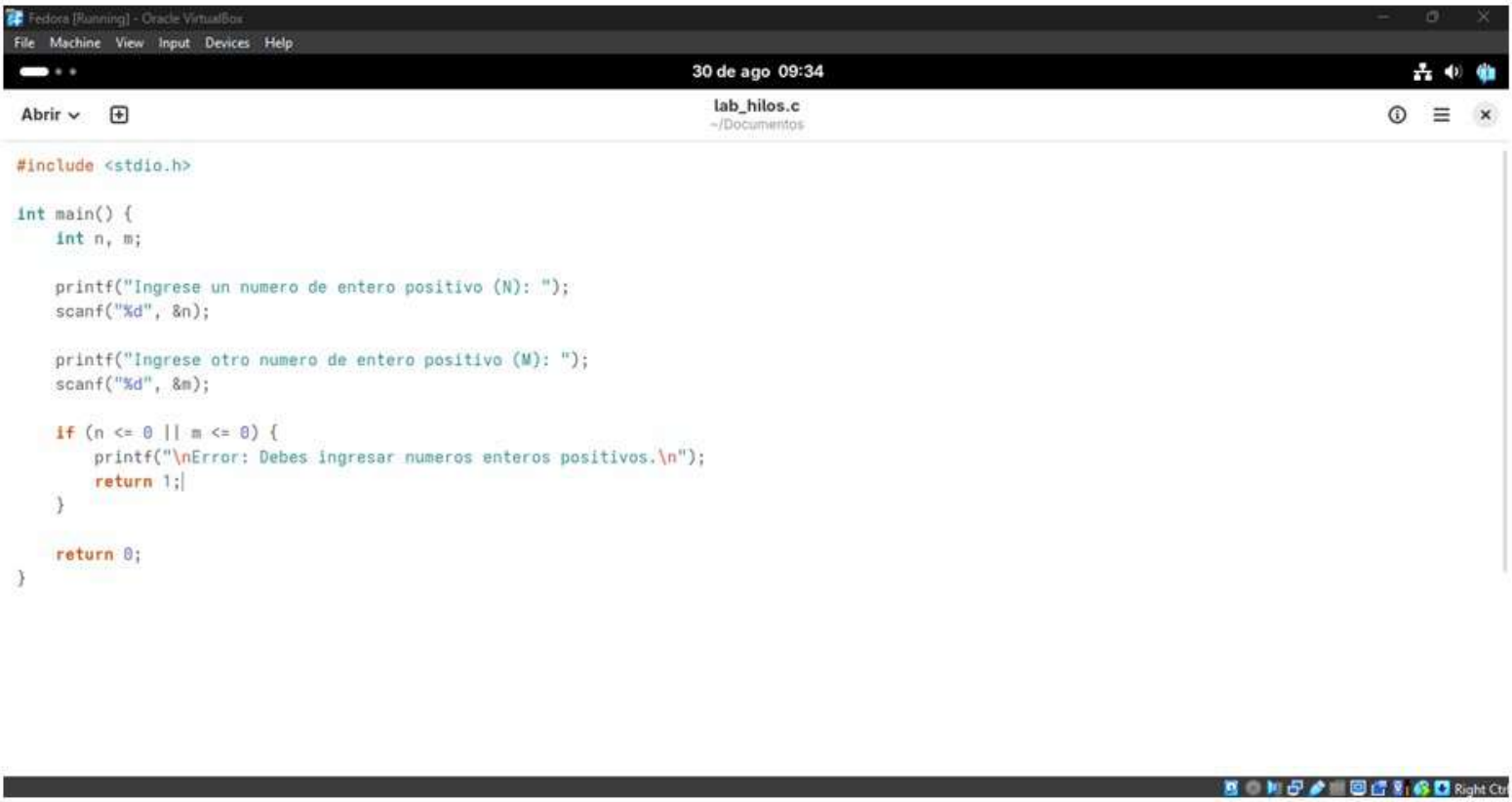
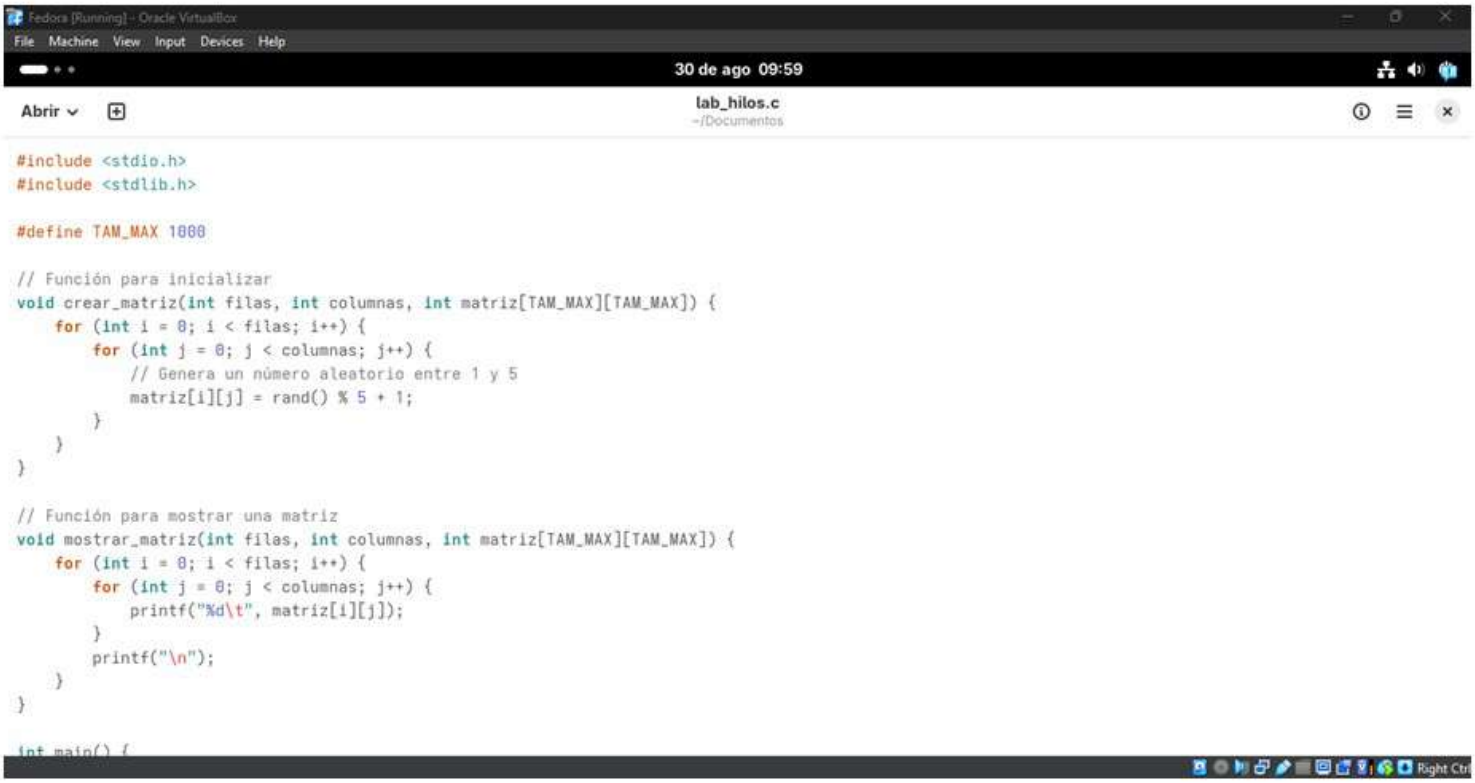
Hice un programa en C con las siguientes funciones:

