Reto 1 - Ecuación de Poisson unidimensional mediante el método de Jacobi

Presentado por:
Santiago Sosa Herrera

Presentado a:
Ramiro Andrés Barrios Valencia

HPC: High Performance Computing
Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira
2023

Tabla de contenido

Resumen **4**
Introducción **5**
Marco Conceptual **6**
High Performance Computing 6
Jacobi 6
Complejidad Computacional 6
Programación paralela 6
Hilos 6
Speedup 6
Marco Contextual **7**
Características de la máquina 7
Desarrollo 7
Pruebas 8
Graficas comparativas 9
Conclusiones 10
Bibliografía **11**

Resumen

El trabajo consiste en la optimización de un programa en lenguaje C que resuelve una ecuación de Poisson unidimensional mediante el método de Jacobi. La ecuación de Poisson es una ecuación diferencial parcial que se utiliza en muchos campos, como la física, la ingeniería y las matemáticas. El método de Jacobi es un método iterativo que se utiliza para resolver ecuaciones diferenciales parciales y que se puede implementar en paralelo para acelerar el proceso de cálculo.

El objetivo del trabajo es optimizar el programa para reducir el tiempo de ejecución y compararlo con una implementación del mismo programa utilizando hilos y procesos. Para medir el tiempo de ejecución, se utilizarán herramientas de medición de tiempo en C. Se espera que la implementación con hilos reduzca el tiempo de ejecución en comparación con la implementación secuencial original.

Se necesitará experiencia en el uso de herramientas de medición de tiempo y en la interpretación de los resultados de la medición de tiempo. También será importante tener habilidades para analizar y comparar el rendimiento de diferentes implementaciones del mismo programa.

Al final del trabajo, se incluye la descripción de las técnicas de optimización utilizadas, los resultados de la medición de tiempo y las conclusiones sobre el rendimiento de las diferentes implementaciones del programa. El informe también puede incluir recomendaciones para futuras mejoras en el programa.

Introducción

La ecuación de Poisson unidimensional es una ecuación diferencial parcial que se utiliza para modelar una amplia variedad de fenómenos físicos y matemáticos en una dimensión. En particular, la ecuación de Poisson es útil en la modelación de campos eléctricos, gravitatorios y de temperatura.

El método de Jacobi es un método iterativo que se utiliza para resolver ecuaciones diferenciales parciales como la ecuación de Poisson. Este método funciona al actualizar los valores de las soluciones en cada punto de la malla en cada iteración, utilizando los valores de la iteración anterior. Este proceso se repite hasta que se cumple un criterio de convergencia.

La optimización con hilos es una técnica que se utiliza para acelerar la ejecución de programas paralelos. La idea es dividir la carga de trabajo entre múltiples hilos de ejecución, de modo que cada hilo pueda procesar una parte del problema. En el caso del método de Jacobo, cada hilo puede actualizar los valores de la solución en una sección diferente de la malla. Esto puede mejorar significativamente el tiempo de ejecución del programa.

Sin embargo, la implementación del método de Jacobi con hilos puede presentar algunos desafíos. En particular, es importante asegurarse de que los hilos no se interfieran entre sí al actualizar los valores de la solución en la malla. Esto puede requerir el uso de técnicas de sincronización, como semáforos o mutex.

La optimización del método de Jacobi con hilos puede ser un problema complejo y requiere conocimientos avanzados de programación en C y técnicas de programación paralela. También es importante tener experiencia en la medición y análisis del rendimiento del programa, para poder identificar y solucionar cuellos de botella en la implementación. Sin embargo, la implementación exitosa del método de Jacobi con hilos puede resultar en una mejora significativa en el tiempo de ejecución del programa, lo que puede ser beneficioso en una amplia variedad de aplicaciones.

Marco conceptual

High Performance Computing: High Performance Computing es un área de la informática que se dedica a crear sistemas informáticos y software que puedan manejar tareas de alta complejidad y procesamiento de grandes volúmenes de datos en un período de tiempo muy corto.

Jacobi: El método de Jacobi es un algoritmo iterativo utilizado para resolver sistemas de ecuaciones lineales. Es particularmente útil en la resolución de ecuaciones diferenciales parciales y ha sido utilizado en una amplia variedad de campos como la física, la ingeniería y las matemáticas.

