Pruebas sobre el comportamiento de la memoria caché: BUCLES ANIDADOS

Patrick Xavier Márquez Choque

1. Introducción

Este trabajo tiene como objetivo la explicación de la implementación, resultados y análisis de la ejecución de la comparación de 2 bucles anidados FOR que se encuentran en el capitulo 2 del libro ”Introduction to Parallel Programming” del autor Morgan Kaufmann.

2. Implementación

El link al repositorio donde se encuentra implementado los códigos mencionados se encuentra en el siguiente enlace: `https://github.com/patrick03524/Programacion-Paralela-y-Distribuida/tree/main/Primer%20Trabajo`

En primer lugar para la implementación de estos problemas son necesarios la implementación de las Declaraciones y funciones para inicializar e imprimir los valores de la siguiente manera:

```
#include <iomanip>
using namespace std;
using namespace std::chrono;
#define MAX 4
double **A, x[MAX], y[MAX];
void initialize();
void print_();

void initialize(){
  /*  Initialize A  */
  A = new double*[MAX];
  for(int i = 0; i<MAX; ++i){
    A[i] = new double[MAX];
  }
  int numero_aleatorio;
  for(int i = 0; i<MAX; ++i){
    for(int j = 0; j<MAX; ++j){
      numero_aleatorio = 1 + rand() % (101 - 1);  ///numeros entre 1-100
      A[i][j] = numero_aleatorio;
    }
  }
  /*  Initialize x  */
  for(int i = 0; i<MAX; ++i){
```


doi:10.1017/xxxxxxx xxxx

```
    numero_aleatorio = 1 + rand() % (101 - 1);  ///numeros entre 1-100
    x[i] = numero_aleatorio;
  }
  /*  Assign y = 0  */
  for(int i = 0; i<MAX; ++i){
    y[i] = 0;
  }
}

void print_(){
  cout<<"MATRIZ A"<<endl;
  for(int i = 0; i<MAX; ++i){
    for(int j = 0; j<MAX; ++j){
      cout<<A[i][j]<<" ";
    }
    cout<<endl;
  }
  cout<<endl;
  cout<<"ARRAY X"<<endl;
  for(int i = 0; i<MAX; ++i){
    cout<<x[i]<<" ";
  }
  cout<<endl;
  cout<<"ARRAY Y"<<endl;
  for(int i = 0; i<MAX; ++i){
    cout<<y[i]<<endl;
  }
  cout<<endl;
}
```

Listing 1: Implementación de las funciones necesarias

La implementación de los bubles anidados mostrados en el libros son lo siguiente:

```
int main(int argc, char *argv[]) {
  srand (time(NULL));
  initialize();
  /*  First pair of loops */
  for(int i = 0; i<MAX; ++i){
    for(int j = 0; j<MAX; ++j){
      y[i] += A[i][j] * x[j];
    }
  }
  /*  Second pair of loops  */
  for(int j = 0; j<MAX; ++j){
    for(int i = 0; i<MAX; ++i){
      y[i] += A[i][j] * x[j];
    }
  }
  /*  Print */
  print_();
  return 0;
}
```

Listing 2: Bucles Anidados FOR

La implementación del anterior código fue realizada en C++ según el libro con la utilización de un doble puntero para la utilización de una matriz y arrays globables para el acceso a funciones sin ningún problema. Para la experimentación de una cantidad considerable de elementos (aproximadamente hasta un MAX de 10 000) se utilizó la instrucción de new para la reserva de memoria en el Heap y posteriormente un llenado de numeros aleatorios entre el rango de 1 a 100 para los elementos tanto de la matriz A, y los arreglos x & y.

Además se utilizó una librería llamada chrono para la evaluación de los tiempos que se analizaran en el apartado de resultados.

El link al repositorio donde se encuentra implementado los códigos mencionados se encuentra en el siguiente enlace: `https://github.com/patrick03524/Programacion-Paralela-y-Distribuida/tree/main/Primer%20Trabajo`

3. Resultados

Se hicieron varias ejecuciones de los programas de cada uno de los bucles FOR anidados y se realizó un cuadro comparativo mostrando la cantidad del elementos que utilizaban como tamaño de los arreglos que corresponde a la variable MAX que es el siguiente:

Loops	Tamaño del Array	Tiempo promedio	Resultados en segundos									
Primer Loop	100	0	0	0	0	0	0	0	0	0	0	0
	1000	0.0055063	0.004986	0.009973	0.005984	0.00798	0.004988	0.004987	0.00299	0.004986	0.00399	0.004199
	10000	0.4117827	0.434868	0.356047	0.496209	0.306181	0.323135	0.413893	0.53903	0.378823	0.43461	0.435031
Segundo Loop	100	0	0	0	0	0	0	0	0	0	0	0
	1000	0.0229594	0.01496	0.013963	0.013963	0.015965	0.09296	0.02193	0.013963	0.017953	0.015959	0.007978
	1000	12.777716	10.805747	8.166066	11.778977	13.959107	10.877607	16.451411	10.24004	14.860978	15.197362	15.439868

Figura 1: Cuadro comparativo entre los resultados de la ejecución del programa

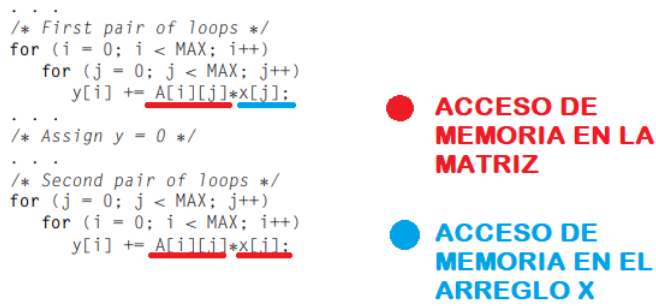
Se realizaron alrededor de 10 ejecuciones por cada loop y por cada tamaño utilizando desde 100, 1000 y 10 000 elementos para nuestros arreglos y para la matriz A como se puede apreciar en la Figura 1.

En primer lugar lo que se puede concluir cuando se utiliza un tamaño MAX de 100 elementos es que los tiempos que retorna la librería chronos es que los tiempos que retorna son 0, eos significa que estos tiempos son tan pequeños que son despreciables para nuestro experimento y no se puede concluir nada concreto de los mismos.

A comparación cuando se utilizan una gran cantidad de elementos entonces es donde se puede concluir que **El Primer Loop del libro es mucho más rapido en cuestión de tiempos a comparación del Segundo Loop** ya que a comparación del primer loop que tiene un tiempo promedio de 0.5 segundos a comparación del segundo que tiene un tiempo promedio de 12 segundos.

4. Análisis de la Ejecución

La interpretación para que el primer bucle sea mucho más rápido a comparación se debe a la característica con la que se analiza y realiza cada uno de los bucles anidados.



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]

Figura 2: Acceso de Memoria en la Matriz A y en el Arreglo x

Podemos observar mediante el código que la principal diferencia es que el primer bucle no sufre de una gran cantidad de cache misses en cuestión cuando se quiere hacer las consultas en la matriz, ya que según este orden lo que hace el computador es que guardan en memoria cache cada una de las filas de la matriz, y cuando ingrese al segundo bucle es donde se aprovecha mucho más la cache para guardar la memoria continua de cada uno de las filas de nuestra matriz y esa es la razón primordial de porque es mucho más rápido a comparación del segundo como se puede apreciar en la Figura 2.

En el segundo bucle FOR anidado se recorre dependiendo de los índices j y es cuando se puede apreciar que ocurren muchos más cache misses porque cuando se realiza el primer acceso del primer elemento del primer arreglo de la matriz eso se guarda en caché, entonces cuando se continua para el segundo acceso es donde ocurre este fenómeno, ya que tiene que acceder a la segunda fila de la matriz así sucesivamente; es aquí donde se puede apreciar el cache miss antes mencionado ya que al cambiar de filas también la instrucción de acceso que se realiza con la operación de doble corchete es también costosa y debe considerarse como un gran factor para que aumente mucho el tiempo ya que esto se realiza un gran cantidad de veces de

manera exponencial y en ejemplo cuando se realiza utilizando un número máximo de 10 000 elementos entonces esta operación se realiza $10000 * 10000$ número de veces

También se tiene que considerar el problema con el acceso específico con el arreglo x, ay que esto en el primer ejemplo representa un acceso a memoria adicional en cada bucle del FOR anidado y por lo tanto una cache miss en caso que se quiera guardar el dato del arreglo x, esto por ejemplo es mucho más lento a comparación del segundo ejemplo ya que el mismo al no cambiar de índice i entonces puede guardar en la cache cada valor del arreglo x y por lo tanto requerir menos tiempo pero esta operación de corchetes simples para acceder a los elementos de nuestro arreglo x en comparación del acceso a la matriz A es mucho más lento y es la razón de la cual se tienen los resultados con una gran margen de comparación.