Pablo
Email: blas.cruz@ucsp.edu.pe

April 1, 2021

GitHub : https://github.com/blasdch18/Computacion_Paralela_-Dist.git

Abstract

Se implementará los algoritmos de Multiplicación de matrices en forma clásica y modelo por Bloques, la dinamica es usar matrices de dimensiones como 100 x 100 hasta 800 x 800. Todo esto para el analisis de nuestro presente trabajo.

1 INTRODUCCION

Se realizará el analisis del Comportamiento de la Memoria Caché mediante el algoritmo de multiplicación de matrices clasico vs por Bloques. Para esto desarrollaremos ambos metodos que se utilizaron para realizar el siguiente trabajo.

2 MULTIPLICACION CLASICA

Solo se puede multiplicar dos matrices si sus dimensiones son compatibles, lo que significa que el número de columnas en la primera matriz es igual al número de filas en la segunda matriz.

El algoritmo clásico de multiplicación de matrices tiene una complejidad de $O(n^3)$. Se ha conjeturado que el algoritmo (clásico) óptimo debe alcanzar $O(n^2)$ en tiempo.

El algoritmo tiene $2n^3 = O(n^3)$ operaciones aritméticas (n elementos de la i -ésima fila de A multiplicada por la j -ésima columna de B) * (n columnas de B) * (n filas de A) * 2. 2 es tener en cuenta cada adición de $C[i, j]$ opera en $3 * n^2$ palabras de memoria, para A , B y res cada una de n^2 tamaño

```

(base) hp@Legion-Y540-15IRH-PG0:~$ lscpu
Arquitectura: x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes: Little Endian
Address sizes: 39 bits physical, 48 bits virtual
CPU(s): 8
Lista de la(s) CPU(s) en línea: 0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»: 4
«Socket(s)»: 1
Modo(s) NUMA: 1
ID de fabricante: GenuineIntel
Familia de CPU: 6
Modelo: 158
Nombre del modelo: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
Revisión: 10
CPU MHz: 2400.000
CPU MHz máx.: 4100.0000
CPU MHz mín.: 800.0000
BogoMIPS: 4800.00
Virtualización: VT-x
Caché L1d: 128 KiB
Caché L1i: 128 KiB
Caché L2: 1 MiB
Caché L3: 8 MiB
CPU(s) del nodo NUMA 0: 0-7
Vulnerability Itlb multihit: KVM: Mitigation: VMX disabled
Vulnerability L1tf: Mitigation; PTE Inversion; VMX conditional
cache flushes, SMT vulnerable
Vulnerability Mds: Mitigation; Clear CPU buffers; SMT vulnerab
le
Vulnerability Meltdown: Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabl
ed via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __
user_pointer sanitization

```

Figure 1: Características del Computador

2.1 Algoritmo

```

Matrix res(height, width);
// Lectura rápida en fila i de A
for (int i=0 ; i<height ; i++)
    // Lectura rápida en res[i][j]
    // Lectura rápida en Col j de B
    for (int j=0 ; j<width ; j++)
        for (int k = 0; k < width; k++)
            // Escritura Lenta en res[i][j]
            res.array[i][j] += arrayA[i][k] * m.arrayB[k][j];

```

3 MULTIPLICACION POR BLOQUES

Al multiplicar dos matrices por bloques, hay que dividir cada una de las matrices en una retícula rectangular de submatrices. Si queremos calcular $C = A*B$, de lo que nos tenemos que asegurar es de que la división de las n columnas de A coincida con la división de las n filas de B .

```

(base).hp@Legion-Y540-15IRH-PG0:~/CompParalelaDistribuida/MatrixMultiplicationBlocked$ valgrind --t
ool=callgrind --simulate-cache=yes ./matrix100
==5174== Callgrind, a call-graph generating cache profiler
==5174== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==5174== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==5174== Command: ./matrix100
==5174==
--5174-- warning: L3 cache found, using its data for the LL simulation.
==5174== For interactive control, run 'callgrind_control -h'.
|
|      Execution Normal Matr= 100x100 Multiplication Time: 3.11184
==5174==
==5174== Events      : Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLMw
==5174== Collected : 130060508 46168961 26583669 2275 82182 7478 2109 8214 5149
==5174==
==5174== I   refs:      130,060,508
==5174== I1  misses:      2,275
==5174== L1i misses:      2,109
==5174== I1  miss rate:      0.00%
==5174== L1i miss rate:      0.00%
==5174==
==5174== D   refs:      72,752,630 (46,168,961 rd + 26,583,669 wr)
==5174== D1  misses:      89,660 ( 82,182 rd +   7,478 wr)
==5174== L1d misses:      13,363 (  8,214 rd +   5,149 wr)
==5174== D1  miss rate:      0.1% (  0.2% +   0.0% )
==5174== L1d miss rate:      0.0% (  0.0% +   0.0% )
==5174==
==5174== LL refs:      91,935 ( 84,457 rd +   7,478 wr)
==5174== LL  misses:      15,472 ( 10,323 rd +   5,149 wr)
==5174== LL  miss rate:      0.0% (  0.0% +   0.0% )

