UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio 2 - Paralelismo orientado al objeto

Nombre: John Serrano Carrasco

Curso: Sistemas Distribuidos y Paralelos

Sección: 13329-0-A-1

Profesor: Fernando Rannou Fuentes

22 de Octubre de 2023

Tabla de contenidos

1. Introducción

El siguiente informe tiene como objetivo principal apoyar y explicar las soluciones creadas para el laboratorio 2, donde se utiliza uC++ para realizar paralelismo orientado al objeto. Se incluyen dos archivos de solución; el primero, **gridding.cc** corresponde a la solución con matriz compartida, mientras que el archivo **gridding_matriz_propia.cc** corresponde a la solución donde cada tarea tiene su matriz propia.

La estructura del informe es la siguiente: En primer lugar, se realiza una breve descripción de los objetos utilizados en ambas soluciones. Luego, se explican las dos soluciones implementadas. Posterior a eso, se muestran los resultados, donde se revela cual fue la imagen resultante obtenida, los tiempos de ejecución de ambas soluciones con diferentes cantidad de tareas y chunks en un computador personal y en el cluster del DIINF, junto con algunas conclusiones generales.

2. Objetos

A continuación, se realiza una breve descripción de los objetos utilizados en la solución de matriz propia y solución de matriz compartida.

2.1. Reader

De tipo **_Mutex _Coroutine.** Al ser de tipo Coroutine, no tiene una hebra de ejecución propia pero si un estado de ejecución. Cuando una tarea entra a la Corrutina, esta encarga de leer el archivo de entrada y obtener las lineas leídas para su procesamiento. Al ser de tipo Mutex, todos sus miembros públicos tienen exclusión mutua. Cuenta con un atributo ifstream que es el archivo de entrada. Además, cuenta con un atributo chunkSize que indica la cantidad de lineas que se le entregaran a cada tarea y un atributo numberOfTasks que indica la cantidad de tareas que creadas. La clase cuenta con un método **siguiente** que es el encargado de entregar las lineas leídas y activar el main de la corrutina y un método **main** que es el encargado de leer el archivo y guardar las lineas leídas para que la tarea que está dentro de la Corrutina pueda guardarlas en su propio estado de ejecución.

2.2. Gridder

De tipo **_Task.** Al ser de tipo Task, tiene una hebra propia, un estado de ejecución propia y exclusión mutua. Es la encargada de procesar los datos entregados por la Corrutina Reader. Cuenta con un atributo id que indica el id de la tarea, un atributo r que es un puntero a la Corrutina Reader, un atributo delta_x que indica el tamaño de la celda y un atributo N que indica el tamaño de la matriz La clase cuenta con un método **main** que es el encargado de procesar los datos entregados por la Corrutina Reader. En la solución de matriz compartida, tiene un atributo m que es un puntero a la clase Matrix. En la solución de matriz propia, crea una matriz para la parte real, la parte imaginaria y el peso en su constructor.

2.3. Matrix

De tipo _**Mutex Class.** Al ser de tipo Mutex, todos sus miembros públicos tienen exclusión mutua. Es la encargada de almacenar los datos procesados por las Tareas. Esta de es de acceso compartido, por lo que todas las tareas pueden acceder a ella. La clase cuenta con 3 matrices de tipo double, una para la parte real, otra para la parte imaginaria y otra para el peso. Además, cuenta con un atributo N que indica el tamaño de la matriz. Esta clase solo se utiliza en la solución de la matriz compartida.

3. La solución

En primer lugar luego de recibir los argumentos de la linea de comando mediante getopt, se inicializa y abre el archivo de entrada. Se verifica que este se haya abierto correctamente, se inicializa la Corrutina Reader y luego se calcula el valor de delta x en radianes.

3.1. Solución con matriz compartida

Se inicializa la clase Matrix, donde se crean la matriz real, matriz imaginaria y matriz peso. Se crea un arreglo de tareas, dando espacio a la cantidad de tareas indica por la linea de comando y se procede a crear las tareas.

Cada tarea tiene un vector de string público, llamado lineas, donde se guardan las lineas a procesar cuando la tarea ya ha leído su chunk correspondiente. Cuando la tarea es creada, ejecuta inmediatamente su main, e ingresa a un while(true) hasta que se hayan leído y procesado todas las lineas del archivo.

