Pruebas sobre el Comportamiento de la Memoria Cache: BUCLES ANIDADOS

Iñigo Manuel Diez Canseco Fuentes

Marzo 2022

1. Introducción

Este documento se realizó para resolver la pregunta propuesta en el aula el cual trata de los *FOR* anidados mostrados en la sesión 2 del día Miércoles 17 de Marzo de la clase de Computación Paralela y Distribuida.

Se implementó y comparó los 2-bucles anidados presentados en clase, además de contar con un análisis explicando los resultados y por último tener un enlace a un repositorio de *Github* donde se encontrará el trabajo.

En la siguiente figura podemos observar los 2-bucles que fueron mostrados en clase.

```
double A[MAX][MAX], x[MAX], y[MAX];
. . .
/* Initialize A and x, assign y = 0 */
. . .
/* First pair of loops */
for (i = 0; i < MAX; i++)
   for (j = 0; j < MAX; j++)
      y[i] += A[i][j]*x[j];
. . .
/* Assign y = 0 */
. . .
/* Second pair of loops */
for (j = 0; j < MAX; j++)
   for (i = 0; i < MAX; i++)
      y[i] += A[i][j]*x[j];
```

Figura 1: Pseudocódigo de los "for" del libro *An Introduction to Parallel Programming*

2. Implementación

El *link* del repositorio se encontrará en el siguiente enlace[1].

En la implementación hay una variación en los bucles que es vital para el funcionamiento y la forma en el cual se comportarían y que al ejecutarse iban a tener tiempos diferentes, lo cual se tuvo que utilizar librerías de tiempo para poder saber cuál bucle era más rápido que el otro.

A continuación está la implementación en el lenguaje de programación C++:

```
#include <ctime>
#include <iomanip>
#include <chrono>
#include <cstdlib>
#include <string>
#include <iostream>

using namespace std;
using namespace std::chrono;

#define MAX 15000
double A[MAX][MAX], x[MAX], y[MAX];

void primer_bucle() {
    for (size_t i = 0; i < MAX ; i++) {
        for (size_t j = 0; j < MAX ; j++) {
            y[i] += A[i][j] * x[j];
        }
    }
}

void segundo_bucle() {
    for (size_t j = 0; j < MAX ; j++) {
        for (size_t i = 0; i < MAX ; i++) {
            y[i] += A[i][j] * x[j];
        }
    }
}

void relleno(){
    for (size_t i = 0; i < MAX; ++i) {
        for (size_t j = 0; j < MAX; ++j) {
            A[i][j] = rand() % MAX;
        }
    }
    for (size_t i = 0; i < MAX; ++i) {
        x[i] = rand() % MAX;
        y[i] = rand() % MAX;
    }
}


```

[1] https://github.com/inigomanuel/Computacion$_P aralela_y D istribuida$

```cpp
void Tiempo_Primer_Bucle() {
    std::chrono::time_point<std::chrono::high_resolution_clock>
    start, end;
    start = std::chrono::high_resolution_clock::now();
    primer_bucle();
    end = std::chrono::high_resolution_clock::now();
    int64_t duration = std::chrono::duration_cast<std::chrono::
    microseconds>(end - start).count();
    std::cout  << std::setw(30) << "Microseg: " + std::to_string(
    duration) + " us\n";
}

void Tiempo_Segundo_Bucle() {
    std::chrono::time_point<std::chrono::high_resolution_clock>
    start, end;
    start = std::chrono::high_resolution_clock::now();
    segundo_bucle();
    end = std::chrono::high_resolution_clock::now();
    int64_t duration = std::chrono::duration_cast<std::chrono::
    microseconds>(end - start).count();
    std::cout  << std::setw(30) << "Microseg: " + std::to_string(
    duration) + " us\n";
}

int main() {
    relleno();
    cout << "Primer Bucle: " << endl;
    Tiempo_Primer_Bucle();
    cout << endl;
    cout << "Segundo Bucle: " << endl;
    Tiempo_Segundo_Bucle();
    return 0;
}
```

Listing 1: Laboratorio$_1$.*cpp*

3. Resultados

En la siguiente imagen donde se encuentra la tabla de resultados, el primer bucle es más rápido debido a que se obtiene el valor avanzando por filas en una columna, en cambio, en el segundo obtiene el valor avanzando por columnas en una fila. En estos movimientos los índices toman un papel importante.

Valor de Elementos	Primer Bucle		Segundo Bucle	
	Microsegundos	Segundos	Microsegundos	Segundos
5000	58399	0.058399	145858	0.145858
10000	59597	0.059597	157669	0.157669
15000	521570	0.52157	2105226	2.105226

Figura 2: Tabla de tiempo de los resultados de la comparación en microsegundos y segundos.

Para lograr estos resultados sé utilizó entre 5000 a 15000 elementos para los arreglos y la matriz.

4. Análisis de la Ejecución

Primero hay que mencionar que nosotros al programar no podemos determinar que *data* o cuál instrucción está en la *cache*. Al tomar en cuenta esto, primero explicamos que sucede con ambos bucles, primero tenemos que el primer bucle obtiene el valor avanzando por filas en una columna, esto significa, que si estamos en A[0][0] y como este dato no está en la *cache* entonces el resultado será un *cache miss* y el sistema leerá la línea que consta en la primera fila de A, estos serian A[0][0], A[0][1], A[0][2], A[0][3] en la *cache*, debido a esto el primer bucle accederá a A[0][1], A[0][2], A[0][3], que ya se encontrarían en la *cache*.

En el segundo bucle, necesitará acceder a A[1][0], A[2][0], A[3][0], ya que primero se encuentra en A[0][0], entonces ninguno de estos está en la *cache* entonces, los próximos tres accesos de A también resultarán en *misses*. Desde que A[2][0] está en la *cache* en la línea 2, leer su línea desalojará la línea 0 y leer A[3][0] desalojará la línea 1. Después a través del bucle exterior, se necesita acceder a A[0][1] el cual ya había sido desalojado antes, lo que implica que si cada vez que se lee un elemento de A, tendremos un *miss*.

En resumen, en el segundo bucle habrá más *cache misses* y accederá más a memoria que en el primer bucle, debido a esto el primer bucle es mucho más rápido que el segundo.