

Examen 1. SAXPY.

Argüello León Dante Moisés
dante.argleon@gmail.com

Castelan Ramos Carlos
carlosscastelan@gmail.com

Robledo Aguirre Eduardo
robledoaguirre98@gmail.com

Ingeniería en computación,
FI, UNAM
CDMX, México

I. Introducción.

En el documento de partida inicial se menciona que SAXPY (Single-Precision A X Plus Y) es una función utilizada en la biblioteca estándar de BLAS (Basic Linear Algebra Subroutines). SAXPY representa una operación matemática fundamental que involucra la multiplicación de un número escalar (S) por un vector (X) y la suma del resultado al vector Y. Esta operación es especialmente relevante en la programación paralela y la optimización de cálculos numéricos, ya que puede ser altamente eficiente cuando se implementa en hardware especializado o en GPU's.

Desempeñando así un papel crucial en una amplia gama de aplicaciones, desde simulaciones científicas hasta el procesamiento de señales, y ha contribuido significativamente al avance de la computación de alto rendimiento y la capacidad de resolver problemas complejos en la ciencia y la ingeniería.

II. Ejecución del Código en S.O. Windows 10.

A. Crear un programa en C que multiplique 2 matrices cuadradas y el resultado lo almacene en una tercera matriz.

A partir del código base proporcionado en el documento de partida inicial, generamos modificaciones necesarias para el cumplimiento de la funcionalidad requerida, tales fueron la declaración de las matrices A, B y C así como su inicialización en el programa, incluyendo también el llenado de las dos primeras

El resultado de las modificaciones anteriores es el siguiente:

```
#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;
    int **matrizB;
    int **matrizC;
    double S1, E1;

    //Iniciando matrices
    matrizA = (int **)malloc(n*sizeof( int *));
```

```
matrizB = (int **)malloc(n*sizeof( int *));
matrizC = (int **)malloc(n*sizeof( int *));
for(i=0;i<n;i++){
    *(matrizA+i) = (int *)malloc(n*sizeof(int));
    *(matrizB+i) = (int *)malloc(n*sizeof(int));
    *(matrizC+i) = (int *)malloc(n*sizeof(int));
}

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

get_walltime(&S1);

for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

get_walltime(&E1);

printf("Tiempo metodo ijk: %f s\n", (E1-S1));
return 0;
}
```

Figura 1. Código base modificado para matrices A, B y C.

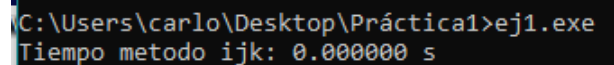


Figura 2. Ejecución del código base de la multiplicación de matrices de 3 por 3.

B. Matrices de 100 * 100.

La modificación residió en el cambio del valor de la variable n, asignando el valor de 100.

```
int main(int argc, char* argv[]){
    int i, j, n=100;
    int **matrizA;
    int **matrizB;
    int **matrizC;
    double S1, E1;
```

Figura 3. Modificación del valor de n para matrices cuadradas de 100 x 100.

A partir de la modificación anterior, el valor de n definirá la inicialización de las matrices de 100 x 100 en los ciclos for con el método ijk para la obtención de la matriz C.

```

for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 4. Método ijk de matriz 100x100 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo ijk: 0.003471 s
-----
Process exited after 0.05582 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 5. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método ikj en matrices de 100x100.

```

for (int i = 0; i < n; ++i)
{
    for (int k = 0; k < n; ++k)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 4. Método ijk de matriz 100x100 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo ikj: 0.002976 s
-----
Process exited after 0.0185 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 7. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método jik en matrices de 100x100.

```

for (int j = 0; j < n; ++j)
{
    for (int i = 0; i < n; ++i)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 8. Método jik de matriz 100x100 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo jik: 0.002976 s
-----
Process exited after 0.01774 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 9. Tiempo de ejecución de obtención de matriz C con en el método jik.

Para el método jki en matrices de 100x100.

```

for (int j = 0; j < n; ++j)
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 10. Método jki de matriz 100x100 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo jki: 0.003473 s
-----
Process exited after 0.01756 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 11. Tiempo de ejecución de obtención de matriz C con en el método jki.

