Identificación de Matriz Dispersa con Procesos e Hilos

Pontificia Universidad Javeriana

Proyecto - Sistemas Operativos

John Jairo Corredor Franco

Diego Fernando Castrillón Cortes, Sebastián Vargas Casquete, Thomas Arévalo Rodríguez

Mayo del 2025

1. Introducción

Este documento describe la implementación, diseño y evaluación de un proyecto desarrollado en el marco del curso de Sistemas Operativos. El objetivo central del proyecto es determinar si una matriz es dispersa o no, entendiendo por matriz dispersa aquella que contiene un alto porcentaje de ceros en comparación con su tamaño total. Para lograr esta tarea de manera eficiente, se utilizaron enfoques concurrentes basados tanto en **procesos** como en **hilos**, implementados en lenguaje C, aplicando los conceptos fundamentales de concurrencia y comunicación entre procesos.

2. Objetivo General

Implementar dos programas en lenguaje C que analicen una matriz almacenada en un archivo de texto y determine, de manera concurrente con procesos e hilos, si cumple con un umbral de disparidad (porcentaje de ceros).

3. Contenido del Proyecto

El contenido del programa realizado con procesos es el siguiente:

- Lectura y representación de matrices.
- División del trabajo entre múltiples hilos POSIX.
- Sincronización y recolección de resultados mediante variables compartidas (sin necesidad de pipes).
- Cálculo y validación del porcentaje de ceros.
- Impresión de resultados en consola.
- Compilación mediante Makefile.
- Uso de la biblioteca pthread para la creación y manejo de hilos.
- Asignación equitativa de filas a cada hilo, manejando filas sobrantes.
- Espera a la finalización de todos los hilos mediante pthread_join.
- Liberación de recursos y memoria utilizada por los hilos y la matriz.

4. Desarrollo del Programa con Procesos

4.1. Archivos del Programa

- **main.c:** Contiene la función principal, validación de argumentos y coordinación de los procesos.
- **matriz.h y matriz.c:** Definen e implementan las operaciones sobre la matriz (lectura, conteo, cálculo e impresión).}
- **utilidades_proceso.h y utilidades_proceso.c:** Manejan la división del trabajo, creación de procesos y recolección de resultados mediante pipes.
- **Makefile:** Automatiza la compilación del ejecutable pdispersa.

4.2. División de la Matriz

La matriz se divide por filas entre los procesos hijos. La división es equitativa, asignando:

- filas / N: Filas a cada proceso.
- Las filas sobrantes se asignan a los primeros procesos (los de menor ID).

4.3. Comunicación entre Procesos (pipes)

Para la comunicación desde Padre a Hijo se realiza indirectamente mediante la herencia de memoria luego del fork(). Cada hijo accede a la matriz ya cargada. Por otro lado, la comunicación desde Hijo a Padre se realiza mediante pipes anónimos, cada hijo escribe su conteo de elementos diferentes de cero en un pipe, y el padre lo lee una vez el hijo ha finalizado con su ejecución.

4.4. Estructuras de Datos

Las estructuras de datos definidas fueron las siguientes:

1. **Matriz:** Estructura que almacena la matriz cargada desde un archivo.
    a. **filas:** Variable de tipo entero que almacena el número de filas de la matriz.
    b. **columnas:** Variable de tipo entero que almacena el número de columnas de la matriz.
    c. **datos:** Variable de tipo apuntador doble que almacena la matriz de enteros.
2. **TrabajoProceso:** Estructura que define el rango de filas a procesar por cada hijo.

a. **filaInicio:** Variable de tipo entero que almacena el número de fila desde donde empezará el trabajo del hijo actual.

b. **filaFin:** Variable de tipo entero que almacena el número de fila donde **terminará el trabajo del hijo actual**

c. **idProceso:** Variable de tipo entero que almacena el Id del proceso actual

4.5. Lógica General del Programa

1. El programa recibe parámetros por línea de comandos.
2. Lee la matriz desde el archivo.
3. Divide el trabajo entre procesos hijos.
4. Cada hijo cuenta los elementos distintos de cero en su rango asignado.
5. Los resultados son enviados al padre mediante pipes de comunicación.
6. El padre suma los conteos y evalúa si la matriz es dispersa.

5. Desarrollo del Programa con Hilos

5.1. Archivos del programa

- mainhilos.c : Contiene la función principal, validación de argumentos y coordinación de los hilos.
- matriz.h y matriz.c: Definen e implementan las operaciones sobre la matriz (lectura, conteo, cálculo e impresión).
- utilidades_hilo.h y utilidades_hilo.c: Manejan la división del trabajo, creación de hilos y recolección de resultados mediante variables compartidas.
- Makefile: Automatiza la compilación del ejecutable hdispersa.

5.2. División de la Matriz

La división de trabajo se hace por filas: cada hilo procesa un bloque contiguo de filas. Primero calculamos la cantidad base de filas por hilo (filas/numHilos) y luego distribuimos las filas sobrantes (filas%numHilos) una a una a los primeros hilos. Así logramos que todos los hilos tengan una carga casi idéntica y ninguno procese muchas más filas que el resto.

5.3. Comunicación entre hilos

- Todos los hilos acceden a la matriz cargada en memoria.
- Cada hilo almacena su conteo de elementos distintos de cero en una posición de un arreglo compartido o en una estructura protegida

    - El hilo principal (main) espera a que todos los hilos terminen usando pthread_join y luego suma los resultados parciales.