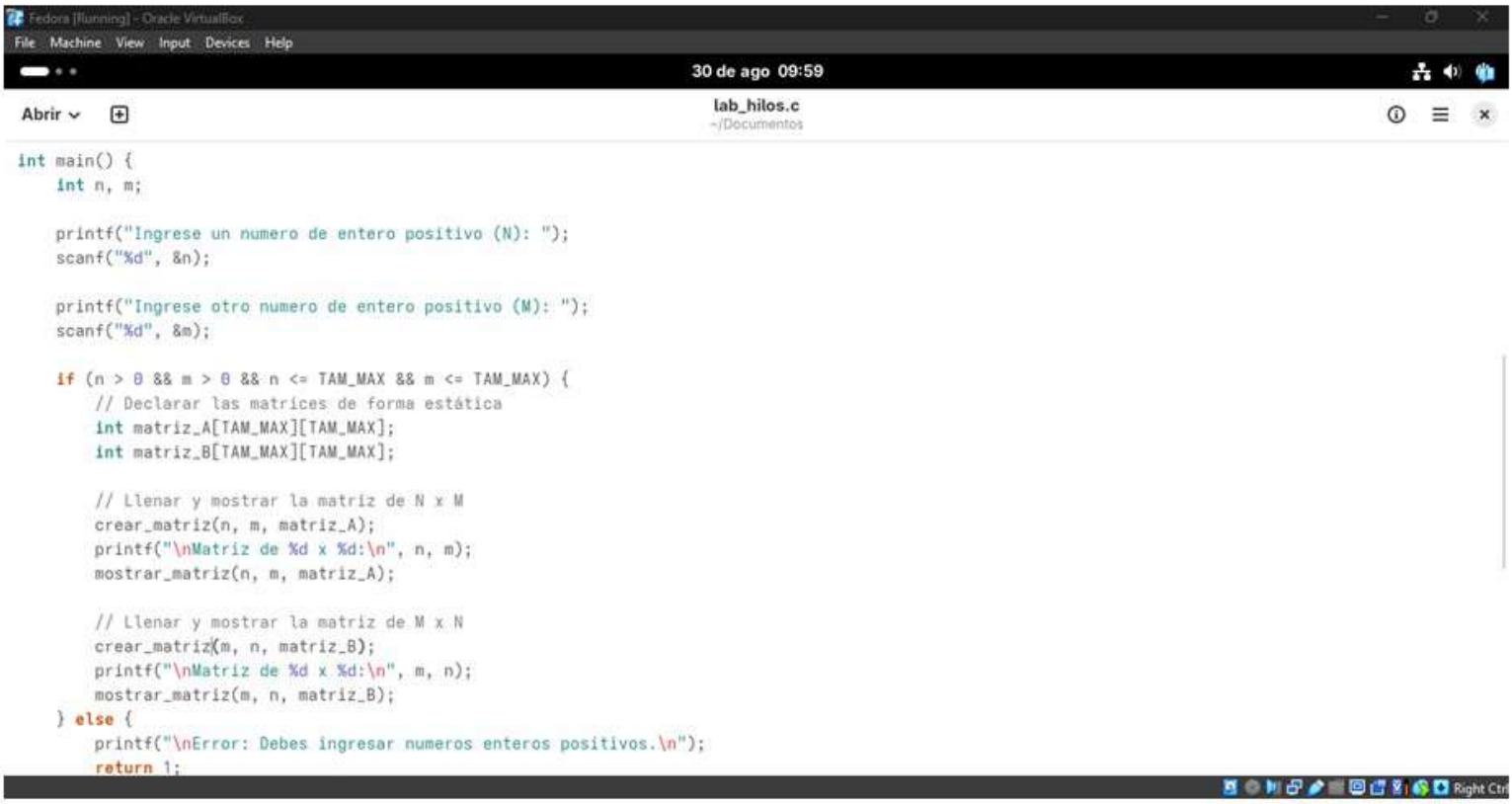
Recibí dos números enteros positivos N y M



Generé una matriz aleatoria de tamaño N x M y otra matriz aleatoria de tamaño M x N, ambas con valores en las celdas en el intervalo [1,5]



Tuve que invertir la lógica de validación de números positivos para evitar poner el código funcional en el else



File Machine View Input Devices Help

30 de ago 10:00

Abrir

lab_hilos.c
~/Documentos

```
    printf("Ingrese otro numero de entero positivo (M): ");
    scanf("%d", &m);

    if (n > 0 && m > 0 && n <= TAM_MAX && m <= TAM_MAX) {
        // Declarar las matrices de forma estática
        int matriz_A[TAM_MAX][TAM_MAX];
        int matriz_B[TAM_MAX][TAM_MAX];

        // Llenar y mostrar la matriz de N x M
        crear_matriz(n, m, matriz_A);
        printf("\nMatriz de %d x %d:\n", n, m);
        mostrar_matriz(n, m, matriz_A);

        // Llenar y mostrar la matriz de M x N
        crear_matriz(m, n, matriz_B);
        printf("\nMatriz de %d x %d:\n", m, n);
        mostrar_matriz(m, n, matriz_B);
    } else {
        printf("\nError: Debes ingresar numeros enteros positivos.\n");
        return 1;
    }

    return 0;
}
```

Right Ctr

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 10:00

nicole@vbox:~/Documentos
~/Documentos

```
nicole@vbox:~/Documentos$ gcc -o lab_hilos lab_hilos.c
nicole@vbox:~/Documentos$ ./lab_hilos
Ingrese un numero de entero positivo (N): 5
Ingrese otro numero de entero positivo (M): 3

Matriz de 5 x 3:
4       2       3
1       4       1
2       3       5
2       3       3
1       5       4

Matriz de 3 x 5:
2       1       2       3       2
2       4       3       5       3
1       3       4       3       1
nicole@vbox:~/Documentos$
```

Multipliqué ambas matrices en una matriz resultante R:MxM

Tuve que cambiar la declaración de las variables para evitar un segmentation fault

```c
void mostrar_matriz(int filas, int columnas, int matriz[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }
}

// Función para multiplicar dos matrices y guardar el resultado en una tercera matriz
void multiplicar_matrices(int m, int n, int matriz_B[TAM_MAX][TAM_MAX], int matriz_A[TAM_MAX][TAM_MAX], int matriz_C[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            matriz_C[i][j] = 0; // Inicializa el elemento C[i][j] a 0
            for (int k = 0; k < n; k++) {
                matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
            }
        }
    }
}

int main() {
    int n, m;

    printf("Ingrese un numero de entero positivo (N): ");
    scanf("%d", &n);
```



```c
    int n, m;

    printf("Ingrese un numero de entero positivo (N): ");
    scanf("%d", &n);

    printf("Ingrese otro numero de entero positivo (M): ");
    scanf("%d", &m);

    if (n > 0 && m > 0 && n <= TAM_MAX && m <= TAM_MAX) {
        // Llenar y mostrar la matriz de N x M
        crear_matriz(n, m, matriz_A);
        printf("\nMatriz de %d x %d:\n", n, m);
        mostrar_matriz(n, m, matriz_A);

        // Llenar y mostrar la matriz de M x N
        crear_matriz(m, n, matriz_B);
        printf("\nMatriz de %d x %d:\n", m, n);
        mostrar_matriz(m, n, matriz_B);

        // Multiplicar B x A y almacenar el resultado en C
        multiplicar_matrices(m, n, matriz_B, matriz_A, matriz_C);

        // Mostrar la matriz resultante C (m x m)
        printf("\n--- Matriz C (%d x %d) = B x A ---\n", m, m);
        mostrar_matriz(m, m, matriz_C);
    } else {
        printf("\nError: Debes ingresar numeros enteros positivos y no mayores de %d.\n", TAM_MAX);
```