Para el método kij en matrices de 100x100.

```

for (int k = 0; k < n; ++k)
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 12. Método kij de matriz 100x100 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo kij: 0.002976 s
-----
Process exited after 0.01724 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 13. Tiempo de ejecución de obtención de matriz C con en el método kij.

Finalmente tenemos el método kji en matrices de 100x100.

```

for (int k = 0; k < n; ++k)
{
    for (int j = 0; j < n; ++j)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 14. Método kji de matriz 100x100 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo kji: 0.003473 s

-----
Process exited after 0.01836 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 15. Tiempo de ejecución de obtención de matriz C con en el método kji.

Mientras tanto el llenado de matrices de 100x100:

Para A

```

Tiempo de llenado Matriz A: 0.000000 s
Tiempo de llenado Matriz B: 0.000496 s

-----
Process exited after 0.01485 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 16 y 17. Tiempo de ejecución de llenado de matrices A y B.

C. Matrices de 500 * 500

La modificación residió en el cambio del valor de la variable n, asignando el valor de 500.

```

int main(int argc, char* argv){
    int i, j, n=500;
    int **matrizA;
    int **matrizB;
    int **matrizC;
    double S1, E1;

```

Figura 18. Modificación del valor de n para matrices cuadradas de 500 x 500.

A partir de la modificación anterior, el valor de n definirá la inicialización de las matrices de 500 x 500 en los ciclos for con el método ijk para la obtención de la matriz C.

```

for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 19. Método ijk de matriz 500x500 para la obtención de C.

```

Tiempo metodo ijk: 0.473185 s

-----
Process exited after 0.4935 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 20. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método ikj en matrices de 500x500.

```

for (int i = 0; i < n; ++i)
{
    for (int k = 0; k < n; ++k)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 21. Método ikj de matriz 500x500 para la obtención de C.

```

Tiempo metodo ikj: 0.382911 s

-----
Process exited after 0.4409 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 22. Tiempo de ejecución de obtención de matriz C con en el método.

Para el método jik en matrices de 500x500.

```

for (int j = 0; j < n; ++j)
{
    for (int i = 0; i < n; ++i)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 23. Método jik de matriz 500x500 para la obtención de C.

```

Tiempo metodo jik: 0.452849 s

-----
Process exited after 0.4727 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 24. Tiempo de ejecución de obtención de matriz C con en el método jik.

Para el método jki en matrices de 500x500.

```

for (int j = 0; j < n; ++j)
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 25. Método jki de matriz 500x500 para la obtención de C.

```
Tiempo metodo jki: 0.637856 s
-----
Process exited after 0.6568 seconds with return value 0
Presione una tecla para continuar . . . _
```

Figura 26. Tiempo de ejecución de obtención de matriz C con en el método jki.

Para el método kij en matrices de 500x500.

```
for (int k = 0; k < n; ++k)
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 27. Método kij de matriz 500x500 para la obtención de C.

```
Tiempo metodo kij: 0.374485 s
-----
Process exited after 0.3948 seconds with return value 0
Presione una tecla para continuar . . . _
```

Figura 28. Tiempo de ejecución de obtención de matriz C con en el método kij.

Finalmente tenemos el método kji en en matrices de 500x500.

```
for (int k = 0; k < n; ++k)
{
    for (int j = 0; j < n; ++j)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 29. Método kji de matriz 500x500 para la obtención de C.

```
Tiempo metodo kji: 0.637888 s
-----
Process exited after 0.6585 seconds with return value 0
Presione una tecla para continuar . . . _
```

Figura 30. Tiempo de ejecución de obtención de matriz C con en el método kji.

Mientras tanto el llenado de matrices de 500x500:

Para A

```
Tiempo de llenado Matriz A: 0.002480 s
Tiempo de llenado Matriz B: 0.001984 s
-----
Process exited after 0.0186 seconds with return value 0
Presione una tecla para continuar . . . _
```

Figura 31 y 32. Tiempo de ejecución de llenado de matrices A y B.

D. Matrices de 1000 * 1000.

La modificación residió en el cambio del valor de la variable n, asignando el valor de 1000.

```
int main(int argc, char* argv[]){
    int i, j, n=1000;
    int **matrizA;
    int **matrizB;
    int **matrizC;
    double s1, e1;
```

Figura 33. Modificación del valor de n para matrices cuadradas de 1000 x 1000.

