Pruebas sobre el comportamiento de la memoria caché

Fabrizio Miguel Mattos Cahui,
Ciencia de la Computación
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú

I. Implementación de multiplicación de matrices clásica

A. Código

```
void matrix_multiplication(int**& mat1,
int**& mat2,int**& result, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0;k < size; k++)
            {
                result[i][j] += mat1[i][k]
                * mat2[k][j];
            }
        }

    }
}
```

B. Ejecución

1) Matriz de tamaño 100: Ejecución con una matriz de tamaño 100 por 100.

```
Time taken by first pair of loops: 4099 microseconds

Press any key to continue . . .
```

Fig. 1: Resultado con 100 elementos

2) Matriz de tamaño 500: Ejecución con una matriz de tamaño 500 por 500.

```
Time taken by first pair of loops: 598166 microseconds

Press any key to continue . . .
```

Fig. 2: Resultado con 500 elementos

3) Matriz de tamaño 1000: Ejecución con una matriz de tamaño 1000 por 1000.

```
Time taken by first pair of loops: 5738980 microseconds

Press any key to continue . . .
```

Fig. 3: Resultado con 1000 elementos

4) Matriz de tamaño 1500: Ejecución con una matriz de tamaño 1500 por 1500.

```
Time taken by first pair of loops: 21551587 microseconds

Press any key to continue . . .
```

Fig. 4: Resultado con 1500 elementos

5) Matriz de tamaño 2000: Ejecución con una matriz de tamaño 2000 por 2000.

```
Time taken by first pair of loops: 76170247 microseconds

Press any key to continue . . .
```

Fig. 5: Resultado con 2000 elementos

6) Matriz de tamaño 2500: Ejecución con una matriz de tamaño 2500 por 2500.

```
Time taken by first pair of loops: 161862488 microseconds

Press any key to continue . . .
```

Fig. 6: Resultado con 2500 elementos

C. Explicación

La función realiza un bucle anidado triple para recorrer las matrices mat1 y mat2 y realizar la multiplicación de matrices. Este bucle anidado garantiza un acceso a memoria con localidad espacial. Esto significa que cuando se accede a un elemento de mat1[i][k], es probable que los elementos adyacentes en la misma fila mat1[i][k+1], mat1[i][k+2], etc., también estén en caché. Lo mismo ocurre para mat2[k][j]. Esto minimiza los fallos de caché y mejora el rendimiento.

II. Implementación de multiplicación de matrices por bloques

A. Código

```
void blockMultiplication(int**& mat1,
int**& mat2,int**& result,
int size, int blockSize)
{
for(int i=0; i<size; i+=blockSize)
    for(int j=0; j<size; j+=blockSize)
        for(int k=0;k<size;k+=blockSize)
            for(int x=i; x <
            std::min(i + blockSize
            , size);x++)
                for(int y=j;
                y<std::min(j + blockSize
                , size);y++)
                    for(int z=k;
                    z<std::min(k+blockSize
                    ,size);z++)
                        result[x][y]+=
                    mat1[x][z]*mat2[z][y];
}
```

B. Ejecución

1) Matriz de tamaño 100: Ejecución con una matriz de tamaño 100 por 100.

```
Time taken: 17510 microseconds

Press any key to continue . . .
```

Fig. 7: Resultado con 100 elementos

2) Matriz de tamaño 500: Ejecución con una matriz de tamaño 500 por 500.

```
Time taken: 1157808 microseconds

Press any key to continue . . .
```

Fig. 8: Resultado con 500 elementos

3) Matriz de tamaño 1000: Ejecución con una matriz de tamaño 1000 por 1000.

```
Time taken: 8390899 microseconds

Press any key to continue . . .
```

Fig. 9: Resultado con 1000 elementos

4) Matriz de tamaño 1500: Ejecución con una matriz de tamaño 1500 por 1500.

```
Time taken: 22051207 microseconds

Press any key to continue . . .
```