5.4. Estructuras de Datos

Las estructuras de datos definidas son las siguientes:

- Matriz: Estructura que almacena la matriz cargada desde un archivo.
    - filas: Variable de tipo entero que almacena el número de filas de la matriz.
    - columnas: Variable de tipo entero que almacena el número de columnas de la matriz.
    - datos: Variable de tipo apuntador doble que almacena la matriz de enteros.
- TrabajoHilo: Estructura que define el rango de filas a procesar por cada hilo.
    - filaInicio: Variable de tipo entero que almacena el número de fila desde donde empezará el trabajo del hilo actual.
    - filaFin: Variable de tipo entero que almacena el número de fila donde terminará el trabajo del hilo actual.
    - idHilo: Variable de tipo entero que almacena el Id del hilo actual.
    - resultado: Apuntador a la variable donde el hilo almacenará su conteo.

5.5. Lógica del programa

El programa recibe parámetros por línea de comandos.

1. Lee la matriz desde el archivo.
2. Divide el trabajo entre hilos.
3. Cada hilo cuenta los elementos distintos de cero en su rango asignado.
4. Los resultados son almacenados en variables compartidas.
5. El hilo principal suma los conteos y evalúa si la matriz es dispersa.

6. Plan de Pruebas para Procesos

El plan de pruebas se desarrolló con la intención de ser capaces de validar si el objetivo del proyecto se cumple exitosamente en los programas realizados, para esto se proponen diferentes enfoques los cuales permitirán un análisis y desarrollo más profundo y amplio.

6.1. Pruebas Unitarias

Pruebas destinadas a comprobar si el programa válida la dispersión o no en una matriz, junto a si es capaz de manejar errores o leer archivos, entre otras. Las imágenes de los resultados se muestran al final.

Prueba	Descripción	Resultado Esperado	Resultado Obtenido
1	Matriz 4x4, 14 ceros, 80% requerido.	Dispersa	Éxito
2	Matriz 8x4, 8 ceros, 60% requerido.	No dispersa	Éxito
3	Matriz 6x6, 32 ceros, 10% requerido.	No dispersa	Éxito
4	Matriz 200x200, 333 ceros, 75% requerido.	No dispersa	Éxito
5	Archivo inexistente	Error al abrir archivo	Éxito
6	Argumentos no válidos	Mensaje de error	Éxito
7	Argumentos en orden diferente	Ningún error	Éxito

Resultados

Prueba 1:

```
painter@Painter:~/Proyecto$ ./pdispersa -f 4 -c 4 -a matrizcorta.txt -n 1 -p 80
La matriz en el archivo tiene un total de 14 ceros (87.5%), por lo tanto, se considera dispersa.
Tiempo total de procesamiento:    186 microsegundos
```

Prueba 2:

```
painter@Painter:~/Proyecto$ ./pdispersa -f 10 -c 4 -a matrizA.txt -n 1 -p 60
La matriz en el archivo tiene un total de 8 ceros (20.0%), por lo tanto, no se considera dispersa.
Tiempo total de procesamiento:    800 microsegundos
```

Prueba 3:

```
painter@Painter:~/Proyecto$ ./pdispersa -f 6 -c 6 -a matrizB.txt -n 1 -p 10
La matriz en el archivo tiene un total de 32 ceros (88.9%), por lo tanto, se considera dispersa.
Tiempo total de procesamiento:    194 microsegundos
```

Prueba 4:

```
painter@Painter:~/Proyecto$ ./pdispersa -f 200 -c 200 -a Matriz200.txt -n 1 -p 75
La matriz en el archivo tiene un total de 333 ceros (0.8%), por lo tanto, no se considera dispersa.
Tiempo total de procesamiento:    363 microsegundos
```

Prueba 5:

```
painter@Painter:~/Proyecto$ ./pdispersa -f 200 -c 200 -a matriz200.txt -n 1 -p 75
Error al abrir el archivo: No such file or directory
```

Prueba 6:

```
painter@Painter:~/Proyecto$ ./pdispersa -f filas -c hola -a matrizcorta.txt -n procesos -p 75
Error: Argumentos inválidos
Uso: ./pdispersa -f M -c N -a Narchivo -n Nprocesos -p porcentaje
Donde:
  M: número de filas de la matriz
  N: número de columnas de la matriz
  Narchivo: nombre del archivo que contiene la matriz
  Nprocesos: número de procesos a crear
  porcentaje: porcentaje de ceros requerido (0-100)
```

Prueba 7:

```
painter@Painter:~/Proyecto$ ./pdispersa -c 200 -f 200 -p 75 -n 1 -a Matriz200.txt
La matriz en el archivo tiene un total de 333 ceros (0.8%), por lo tanto, no se considera dispersa.
Tiempo total de procesamiento:    421 microsegundos
```