A partir de la modificación anterior, el valor de n definirá la inicialización de las matrices de 1000 x 1000 en los ciclos for con el método ijk para la obtención de la matriz C.

```
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 34. Método ijk de matriz 1000x1000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo ijk: 5.854783 s
-----
Process exited after 5.928 seconds with return value 0
Presione una tecla para continuar . . . _
```

Figura 35. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método ikj en matrices de 1000x1000.

```
for (int i = 0; i < n; ++i)
{
    for (int k = 0; k < n; ++k)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 36. Método ikj de matriz 1000x1000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo ikj: 3.073711 s
-----
Process exited after 3.11 seconds with return value 0
Presione una tecla para continuar . . . _
```

Figura 37. Tiempo de ejecución de obtención de matriz C con en el método ikj.

Para el método jik en matrices de 1000x1000.

```
for (int j = 0; j < n; ++j)
{
    for (int i = 0; i < n; ++i)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 38. Método jik de matriz 1000x1000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo jik: 5.815185 s
-----
Process exited after 5.889 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 39. Tiempo de ejecución de obtención de matriz C con en el método jik.

Para el método jki en matrices de 1000x1000.

```
for (int j = 0; j < n; ++j)
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 40. Método jki de matriz 1000x1000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo jki: 7.664191 s
-----
Process exited after 7.7 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 41. Tiempo de ejecución de obtención de matriz C con en el método jki.

Para el método kij en matrices de 1000x1000.

```
for (int k = 0; k < n; ++k)
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 42. Método kij de matriz 1000x1000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo kij: 3.001295 s
-----
Process exited after 3.037 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 43. Tiempo de ejecución de obtención de matriz C con en el método kij.

Finalmente tenemos el método kji en matrices de 1000x1000.

```
for (int k = 0; k < n; ++k)
{
    for (int j = 0; j < n; ++j)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 44. Método kji de matriz 1000x1000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo kji: 7.563008 s
-----
Process exited after 7.639 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 45. Tiempo de ejecución de obtención de matriz C con en el método kji.

Mientras tanto el llenado de matrices de 1000x1000:

Para A

```
Tiempo de llenado Matriz A: 0.009423 s
Tiempo de llenado Matriz B: 0.008928 s
-----
Process exited after 0.07348 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 46 y 47. Tiempo de ejecución de llenado de matrices A y B.

E. Matrices de 5000 * 5000.

La modificación residió en el cambio del valor de la variable n, asignando el valor de 5000.

```
int main(int argc, char* argv){
    int i, j, n=5000;
    int **matrizA;
    int **matrizB;
    int **matrizC;
    double S1, E1;
```

Figura 48. Modificación del valor de n para matrices cuadradas de 5000 x 5000.

A partir de la modificación anterior, el valor de n definirá la inicialización de las matrices de 5000 x 5000 en los ciclos for con el método ijk para la obtención de la matriz C.


```

for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 49. Método ijk de matriz 5000x5000 para la obtención de C.

Tomamos así su tiempo de ejecución tenemos que.

```

Tiempo metodo ijk: 1074.732802 s
-----
Process exited after 1076 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 50. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método ikj en matrices de 5000x5000.

```

for (int i = 0; i < n; ++i)
{
    for (int k = 0; k < n; ++k)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 51. Método ijk de matriz 5000x5000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo ikj: 376.474634 s
-----
Process exited after 377 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 52. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método jik en matrices de 5000x5000.

```

for (int j = 0; j < n; ++j)
{
    for (int i = 0; i < n; ++i)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 53. Método jik de matriz 5000x5000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo jik: 862.550300 s
-----
Process exited after 863.5 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 54. Tiempo de ejecución de obtención de matriz C con en el método jik.

Para el método jki en matrices de 5000x5000.

```

for (int j = 0; j < n; ++j)
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 55. Método jki de matriz 5000x5000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo jki: 1270.893193 s
-----
Process exited after 1272 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 56. Tiempo de ejecución de obtención de matriz C con en el método jki.