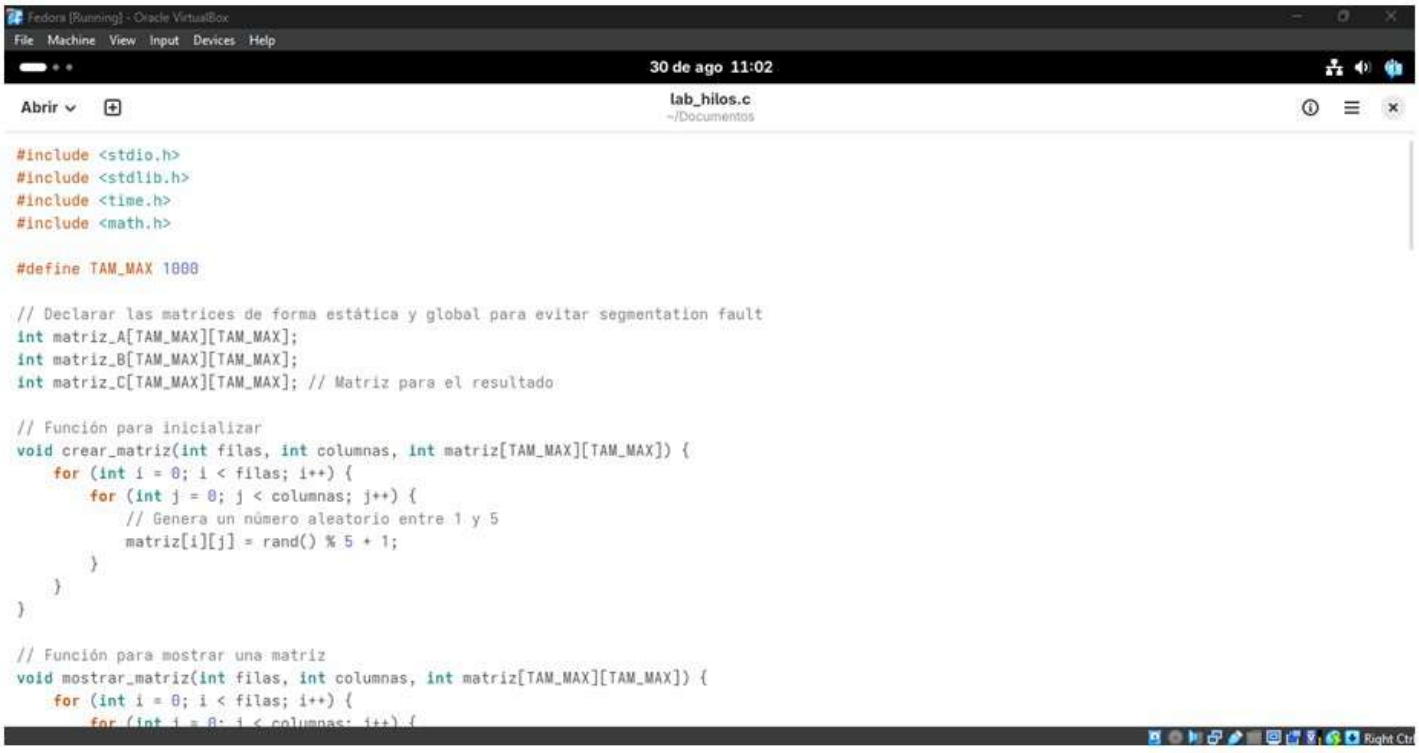
```c
    if (n > 0 && m > 0 && n <= TAM_MAX && m <= TAM_MAX) {
        // Llenar y mostrar la matriz de N x M
        crear_matriz(n, m, matriz_A);
        printf("\nMatriz de %d x %d:\n", n, m);
        mostrar_matriz(n, m, matriz_A);

        // Llenar y mostrar la matriz de M x N
        crear_matriz(m, n, matriz_B);
        printf("\nMatriz de %d x %d:\n", m, n);
        mostrar_matriz(m, n, matriz_B);

        // Multiplicar B x A y almacenar el resultado en C
        multiplicar_matrices(m, n, matriz_B, matriz_A, matriz_C);

        // Mostrar la matriz resultante C (m x m)
        printf("\n--- Matriz C (%d x %d) = B x A ---\n", m, m);
        mostrar_matriz(m, m, matriz_C);
    } else {
        printf("\nError: Debes ingresar numeros enteros positivos y no mayores de %d.\n", TAM_MAX);
        return 1;
    }

    return 0;
}
```

Lo ejecuté 100 veces con los valores N:1000 y M:1000, promediando los tiempos de ejecución y calculando la desviación estándar

Tuve que modificar mi código para poder ejecutarlo en una cantidad de veces variable y poder hacer los cálculos necesarios, además comenté los prints de matrices para que no afecten los resultados

Lo ejecuté 100 veces con los valores N:1000 y M:1000, promediando los tiempos de ejecución y calculando la desviación estándar

Tuve que modificar mi código para poder ejecutarlo en una cantidad de veces variable y poder hacer los cálculos necesarios, además comenté los prints de matrices para que no afecten los resultados



```
        for (int j = 0; j < columnas; j++) {
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }
}

// Función para multiplicar dos matrices y guardar el resultado en una tercera matriz
void multiplicar_matrices(int m, int n, int matriz_B[TAM_MAX][TAM_MAX], int matriz_A[TAM_MAX][TAM_MAX], int matriz_C[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            matriz_C[i][j] = 0; // Inicializa el elemento C[i][j] a 0
            for (int k = 0; k < n; k++) {
                matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
            }
        }
    }
}

int main() {
    int n, m, ciclos;
    double tiempos[100]; // Array para almacenar los tiempos de cada ciclo
    double suma, promedio, desviacion = 0.0;

    printf("Ingrese un numero de entero positivo (N): ");
    scanf("%d", &n);
```



```
    double suma, promedio, desviacion = 0.0;

    printf("Ingrese un numero de entero positivo (N): ");
    scanf("%d", &n);

    printf("Ingrese otro numero de entero positivo (M): ");
    scanf("%d", &m);

    printf("Ingrese otro numero de entero positivo (Ciclos): ");
    scanf("%d", &ciclos);

    if (n > 0 && m > 0 && n <= TAM_MAX && m <= TAM_MAX && ciclos > 0) {
        for (int i = 0; i < ciclos; i++) {
            // Medir tiempo de inicio
            clock_t inicio = clock();

            // Llenar y mostrar la matriz de N x M
            crear_matriz(n, m, matriz_A);
            //printf("\nMatriz de %d x %d:\n", n, m);
            //mostrar_matriz(n, m, matriz_A);

            // Llenar y mostrar la matriz de M x N
            crear_matriz(m, n, matriz_B);
            //printf("\nMatriz de %d x %d:\n", m, n);
            //mostrar_matriz(m, n, matriz_B);
```

File Machine View Input Devices Help

30 de ago 11:04

Abrir

lab_hilos.c
~/Documentos

```
        crear_matriz(m, n, matriz_B);
        //printf("\nMatriz de %d x %d:\n", m, n);
        //mostrar_matriz(m, n, matriz_B);

        // Multiplicar B x A y almacenar el resultado en C
        multiplicar_matrices(m, n, matriz_B, matriz_A, matriz_C);

        // Medir tiempo de inicio
        clock_t fin = clock();

        // Calcular tiempo transcurrido en segundos
        double tiempo = ((double)(fin - inicio)) / CLOCKS_PER_SEC;
        tiempos[i] = tiempo;
        suma += tiempo;

        // Mostrar la matriz resultante C (m x m)
        //printf("\n--- Matriz C (%d x %d) = B x A ---\n", m, m);
        //mostrar_matriz(m, m, matriz_C);
    }
    // Calcular promedio
    promedio = suma / ciclos;

    // Calcular desviación estándar
    for (int i = 0; i < ciclos; i++) {
        desviacion += pow(tiempos[i] - promedio, 2);
    }
```