6.2. Pruebas de Concurrencia

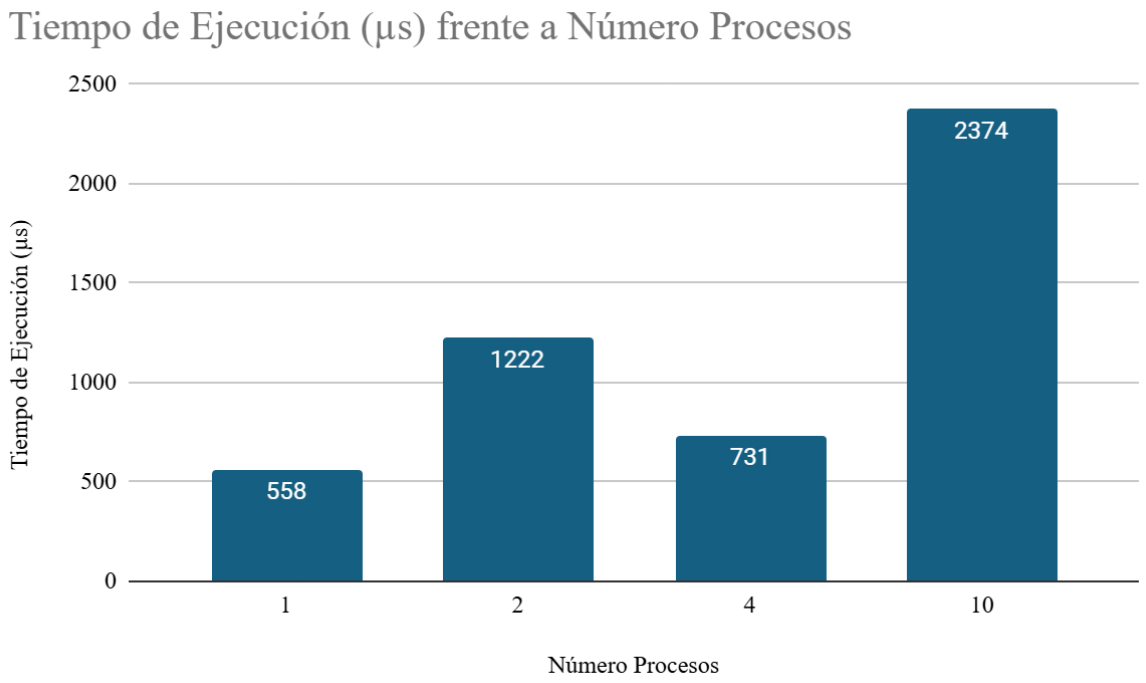
Pruebas destinadas a corroborar que el uso de más procesos o hilos aumente el rendimiento y satisfaga la eficiencia en el proyecto.

Prueba	Descripción	Procesos
1	Matriz 100x100, 500 ceros, 50% requerido.	1
2	Matriz 100x100, 500 ceros, 50% requerido.	2
3	Matriz 100x100, 500 ceros, 50% requerido.	4
4	Matriz 100x100, 500 ceros, 50% requerido.	10
5	Matriz 500x500, 125.000 ceros, 80% requerido.	1
6	Matriz 500x500, 125.000 ceros, 80% requerido.	2
7	Matriz 500x500, 125.000 ceros, 80% requerido.	4
8	Matriz 500x500, 125.000 ceros, 80% requerido.	8
9	Matriz 500x500, 125.000 ceros, 80% requerido.	16
10	Matriz 1000x1000, 500.000 ceros, 10% requerido.	1
11	Matriz 1000x1000, 500.000 ceros, 10% requerido.	4

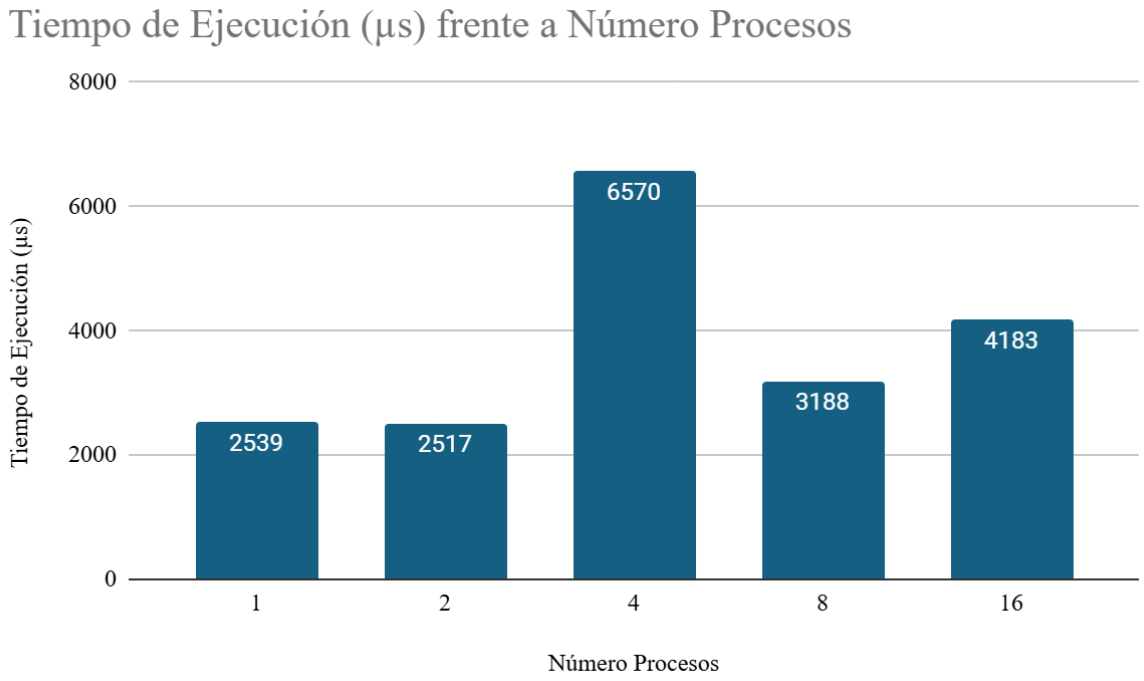
12	Matriz 1000x1000, 500.000 ceros, 10% requerido.	8
13	Matriz 1000x1000, 500.000 ceros, 10% requerido.	16
14	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	1
15	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	2
16	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	4
17	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	8
18	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	16
19	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	32
20	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	1
20	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	2
20	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	4
20	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	8

Resultados (Gráficas)

