

Memoria Cache ,Bucles Anidados

Angel Andres Bejar Merma
Universidad Nacional de San Agustin
abejar@unsa.com
Ciudad de Arequipa

18 de septiembre de 2023

Resumen

Se implementara y analizara el codigo de bucles anidados For ademas de comprender la función de la memoria cache.

Keywords— Computación paralela, bucles, memoria, caché

1. Actividad

Implementar y comparar los 2-bucles anidados FOR presentados en el cap. 2 del libro, pag 22.

| Cache Line | Elements of A | | | |
|------------|---------------|---------|---------|---------|
| 0 | A[0][0] | A[0][1] | A[0][2] | A[0][3] |
| 1 | A[1][0] | A[1][1] | A[1][2] | A[1][3] |
| 2 | A[2][0] | A[2][1] | A[2][2] | A[2][3] |
| 3 | A[3][0] | A[3][1] | A[3][2] | A[3][3] |

En los siguientes funciones de código 1, esperaríamos que el primer par de bucles anidados tuviera un rendimiento mucho mejor que el segundo, ya que accede a los datos del array de 2 bidimensional en bloques contiguos.

Entonces, por ejemplo, $A[0][1]$ se almacena inmediatamente después de $A[0][0]$ y $A[1][0]$ se almacena inmediatamente después de $A[0][3]$. Supongamos que ninguno de los elementos de A está en el caché cuando cada par de bucles comienza a ejecutarse. Supongamos también que una línea de caché consta de cuatro elementos de A y $A[0][0]$ es el primer elemento de una línea de caché. Finalmente, supondremos que el caché está mapeado directamente y solo puede almacenar ocho elementos de A , o dos líneas de caché. (No nos preocuparemos por xey). Ambos pares de bucles intentan acceder primero a $A[0][0]$. Como no está en el caché, esto resultará en una pérdida de caché y el sistema leerá la línea que consta de la primera fila de A , $A[0][0]$, $A[0][1]$, $A[0][2]$, $A[0][3]$, en el caché. Luego, el primer par de bucles accede a $A[0][1]$, $A[0][2]$, $A[0][3]$, todos los cuales están en el caché, y el siguiente error en el primer par de bucles ocurre cuando el código accede a $A[1][0]$. Siguiendo de esta manera, vemos que el primer par de bucles dará como resultado un total de cuatro fallos cuando acceda a los elementos de A , uno para cada fila. Tenga en cuenta que dado que nuestro caché hipotético solo puede almacenar dos líneas u ocho elementos de A , cuando leemos el primer elemento de la fila dos y el primer elemento de la fila tres, una de las líneas que ya está en el caché tendrá que ser desalojada del caché, pero una vez que se expulsa una línea, el primer par de bucles no necesitará acceder a los elementos de esa línea nuevamente. Después de leer la primera fila en el caché, el segundo par de bucles debe acceder a $A[1][0]$, $A[2][0]$, $A[3][0]$, ninguno de los cuales está en el caché. Entonces los próximos tres accesos de A también resultarán en errores. Además, debido a que el caché es pequeño, las lecturas de $A[2][0]$ y $A[3][0]$ requerirán que se expulsen las líneas que ya están en el caché.

Dado que $A[2][0]$ está almacenado en la línea de caché 2, leer su línea desalojará la línea 0, y leer $A[3][0]$ desalojará la línea 1. Después de terminar el primer paso por el bucle externo, A continuación, deberá acceder a $A[0][1]$, que fue desalojado con el resto de la primera fila. Entonces vemos que cada vez que leemos un elemento de A , fallaremos, y el segundo par de bucles resulta en 16 errores. Por lo tanto, esperaríamos que el primer par de bucles anidados fuera mucho más rápido que el segundo. De hecho, si ejecutamos el código en uno de nuestros sistemas con $MAX = 1000$, el primer par de bucles anidados es aproximadamente tres veces más rápido que el segundo par Pacheco, 2011.

2. Análisis

Se hizo 5 pruebas con memoria cache, de los cuales el segundo par de bucles anidado es mucha mas lento en cuanto a tiempo de ejecución, esto se puede ver en la Fig 1 ,

Esto debido a :

- Primero :Teniendo en cuenta que la cache solo puede almacenar 2 líneas o 8 elementos de A y $A_{m \times n}$. Primer par de bucles; para acceder a el primer elemento de la matriz, esta resultará en una falla de caché, ya que esta no se encuentra en la caché, luego el sistema leerá toda la primera fila de A. Así sucesivamente cometiendo m fallos.
- Segundo : Después de leer la primera fila en la caché, este necesita acceder a $A[1][0]$, $A[2][0]$,... $A[m][0]$, los cuales no se encuentran en caché por ende estos accesos resultarán en fallos, y debido a que la caché es muy pequeña las lecturas para acceder a las demás filas de A se requerirá que las líneas anteriores sean desalojadas. Esto se genera n Veces, por lo cual se tendrá $m \cdot n$ fallosCormen et al., 2022 .