```

Figure 2: Cache Monitor Mult. Matriz 100x100

```

Matrix res (size, size);
// Lectura Rapida de Bloque res[jj][kk]
for (int jj=0; jj<size; jj+=block_size)
    for (int kk=0; kk<size; kk+=block_size)
        // Lectura Rapida de Bloque A[jj][kk]
        // Lectura Rapida de Bloque B[kk][i]
        for (int i=0; i<size; i++)
            for (int j=jj; j<((jj+block_size)>size ? size:(jj+block_size)); j++)
                tmp = 0;
                for (int k=kk; k<((kk+block_size)>size ? size:(kk+block_size)); k++)
                    // Escritura Rapida de Bloque res[i][jj]
                    tmp += A.array[i][k] * B.array[k][j];
                    res.array[i][j] += tmp;
return res;

```

3.1 Algoritmo

El algoritmo por bloques de multiplicación de matrices tiene una complejidad de $O(n^3)$ igual que la forma clásica; sin embargo en este caso se está ejecutando un algoritmo paralelo ya que se divide cada bloque para así ser operado, y así nos da menor tiempo en sus resultados. Fig. 2

```

(base) hp@Legion-Y540-15IRH-PG0:~/CompParalelaDistribuida/MatrixMultiplicationBlocked$ valgrind --t
ool=callgrind --simulate-cache=yes ./matrix800
==7989== Callgrind, a call-graph generating cache profiler
==7989== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==7989== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==7989== Command: ./matrix800
==7989==
--7989-- warning: L3 cache found, using its data for the LL simulation.
==7989== For interactive control, run 'callgrind_control -h'.
||||||| Execution Normal Matr= 800x800 Multiplication Time: 1593.99
==7989==
==7989== Events      : Ir Dr Dw I1mr D1mr D1mw ILmr DLMr DLMw
==7989== Collected : 64526485717 23045988611 13314727558 2344 736539390 127134 2173 8334 124646
==7989==
==7989== I   refs:      64,526,485,717
==7989== I1  misses:      2,344
==7989== L1i misses:      2,173
==7989== I1  miss rate:      0.00%
==7989== L1i miss rate:      0.00%
==7989==
==7989== D   refs:      36,360,716,169 (23,045,988,611 rd + 13,314,727,558 wr)
==7989== D1  misses:      736,666,524 ( 736,539,390 rd +      127,134 wr)
==7989== L1d misses:      132,980 (      8,334 rd +      124,646 wr)
==7989== D1  miss rate:      2.0% (      3.2% +      0.0% )
==7989== L1d miss rate:      0.0% (      0.0% +      0.0% )
==7989==
==7989== LL refs:      736,668,868 ( 736,541,734 rd +      127,134 wr)
==7989== LL misses:      135,153 (      10,507 rd +      124,646 wr)
==7989== LL miss rate:      0.0% (      0.0% +      0.0% )

```

Figure 3: Cache Monitor Mult. Matriz 800x800

4 Conclusión

Este resultado es importante ya que muestra que podemos aumentar la intensidad computacional eligiendo un valor de tamaño grande para cada bloque y puede ser más rápido que la multiplicación clásica.