Para el método kij en matrices de 5000x5000.

```

for (int k = 0; k < n; ++k)
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 57. Método kij de matriz 5000x5000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo kij: 378.144917 s
-----
Process exited after 379.2 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 58. Tiempo de ejecución de obtención de matriz C con en el método kij.

Finalmente tenemos el método kji en matrices de 5000x5000.

```

for (int k = 0; k < n; ++k)
{
    for (int j = 0; j < n; ++j)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 59. Método kji de matriz 5000x5000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo kji: 1114.804541 s
-----
Process exited after 1115 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 60. Tiempo de ejecución de obtención de matriz C con en el método kji.

Mientras tanto el llenado de matrices de 5000x5000:

Para A

```

Tiempo de llenado Matriz A: 0.244033 s
Tiempo de llenado Matriz B: 0.243536 s
-----
Process exited after 0.5531 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 61 y 62. Tiempo de ejecución de llenado de matrices A y B.

F. Matrices de 10,000 * 10,000.

La modificación residió en el cambio del valor de la variable n, asignando el valor de 10000.

```

int main(int argc, char* argv[]){
    int i, j, n=10000;
    int **matrizA;
    int **matrizB;
    int **matrizC;
    double S1, E1;
}

```

Figura 63. Modificación del valor de n para matrices cuadradas de 10000 x 10000.

A partir de la modificación anterior, el valor de n definirá la inicialización de las matrices de 10000 x 10000 en los ciclos for con el método ijk para la obtención de la matriz C.

```

for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 64. Método ijk de matriz 10000x10000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo ijk: 5102.433769
-----
Process exited after 5013 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 65. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método ikj en matrices de 10000x10000.

```

for (int i = 0; i < n; ++i)
{
    for (int k = 0; k < n; ++k)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 66. Método ijk de matriz 10000x10000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo ikj: 4010.655847 s
-----
Process exited after 4013 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 67. Tiempo de ejecución de obtención de matriz C con en el método ijk.

Para el método jik en matrices de 10000x10000.

```

for (int j = 0; j < n; ++j)
{
    for (int i = 0; i < n; ++i)
    {
        for (int k = 0; k < n; ++k)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}

```

Figura 68. Método jik de matriz 10000x10000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```

Tiempo metodo jik: 5430.108234
-----
Process exited after 5431 seconds with return value 0
Presione una tecla para continuar . . .

```

Figura 69. Tiempo de ejecución de obtención de matriz C con en el método jik.

Para el método jki en matrices de 10000x10000.

```
for (int j = 0; j < n; ++j)
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 70. Método jki de matriz 10000x10000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo jki: 6053.448390
-----
Process exited after 6055 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 71. Tiempo de ejecución de obtención de matriz C con en el método jki.

Para el método kij en matrices de 10000x10000.

```
for (int k = 0; k < n; ++k)
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 72. Método kij de matriz 10000x10000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo kij: 3876.973443
-----
Process exited after 3878 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 73. Tiempo de ejecución de obtención de matriz C con en el método kij.

Finalmente tenemos el método kji en matrices de 10000x10000.

```
for (int k = 0; k < n; ++k)
{
    for (int j = 0; j < n; ++j)
    {
        for (int i = 0; i < n; ++i)
        {
            matrizC[i][j] += matrizA[i][k] * matrizB[k][j];
        }
    }
}
```

Figura 74. Método kji de matriz 10000x10000 para la obtención de C.

Tomando así su tiempo de ejecución tenemos que.

```
Tiempo metodo kji: 5215.746188
-----
Process exited after 5216 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 75. Tiempo de ejecución de obtención de matriz C con en el método kji.

Mientras tanto el llenado de matrices de 10000x10000:

Para A

```
Tiempo de llenado Matriz A: 0.992992 s
Tiempo de llenado Matriz B: 0.984560 s
-----
Process exited after 2.143 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 76 y 77. Tiempo de ejecución de llenado de matrices A y B.

G. Compara los resultados y explica cuál disposición de ciclos for se ejecuta más rápido y cuál más lento. ¿Por qué ocurre esto?

Se muestran las características de la computadora donde fueron ejecutadas las pruebas:

- Procesador: i7 10^oGen
- RAM: 16 GB
- GPU: RTX 2060
- SSD (NVMe): 1 Tb

A continuación, se presentan en forma de tabla los tiempos obtenidos.