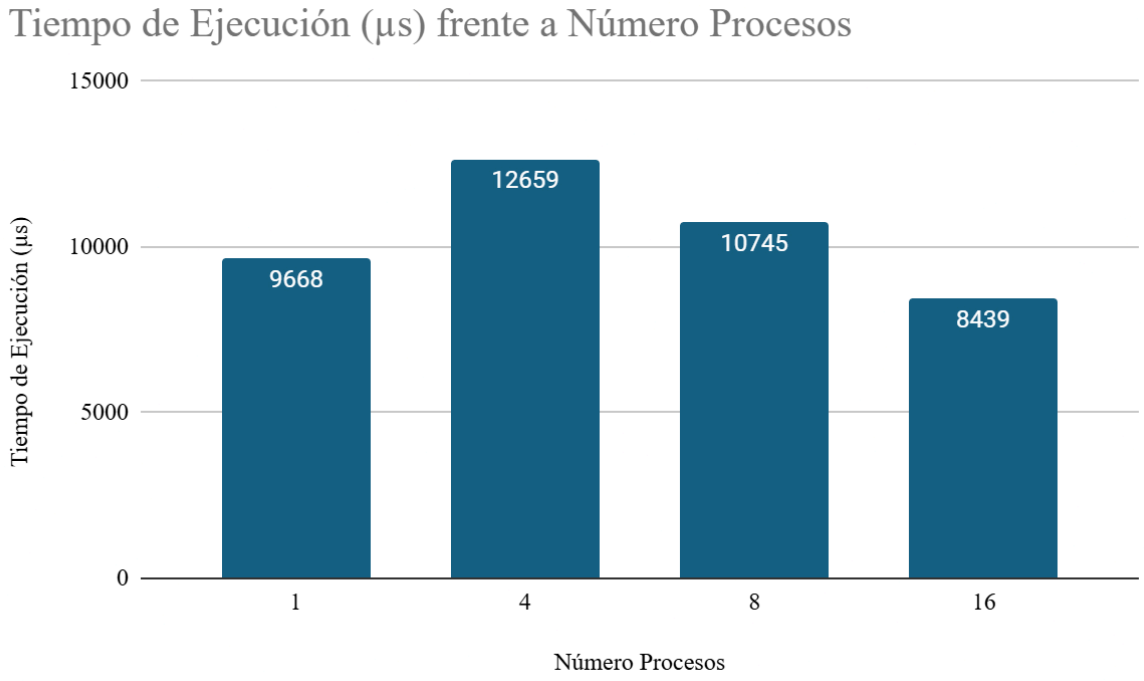
Matriz 100x100:



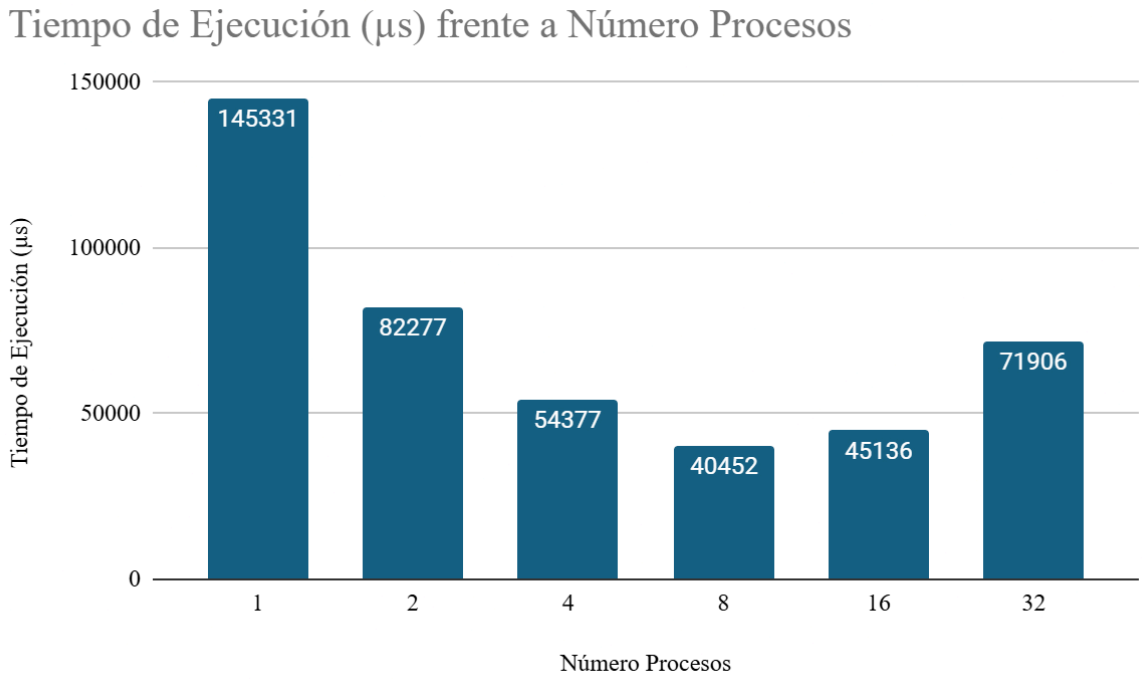
Matriz 500x500:



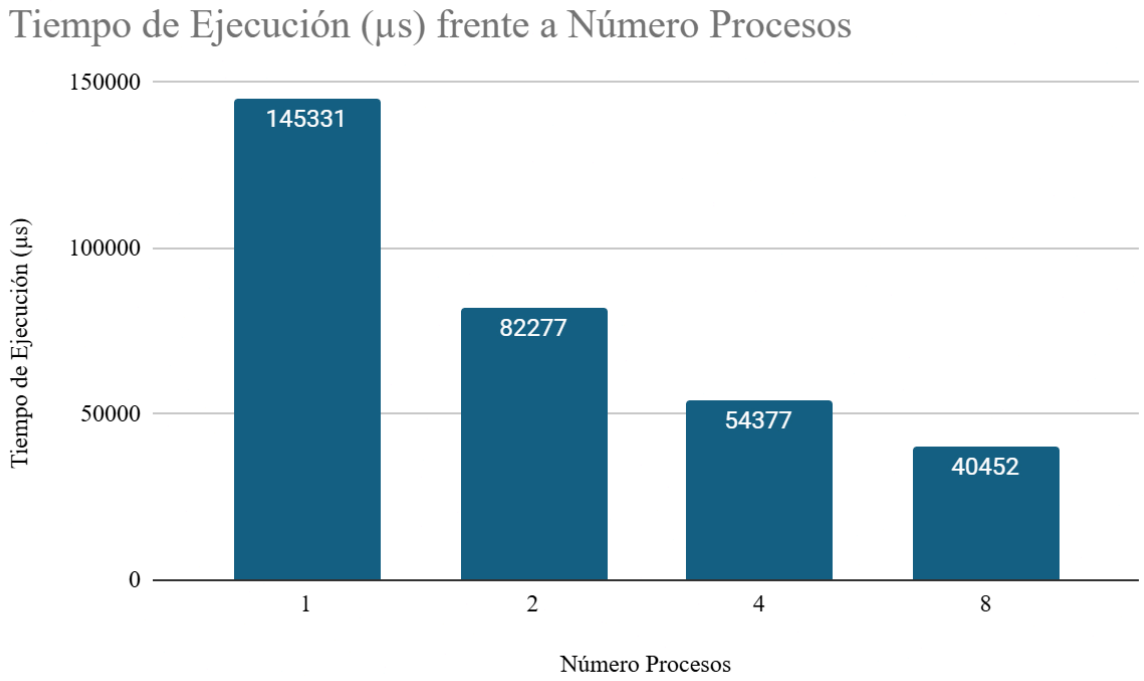
Matriz 1000x1000:



Matriz 5000x5000:



Matriz 10000x10000:



Basado en los resultados de las gráficas se puede reflexionar que el overhead que hay en el código debido a los llamados a funciones, inclusión de dependencias, uso de pipes para la comunicación y la simple creación del proceso puede jugar un

papel en contra al momento de reducir los tiempos de ejecución, puesto que en matrices pequeñas se ve la notoria falta de rendimiento en un mayor número de procesos, por lo que este programa estaría destinado a matrices de grandes dimensiones como por ejemplo 5.000x5.000 o 10.000x10.000 donde es más notoria la ventaja de tener mayor número de procesos, puesto que el beneficio en tiempo es mayor que el overhead que puede causar el crear y comunicar estos procesos.

7. Plan de pruebas para hilos

Ahora se desarrollara el mismo plan de prueba para la ejecución con hilos, para validar si el objetivo del proyecto se cumple exitosamente

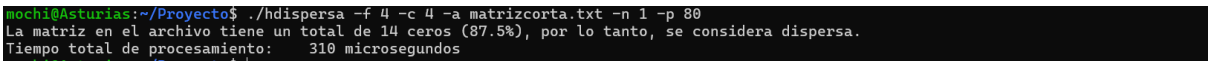
7.1. Pruebas Unitarias

Pruebas destinadas a comprobar si el programa válida la dispersión o no en una matriz, junto a si es capaz de manejar errores o leer archivos, entre otras. Las imágenes de los resultados se muestran al final.

Prueba	Descripción	Resultado Esperado	Resultado Obtenido
1	Matriz 4x4, 14 ceros, 80% requerido.	Dispersa	Éxito
2	Matriz 8x4, 8 ceros, 60% requerido.	No dispersa	Éxito
3	Matriz 6x6, 32 ceros, 10% requerido.	No dispersa	Éxito
4	Matriz 200x200, 333 ceros, 75% requerido.	No dispersa	Éxito
5	Archivo inexistente	Error al abrir archivo	Éxito
6	Argumentos no válidos	Mensaje de error	Éxito
7	Argumentos en orden diferente	Ningún error	Éxito

Resultados

Prueba 1:



Prueba 2:

```
mochi@Asturias:~/Proyecto$ ./hdispersa -f 10 -c 4 -a matrizA.txt -n 1 -p 60
La matriz en el archivo tiene un total de 8 ceros (20.0%), por lo tanto, no se considera dispersa.
Tiempo total de procesamiento:    420 microsegundos
```

Prueba 3:

```
mochi@Asturias:~/Proyecto$ ./hdispersa -f 6 -c 6 -a matrizB.txt -n 1 -p 10
La matriz en el archivo tiene un total de 32 ceros (88.9%), por lo tanto, se
 considera dispersa.
Tiempo total de procesamiento:    189 microsegundos
```

Prueba 4:

```
mochi@Asturias:~/Proyecto$ ./hdispersa -f 200 -c 200 -a Matriz200.txt -n 1 -
p 75
La matriz en el archivo tiene un total de 333 ceros (0.8%), por lo tanto, no
 se considera dispersa.
Tiempo total de procesamiento:    504 microsegundos
```

Prueba 5:

```
mochi@Asturias:~/Proyecto$ ./hdispersa -f 200 -c 200 -a matriz200.txt -n
 1 -p 75
Error al abrir el archivo: No such file or directory
Error al leer la matriz desde el archivo matriz200.txt
```

Prueba 6:

Prueba 7:

```
mochi@Asturias:~/Proyecto$ ./hdispersa -c 200 -f 200  -p 75 -a Matriz200
.txt -n 1
La matriz en el archivo tiene un total de 333 ceros (0.8%), por lo tanto
, no se considera dispersa.
Tiempo total de procesamiento:    690 microsegundos
```