| NxN Matriz | Tiempo | I1 | LLi | D1 | LLd |
|------------|---------|------|------|-----------|--------|
| 100 | 3.11184 | 2275 | 2109 | 89660 | 13363 |
| 200 | 24.6026 | 2305 | 2133 | 564758 | 19195 |
| 300 | 80.5403 | 2337 | 2153 | 2778571 | 28761 |
| 400 | 193.071 | 2345 | 2158 | 71565584 | 42080 |
| 500 | 378.123 | 2345 | 2158 | 180084582 | 59149 |
| 600 | 647.207 | 2341 | 2156 | 311244345 | 79968 |
| 700 | 1069.47 | 2341 | 2156 | 382855178 | 104536 |
| 800 | 1593.99 | 2344 | 2173 | 736666524 | 132980 |

Table 1: Multiplicación Clasica : Dimensión y Misses

| NxN Matriz | I1 miss rate% | LLi miss rate% | D1 miss rate% | LLd miss rate% |
|------------|---------------|----------------|---------------|----------------|
| 100 | 0.00 | 0.00 | 0.1 | 0.0 |
| 200 | 0.00 | 0.00 | 0.1 | 0.0 |
| 300 | 0.00 | 0.00 | 0.1 | 0.0 |
| 400 | 0.00 | 0.00 | 1.6 | 0.0 |
| 500 | 0.00 | 0.00 | 2.0 | 0.0 |
| 600 | 0.00 | 0.00 | 2.0 | 0.0 |
| 700 | 0.00 | 0.00 | 1.6 | 0.0 |
| 800 | 0.00 | 0.00 | 2.0 | 0.0 |

Table 2: Multiplicación Clasica : Dimensión y Misses Rate

References

- [1] Pacheco P.: An Introduction to Parallel Programming Ed. MK, San Francisco, USA 2011.

```

(base) hp@Legion-V540-15IRH-PG0:~/CompParalelaDistribuida/MatrixMultiplicationBlocked$ valgrind --tool=callgrind
--simulate-cache=yes ./matrixB100
==12861== Callgrind, a call-graph generating cache profiler
==12861== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==12861== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==12861== Command: ./matrixB100
==12861==
--12861-- warning: L3 cache found, using its data for the LL simulation.
==12861== For interactive control, run 'callgrind_control -h'.

/      Execution Blocks Matr= 100x100 Multiplication Time: 2.24422
==12861==
==12861== Events      : Ir Dr Dw I1mr D1mr D1mw ILmr D1mr D1mw
==12861== Collected : 103856254 38799678 20693940 2314 32084 8188 2104 8203 5836
==12861==
==12861== I   refs:      103,856,254
==12861== I1  misses:      2,314
==12861== LLi misses:      2,104
==12861== I1  miss rate:      0.00%
==12861== LLi miss rate:      0.00%
==12861==
==12861== D   refs:      59,493,618 (38,799,678 rd + 20,693,940 wr)
==12861== D1  misses:      40,272 ( 32,084 rd +  8,188 wr)
==12861== LLd misses:      14,039 (  8,203 rd +  5,836 wr)
==12861== D1  miss rate:      0.1% (  0.1% +  0.0% )
==12861== LLd miss rate:      0.0% (  0.0% +  0.0% )
==12861==
==12861== LL refs:      42,586 ( 34,398 rd +  8,188 wr)
==12861== LL misses:      16,143 ( 10,307 rd +  5,836 wr)
==12861== LL miss rate:      0.0% (  0.0% +  0.0% )

```

Figure 4: Cache Monitor Mult.Bloques 100x100

| NxN Matriz | Tiempo | I1 | LLi | D1 | LLd |
|------------|---------|------|------|----------|--------|
| 100 | 2.24422 | 2314 | 2104 | 40272 | 14039 |
| 200 | 19.0681 | 2355 | 2129 | 672678 | 21809 |
| 300 | 62.9421 | 2362 | 2135 | 2510103 | 34565 |
| 400 | 148.410 | 2371 | 2143 | 5929912 | 52322 |
| 500 | 286.986 | 2368 | 2140 | 11607945 | 75078 |
| 600 | 492.369 | 2368 | 2140 | 19733445 | 102834 |
| 700 | 771.523 | 2371 | 2197 | 30176106 | 135807 |
| 800 | 1152.33 | 2371 | 2280 | 46993632 | 254302 |