Right Ctr

Fedora [Running] - Oracle VirtualBox

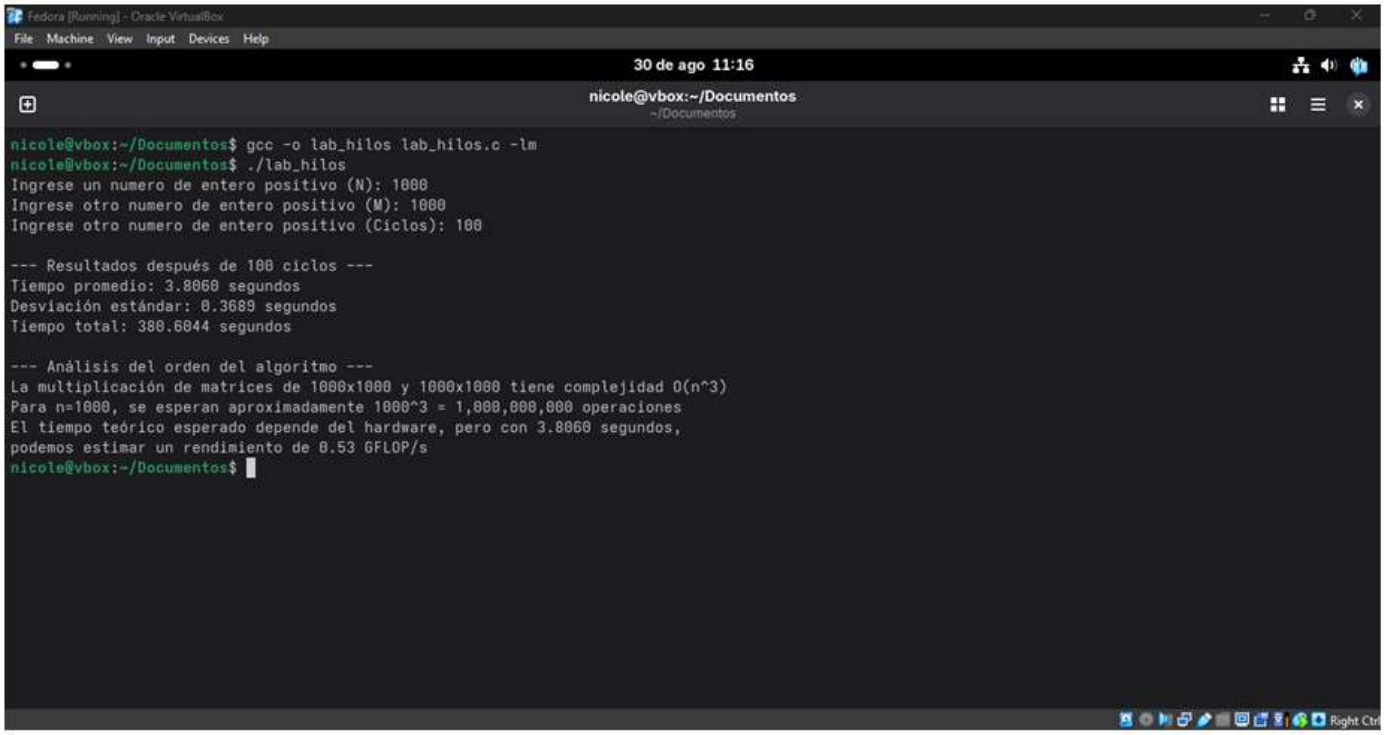
File Machine View Input Devices Help

30 de ago 11:04

Abrir

lab_hilos.c
~/Documentos

```
    // Calcular desviación estándar
    for (int i = 0; i < ciclos; i++) {
        desviacion += pow(tiempos[i] - promedio, 2);
    }
    desviacion = sqrt(desviacion / ciclos);

    // Mostrar resultados
    printf("\n--- Resultados después de %d ciclos ---\n", ciclos);
    printf("Tiempo promedio: %.4f segundos\n", promedio);
    printf("Desviación estándar: %.4f segundos\n", desviacion);
    printf("Tiempo total: %.4f segundos\n", suma);

    // Análisis del orden del algoritmo
    printf("\n--- Análisis del orden del algoritmo ---\n");
    printf("La multiplicación de matrices de %dx%d y %dx%d tiene complejidad O(n^3)\n", m, n, n, m);
    printf("Para n=1000, se esperan aproximadamente 1000^3 = 1,000,000,000 operaciones\n");
    printf("El tiempo teórico esperado depende del hardware, pero con %.4f segundos,\n", promedio);
    printf("podemos estimar un rendimiento de %.2f GFLOP/s\n",
           (2.0 * n * n * n) / (promedio * 1e9)); // 2 operaciones por elemento (multiplicación y suma)
} else {
    printf("\nError: Debes ingresar numeros enteros positivos y no mayores de %d.\n", TAM_MAX);
    return 1;
}

return 0;
```



Por ahora, el tanto el tiempo total, como el tiempo promedio y la desviación estándar no parecen ser muy altos, siendo el tiempo total alrededor de 6 minutos con 20 segundos, tomando en cuenta que cada creación de ambas matrices y sus multiplicaciones tomó en promedio 3.8 segundos me parece un resultado bastante aceptable. Incluso con la desviación estándar al dar un resultado cercano a 0.37 segundos se nota que cada ciclo tuvo tiempos muy similartes. Por último, tomé en cuenta la fórmula usada por las documentaciones de Nvidia y Jax en sus multiplicaciones de matrices para calcular los GigaFLOP/s o, dicho de otra forma, operaciones de punto flotante por segundo y vemos un rendimiento algo bajo con 0.53 GFLOP/s, algo normal al ser una máquina virtual (Matrix Multiplication Background User's Guide - NVIDIA Docs, 2023; Matrix Multiplication — JAX Documentation, s. f.).

Modifiqué el programa de forma que cada celda de R se calcule en un hilo usando pthread.

Tuve que hace una modificación original para que la cantidad de hilos no fuera excesiva y resultara en un segmentation fault, el programa calcula la cantidad óptima de hilos para no caer en error.

30 de ago 12:10

Abrir

lab_hilos.c
~/Documentos

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <pthread.h>
#include <unistd.h>

#define TAM_MAX 1000
#define MAX_HILOS 16  // Número máximo de hilos trabajando simultáneamente

// Declarar las matrices de forma estática y global para evitar segmentation fault
int matriz_A[TAM_MAX][TAM_MAX];
int matriz_B[TAM_MAX][TAM_MAX];
int matriz_C[TAM_MAX][TAM_MAX]; // Matriz para el resultado

// Variables globales para el pool de hilos
int m_global, n_global;
int siguiente_celda = 0;
int total_celdas = 0;
pthread_mutex_t mutex_trabajo = PTHREAD_MUTEX_INITIALIZER;

// Función para inicializar
void crear_matriz(int filas, int columnas, int matriz[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            // Genera un número aleatorio entre 1 y 5
```

Right Ctrl

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 12:10

Abrir

lab_hilos.c
~/Documentos

```c
            // Genera un número aleatorio entre 1 y 5
            matriz[i][j] = rand() % 5 + 1;
        }
    }
}

// Función para mostrar una matriz
void mostrar_matriz(int filas, int columnas, int matriz[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }
}

// Función para multiplicar dos matrices y guardar el resultado en una tercera matriz
void multiplicar_matrices(int m, int n, int matriz_B[TAM_MAX][TAM_MAX], int matriz_A[TAM_MAX][TAM_MAX], int matriz_C[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            matriz_C[i][j] = 0; // Inicializa el elemento C[i][j] a 0
            for (int k = 0; k < n; k++) {
                matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
            }
        }
    }
```

Right Ctrl

```
                matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
            }
        }
    }
}

// Función para obtener la siguiente celda a calcular
int obtener_siguiente_celda(int* i, int* j) {
    pthread_mutex_lock(&mutex_trabajo);

    if (siguiente_celda >= total_celdas) {
        pthread_mutex_unlock(&mutex_trabajo);
        return 0; // No hay más trabajo
    }

    *i = siguiente_celda / m_global;
    *j = siguiente_celda % m_global;
    siguiente_celda++;

    pthread_mutex_unlock(&mutex_trabajo);
    return 1; // Hay trabajo disponible
}

// Función que cada hilo ejecutará
void* trabajador_hilos(void* arg) {
    int i, j;
```



```
    int i, j;

    // Cada hilo procesa múltiples celdas hasta que no haya más trabajo
    while (obtener_siguiente_celda(&i, &j)) {
        // Calcular la celda [i][j]
        matriz_C[i][j] = 0;
        for (int k = 0; k < n_global; k++) {
            matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
        }
    }

    pthread_exit(NULL);
}

// Función para multiplicar matrices usando pool de hilos
void multiplicar_matrices_con_hilos(int m, int n) {
    // Configurar variables globales
    m_global = m;
    n_global = n;
    siguiente_celda = 0;
    total_celdas = m * m;

    // Determinar número óptimo de hilos (no más que el número de CPUs)
    int num_cpus = sysconf(_SC_NPROCESSORS_ONLN);
    int num_hilos = (num_cpus > MAX_HILOS) ? MAX_HILOS : num_cpus;
```

Abrir

lab_hilos.c
~/Documentos

```c
    // Determinar número óptimo de hilos (no más que el número de CPUs)
    int num_cpus = sysconf(_SC_NPROCESSORS_ONLN);
    int num_hilos = (num_cpus > MAX_HILOS) ? MAX_HILOS : num_cpus;

    // Crear array de hilos
    pthread_t hilos[MAX_HILOS];

    // Crear los hilos trabajadores
    for (int i = 0; i < num_hilos; i++) {
        if (pthread_create(&hilos[i], NULL, trabajador_hilos, NULL) != 0) {
            perror("Error al crear hilo");
            exit(EXIT_FAILURE);
        }
    }

    // Esperar a que todos los hilos terminen
    for (int i = 0; i < num_hilos; i++) {
        pthread_join(hilos[i], NULL);
    }
}

int main() {
    int n, m, ciclos;
    double tiempos[100]; // Array para almacenar los tiempos de cada ciclo
    double suma, promedio, desviacion = 0.0;
```

Right Ctrl

También agregué 2 prints, uno para saber cuántos hilos está usando y otro para saber el progreso cada vez que llegue a un ciclo múltiplo de 10.

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 12:15

Abrir

lab_hilos.c
~/Documentos

```c
    int n, m, ciclos;
    double tiempos[100]; // Array para almacenar los tiempos de cada ciclo
    double suma, promedio, desviacion = 0.0;

    printf("Ingrese un numero de entero positivo (N): ");
    scanf("%d", &n);

    printf("Ingrese otro numero de entero positivo (M): ");
    scanf("%d", &m);

    printf("Ingrese otro numero de entero positivo (Ciclos): ");
    scanf("%d", &ciclos);

    if (n > 0 && m > 0 && n <= TAM_MAX && m <= TAM_MAX && ciclos > 0) {
        printf("Usando hasta %d hilos simultáneos\n", (sysconf(_SC_NPROCESSORS_ONLN) > MAX_HILOS) ? MAX_HILOS : sysconf(_SC_NPROCESSORS_ONLN));
        for (int i = 0; i < ciclos; i++) {
            // Mostrar progreso
            if (i % 10 == 0) {
                printf("Ciclo %d/%d...\n", i + 1, ciclos);
            }
            // Medir tiempo de inicio
            clock_t inicio = clock();