TABLA 1: Resultados de las pruebas.

Tamaño de Matrices	Método	Tiempo [s]
100 x 100	IJK	0.003471
	IKJ	0.002976
	JIK	0.002976
	JKI	0.003473
	KIJ	0.002976
	KJI	0.003473
500 x 500	IJK	0.473185
	IKJ	0.382911
	JIK	0.452849
	JKI	0.637856
	KIJ	0.374485
	KJI	0.637888
1000 x 1000	IJK	5.854783
	IKJ	3.073711
	JIK	5.815185
	JKI	7.664191
	KIJ	3.001295
	KJI	7.563008

5000 x 5000	IJK	1074.7328
	IKJ	376.474634
	JKI	862.5503
	KJI	1270.89319
	KIJ	378.144917
	KJI	1114.80454
10000 x 10000	IJK	5102.43377
	IKJ	4010.65585
	JKI	5430.10823
	KJI	6053.44839
	KIJ	3876.97344
	KJI	5215.74619

Y de manera gráfica con base en la tabla anterior.

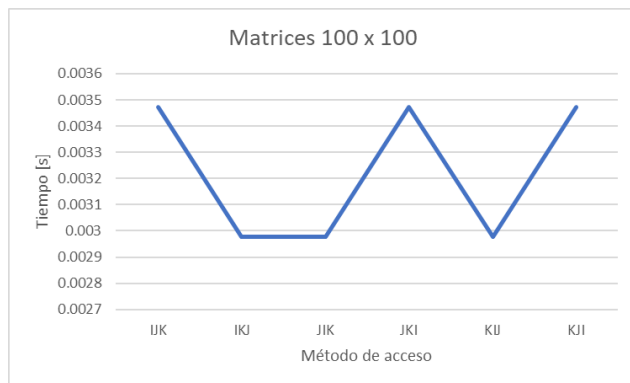


Figura 78. Gráfica de tiempo vs. método en 100 x 100

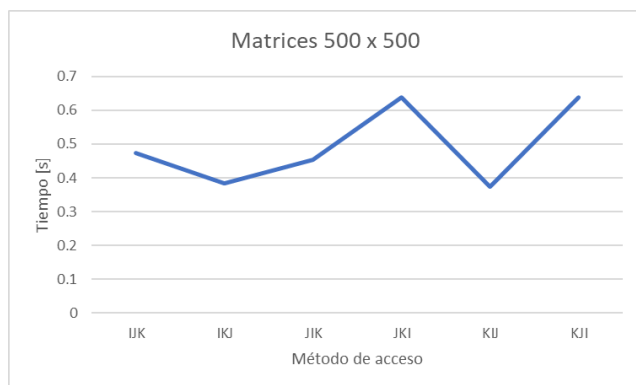


Figura 79. Gráfica de tiempo vs. método en 500 x 500

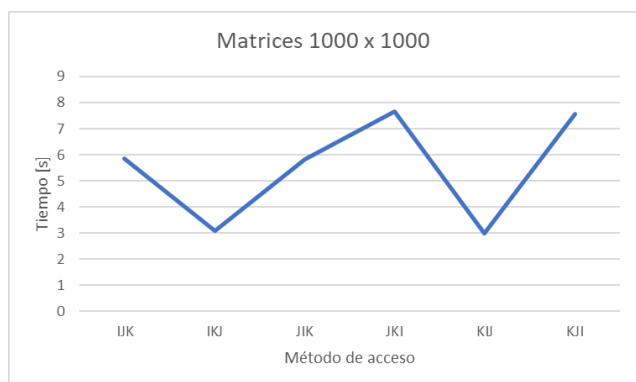


Figura 80. Gráfica de tiempo vs. método en 1000 x 1000

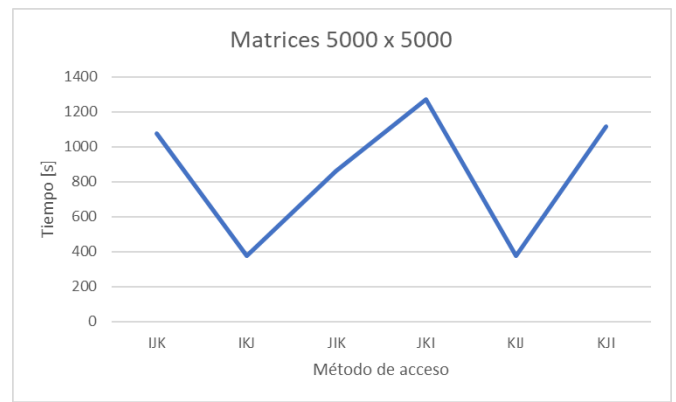


Figura 81. Gráfica de tiempo vs. método en 5000 x 5000

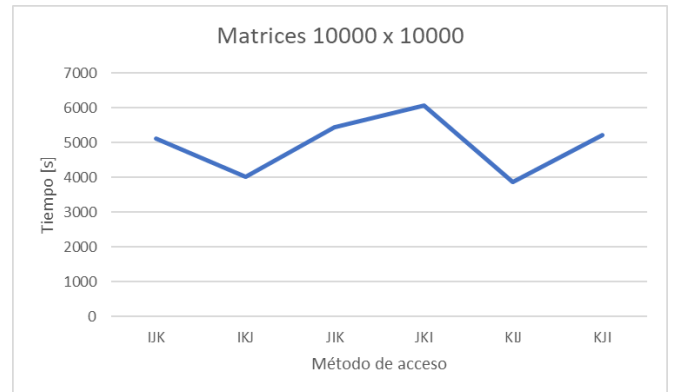


Figura 82. Gráfica de tiempo vs. método en 10000 x 10000

Se observa que, para todos los tamaños de matrices, el método de acceso más lento es JKI porque el bucle más externo da prioridad a las columnas de la matriz resultante, luego a la iteración y al final a las filas, por lo que en cada instante estaremos accediendo a elementos de diferentes filas de las matrices de entrada, generando un patrón de recorrido no contiguo en la memoria y aumentando el tiempo de acceso a la memoria principal. Mientras que el método de acceso más rápido es KIJ debido a que mantiene una naturaleza más acorde al esquema de almacenamiento *Row major order*, ya que el bucle más externo K garantiza una iteración consistente a través de las matrices, luego con I a través de las filas de una manera contigua para obtener de manera individual los valores de las columnas con J y si bien aplica a las matrices de entrada beneficia también al acceso de la resultante.