Table 3: Multiplicación Por Bloques : Dimensión y Misses

```

(base) hp@Legion-Y540-15IRH-PG0:~/CompParalelaDistribuida/MatrixMultiplicationBlocked$ valgrind --tool=callgrind
--simulate-cache=yes ./matrix8800
==15858== Callgrind, a call-graph generating cache profiler
==15858== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==15858== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==15858== Command: ./matrix8800
==15858==
--15858-- warning: L3 cache found, using its data for the LL simulation.
==15858== For interactive control, run 'callgrind_control -h'.

==15858== brk segment overflow in thread #1: can't grow to 0x484f000
==15858== (see section Limitations in user manual)
==15858== NOTE: further instances of this message will not be shown
/////////
Execution Blocks Matr= 800x800 Multiplication Time: 1152.33
==15858==
==15858== Events      : Ir Dr Dw I1mr D1mr D1mw I1mr D1mr D1mw
==15858== Collected : 51009489289 19243335129 10275677396 2371 46825951 167681 2280 89107 165195
==15858==
==15858== I  refs:      51,009,489,289
==15858== I1 misses:      2,371
==15858== L1i misses:      2,280
==15858== I1 miss rate:      0.00%
==15858== L1i miss rate:      0.00%
==15858==
==15858== D  refs:      29,519,012,525 (19,243,335,129 rd + 10,275,677,396 wr)
==15858== D1 misses:      46,993,632 ( 46,825,951 rd + 167,681 wr)
==15858== L1d misses:      254,302 ( 89,107 rd + 165,195 wr)
==15858== D1 miss rate:      0.2% ( 0.2% + 0.0% )
==15858== L1d miss rate:      0.0% ( 0.0% + 0.0% )
==15858==
==15858== LL refs:      46,996,003 ( 46,828,322 rd + 167,681 wr)
==15858== LL misses:      256,582 ( 91,387 rd + 165,195 wr)
==15858== LL miss rate:      0.0% ( 0.0% + 0.0% )

```

Figure 5: Cache Monitor Mult. Bloques 800x800

| NxN Matriz | I1 miss rate% | L1i miss rate% | D1 miss rate% | L1d miss rate% |
|------------|---------------|----------------|---------------|----------------|
| 100 | 0.00 | 0.00 | 0.1 | 0.0 |
| 200 | 0.00 | 0.00 | 0.1 | 0.0 |
| 300 | 0.00 | 0.00 | 0.2 | 0.0 |
| 400 | 0.00 | 0.00 | 0.2 | 0.0 |
| 500 | 0.00 | 0.00 | 0.2 | 0.0 |
| 600 | 0.00 | 0.00 | 0.2 | 0.0 |
| 700 | 0.00 | 0.00 | 0.2 | 0.0 |
| 800 | 0.00 | 0.00 | 0.2 | 0.0 |