Fig. 10: Resultado con 1500 elementos

5) Matriz de tamaño 2000: Ejecución con una matriz de tamaño 2000 por 2000.

```
Time taken: 47902351 microseconds

Press any key to continue . . .
```

Fig. 11: Resultado con 2000 elementos

6) Matriz de tamaño 2500: Ejecución con una matriz de tamaño 2500 por 2500.

```
Time taken: 90217413 microseconds

Press any key to continue . . .
```

Fig. 12: Resultado con 2500 elementos

C. Explicación

El manejo de la caché en esta función ‘blockMatrixMultiplication‘ es similar al enfoque explicado anteriormente en la función de multiplicación de matrices sin bloqueo. Sin embargo, aquí se utilizan bloques más pequeños (‘blockSize x blockSize‘) para mejorar la eficiencia del acceso a la caché. A continuación, se explica cómo funciona el manejo de la caché en esta función:

- Localidad espacial: Al dividir las matrices en bloques de tamaño ‘blockSize x blockSize‘, se explora un enfoque de localidad espacial. Esto significa que cuando se accede a un elemento de una matriz dentro de un bloque, es más probable que otros elementos del mismo bloque se encuentren en caché. Esto minimiza los fallos de caché y mejora el rendimiento al aprovechar la localidad espacial.
- Bucles anidados para bloques: Los bucles ‘for‘ externos (‘i‘, ‘j‘, y ‘k‘) dividen las matrices en bloques y controlan la forma en que se procesan. Estos bucles permiten un acceso a memoria más eficiente al garantizar que los elementos del bloque actual estén en caché mientras se realizan las operaciones de multiplicación.
- Bucles anidados para elementos dentro del bloque: Los bucles ‘for‘ internos (‘x‘, ‘y‘, y ‘z‘) se utilizan para recorrer y multiplicar los elementos dentro de cada bloque. Al recorrer estos elementos de manera consecutiva en memoria, se mejora la localidad espacial y se minimizan los fallos de caché.

III. Valgrind

Para realizar las pruebas usando Valgrind, se instalo la distribución Linux Mint 21.2 en la máquina virtual VMware.

A. Multiplicación Clásica

1) Resultado primer caso: Multiplicación de matriz de 100 por 100.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./threelo
op
==3460== Cachegrind, a cache and branch-prediction profiler
==3460== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==3460== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3460== Command: ./threeloop
==3460==
--3460-- warning: L3 cache found, using its data for the LL simulation.
==3460==
==3460== I   refs:      57,204,558
==3460== I1  misses:         1,971
==3460== LLi misses:         1,866
==3460== I1  miss rate:       0.00%
==3460== LLi miss rate:       0.00%
==3460==
==3460== D   refs:      28,197,069  (26,948,416 rd   + 1,248,653 wr)
==3460== D1  misses:        84,560  (    79,989 rd   +     4,571 wr)
==3460== LLd misses:        11,241  (     7,666 rd   +     3,575 wr)
==3460== D1  miss rate:        0.3% (       0.3%     +       0.4%  )
==3460== LLd miss rate:        0.0% (       0.0%     +       0.3%  )
==3460==
==3460== LL refs:           86,531  (    81,960 rd   +     4,571 wr)
==3460== LL misses:         13,107  (     9,532 rd   +     3,575 wr)
==3460== LL miss rate:         0.0% (       0.0%     +       0.3%  )
```

Fig. 13: Ejecución con matriz de tamaño 100

2) Resultado segundo caso: Multiplicación de matriz de 500 por 500.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./threelo
op
==3471== Cachegrind, a cache and branch-prediction profiler
==3471== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==3471== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3471== Command: ./threeloop
==3471==
--3471-- warning: L3 cache found, using its data for the LL simulation.
==3471==
==3471== I   refs:      6,773,659,143
==3471== I1  misses:            1,968
==3471== LLi misses:            1,872
==3471== I1  miss rate:          0.00%
==3471== LLi miss rate:          0.00%
==3471==
==3471== D   refs:      3,386,380,133  (3,260,130,971 rd   + 126,249,162 wr)
==3471== D1  misses:      136,314,783  (  136,263,343 rd   +      51,440 wr)
==3471== LLd misses:           56,803  (        7,778 rd   +      49,025 wr)
==3471== D1  miss rate:           4.0% (          4.2%     +         0.0%  )
==3471== LLd miss rate:           0.0% (          0.0%     +         0.0%  )
==3471==
==3471== LL refs:         136,316,751  (  136,265,311 rd   +      51,440 wr)
==3471== LL misses:            58,675  (        9,650 rd   +      49,025 wr)
==3471== LL miss rate:            0.0% (          0.0%     +         0.0%  )
```