H. Conclusiones.

La elección del orden de acceso a la memoria es importante en el contexto del cómputo de alto rendimiento debido a su impacto en la optimización del tiempo. Pues muchas aplicaciones se basan en instrucciones que operan en un conjunto de datos extensos, y aunque el orden de acceso a memoria puede no parecer importante cuando tenemos pocos datos, observamos que en las pruebas de matrices de 10000 x 10000 puede existir una diferencia muy grande en términos de tiempo. Conocer la arquitectura de la memoria y cómo es que se almacenan los datos permitirá lograr una mayor eficiencia en nuestras implementaciones de algoritmos que requieran de acceso a memoria.

Enlace del Repositorio

<https://github.com/CarlosCR07/Sistemas-Distribuidos.git>

Referencias

[1]Foulser, D. E., & Schreiber, R. (1987). The Saxpy Matrix-1: A general-purpose systolic computer. *Computer*, 20(07), 35-43.

[2] IBM. (2021, marzo 03). *P SAXPY, DAXPY, CAXPY, and ZAXPY (Multiply a Vector X by a Scalar, Add to a Vector Y, and Store in the Vector Y)*. [Online]. Available: <https://www.ibm.com/docs/en/essl/6.2?topic=vss-saxpy-daxpy-caxpy-zaxpy-multiply-vector-by-scalar-add-vector-store-in-vector>

[3]Tabik, S., Ortega, G., & Garzón, E. M. (2014). Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study. *The Journal of Supercomputing*, 70(2), 577-587.

[4] National Laboratory for High Performance Computing Chile. (2023). *Ejemplos de programación: Seis formas de implementar SAXPY en GPU*. [Online]. Available: https://www.nlhpc.cl/wp-content/uploads/2012/09/4_saxpy.pdf