Universidad Nacional de San Agustin

Escuela Profesional de
Ciencia de la Computación

Algoritmos Paralelos

Impacto de Cache

Alumna :
Rosa Yuliana Gabriela
Paccotacya Yanque
Profesor:
Mg.Alvaro Henry Mamani
Aliaga

Índice

1. Multiplicacion de matrices con 3 bucles anidados **2**

2. Multiplicacion de matrices con 6 bloques **3**

3. Experimentos y Resultados **7**

1. Multiplicacion de matrices con 3 bucles anidados

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define min( a, b ) ( ((a) < (b)) ? (a) : (b) )

typedef int NUM;

struct Matrix
{
        int r, c;
        NUM **m;
};

void print(struct Matrix m) {
        int i, j;
        for (i = 0; i < m.r; ++i) {
                for (j = 0; j < m.c; ++j)
                        printf("%d\t", m.m[i][j]);
                printf("\n");
        }
}

void generate(struct Matrix *m) {
        int i, j;
        m->m = malloc(m->r * sizeof(NUM*));
        for (i = 0; i < m->r; ++i) {
                m->m[i] = malloc(m->c * sizeof(NUM));
                for (j = 0; j < m->c; ++j)
                        m->m[i][j] = (rand() % 10);
        }
}

struct Matrix three_nested_loop(struct Matrix a, struct Matrix b) {
        struct Matrix m;
        m.r = a.r; m.c = b.c;
```

```
38          int i, j, k;
39          m.m = malloc(m.r * sizeof(NUM*));
40          for (i = 0; i < m.r; ++i) {
41                  m.m[i] = malloc(m.c * sizeof(NUM));
42                  for (j = 0; j < m.c; ++j) {
43                          m.m[i][j] = 0;
44                          for (k = 0; k < a.c; ++k)
45                                  m.m[i][j] = m.m[i][j] + (a.m[i][k] * b.m[k][j]);
46                  }
47          }
48          return m;
49  }
50
51  int main(int argc, char *argv[])
52  {
53          srand(time(NULL));
54
55          int n = 800, blk = 10;
56          struct Matrix m1, m2, p1, p2;
57          m1.r = n; m1.c = n; m2.r = n; m2.c = n;
58
59          generate(&m1);
60          generate(&m2);
61
62
63
64          clock_t start = clock(), end;
65          p1 = three_nested_loop(m1, m2);
66          end = clock();
67          printf("%f\n", ((double)(end - start) / CLOCKS_PER_SEC));
68          //print(p1);
69
70          return 0;
71  }
```

2. Multiplicacion de matrices con 6 bloques

```
1  #include <time.h>
2  #include <stdlib.h>
```

```
3  #include <stdio.h>
4  #include <math.h>
5
6  #define min( a, b ) ( ((a) < (b)) ? (a) : (b) )
7
8  typedef int NUM;
9
10 struct Matrix
11 {
12         int r, c;
13         NUM **m;
14 };
15
16 void print(struct Matrix m) {
17         int i, j;
18         for (i = 0; i < m.r; ++i) {
19                 for (j = 0; j < m.c; ++j)
20                         printf("%d\t", m.m[i][j]);
21                 printf("\n");
22         }
23 }
24
25 void generate(struct Matrix *m) {
26         int i, j;
27         m->m = malloc(m->r * sizeof(NUM*));
28         for (i = 0; i < m->r; ++i) {
29                 m->m[i] = malloc(m->c * sizeof(NUM));
30                 for (j = 0; j < m->c; ++j)
31                         m->m[i][j] = (rand() % 10);
32         }
33 }
34
35 struct Matrix bloqued_version(struct Matrix a, struct Matrix b, int blocks) {
36         struct Matrix m;
37         m.r = a.r; m.c = b.c;
38         int i, j, k, ii, jj, kk;
39         m.m = malloc(m.r * sizeof(NUM*));
40         for(i = 0; i < m.r; i++){
41                 m.m[i] = malloc(sizeof(NUM) * m.c);
42         }
```