Fig. 14: Ejecución con matriz de tamaño 500

3) Resultado tercer caso: Multiplicación de matriz de 1000 por 1000.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./threeloop
==3494== Cachegrind, a cache and branch-prediction profiler
==3494== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==3494== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3494== Command: ./threeloop
==3494==
--3494-- warning: L3 cache found, using its data for the LL simulation.
==3494== brk segment overflow in thread #1: can't grow to 0x485b000
==3494== (see section Limitations in user manual)
==3494== NOTE: further instances of this message will not be shown
==3494==
==3494== I   refs:      54,087,024,897
==3494== I1  misses:             2,001
==3494== LLi misses:             1,929
==3494== I1  miss rate:           0.00%
==3494== LLi miss rate:           0.00%
==3494==
==3494== D   refs:      27,043,013,688  (26,038,709,258 rd   + 1,004,304,430 wr)
==3494== D1  misses:     1,188,520,691  ( 1,188,326,489 rd   +       194,202 wr)
==3494== LLd misses:           350,452  (       160,090 rd   +       190,362 wr)
==3494== D1  miss rate:            4.4% (           4.6%     +           0.0%  )
==3494== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==3494==
==3494== LL refs:        1,188,522,692  ( 1,188,328,490 rd   +       194,202 wr)
==3494== LL misses:            352,381  (       162,019 rd   +       190,362 wr)
==3494== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

Fig. 15: Ejecución con matriz de tamaño 1000

4) Resultado cuarto caso: Multiplicación de matriz de 1500 por 1500.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./threeloop
==3524== Cachegrind, a cache and branch-prediction profiler
==3524== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==3524== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3524== Command: ./threeloop
==3524==
--3524-- warning: L3 cache found, using its data for the LL simulation.
==3524== brk segment overflow in thread #1: can't grow to 0x4845000
==3524== (see section Limitations in user manual)
==3524== NOTE: further instances of this message will not be shown
==3524==
==3524== I   refs:      182,442,427,273
==3524== I1  misses:              1,996
==3524== LLi misses:              1,923
==3524== I1  miss rate:            0.00%
==3524== LLi miss rate:            0.00%
==3524==
==3524== D   refs:       91,220,658,548  (87,836,296,064 rd   + 3,384,362,484 wr)
==3524== D1  misses:      3,586,914,725  ( 3,586,483,897 rd   +       430,828 wr)
==3524== LLd misses:         47,189,744  (    46,763,911 rd   +       425,833 wr)
==3524== D1  miss rate:             3.9% (           4.1%     +           0.0%  )
==3524== LLd miss rate:             0.1% (           0.1%     +           0.0%  )
==3524==
==3524== LL refs:         3,586,916,721  ( 3,586,485,893 rd   +       430,828 wr)
==3524== LL misses:          47,191,667  (    46,765,834 rd   +       425,833 wr)
==3524== LL miss rate:              0.0% (           0.0%     +           0.0%  )
```

Fig. 16: Ejecución con matriz de tamaño 1500

5) Resultado quinto caso: Multiplicación de matriz de 2000 por 2000.