            // Llenar y mostrar la matriz de N x M
            crear_matriz(n, m, matriz_A);
            //printf("\nMatriz de %d x %d:\n", n, m);
```

File Machine View Input Devices Help

30 de ago 12:15

Abrir

lab_hilos.c
~/Documentos

```
    // Llenar y mostrar la matriz de N x M
    crear_matriz(n, m, matriz_A);
    //printf("\nMatriz de %d x %d:\n", n, m);
    //mostrar_matriz(n, m, matriz_A);

    // Llenar y mostrar la matriz de M x N
    crear_matriz(m, n, matriz_B);
    //printf("\nMatriz de %d x %d:\n", m, n);
    //mostrar_matriz(m, n, matriz_B);

    // Multiplicar B x A y almacenar el resultado en C
    //multiplicar_matrices(m, n, matriz_B, matriz_A, matriz_C);

    // Multiplicar matrices usando hilos
    multiplicar_matrices_con_hilos(m, n);

    // Medir tiempo de inicio
    clock_t fin = clock();

    // Calcular tiempo transcurrido en segundos
    double tiempo = ((double)(fin - inicio)) / CLOCKS_PER_SEC;
    tiempos[i] = tiempo;
    suma += tiempo;

    // Mostrar la matriz resultante C (m x m)
```

Right Ctrl

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 12:16

Abrir

lab_hilos.c
~/Documentos

```
    tiempos[i] = tiempo;
    suma += tiempo;

    // Mostrar la matriz resultante C (m x m)
    //printf("\n--- Matriz C (%d x %d) = B x A ---\n", m, m);
    //mostrar_matriz(m, m, matriz_C);
}
// Calcular promedio
promedio = suma / ciclos;

// Calcular desviación estándar
for (int i = 0; i < ciclos; i++) {
    desviacion += pow(tiempos[i] - promedio, 2);
}
desviacion = sqrt(desviacion / ciclos);

// Mostrar resultados
printf("\n--- Resultados después de %d ciclos ---\n", ciclos);
printf("Tiempo promedio: %.4f segundos\n", promedio);
printf("Desviación estándar: %.4f segundos\n", desviacion);
printf("Tiempo total: %.4f segundos\n", suma);

// Análisis del orden del algoritmo
printf("\n--- Análisis del orden del algoritmo ---\n");
printf("La multiplicación de matrices de %dx%d y %dx%d tiene complejidad O(n^3)\n", m, n, n, m);
printf("Para n=1000, se esperan aproximadamente 1000^3 = 1,000,000,000 operaciones\n");
```

```c
        }
        desviacion = sqrt(desviacion / ciclos);

        // Mostrar resultados
        printf("\n--- Resultados después de %d ciclos ---\n", ciclos);
        printf("Tiempo promedio: %.4f segundos\n", promedio);
        printf("Desviación estándar: %.4f segundos\n", desviacion);
        printf("Tiempo total: %.4f segundos\n", suma);

        // Análisis del orden del algoritmo
        printf("\n--- Análisis del orden del algoritmo ---\n");
        printf("La multiplicación de matrices de %dx%d y %dx%d tiene complejidad O(n^3)\n", m, n, n, m);
        printf("Para n=1000, se esperan aproximadamente 1000^3 = 1,000,000,000 operaciones\n");
        printf("El tiempo teórico esperado depende del hardware, pero con %.4f segundos,\n", promedio);
        printf("podemos estimar un rendimiento de %.2f GFLOP/s\n",
               (2.0 * n * n * n) / (promedio * 1e9)); // 2 operaciones por elemento (multiplicación y suma)
    } else {
        printf("\nError: Debes ingresar numeros enteros positivos y no mayores de %d.\n", TAM_MAX);
        return 1;
    }

    return 0;
}
```

Volví a ejecutar 100 veces con N:1000 y M:1000, promedié los tiempos y calculé la desviación estándar.

```
Fedora [Running] - Oracle VirtualBox
File  Machine  View  Input  Devices  Help
30 de ago 12:24
nicole@vbox:~/Documentos
~/Documentos

nicole@vbox:~/Documentos$ gcc -o lab_hilos lab_hilos.c -lm
nicole@vbox:~/Documentos$ ./lab_hilos
Ingrese un numero de entero positivo (N): 1000
Ingrese otro numero de entero positivo (M): 1000
Ingrese otro numero de entero positivo (Ciclos): 100
Usando hasta 5 hilos simultáneos
Ciclo 1/100...
Ciclo 11/100...
Ciclo 21/100...
Ciclo 31/100...
Ciclo 41/100...
Ciclo 51/100...
Ciclo 61/100...
Ciclo 71/100...
Ciclo 81/100...
Ciclo 91/100...

--- Resultados después de 100 ciclos ---
Tiempo promedio: 26.9539 segundos
Desviación estándar: 1.1160 segundos
Tiempo total: 2695.3947 segundos

--- Análisis del orden del algoritmo ---
La multiplicación de matrices de 1000x1000 y 1000x1000 tiene complejidad O(n^3)
Para n=1000, se esperan aproximadamente 1000^3 = 1,000,000,000 operaciones
El tiempo teórico esperado depende del hardware, pero con 26.9539 segundos,
podemos estimar un rendimiento de 0.07 GFLOP/s
nicole@vbox:~/Documentos$
```

El tiempo total aumentó bastante de 6 minutos anteriormente hasta 44 minutos ahora con el manejo de hilos, así mismo el tiempo promedio tuvo un gran incremento de 3.8 segundos hasta 26.95 segundos. No obstante, el incremento de la desviación estándar no fue tan drástico puesto que solo pasó de 0.37 segundos a 1.12 segundos, fue más que el doble del anterior, pero aún así los ciclos tienen un tiempo similar. Algo interesante es que los GLOP/s se redujeron de 0.53 hasta 0.07. En conclusión, se nota que el uso de hilos solo aumentó la complejidad del programa, por ello los tiempos aumentaron resultando en un algoritmo mucho más ineficiente.

Investigué qué es PIPE y MMAP para comunicación entre procesos (IPC), cómo funcionan ambos en C, elegí uno

PIPE son un tipo de técnica de comunicación entre procesos que utiliza una sección de memoria compartida y permite que dos o más procesos se comuniquen entre sí mediante la creación de un canal entre ellos que puede ser unidireccional o bidireccional, estas se implementan mediante llamadas al sistema en la mayoría de los sistema operativos modernos incluyendo Linux, Windos y macOS (Combeau, 2022; GeeksforGeeks, 2025).

Combeau (2022) indica que la forma de crear un pipe es con el siguiente código:

int pipe(int pipefd[2]);

Pipe recibe una matriz de dos enteros que representan los extremos del pipe, pipefd[0] para lectura y pipefd[1] para escritura. En caso de éxito, pipe retorna 0, sino retorna -1. Para establecer la comunicación entre procesos primero se debe crear el pipe en el proceso padre, luego se crea al proceso hijo, de esta forma el hijo tendrá un duplicado de los descriptores del pipe.

Según Combeau (2022), los datos se transmiten como un flujo de bytes, mientras que el pipe actúa como una cola FIFO (First In, First Out). Además sincrónico, lo que significa que si no hay datos para leer, el proceso lector se bloquea hasta que llegan datos. Para comunicación bidireccional, se necesitan dos pipes y se suele utilizar la biblioteca <unistd.h>.

Por otro lado, Lee (2024) indica que MMAP (memory mapping) es un mecanismo de comunicación entre procesos que permite mapear un archivo o un objeto de memoria compartida al espacio de direcciones virtual de múltiples procesos, de ese modo, permitiendo que puedan acceder a la misma región de memoria como si fuera parte de su propio espacio, asimismo facilitando la comunicación bidireccional y eficiente sin la necesidad de copias explícitas de datos, puesto que los cambios en memoria se reflejan inmediatamente en todos los procesos que la comparten.

Para utilizar MMAP en C, Mishra (2024) señala que se debe usar la llamada al sistema mmap() del header <sys/mman.h>. Después, se crea un objeto de memoria compartida usando shm_open() (para memoria anónima) o open() (para un archivo), seguido de ftruncate() para establecer el tamaño. Luego, mmap() mapea la región con flags como MAP_SHARED para compartirla entre procesos. Los parámetros incluyen la dirección deseada (generalmente NULL), el tamaño, protecciones (PROT_READ | PROT_WRITE), flags, el descriptor de archivo y el offset. Los procesos escriben y leen directamente en la dirección retornada por mmap(). Al finalizar, se desmapea con munmap() y se cierra con shm_unlink() si es necesario. La sincronización (e.g., con semáforos) es crucial para evitar condiciones de carrera, ya que no hay bloqueo inherente.

En conclusión, elijo MMAP porque aunque PIPE es más simple y adecuado para flujos de datos unidireccionales y procesos relacionados, MMAP ofrece una mayor eficiencia en términos de rendimiento, especialmente para grandes volúmenes de datos y dado que antes tuve dificultades al intentar crear simultáneamente todos los hilos necesarios para la multiplicación de matrices e igualmente al final terminé con un resultado mucho menos eficiente quiero darle la oportunidad a MMAP para ver si al menos logra superar los tiempos de pthread.

Modifiqué el programa de forma que cada celda de R se calcule en un subproceso usando fork y alguno de los dos métodos de IPC.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include <signal.h>

#define TAM_MAX 1000
#define MAX_HILOS 16  // Número máximo de hilos trabajando simultáneamente
#define MAX_PROCESOS_SIMULTANEOS 100

// Declarar las matrices de forma estática y global para evitar segmentation fault
int matriz_A[TAM_MAX][TAM_MAX];
int matriz_B[TAM_MAX][TAM_MAX];
int matriz_C[TAM_MAX][TAM_MAX]; // Matriz para el resultado

// Variables globales para el pool de hilos
int m_global, n_global;
int siguiente_celda = 0;
```



```c
int siguiente_celda = 0;
int total_celdas = 0;
pthread_mutex_t mutex_trabajo = PTHREAD_MUTEX_INITIALIZER;

// Estructura para compartir datos entre procesos usando mmap
typedef struct {
    int matriz_A[TAM_MAX][TAM_MAX];
    int matriz_B[TAM_MAX][TAM_MAX];
    int matriz_C[TAM_MAX][TAM_MAX];
    int m;  // dimensiones
    int n;
    int proceso_terminado[TAM_MAX * TAM_MAX];  // flag para cada celda
} shared_data_t;

// Variables globales
shared_data_t *shared_mem = NULL;
int shm_fd = -1;

// Función para inicializar
void crear_matriz(int filas, int columnas, int matriz[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            // Genera un número aleatorio entre 1 y 5
            matriz[i][j] = rand() % 5 + 1;
        }
    }
```

```
            // Genera un número aleatorio entre 1 y 5
            matriz[i][j] = rand() % 5 + 1;
        }
    }
}

// Función para mostrar una matriz
void mostrar_matriz(int filas, int columnas, int matriz[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }
}

// Función para multiplicar dos matrices y guardar el resultado en una tercera matriz
void multiplicar_matrices(int m, int n, int matriz_B[TAM_MAX][TAM_MAX], int matriz_A[TAM_MAX][TAM_MAX], int matriz_C[TAM_MAX][TAM_MAX]) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            matriz_C[i][j] = 0; // Inicializa el elemento C[i][j] a 0
            for (int k = 0; k < n; k++) {
                matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
            }
        }
    }
```



```
                matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
            }
        }
    }
}

// Función para obtener la siguiente celda a calcular
int obtener_siguiente_celda(int* i, int* j) {
    pthread_mutex_lock(&mutex_trabajo);

    if (siguiente_celda >= total_celdas) {
        pthread_mutex_unlock(&mutex_trabajo);
        return 0; // No hay más trabajo
    }

    *i = siguiente_celda / m_global;
    *j = siguiente_celda % m_global;
    siguiente_celda++;

    pthread_mutex_unlock(&mutex_trabajo);
    return 1; // Hay trabajo disponible
}

// Función que cada hilo ejecutará
void* trabajador_hilos(void* arg) {
    int i, j;
```