```
43          for(i = 0; i < m.r; i++){
44                  for(j = 0; j < m.c; j++){
45                          m.m[i][j] = 0;
46                  }
47          }
48          for(ii = 0; ii < m.r; ii += blocks) {
49                  for(jj = 0; jj < m.c; jj +=  blocks) {
50                          for(kk = 0; kk < a.c; kk += blocks) {
51                                  for(i = ii; i < min(ii + blocks, m.r); ++i) {
52                                          for(j = jj; j < min(jj + blocks, m.c); ++j)
53                                                  for(k = kk; k < min(kk + blocks, a.
54                                                          m.m[i][j] = m.m[i][j] + (a.
55                                          }
56                                  }
57                          }
58                  }
59          }
60          return m;
61  }
62
63  int main(int argc, char *argv[])
64  {
65          srand(time(NULL));
66
67          int n = 1500, blk = 10;
68          struct Matrix m1, m2, p1, p2;
69          m1.r = n; m1.c = n; m2.r = n; m2.c = n;
70
71          generate(&m1);
72          generate(&m2);
73
74          /*print(m1);
75          printf("\n");
76          print(m2);
77          printf("\n");*/
78
79          clock_t start = clock(), end;
80          p2 = bloqued_version(m1, m2, n / blk);
81          end = clock();
82          printf("%f\n", ((double)        (end - start) / CLOCKS_PER_SEC));
```