Cuando la tarea ingresa al ciclo while, realiza un llamado al método siguiente() de la Corrutina Reader. Este método verifica si se ha llegado al final del archivo y si no es así, ejecuta **resume()**, lo cual ejecuta el main de la Corrutina. En este main, se inicializa un contador de lineas leídas y se ingresa a un ciclo while(true). Dentro de este ciclo, se utiliza getline para obtener una linea del archivo y luego esta linea se guarda en un vector de strings. Se aumenta el contador de lineas leídas y se repite el proceso hasta que se llegue a la cantidad del chunk, donde se aplica **suspend()** y se retornan las lineas a la tarea, la cual sale

de la Corrutina. La tarea procesa estas lineas y realiza los cálculos correspondientes, para guardar los resultados en las matrices. Se repite el proceso hasta que ya no quedan lineas por leer, **donde se retorna un vector vació para indicar que ya no quedan mas lineas por leer y cuando las tareas verifican que se está entregando un vector vació, comienzan a romper el ciclo while y terminar su ejecución.**

En el main principal, se realiza la normalización de las matrices y luego se crean los archivos de la parte real y la parte imaginaria. La solución utiliza delete para eliminar toda la memoria reservada y finaliza su ejecución. El tiempo de ejecución es calculado desde antes de la creación de las tareas hasta antes de la escritura de los archivos de salida, utilizando **clock().**

Los archivos de salida de esta solución son **datosgrideadosr.raw** y **datosgrideadosi.raw**.

3.2. Solución con matriz propia

Si bien la solución con matriz propia es similar a la solución anterior, esta vez las matrices son parte de cada tarea, por lo que al momento de crear una tarea, **esta se crea con la matriz real, matriz imaginaria y matriz peso que solo ella puede acceder.** No existe la clase Matrix en esta solución, dado lo anterior.

Cuando las tareas están procesando, en vez de llamar a un getter() para acceder a la matriz Mutex correspondiente, simplemente realizan el guardado de información dentro de su propia matriz. En el main principal, se crean las tres matrices, que serán las matrices finales y resultantes. Por cada tarea, las matrices propias se van sumando y guardando en las matrices finales. Una vez que ya se realizó todo el proceso anterior, se realiza la normalización de las matrices y se escriben los datos finales.

Los archivos de salida de esta solución son **datosgrideadosr_mp.raw** y **datosgrideadosi_mp.raw**.

4. Resultados

Los siguientes resultantes fueron alcanzados utilizando los siguientes datos de entrada: Archivo de entrada = **hltau_completo_uv.csv**, N = **2048** y d = **0.003**.

4.1. Imagen resultante

Una vez que se tienen los archivos de salida de la parte real y la parte imaginaria, se puede crear un código de MATLAB para comprobar la solución. Tras cargar los archivos en MATLAB y ejecutar el código, la imagen resultante tras aplicar ambas soluciones es la siguiente:

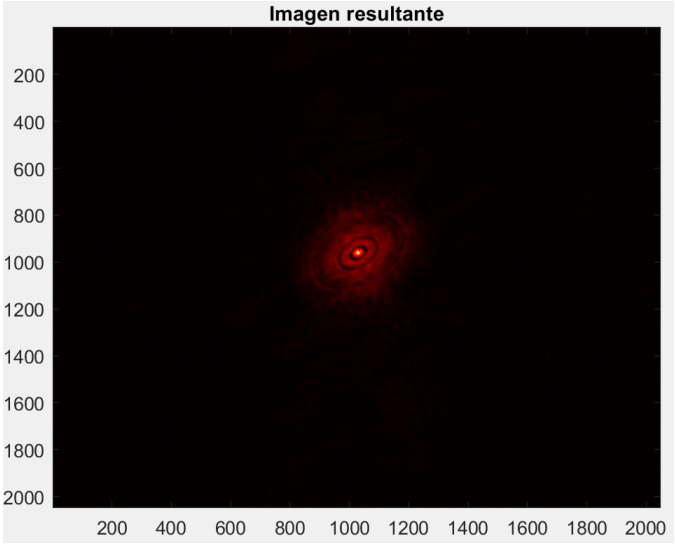

Figura N°1: Imagen resultante en MATLAB.

4.2. Rendimiento Computacional

Se realizaron pruebas con ambas soluciones en un computador personal y en el cluster del DIINF. Cabe destacar que ambas soluciones se compilaron utilizando u++ -multi -nodebug.

4.2.1. Computador personal

Se realizaron distintas pruebas en un computador con un procesador AMD Ryzen 9 5900HS with Radeon Graphics, 3301 Mhz, 8 procesadores principales, 16 procesadores

lógicos, corriendo una maquina virtual que ejecuta Ubuntu 22.04.3 LTS.