Complejidad Computacional: La complejidad computacional es una forma de medir los recursos que se necesitan para resolver un problema computacional. Sirve para determinar la dificultad de un problema y para comparar la eficiencia de distintos algoritmos.

Programación paralela: La programación paralela es una técnica de programación que se basa en dividir un problema en tareas más pequeñas y ejecutarlas en múltiples procesadores al mismo tiempo. Esta técnica se utiliza para mejorar la eficiencia de los sistemas informáticos y acelerar la resolución de problemas.

Hilos: Los hilos son una forma de dividir la ejecución de un programa en unidades más pequeñas y manejables, lo que permite que varias tareas se realicen simultáneamente. Cada hilo funciona de manera independiente, con su propia pila de memoria y compartiendo recursos con otros hilos dentro del mismo proceso. El uso de hilos puede mejorar significativamente la eficiencia y velocidad de ejecución de una aplicación.

Speedup: El speedup se utiliza para medir la mejora en la velocidad de ejecución de un programa cuando se ejecuta en un sistema de cómputo más rápido o en paralelo. Se mide en términos de una relación entre el tiempo que tarda un programa en ejecutarse en un sistema de cómputo determinado y el tiempo que tarda en ejecutarse en un

sistema más rápido. Cuanto mayor sea el speedup, mayor será la mejora en el rendimiento.

Marco Contextual

Características de la maquina

Las características del pc donde se realizaron las pruebas son los siguientes:

- Procesador AMD A8-7410 apu with amd Radeon r5 graphics x 4
- Memoria 6,7 GiB
- SO 64 bits
- Disco de 295 GB
- Sistema operativo Linux Mint 19
    - Gnome 3.28.2

Desarrollo

Para el trabajo básicamente se estudió el código que tiene la solución del método de Jacobi para poder optimizarlo, para ello se empezó con el método de hilos para el cual se utilizó la biblioteca pthreads de C. En este caso, la estrategia que se podría usar es dividir el cálculo de la solución en partes iguales y asignar cada parte a un hilo diferente. Los hilos calcularían sus partes y al final se combinarían para obtener la solución completa.

Para hacer esto, se podría crear una función jacobi_thread que realiza una sola iteración de Jacobi para una sección específica de la malla. Esta función tomaría como argumentos un puntero al arreglo de la solución, un puntero al arreglo del lado derecho de la ecuación, el número de iteraciones que deben realizarse en esta sección y el índice de la primera celda de la sección en la malla. Dentro de la función, se iteraría el número de veces especificado, calculando la solución para cada celda en la sección.

Luego, en la función principal, se crearían varios hilos y se asignaría cada sección de la malla a un hilo diferente. Cada hilo llamaría a la función jacobi_thread con los

argumentos correspondientes. Después de que todos los hilos hayan terminado, la solución completa estaría disponible en el arreglo.

En el caso de los procesos se puede crear un proceso padre que se encargue de dividir la tarea en varias subtareas y distribuirlas entre los procesos hijos. Cada proceso hijo realizaría una parte de la tarea, y el proceso padre se encargaría de recolectar los resultados de cada proceso hijo y combinarlos para obtener el resultado final.

Pruebas

1. Resultados de ejecutar el algoritmo en forma secuencial, con 2 hilos y con 8 procesos: (tener en cuenta que el programa esta compilado con O3)

Secuencial	Hilos	Procesos
0,418756	0,763606	0,001455
0,483259	0,764882	0,001267
0,470164	0,722973	0,001412
0,487788	0,772238	0,001552
0,443308	0,758729	0,001541
0,463342	0,745735	0,001558
0,458632	0,74743	0,001447
0,4586	0,763233	0,001713
0,456892	0,741627	0,001586