```
83          //print(p2);
84
85          return 0;
86  }
```

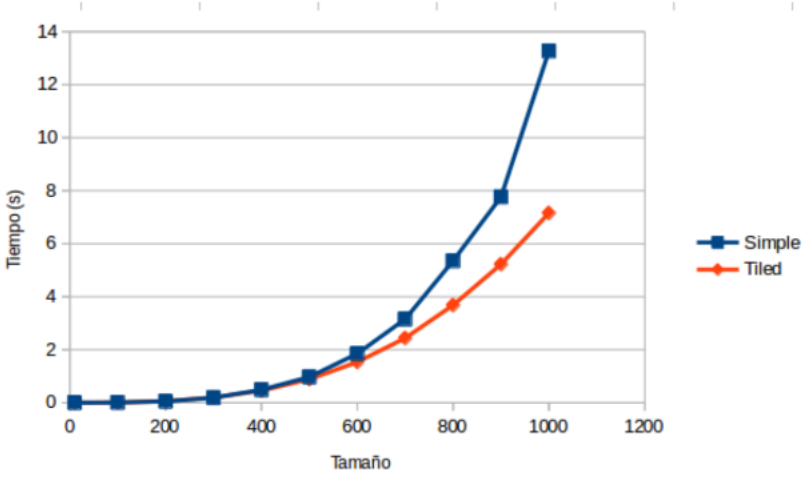
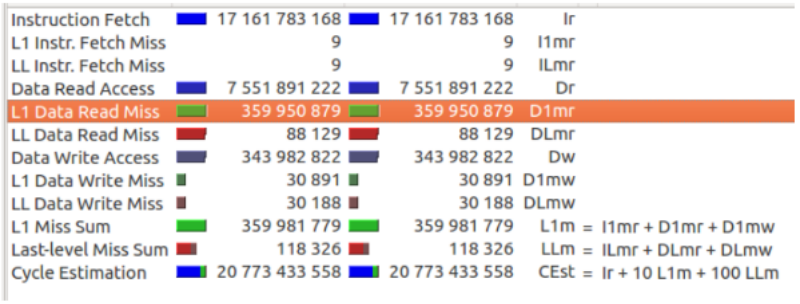


Figura 1: Comparacion entre los dos tipos de multiplicación

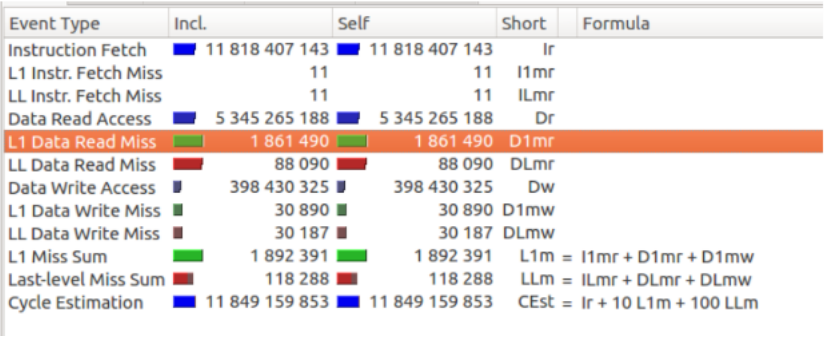
3. Experimentos y Resultados

En la Figura 1 se muestra un grafico con el tiempo de demora de los algoritmos respecto al tamaño de las matrices. Claramente se observa que el algoritmo con bloques es mas rapido que el algoritmo con 3 loops cuando el tamaño aumenta. El Algoritmo con bloques divide la matriz en bloques mas pequeños, de tal manera que una fila de esos bloques quepa en una lınea de cache y poder utilizar esa fila lo mas que se pueda, de modo que los cache misses se reduzcan considerablemente. En este experimento se supuso un tamaño de lınea de cache de 64 bytes. Ya que los enteros ocupan 4 bytes, las divisiones hechas en el experimento son de 16 numeros por bloque.El Algoritmo con 3 loops no divide la matriz. Si el tamaño de una fila es mas grande que el tamaño de una linea de cache, al querer utilizar esa fila para la multiplicacion, se generan muchos mas cache misses. Se utilizo valgrind y kcachegrind para ver el manejo de la memoria de ambos algoritmos, para esto se utilizaron matrices de 700x700 (Figura 2 y Figura 3).



Instruction Fetch	17 161 783 168	17 161 783 168	Ir	
L1 Instr. Fetch Miss	9	9	I1mr	
LL Instr. Fetch Miss	9	9	ILmr	
Data Read Access	7 551 891 222	7 551 891 222	Dr	
L1 Data Read Miss	359 950 879	359 950 879	D1mr	
LL Data Read Miss	88 129	88 129	DLmr	
Data Write Access	343 982 822	343 982 822	Dw	
L1 Data Write Miss	30 891	30 891	D1mw	
LL Data Write Miss	30 188	30 188	DLmw	
L1 Miss Sum	359 981 779	359 981 779	L1m	= I1mr + D1mr + D1mw
Last-level Miss Sum	118 326	118 326	LLm	= ILmr + DLmr + DLmw
Cycle Estimation	20 773 433 558	20 773 433 558	CEst	= Ir + 10 L1m + 100 LLm

Figura 2: Cache misses para Multiplicacion con 3 bucles



Event Type	Incl.	Self	Short	Formula
Instruction Fetch	11 818 407 143	11 818 407 143	Ir	
L1 Instr. Fetch Miss	11	11	I1mr	
LL Instr. Fetch Miss	11	11	ILmr	
Data Read Access	5 345 265 188	5 345 265 188	Dr	
L1 Data Read Miss	1 861 490	1 861 490	D1mr	
LL Data Read Miss	88 090	88 090	DLmr	
Data Write Access	398 430 325	398 430 325	Dw	
L1 Data Write Miss	30 890	30 890	D1mw	
LL Data Write Miss	30 187	30 187	DLmw	
L1 Miss Sum	1 892 391	1 892 391	L1m	= I1mr + D1mr + D1mw
Last-level Miss Sum	118 288	118 288	LLm	= ILmr + DLmr + DLmw
Cycle Estimation	11 849 159 853	11 849 159 853	CEst	= Ir + 10 L1m + 100 LLm

Figura 3: Cache misses para Multiplicacion con bloques