3. Codigo

El código Pacheco, 2011 implementado es el siguiente :

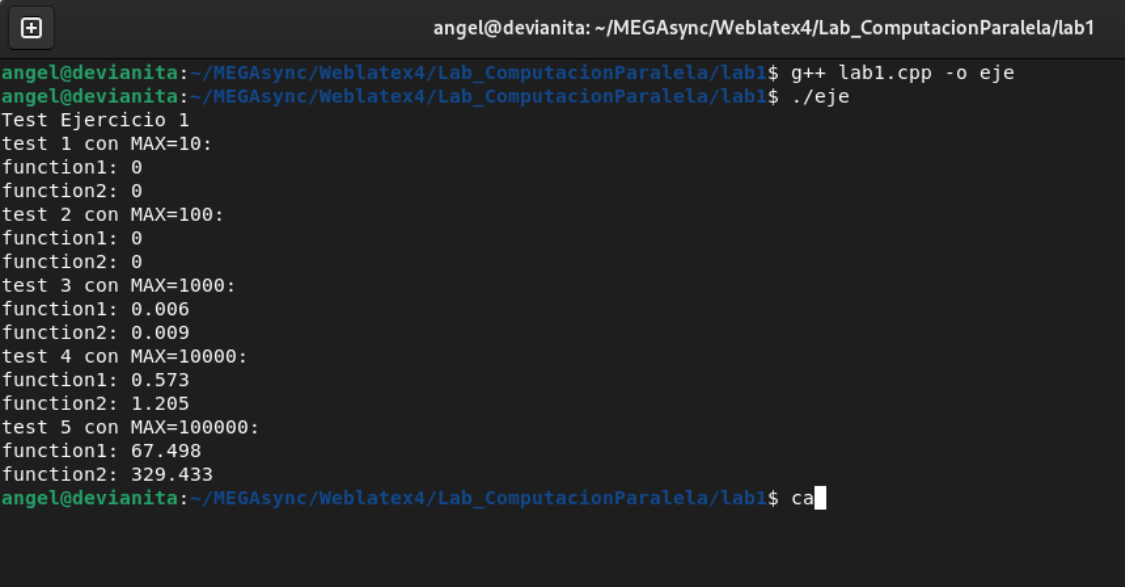
```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void asignarmemoria(double **&A,double *&x,double *&y,int MAX){
5     A=new double*[MAX];
6     x=new double[MAX];
7     y=new double[MAX];
8     for(int i=0;i<MAX;++i){
9         A[i]=new double[MAX];
10    }
11 }
12 void freememoria(double **&A,int MAX){
13     for(int i=0;i<MAX;++i){
14         delete [] A[i];
15     }
16     delete [] A;
17 }
18
19 void function1(int MAX){
20     double **A,*x,*y;
21     asignarmemoria(A,x,y,MAX);
22     int i,j;
```

```

23     auto t1 = chrono::high_resolution_clock::now();
24     for(i=0;i<MAX;i++){
25         for(j=0;j<MAX;j++){
26             y[i] += A[i][j]*x[j];
27         }
28     }
29     auto t2 = chrono::high_resolution_clock::now();
30     auto duration = chrono::duration_cast<chrono::milliseconds>(t2-
t1).count();
31     cout<<"function1: "<< duration/1000.0<<"\n";
32     freememoria(A,MAX);
33     free(x);
34     free(y);
35 }
36
37 void function2(int MAX){
38     double **A,*x,*y;
39     asignarmemoria(A,x,y,MAX);
40     int i,j;
41     auto t1 = std::chrono::high_resolution_clock::now();
42     for(j=0;j<MAX;j++){
43         for(i=0;i<MAX;i++){
44             y[i] += A[i][j]*x[j];
45         }
46     }
47     auto t2 = chrono::high_resolution_clock::now();
48     auto duration = chrono::duration_cast<chrono::milliseconds>(t2-
t1).count();
49     cout<<"function2: "<< duration/1000.0<<"\n";
50     freememoria(A,MAX);
51     free(x);
52     free(y);
53 }
54
55
56 int main(int argc, char const** argv){
57
58     int MAX = 10;
59     cout<<"Test Ejercicio 1 \n";
60     for(int i=0;i<5;++i){
61         cout<<"test "<<i+1<<" con MAX="<<MAX<<": "<<endl;
62         function1(MAX);
63         function2(MAX);
64         MAX *=10;
65     }
66     return 0;

```

4. Anexo



```
angel@devianita: ~/MEGAsync/Weblatex4/Lab_ComputacionParalela/lab1
angel@devianita:~/MEGAsync/Weblatex4/Lab_ComputacionParalela/lab1$ g++ lab1.cpp -o eje
angel@devianita:~/MEGAsync/Weblatex4/Lab_ComputacionParalela/lab1$ ./eje
Test Ejercicio 1
test 1 con MAX=10:
function1: 0
function2: 0
test 2 con MAX=100:
function1: 0
function2: 0
test 3 con MAX=1000:
function1: 0.006
function2: 0.009
test 4 con MAX=10000:
function1: 0.573
function2: 1.205
test 5 con MAX=100000:
function1: 67.498
function2: 329.433
angel@devianita:~/MEGAsync/Weblatex4/Lab_ComputacionParalela/lab1$ ca
```

Figura 1: Prueba

Repositorio: https://github.com/ubuangel/Weblatex4/tree/main/Lab_ComputacionParalela/informes

Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press.
- Pacheco, P. (2011). *An introduction to parallel programming*. Elsevier.