Abrir

lab_hilos.c
~/Documentos

```c
// Función que cada hilo ejecutará
void* trabajador_hilos(void* arg) {
    int i, j;

    // Cada hilo procesa múltiples celdas hasta que no haya más trabajo
    while (obtener_siguiente_celda(&i, &j)) {
        // Calcular la celda [i][j]
        matriz_C[i][j] = 0;
        for (int k = 0; k < n_global; k++) {
            matriz_C[i][j] += matriz_B[i][k] * matriz_A[k][j];
        }
    }

    pthread_exit(NULL);
}

// Función para multiplicar matrices usando pool de hilos
void multiplicar_matrices_con_hilos(int m, int n) {
    // Configurar variables globales
    m_global = m;
    n_global = n;
    siguiente_celda = 0;
    total_celdas = m * m;

    // Determinar número óptimo de hilos (no más que el número de CPUs)
```


30 de ago 14:48

Abrir

lab_hilos.c
~/Documentos

```c
    siguiente_celda = 0;
    total_celdas = m * m;

    // Determinar número óptimo de hilos (no más que el número de CPUs)
    int num_cpus = sysconf(_SC_NPROCESSORS_ONLN);
    int num_hilos = (num_cpus > MAX_HILOS) ? MAX_HILOS : num_cpus;

    // Crear array de hilos
    pthread_t hilos[MAX_HILOS];

    // Crear los hilos trabajadores
    for (int i = 0; i < num_hilos; i++) {
        if (pthread_create(&hilos[i], NULL, trabajador_hilos, NULL) != 0) {
            perror("Error al crear hilo");
            exit(EXIT_FAILURE);
        }
    }

    // Esperar a que todos los hilos terminen
    for (int i = 0; i < num_hilos; i++) {
        pthread_join(hilos[i], NULL);
    }
}

// Función para limpiar memoria compartida
void cleanup() {
```

File Machine View Input Devices Help

30 de ago 14:48

Abrir

lab_hilos.c
~/Documentos

```
}

// Función para limpiar memoria compartida
void cleanup() {
    if (shared_mem != NULL) {
        munmap(shared_mem, sizeof(shared_data_t));
    }
    if (shm_fd != -1) {
        close(shm_fd);
        shm_unlink("/matrix_multiplication");
    }
}

// Manejador de señales para limpieza
void signal_handler(int sig) {
    printf("\nRecibida señal %d, limpiando...\n", sig);
    cleanup();
    exit(1);
}

// Función para inicializar memoria compartida
int inicializar_memoria_compartida() {
    // Crear objeto de memoria compartida
    shm_fd = shm_open("/matrix_multiplication", O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Error en shm_open");
```

Right Ctrl

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 14:49

Abrir

lab_hilos.c
~/Documentos

```
        perror("Error en shm_open");
        return -1;
    }

    // Establecer el tamaño
    if (ftruncate(shm_fd, sizeof(shared_data_t)) == -1) {
        perror("Error en ftruncate");
        close(shm_fd);
        shm_unlink("/matrix_multiplication");
        return -1;
    }

    // Mapear la memoria
    shared_mem = (shared_data_t *)mmap(NULL, sizeof(shared_data_t),
                                       PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        perror("Error en mmap");
        close(shm_fd);
        shm_unlink("/matrix_multiplication");
        return -1;
    }

    return 0;
}

// Función que ejecuta cada proceso hijo para calcular una celda específica
```

File Machine View Input Devices Help

30 de ago 14:49

Abrir

lab_hilos.c
~/Documentos

```c
// Función que ejecuta cada proceso hijo para calcular una celda específica
void calcular_celda_proceso(int fila, int columna) {
    // Cada proceso hijo mapea la memoria compartida
    int child_shm_fd = shm_open("/matrix_multiplication", O_RDWR, 0666);
    if (child_shm_fd == -1) {
        exit(EXIT_FAILURE);
    }

    shared_data_t *child_shared = (shared_data_t *)mmap(NULL, sizeof(shared_data_t),
                                                        PROT_READ | PROT_WRITE, MAP_SHARED,
                                                        child_shm_fd, 0);
    if (child_shared == MAP_FAILED) {
        close(child_shm_fd);
        exit(EXIT_FAILURE);
    }

    // Calcular la celda asignada: C[fila][columna] = suma(B[fila][k] * A[k][columna])
    int resultado = 0;
    for (int k = 0; k < child_shared->n; k++) {
        resultado += child_shared->matriz_B[fila][k] * child_shared->matriz_A[k][columna];
    }

    // Escribir el resultado en la matriz C
    child_shared->matriz_C[fila][columna] = resultado;

    // Marcar como terminado
```