```
==5130== Cachegrind, a cache and branch-prediction profiler
==5130== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==5130== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5130== Command: ./threeloop
==5130==
--5130-- warning: L3 cache found, using its data for the LL simulation.
==5130== brk segment overflow in thread #1: can't grow to 0x484a000
==5130== (see section Limitations in user manual)
==5130== NOTE: further instances of this message will not be shown
==5130==
==5130== I   refs:      432,339,861,405
==5130== I1  misses:              2,000
==5130== LLi misses:              1,926
==5130== I1  miss rate:            0.00%
==5130== LLi miss rate:            0.00%
==5130==
==5130== D   refs:      216,169,314,516  (208,152,890,122 rd   + 8,016,424,394 wr)
==5130== D1  misses:      8,423,093,067  (  8,422,331,522 rd   +       761,545 wr)
==5130== LLd misses:        508,138,949  (    507,381,585 rd   +       757,364 wr)
==5130== D1  miss rate:             3.9% (             4.0%     +           0.0%  )
==5130== LLd miss rate:             0.2% (             0.2%     +           0.0%  )
==5130==
==5130== LL refs:         8,423,095,067  (  8,422,333,522 rd   +       761,545 wr)
==5130== LL misses:         508,140,875  (    507,383,511 rd   +       757,364 wr)
==5130== LL miss rate:              0.1% (             0.1%     +           0.0%  )
```

Fig. 17: Ejecución con matriz de tamaño 2000

B. Multiplicación por bloques

1) Primer caso: Matriz de 100 por 100.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./block
==5193== Cachegrind, a cache and branch-prediction profiler
==5193== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==5193== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5193== Command: ./block
==5193==
--5193-- warning: L3 cache found, using its data for the LL simulation.
==5193==
==5193== I   refs:      88,751,119
==5193== I1  misses:         1,976
==5193== LLi misses:         1,871
==5193== I1  miss rate:       0.00%
==5193== LLi miss rate:       0.00%
==5193==
==5193== D   refs:      46,009,302  (38,554,631 rd   + 7,454,671 wr)
==5193== D1  misses:        26,116  (    21,546 rd   +     4,570 wr)
==5193== LLd misses:        11,244  (     7,669 rd   +     3,575 wr)
==5193== D1  miss rate:        0.1% (       0.1%     +       0.1%  )
==5193== LLd miss rate:        0.0% (       0.0%     +       0.0%  )
==5193==
==5193== LL refs:           28,092  (    23,522 rd   +     4,570 wr)
==5193== LL misses:         13,115  (     9,540 rd   +     3,575 wr)
==5193== LL miss rate:         0.0% (       0.0%     +       0.0%  )
```