Table 4: Multiplicación Por Bloques : Dimensión y Misses Rate

Multiplicacion Clasica/Tiempo y Multiplicacion por Bloques/Tiempo

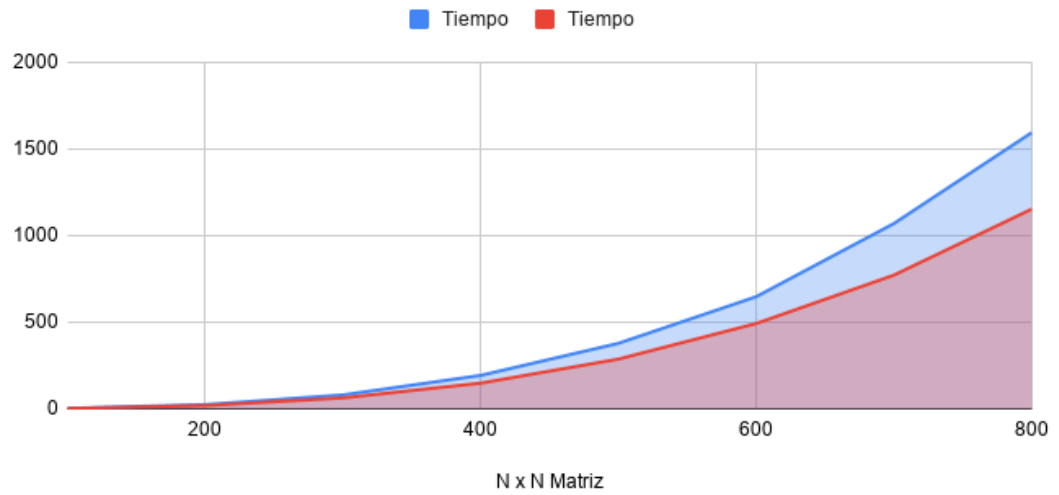


Figure 6: Ejecución de Resultados: Tiempos

Multiplicacion Clasica/LLd y Multiplicacion por Bloques/LLd

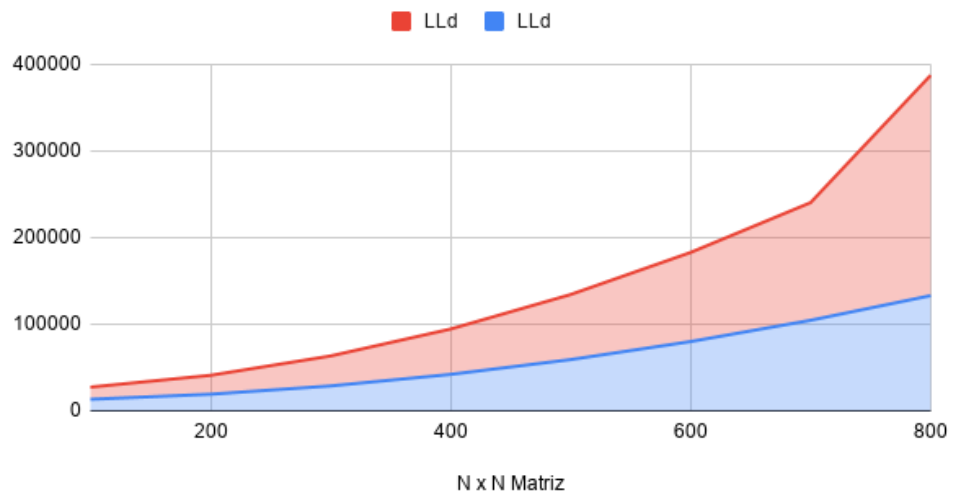


Figure 7: Ejecución de Resultados: Instruction Cache Misses

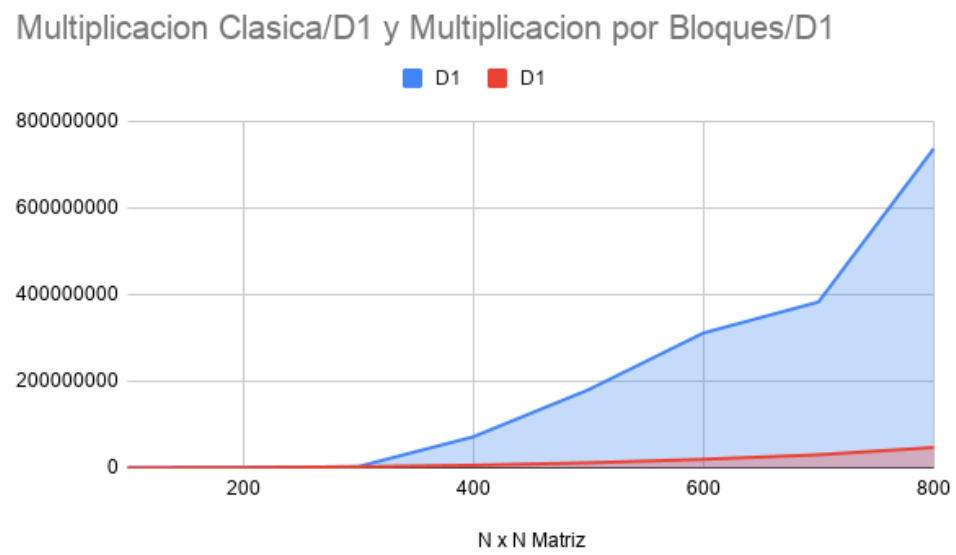


Figure 8: Ejecución de Resultados: Data Cache Misses