Right Ctrl

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 14:49

Abrir

lab_hilos.c
~/Documentos

```c
    // Marcar como terminado
    int celda_index = fila * child_shared->m + columna;
    child_shared->proceso_terminado[celda_index] = 1;

    // Limpiar memoria del proceso hijo
    munmap(child_shared, sizeof(shared_data_t));
    close(child_shm_fd);

    exit(EXIT_SUCCESS);
}

// Función para multiplicar matrices usando procesos fork
void multiplicar_matrices_con_fork(int m, int n) {
    pid_t procesos[TAM_MAX * TAM_MAX];
    int total_celdas = m * m;
    int procesos_activos = 0;
    int celdas_completadas = 0;

    // Inicializar flags de procesos terminados
    memset(shared_mem->proceso_terminado, 0, sizeof(shared_mem->proceso_terminado));

    printf("Creando %d procesos para calcular matriz %dx%d...\n", total_celdas, m, m);

    // Crear procesos en lotes para evitar sobrecargar el sistema
    for (int celda = 0; celda < total_celdas; celda++) {
        int fila = celda / m;
```

30 de ago 14:50

Abrir

lab_hilos.c
~/Documentos

```
// Crear procesos en lotes para evitar sobrecargar el sistema
for (int celda = 0; celda < total_celdas; celda++) {
    int fila = celda / m;
    int columna = celda % m;

    // Limitar procesos simultáneos
    while (procesos_activos >= MAX_PROCESOS_SIMULTANEOS) {
        // Esperar a que termine algún proceso
        int status;
        pid_t pid_terminado = wait(&status);
        if (pid_terminado > 0) {
            procesos_activos--;
            celdas_completadas++;

            // Mostrar progreso cada 100000 celdas
            if (celdas_completadas % 100000 == 0) {
                printf("Completadas %d/%d celdas (%.1f%%)\n",
                       celdas_completadas, total_celdas,
                       (float)celdas_completadas * 100 / total_celdas);
            }
        }
    }

    // Crear nuevo proceso hijo
    pid_t pid = fork();
```

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 14:50

Abrir

lab_hilos.c
~/Documentos

```
    }

    // Crear nuevo proceso hijo
    pid_t pid = fork();

    if (pid == 0) {
        // Proceso hijo: calcular la celda especifica
        calcular_celda_proceso(fila, columna);
    } else if (pid > 0) {
        // Proceso padre: guardar el PID
        procesos[celda] = pid;
        procesos_activos++;
    } else {
        perror("Error en fork");
        exit(EXIT_FAILURE);
    }
}

// Esperar a que terminen todos los procesos restantes
while (procesos_activos > 0) {
    int status;
    pid_t pid_terminado = wait(&status);
    if (pid_terminado > 0) {
        procesos_activos--;
        celdas_completadas++;

        if (celdas_completadas % 1000 == 0) {
```

30 de ago 14:50

Abrir

lab_hilos.c
~/Documentos

```
        if (pid_terminado > 0) {
            procesos_activos--;
            celdas_completadas++;

            if (celdas_completadas % 1000 == 0) {
                printf("Completadas %d/%d celdas (%.1f%%)\n",
                       celdas_completadas, total_celdas,
                       (float)celdas_completadas * 100 / total_celdas);
            }
        }
    }

    printf("Todos los procesos terminaron. Celdas completadas: %d/%d\n",
           celdas_completadas, total_celdas);
}

int main() {
    int n, m, ciclos;
    double tiempos[100]; // Array para almacenar los tiempos de cada ciclo
    double suma, promedio, desviacion = 0.0;

    // Configurar manejadores de señales para limpieza
    signal(SIGINT, signal_handler);
    signal(SIGTERM, signal_handler);

    printf("Ingrese un numero de entero positivo (N): ");
    scanf("%d", &n);
```

Right Ctrl

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

30 de ago 14:51

Abrir

lab_hilos.c
~/Documentos

```
    scanf("%d", &n);

    printf("Ingrese otro numero de entero positivo (M): ");
    scanf("%d", &m);

    printf("Ingrese otro numero de entero positivo (Ciclos): ");
    scanf("%d", &ciclos);

    if (n > 0 && m > 0 && n <= TAM_MAX && m <= TAM_MAX && ciclos > 0) {
        //printf("Usando hasta %d hilos simultáneos\n", (sysconf(_SC_NPROCESSORS_ONLN) > MAX_HILOS) ? MAX_HILOS : sysconf(_SC_NPROCESSORS_ONLN));

        // Inicializar memoria compartida
        if (inicializar_memoria_compartida() == -1) {
            printf("Error al inicializar memoria compartida\n");
            return 1;
        }

        // Configurar dimensiones en memoria compartida
        shared_mem->m = m;
        shared_mem->n = n;

        for (int i = 0; i < ciclos; i++) {
            // Mostrar progreso
            if (i % 10 == 0) {
                printf("Ciclo %d/%d...\n", i + 1, ciclos);
            }
```

Right Ctrl

30 de ago 14:51

Abrir

lab_hilos.c
~/Documentos

```
    // Mostrar progreso
    if (i % 10 == 0) {
        printf("Ciclo %d/%d...\n", i + 1, ciclos);
    }
    // Medir tiempo de inicio
    clock_t inicio = clock();

    // Llenar y mostrar la matriz de N x M
    crear_matriz(n, m, matriz_A);
    //printf("\nMatriz de %d x %d:\n", n, m);
    //mostrar_matriz(n, m, matriz_A);

    // Llenar y mostrar la matriz de M x N
    crear_matriz(m, n, matriz_B);
    //printf("\nMatriz de %d x %d:\n", m, n);
    //mostrar_matriz(m, n, matriz_B);

    // Multiplicar B x A y almacenar el resultado en C
    //multiplicar_matrices(m, n, matriz_B, matriz_A, matriz_C);

    // Multiplicar matrices usando hilos
    //multiplicar_matrices_con_hilos(m, n);

    // Multiplicar matrices usando fork
    multiplicar_matrices_con_fork(m, n);
```