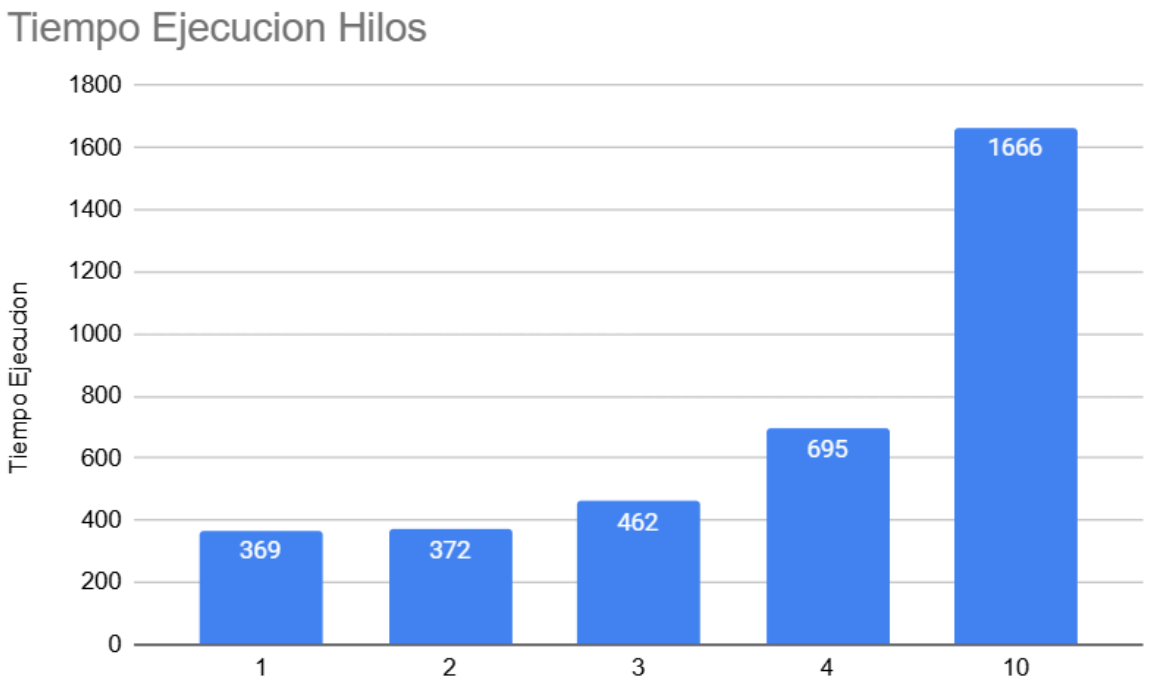
7.2. Pruebas de Rendimiento

Prueba	Descripción	Procesos
1	Matriz 100x100, 500 ceros, 50% requerido.	1
2	Matriz 100x100, 500 ceros, 50% requerido.	2
3	Matriz 100x100, 500 ceros, 50% requerido.	4
4	Matriz 100x100, 500 ceros, 50% requerido.	10
5	Matriz 500x500, 125.000 ceros, 80% requerido.	1

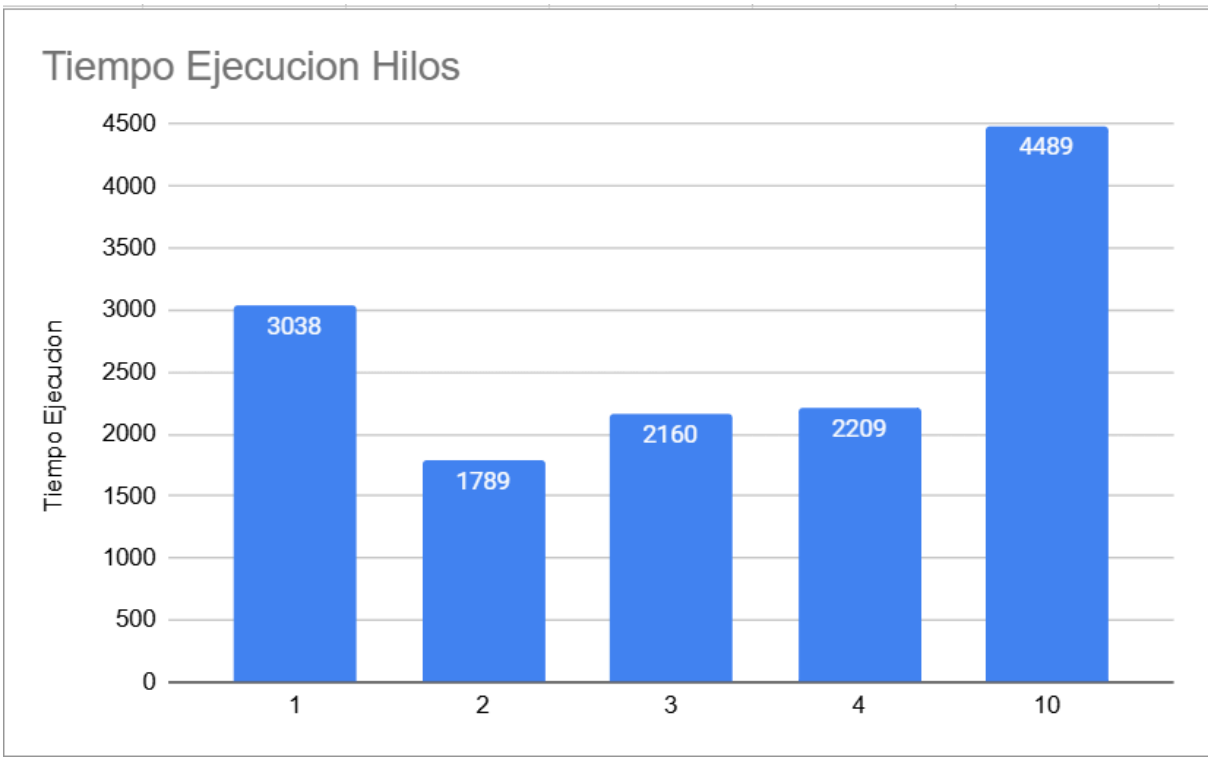
6	Matriz 500x500, 125.000 ceros, 80% requerido.	2
7	Matriz 500x500, 125.000 ceros, 80% requerido.	4
8	Matriz 500x500, 125.000 ceros, 80% requerido.	8
9	Matriz 500x500, 125.000 ceros, 80% requerido.	16
10	Matriz 1000x1000, 500.000 ceros, 10% requerido.	1
11	Matriz 1000x1000, 500.000 ceros, 10% requerido.	4
12	Matriz 1000x1000, 500.000 ceros, 10% requerido.	8
13	Matriz 1000x1000, 500.000 ceros, 10% requerido.	16
14	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	1
15	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	2
16	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	4
17	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	8
18	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	16
19	Matriz 5000x5000, 12.500.000 ceros, 25% requerido.	32
20	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	1
21	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	2
22	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	4
23	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	8
24	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	16
25	Matriz 10000x10000, 25.000.000 ceros, 80% requerido.	32