Se hicieron pruebas con 1, 5, 10 y 15 tareas probando chunks de 1, 10, 100, 1000, 10000 y 100000. Se registraron los tiempos del proceso de gridding en segundos de ambas soluciones, los cuales se pueden observar en la tabla 1.

N° Tasks	N° Chunks	Tiempo Matriz Compartida [s]	Tiempo Matriz Propia [s]
1	1	6.39377	6.69602
1	10	5.44312	5.5607
1	100	5.11441	5.3481
1	1000	5.12081	5.23077
1	10000	5.19956	5.28603
1	100000	5.23215	5.40605
5	1	7.74023	9.45979
5	10	7.24689	8.13278
5	100	7.05438	7.84834
5	1000	7.56173	7.82421
5	10000	7.06645	7.57552
5	100000	7.81759	8.29296
10	1	9.28042	11.9065
10	10	9.1834	10.5098
10	100	8.30758	10.1906
10	1000	7.53552	9.979
10	10000	7.81784	10.963
10	100000	8.17751	11.0699
15	1	8.90177	14.3064
15	10	8.12291	12.6067
15	100	7.12405	12.4103
15	1000	6.20907	12.2355
15	10000	6.83886	12.1195
15	100000	8.26274	12.7088

Tabla N°1: Tiempos de ejecución del proceso de gridding diferentes cantidades de tareas y chunks.

Se puede observar que el tiempo de ejecución aumenta a medida que aumentan la cantidad de tareas. Si bien esto puede parecer contraproducente, es probable que dado los mecanismos de concurrencia de uC++, se sacrifique tiempo de ejecución para evitar problemas de exclusión mutua, dado que para el caso del laboratorio, solo una tarea puede acceder a la corrutina a obtener sus lineas, lo que conlleva que haya una demora que aumenta cada vez que hay mas tareas.

Para el caso de la solución con matriz compartida, el mejor tiempo fue 1 tarea y 100 chunks, mientras que el peor fue con 10 tareas y 1 chunk. Para la solución con matriz propia, el mejor tiempo fue con 1 tarea y 1000 chunk, mientras que el peor fue 15 tareas con 1 chunk. Por lo general los tiempos de la matriz compartida fueron mejor que los tiempos con matriz propia, esto debido a que cada tarea tiene su propia matriz creados con ciclos for y al final del proceso de gridding, todas las matrices deben sumarse para obtener las matrices finales, lo cual conlleva un aumento del tiempo de ejecución. Cabe destacar que los tiempos de ejecución tienen una variación de 1 a 2 segundos promedio, donde los tiempos de la tabla 1 son los mejores tras probar un grupo de tareas y chunks específicos 3 veces.

4.2.2. Cluster del Departamento de Ingeniería en Informática

En el cluster del DIINF se hicieron pruebas con 1, 5, 10 y 15 tareas, utilizando 1000 chunks para ambas soluciones. A continuación se presenta la tabla que contiene los tiempos de ejecución del proceso de gridding (Se adjuntan pruebas en anexo).

De la tabla anterior se puede ver que de manera general, los tiempos de ejecución del proceso de gridding son mucho mejores que los tiempos obtenidos en el computador personal, dada las capacidades del cluster. Para el caso de la matriz compartida, el mejor tiempo fue el de 1 tarea, mientras que el peor fue de 15 tareas. Lo mismo sucede en el caso de la matriz propia. Nuevamente, los tiempos de matriz compartida son menores a los de matriz propia.

N° Tareas	Tiempo Matriz Compartida [s]	Tiempo Matriz Propia [s]
1	4.11048	4.24261
5	6.46687	6.93114
10	6.77456	8.9984
15	7.02456	10.8971

Tabla N°2: Tiempos de ejecución del proceso de gridding diferentes cantidades de tareas y chunks en el cluster del DIINF.

4.3. Sobre los resultados en general

Si bien los tiempos de ejecución son relativamente buenos, si hubiera alguna forma de mejorar la parte de la leída del archivo y no tener problemas de exclusión mutua a la vez, probablemente los tiempos de ejecución serian mejores con una mayor cantidad de tareas. Dada la solución planteada en el enunciado del laboratorio, lamentablemente, no es posible con la solución actual, lo que conlleva a un speedup negativo, dado que el tiempo es mejor con 1 tarea que con 2 o más tareas.

Aún así, es interesante notar de que hay casos donde con una mayor cantidad de tareas y chunks, se obtuvo un tiempo menor, como es el caso que se puede apreciar en la tabla 1 de 15 tareas y 1000 chunks comparado con 5 tareas y 1000 chunks en la matriz compartida, por lo que no siempre al aumentar las tareas y probar con una cantidad de chunk fija puede implicar que el tiempo aumente.

