1. Objetivos

- Entender la forma en que se almacenan los arreglos en memoria.
- Comprender técnicas de optimización de caché.

2. Conceptos teóricos

2.1. SAXPY "Single-Precision A X Plus Y"

Es una función en la biblioteca estándar de Basic Linear Algebra Subroutines (BLAS). Dicha operación es realizada en un tiempo de reloj.

SAXPY es una combinación de una multiplicación escalar y una suma de vectores, y es muy simple:

Toma como entrada dos vectores, comúnmente float (32 bits), ***x[]*** y ***y[]*** con ***n*** elementos cada uno, y un valor escalar ***A***.

Multiplica cada elemento ***x[i]*** por ***a*** y agrega el resultado en ***y[i]***. Una implementación simple en C se muestra en la figura 1.

```
1
2  void saxpy(int n, float a, float * x, float * y)
3  {
4      for (int i = 0; i < n; ++i)
5          y[i] = a * x[i] + y[i];
6  }
```

Figura 1

El ejemplo más común de la utilización de esta función se encuentra en la multiplicación estándar de matrices, donde es necesario la multiplicación del renglón de la primera por la columna de la segunda, más el resultado almacenado en la matriz resultado.

```
1  for (i = 0; i < n; ++i)
2  {
3    for (j = 0; j < n; ++j)
4    {
5      for (k = 0; k < n; ++k)
6      {
7        matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
8      }
9    }
10 }
```

Donde i es la variable que itera los renglones de la matriz A.
j es la variable que itera las columnas de la matriz B.
k la variable temporal que itera elemento a elemento, de ya sean, los renglones o las columnas.

2.2. Orden de almacenamiento

Los arreglos multidimensionales (matrices) son omnipresentes en el cómputo científico. El acceso a datos es un tema crucial en este campo, ya que el mapeo entre elementos debe coincidir con el orden en que el código carga y almacena datos en la computadora para que la localidad en tiempo y espacio pueda ser empleada.

El acceso estricto (**Strided access**) a una matriz unidimensional reduce el espacio en memoria, lo que lleva a una baja utilización del ancho de banda disponible y así, finalmente, a la optimización de recursos. Dependiendo del lenguaje de programación se utiliza un orden de recorrido de los vectores en el caché.

Esquema de almacenamiento de matriz de orden-fila (**Row major order**), utilizado por el lenguaje de programación C. Las filas de la matriz se almacenan consecutivamente en la memoria.

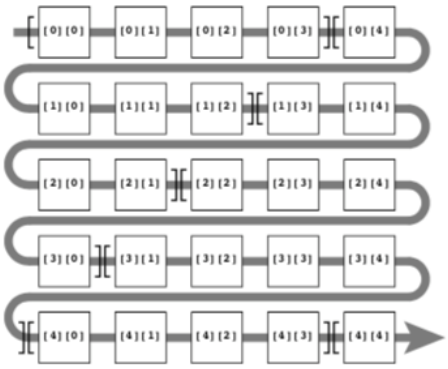

Figura 2

En cambio, en el lenguaje de programación Fortran (figura 3), se rige por el esquema de almacenamiento de matriz de orden-columna (**Column major order**), donde las líneas de caché almacenan las columnas consecutivamente.

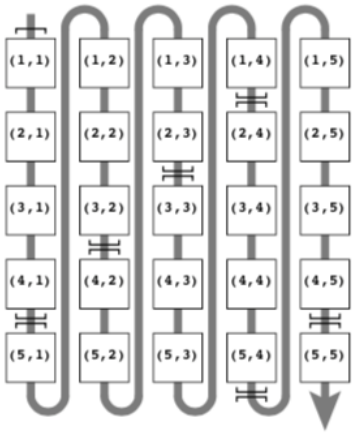

Figura 3

3. Realización de la práctica

- Crear un programa en C que multiplique 2 matrices cuadradas y el resultado lo almacene en una tercera matriz.
- Posteriormente variar el orden de los bucles *for* en sus 6 combinaciones para cambiar la forma de acceso a la información de la estructura de datos.
- Cronometrar el tiempo que tarda cada una en resolverse.
- Ejecutar al menos 3 diferentes sistemas de matrices (100*100),(500*500),(1000*1000).
- Compara los resultados y explica cuál disposición de ciclos *for* se ejecuta más rápido y cuál más lento. ¿Por qué ocurre esto?
- Compilar y ejecutar los programas con tres compiladores diferentes (gcc, pgcc, e icc). ¿Existe alguna diferencia en el tiempo de ejecución?
- Para el punto anterior, también mide el tiempo de llenado de las matrices.

Nota: No olvides incluir la impresión de pantalla de la ejecución del programa.

3.1. Código de apoyo

Te puedes apoyar en el siguiente código:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

void get_walltime(double* wcTime){
  struct timeval tp;
  gettimeofday(&tp,NULL);
  *wcTime=(tp.tv_sec + tp.tv_usec/1000000.0);
}

int main(int argc, char* argv[]){
  int i,j,n=3;
  int **matrizA;
  double S1,E2;

  //Inicializando matrices
  matrizA = (int **)malloc(n*sizeof(int *));
  for(i=0;i<n;i++)
      *(matrizA+i) = (int *)malloc(n*sizeof(int));

  //Llenando matrices
  for (i = 0; i < n; ++i)
  {
    for (j = 0; j < n; ++j)
    {
      matrizA[i][j] = rand() % 6;
```

```
27        }
28      }
29
30      get_walltime(&S1);
31
32      for (i = 0; i < n; ++i)
33      {
34        for (j = 0; j < n; ++j)
35        {
36          for (k = 0; k < n; ++k)
37          {
38            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
39          }
40        }
41      }
42      get_walltime(&E1);
43
44      printf("Tiempo método ijk: %f s\n",(E1-S1));
45      return 0;
46  }
```

Recuerda que la compilación y ejecución desde terminal con gcc es de la forma:

```
1  # gcc -o ejecutable codigo_fuente.c
2  # ./ejecutable
```