Gráficas para el rendimiento:

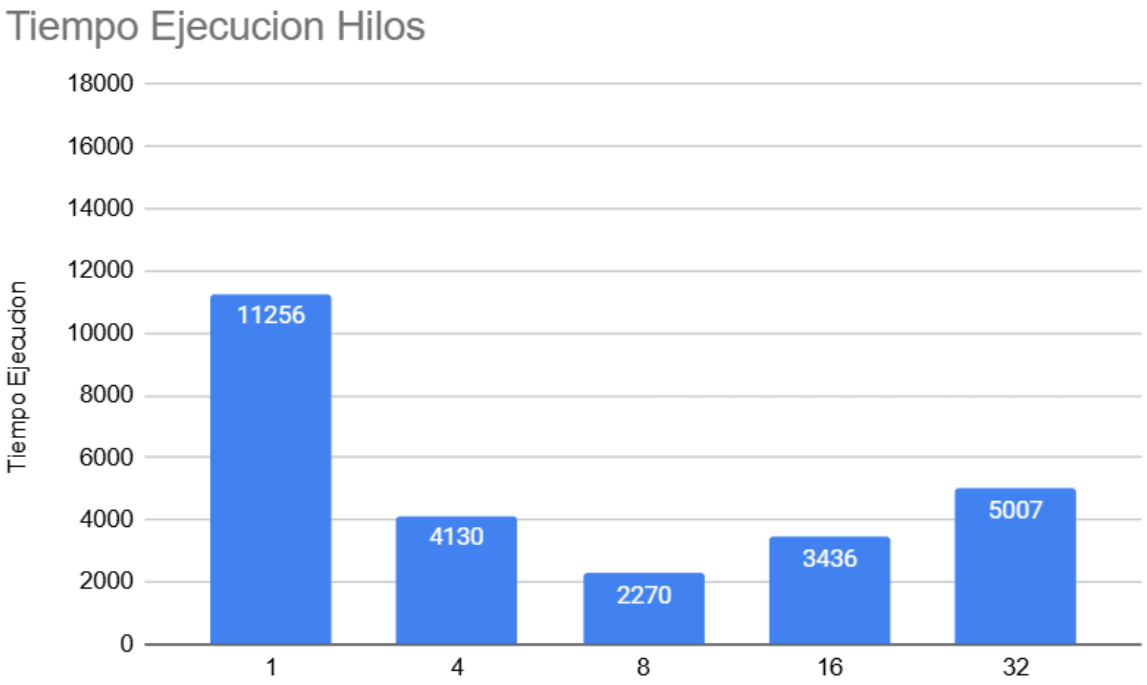
Matriz 100x100:



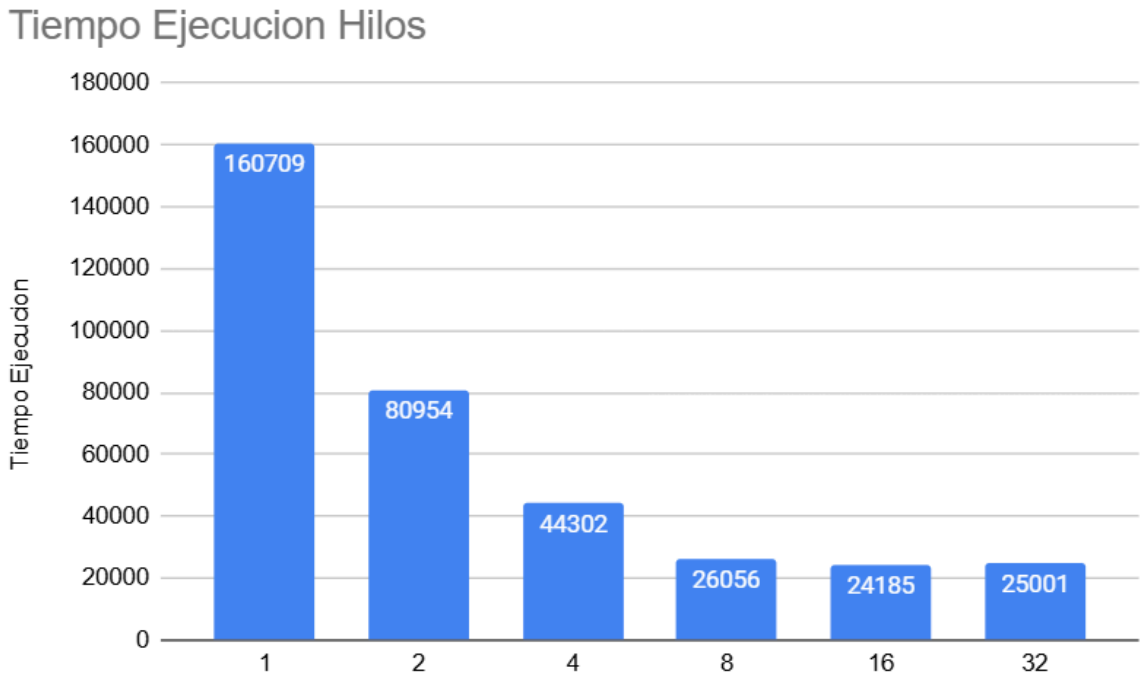
Matriz 500x500:



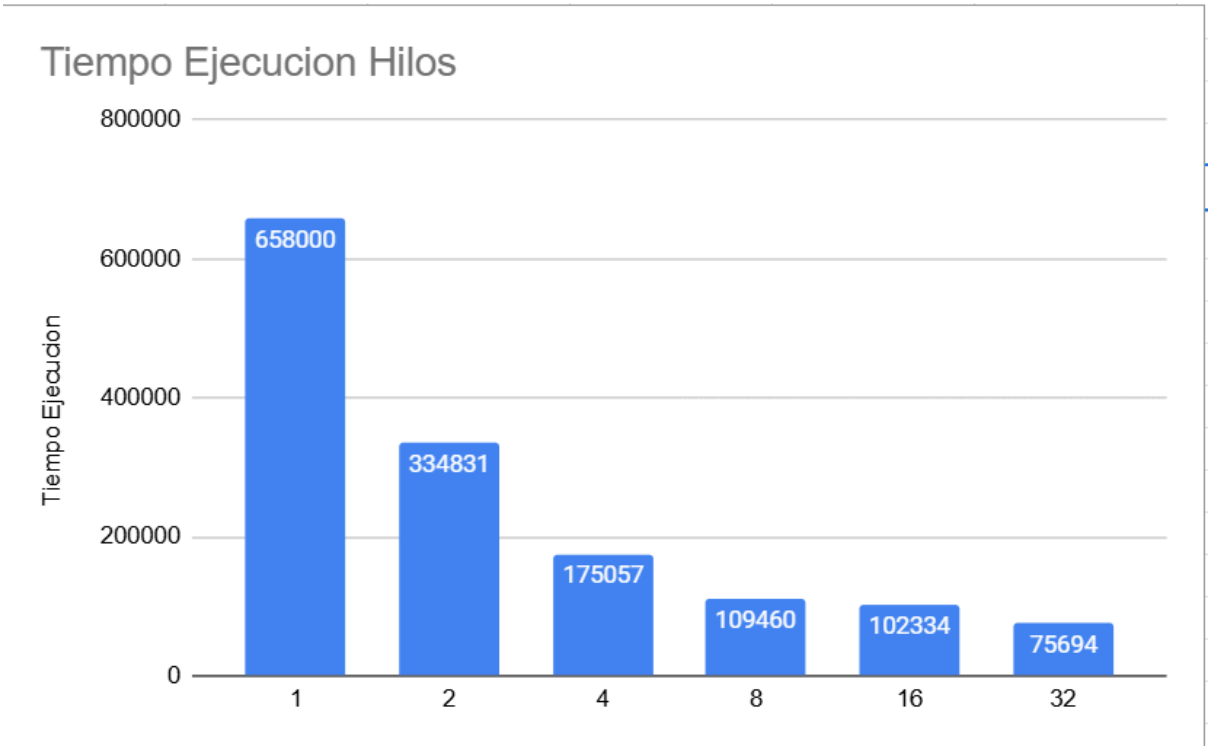
Matriz 1000x1000



Matriz 5000x5000



Matriz 10000x10000



Los resultados obtenidos a partir de las ejecuciones reafirman lo planteado anteriormente. Se observa de manera consistente que, en matrices de dimensiones reducidas, el uso de múltiples divisiones para concurrencia no representa una mejora significativa e incluso puede perjudicar el rendimiento debido al overhead asociado. No obstante, al aumentar considerablemente el tamaño de las matrices, se evidencia una ganancia sustancial en los tiempos de ejecución al incrementar la concurrencia, como en este caso que se logró con hilos.

8. Conclusiones

A partir del desarrollo del proyecto y del análisis de los resultados obtenidos en las distintas pruebas realizadas, se puede concluir que el uso de concurrencia, ya sea mediante procesos o hilos, representa una herramienta poderosa para el procesamiento de matrices de gran tamaño. No obstante, también se evidencia que el desempeño no siempre mejora con el simple aumento del número de unidades de ejecución.

En particular, en matrices pequeñas, el overhead generado por la creación de procesos, el uso de pipes y la comunicación entre ellos puede superar los beneficios del paralelismo, resultando incluso en un peor rendimiento. Por otro lado, en matrices de grandes dimensiones (por ejemplo, 5.000×5.000 o 10.000×10.000), se logra una reducción significativa en los tiempos de ejecución al utilizar múltiples procesos o hilos, justificando así el uso de técnicas concurrentes.

Asimismo, se observó que el enfoque con hilos, al compartir memoria directamente, puede resultar más liviano en cuanto a consumo de recursos, eliminando la necesidad de

comunicación explícita por pipes, aunque requiere una correcta sincronización para evitar condiciones de carrera.

Finalmente, el proyecto permitió aplicar y reforzar conceptos clave del curso de Sistemas Operativos como concurrencia, sincronización, planificación y comunicación entre procesos, dejando en claro que la elección de una técnica sobre otra debe depender del contexto del problema y de los recursos disponibles.