NOTA: Por algún motivo, al compilar y ejecutar el código en el cluster, se obtenía un error de "Division by zero". El error se arregló cambiando el valor de delta_x de 0.003 a 0,003, como se puede apreciar en el anexo. Lo anterior no sucede en el computador personal.

5. Conclusión

En conclusión, se logró cumplir con el objetivo principal de la experiencia, que era lograr crear dos soluciones utilizando paralelismo orientado al objeto con uC++ para recrear un proceso de gridding, utilizando el archivo **hltau**. Se logró probar ambas soluciones en el cluster del DIINF, lo cual complementa los resultados obtenidos. Si bien el speedup es negativo, los tiempos de ejecución son relativamente buenos y se logró obtener la imagen resultante esperada.

Se espera que lo aprendido en este laboratorio sirva para otras experiencias donde haya que trabajar con paralelismo, como también para complementar los contenidos sobre uC++, especialmente, Corrutinas y Tareas.

6. Anexos

```
jserrano@xicpu01:~/test$ make run
./gridding -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 1
Tarea creada: 0
Tiempo del proceso de gridding con matriz compartida: 4.11048[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°2: Prueba en el cluster - Matriz Compartida, 1 tarea, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run2
./gridding -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 5
Tarea creada: 0
Tarea creada: 1
Tarea creada: 2
Tarea creada: 3
Tarea creada: 4
Tiempo del proceso de gridding con matriz compartida: 6.46687[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°3: Prueba en el cluster - Matriz Compartida, 5 tareas, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run3
./gridding -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 10
Tarea creada: 0
Tarea creada: 1
Tarea creada: 2
Tarea creada: 3
Tarea creada: 4
Tarea creada: 5
Tarea creada: 6
Tarea creada: 7
Tarea creada: 8
Tarea creada: 9
Tiempo del proceso de gridding con matriz compartida: 6.77456[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°4: Prueba en el cluster - Matriz Compartida, 10 tareas, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run4
./gridding -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 15
Tarea creada: 0
Tarea creada: 1
Tarea creada: 2
Tarea creada: 3
Tarea creada: 4
Tarea creada: 5
Tarea creada: 6
Tarea creada: 7
Tarea creada: 8
Tarea creada: 9
Tarea creada: 10
Tarea creada: 11
Tarea creada: 12
Tarea creada: 13
Tarea creada: 14
Tiempo del proceso de gridding con matriz compartida: 7.02456[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°5: Prueba en el cluster - Matriz Compartida, 15 tareas, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run5
./gridding_matriz_propia -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 1
Tarea creada: 0
Tiempo del proceso de gridding con matriz propia: 4.24261[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°6: Prueba en el cluster - Matriz Propia, 1 tarea, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run6
./gridding_matriz_propia -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 5
Tarea creada: 0
Tarea creada: 1
Tarea creada: 2
Tarea creada: 3
Tarea creada: 4
Tiempo del proceso de gridding con matriz propia: 6.93114[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°7: Prueba en el cluster - Matriz Propia, 5 tareas, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run7
./gridding_matriz_propia -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 1
0
Tarea creada: 0
Tarea creada: 1
Tarea creada: 2
Tarea creada: 3
Tarea creada: 4
Tarea creada: 5
Tarea creada: 6
Tarea creada: 7
Tarea creada: 8
Tarea creada: 9
Tiempo del proceso de gridding con matriz propia: 8.9984[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°8: Prueba en el cluster - Matriz Propia, 10 tareas, 1000 chunk.

```
jserrano@xicpu01:~/test$ make run8
./gridding_matriz_propia -i /home/XI/sdp22023_data/hltau_completo_uv.csv -o datosgrideados -d 0,003 -N 2048 -c 1000 -t 1
5
Tarea creada: 0
Tarea creada: 1
Tarea creada: 2
Tarea creada: 3
Tarea creada: 4
Tarea creada: 5
Tarea creada: 6
Tarea creada: 7
Tarea creada: 8
Tarea creada: 9
Tarea creada: 10
Tarea creada: 11
Tarea creada: 12
Tarea creada: 13
Tarea creada: 14
Tiempo del proceso de gridding con matriz propia: 10.8971[s]
Archivo de salida de la parte real creado
Archivo de salida de la parte imaginaria creado
```

Figura N°9: Prueba en el cluster - Matriz Propia, 15 tareas, 1000 chunk.