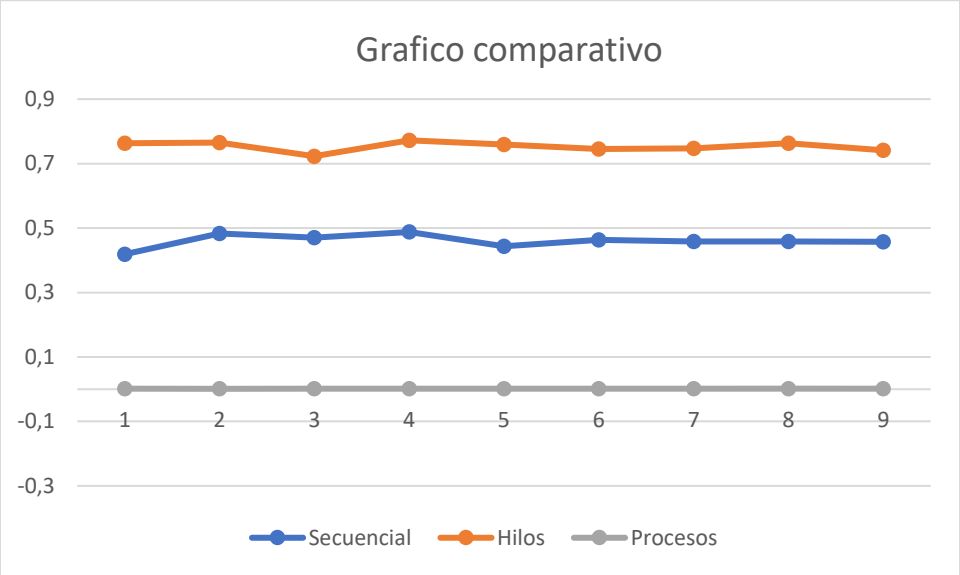
	Promedio
Secuencial	0,46008233
Hilos	0,75338367
Procesos	0,00150344

	Speedup
Hilos	0,610688
Procesos	306,0188

Que podemos ver en estos resultados, primero un interesante caso en el que el sistema en el que se esta probando parece ser que el multithreading no lo maneja bien y de hecho empeora el rendimiento de los cálculos, si se escogieron solo 2 hilos es porque se veía que a medida que se aumentan los hilos se empeora el rendimiento, algo ya probado también en el proyecto de multiplicación de matrices; por otro lado, se puede observar que el método por procesos si funciona.

Graficas comparativas

Para ver un poco mejor y poder ver las diferencias entre las implementaciones, las vemos comparadas en gráficos para poder diferenciarlo de otra manera.



No vale ni la pena ver la diferencia de Speedup entre los hilos y procesos, claramente los procesos en este ambiente de desarrollo es mucho mejor, aunque cabe aclarar que en otros no es así, como por ejemplo en el C online.

Conclusiones

1. Este trabajo podría decir que el uso de hilos podría no ser verdaderamente efectivo por que claramente podemos ver que es mucho peor que hacerlo secuencialmente, pero no podemos decir lo mismo de los procesos.

2. A medida que más hilos se utilizaban más empeora el rendimiento, por lo que puedo concluir que el sistema y el ambiente en el que se está probando quizás no eran el más adecuado, ya que en otros sistemas como el compilador online de C nos dice otra cosa con respecto a los tiempos, mostrando ser muchísimo mejor que la forma secuencial

3. A pesar de que no se implementaron otras formas de optimización aparte de la que se hizo por consola con O3, se podría mejorar el rendimiento del código con otras formas como:

- Almacenamiento en caché temporal: La asignación y des asignación de memoria para utmp en cada iteración del bucle también puede ser costosa. En su lugar, se puede asignar una matriz utmp de tamaño n+1 y reutilizarla en cada iteración, lo que reduce la sobrecarga de la asignación y des asignación de memoria.

- Uso de tipos de datos de tamaño fijo: En lugar de usar el tipo de datos de punto flotante doble, es posible usar tipos de datos de tamaño fijo como float o incluso int si la precisión es suficiente. Esto puede reducir el tamaño de la matriz y, por lo tanto, mejorar la eficiencia del almacenamiento en caché.

Bibliografía

- HPC for Research (Sitio web de recursos)
- Introduction to Matrix Multiplication (Artículo)
- CUDA Programming (Sitio web de recursos)
- Understanding Speedup in Parallel Computing (Artículo)
- "Introduction to Multithreading in C" (artículo)
- Introduction to Scientific Computing in C++