Fig. 18: Primer caso

2) Segundo caso: Matriz de 500 por 500.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./block
==5202== Cachegrind, a cache and branch-prediction profiler
==5202== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==5202== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5202== Command: ./block
==5202==
--5202-- warning: L3 cache found, using its data for the LL simulation.
==5202==
==5202== I   refs:      10,262,173,029
==5202== I1  misses:             1,973
==5202== LLi misses:             1,877
==5202== I1  miss rate:           0.00%
==5202== LLi miss rate:           0.00%
==5202==
==5202== D   refs:       5,345,656,234  (4,528,424,510 rd   + 817,231,724 wr)
==5202== D1  misses:         1,067,346  (    1,015,907 rd   +      51,439 wr)
==5202== LLd misses:            56,811  (        7,786 rd   +      49,025 wr)
==5202== D1  miss rate:            0.0% (          0.0%     +         0.0%  )
==5202== LLd miss rate:            0.0% (          0.0%     +         0.0%  )
==5202==
==5202== LL refs:            1,069,319  (    1,017,880 rd   +      51,439 wr)
==5202== LL misses:             58,688  (        9,663 rd   +      49,025 wr)
==5202== LL miss rate:             0.0% (          0.0%     +         0.0%  )
```

Fig. 19: Segundo caso

3) Tercer caso: Matriz de 1000 por 1000.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./block
==5220== Cachegrind, a cache and branch-prediction profiler
==5220== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==5220== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5220== Command: ./block
==5220==
--5220-- warning: L3 cache found, using its data for the LL simulation.
==5220== brk segment overflow in thread #1: can't grow to 0x485b000
==5220== (see section Limitations in user manual)
==5220== NOTE: further instances of this message will not be shown
==5220==
==5220== I   refs:      81,069,943,682
==5220== I1  misses:             2,006
==5220== LLi misses:             1,934
==5220== I1  miss rate:           0.00%
==5220== LLi miss rate:           0.00%
==5220==
==5220== D   refs:      42,193,576,039  (35,826,810,937 rd   + 6,366,765,102 wr)
==5220== D1  misses:         5,554,138  (     5,359,937 rd   +       194,201 wr)
==5220== LLd misses:           351,543  (       161,155 rd   +       190,388 wr)
==5220== D1  miss rate:            0.0% (           0.0%     +           0.0%  )
==5220== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==5220==
==5220== LL refs:            5,556,144  (     5,361,943 rd   +       194,201 wr)
==5220== LL misses:            353,477  (       163,089 rd   +       190,388 wr)
==5220== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

Fig. 20: Tercer caso

4) Cuarto caso: Matriz de 1500 por 1500.

```
test@test-virtual-machine:~/Downloads/Test$ valgrind --tool=cachegrind ./block
==5232== Cachegrind, a cache and branch-prediction profiler
==5232== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==5232== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5232== Command: ./block
==5232==
--5232-- warning: L3 cache found, using its data for the LL simulation.
==5232== brk segment overflow in thread #1: can't grow to 0x4845000
==5232== (see section Limitations in user manual)
==5232== NOTE: further instances of this message will not be shown
==5232==
==5232== I   refs:      272,245,095,617
==5232== I1  misses:             2,001
==5232== LLi misses:             1,928
==5232== I1  miss rate:           0.00%
==5232== LLi miss rate:           0.00%
==5232==
==5232== D   refs:      141,609,568,049  (120,361,753,083 rd   + 21,247,814,966 wr)
==5232== D1  misses:         19,455,337  (     19,024,510 rd   +       430,827 wr)
==5232== LLd misses:          2,343,986  (      1,918,151 rd   +       425,835 wr)
==5232== D1  miss rate:             0.0% (            0.0%     +           0.0%  )
==5232== LLd miss rate:             0.0% (            0.0%     +           0.0%  )
==5232==
==5232== LL refs:            19,457,338  (     19,026,511 rd   +       430,827 wr)
==5232== LL misses:           2,345,914  (      1,920,079 rd   +       425,835 wr)
==5232== LL miss rate:              0.0% (            0.0%     +           0.0%  )
```

Fig. 21: Cuarto caso

IV. Anexos

El código fuente de la implementación de los algoritmos se encuentra en el siguiente enlace de github: https://github.com/fm1299/Parallel-Computing/tree/main/Lab%2002

V. Conclusión

De acuerdo a los resultados se puede observar que la multiplicación por bloques es más eficiente que la multiplicación normal, esto debido al mejor manejo y acceso a la memoria cache que la multiplicación por bloques realiza, lo que resulta en una menor cantidad de cache misses durante la ejecución del programa, por lo que las operaciones de lectura y escritura son realizadas en menos tiempo. Sin embargo el rendimiento del método por bloques se verá muy afectado en función al valor que se le asigne al tamaño de los bloques en los que se dividirá la matriz, ya que tiene que estar equilibrado con el tamaño de la matriz, ya que si la matriz es de 1000 por 1000 y el tamaño del bloque es de 4, significa que la matriz será dividida en muchas submatrices, que puede resultar en un mal manejo de la memoria cache.

References