Right Ctrl

Fedora [Running] - Oracle VirtualBox

File Machine View Input Devices Help

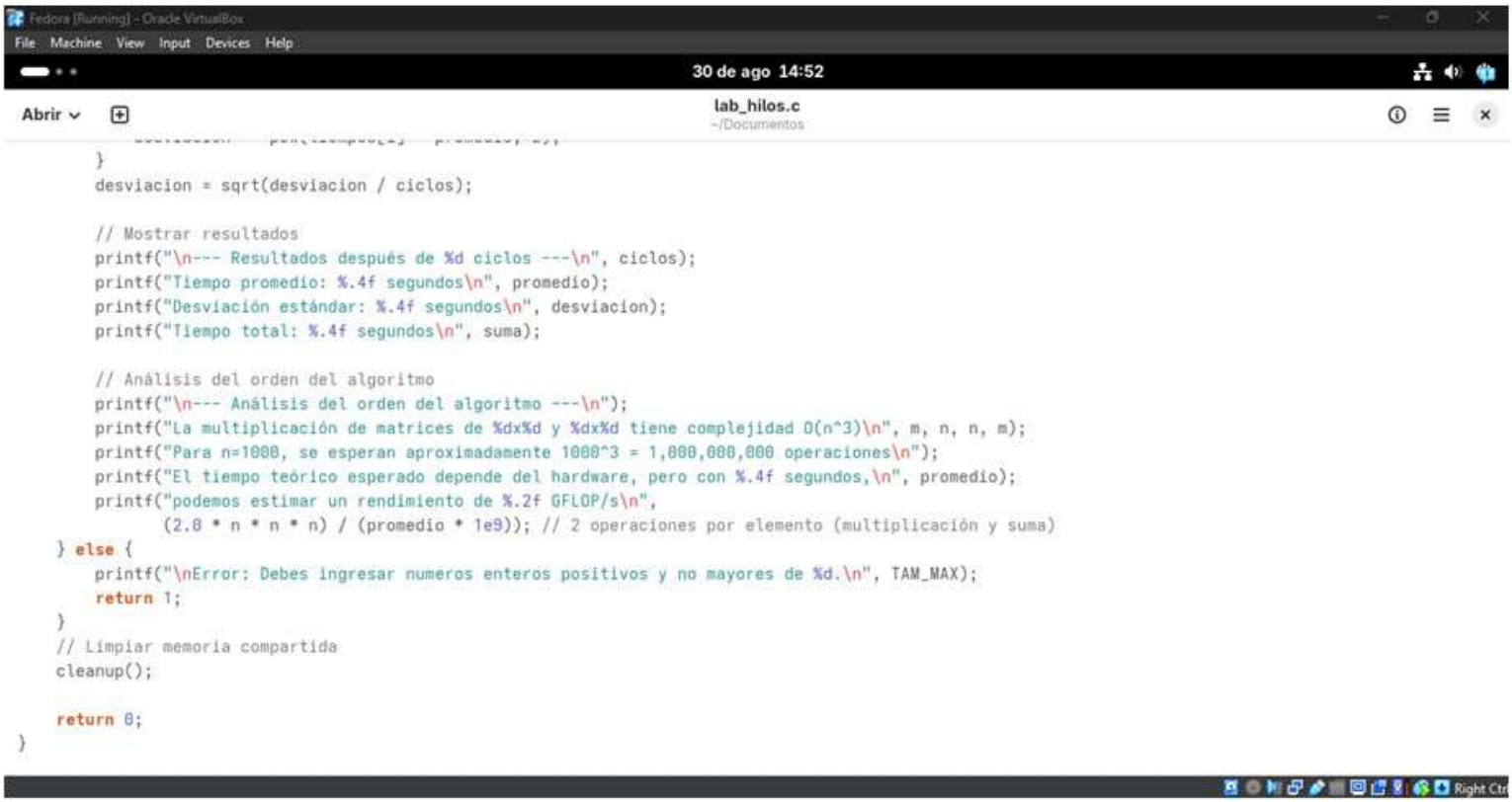
30 de ago 14:51

Abrir

lab_hilos.c
~/Documentos

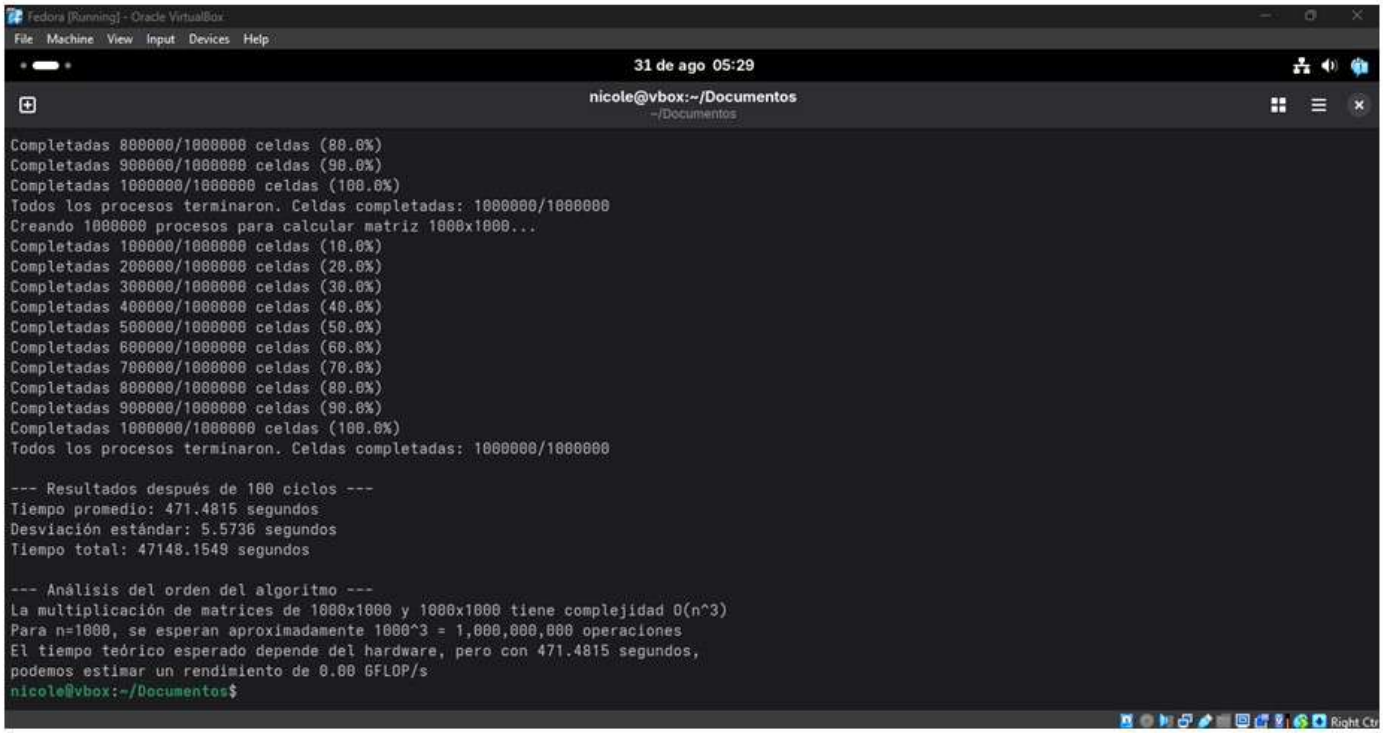
```
    // Multiplicar matrices usando fork
    multiplicar_matrices_con_fork(m, n);

    // Medir tiempo de inicio
    clock_t fin = clock();

    // Calcular tiempo transcurrido en segundos
    double tiempo = ((double)(fin - inicio)) / CLOCKS_PER_SEC;
    tiempos[i] = tiempo;
    suma += tiempo;

    // Mostrar la matriz resultante C (m x m)
    //printf("\n--- Matriz C (%d x %d) = B x A ---\n", m, m);
    //mostrar_matriz(m, m, matriz_C);
}
// Calcular promedio
promedio = suma / ciclos;

// Calcular desviación estándar
for (int i = 0; i < ciclos; i++) {
    desviacion += pow(tiempos[i] - promedio, 2);
}
desviacion = sqrt(desviacion / ciclos);

// Mostrar resultados
```

Right Ctrl



Volví a ejecutar 100 veces con N:1000 y M:1000, promedié los tiempos y calculé la desviación estándar

Volví a ejecutar 100 veces con N:1000 y M:1000, promedié los tiempos y calculé la desviación estándar



Los tiempos empeoraron terriblemente, aunque MMAP prometía ser la opción más eficiente entre ambas, se nota que el uso de un subproceso por cada celda fue una opción aún peor al llegar a un tiempo total aproximado a 13 horas, asimismo el tiempo promedio también sufrió un aumento significativo hasta casi 8 minutos. En cambio, la desviación estándar, aunque también se quintuplicó comparado con los hilos, si hablamos de un promedio de casi 8 minutos una desviación de apenas 5 segundos no me parece muy grave. Algo interesante es que la medida de GFLOP/s que agregué parece que ahora es un número tan pequeño que Fedora decidió representarlo como 0.00.

En conclusión, basado en los experimentos realizados durante esta asignatura, considero que para el cálculo de multiplicación de matrices tanto el manejo de hilos como el manejo de subprocesos empeoran en gran medida el desempeño del programa y la mejor opción es resolverlo de forma lineal, de esta forma cada multiplicación de matrices de tamaño 1000x1000 toma un tiempo aproximado a 3.8 segundos en lugar de 26.95 segundos en el caso del manejo de hilos o casi 8 minutos en el caso del manejo de los subprocesos.

Los tiempos empeoraron terriblemente, aunque MMAP prometía ser la opción más eficiente entre ambas, se nota que el uso de un subproceso por cada celda fue una opción aún peor al llegar a un tiempo total aproximado a 13 horas, asimismo el tiempo promedio también sufrió un aumento significativo hasta casi 8 minutos. En cambio, la desviación estándar, aunque también se quintuplicó comparado con los hilos, si hablamos de un promedio de casi 8 minutos una desviación de apenas 5 segundos no me parece muy grave. Algo interesante es que la medida de GFLOP/s que agregué parece que ahora es un número tan pequeño que Fedora decidió representarlo como 0.00.

En conclusión, basado en los experimentos realizados durante esta asignatura, considero que para el cálculo de multiplicación de matrices tanto el manejo de hilos como el manejo de subprocesos empeoran en gran medida el desempeño del programa y la mejor opción es resolverlo de forma lineal, de esta forma cada multiplicación de matrices de tamaño 1000x1000 toma un tiempo aproximado a 3.8 segundos en lugar de 26.95 segundos en el caso del manejo de hilos o casi 8 minutos en el caso del manejo de los subprocesos.

Referencias

Combeau, M. (2022, 31 octubre). *Pipe: an Inter-Process Communication Method - codequoi*. https://www.codequoi.com/en/pipe-an-inter-process-communication-method/#what-is-a-pipe

GeeksforGeeks. (2025, 12 julio). *IPC technique PIPES*. GeeksforGeeks. https://www.geeksforgeeks.org/operating-systems/ipc-technique-pipes/

Lee, J. (2024, 22 febrero). *How does memory-mapping (mmap) work?* Medium. https://medium.com/@jyjimmylee/how-does-memory-mapping-mmap-work-c8a6a550ba0d

Matrix Multiplication Background User's Guide - NVIDIA Docs. (2023). NVIDIA Docs. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix Multiplication — JAX documentation. (s. f.). https://docs.jax.dev/en/latest/pallas/tpu/matmul.html

Mishra, S. (2024, 30 diciembre). What is MMAP in Linux and how it is useful? *Shashank's Substack*. https://programmingappliedai.substack.com/p/what-is-mmap-in-linux-and-how-it