Reto 2 - Ecuación de Poisson unidimensional mediante el método de Jacobi utilizando OpenMP como método de paralelización

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

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 la api de OpenMP. Para medir el tiempo de ejecución, se utilizarán herramientas de medición de tiempo en C. Se espera que la implementación esta api 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 del método de Jacobi con OpenMP no es un problema complejo y no requiere de muchos conocimientos avanzados de programación en C y técnicas de programación paralela.

También se harán comparaciones con métodos usados en el trabajo anterior entregado de la materia.

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.

OpenMP: OpenMP es una interfaz de programación de aplicaciones para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join.

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 con openMP ya que anteriormente se había realizado con hilos, por lo que el funcionamiento del código seria el siguiente:

El programa comienza leyendo los parámetros de entrada desde la línea de comandos: n, el número de puntos de la malla, y nsteps, el número de barridos de Jacobi que se realizarán. Si se especifica un tercer argumento, se escribe la solución en un archivo con ese nombre. A continuación, el programa reserva memoria para los vectores u y f y los inicializa. El vector f se inicializa con valores crecientes de x, para que la solución se asemeje a una función lineal. El vector u se inicializa con ceros, excepto en los extremos, que se fijan a cero debido a las condiciones de contorno. Luego, el programa ejecuta la función jacobi() para resolver el problema de Poisson. Esta función implementa el método de Jacobi utilizando ordenamiento en barrido y actualiza los valores de la solución en el vector u. La función recibe como argumentos el número de barridos de Jacobi que se realizarán, el número de puntos de la malla, el vector u y el vector f. Finalmente, el programa escribe la solución en un archivo si se especificó un tercer argumento y libera la memoria reservada para los vectores u y f.

Pruebas

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

Secuencial	Hilos	Procesos	OpenMP
0,418756	0,763606	0,001455	0,686131
0,483259	0,764882	0,001267	0,677586
0,470164	0,722973	0,001412	0,681816
0,487788	0,772238	0,001552	0,675593
0,443308	0,758729	0,001541	0,67025
0,463342	0,745735	0,001558	0,72874
0,458632	0,74743	0,001447	0,67146
0,4586	0,763233	0,001713	0,675237
0,456892	0,741627	0,001586	0,70813

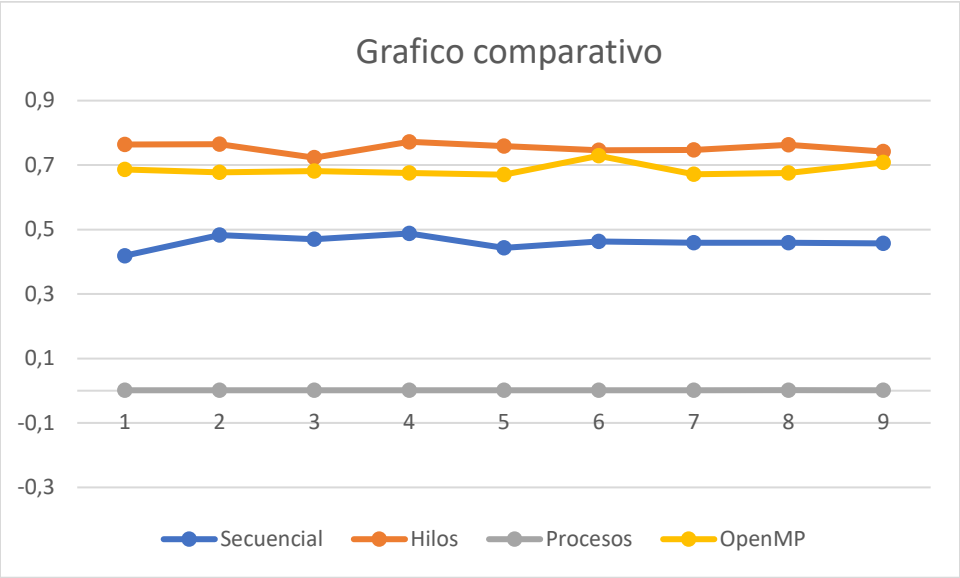
	Promedio
Secuencial	0,46008233
Hilos	0,75338367
Procesos	0,00150344
OpenMP	0,68610478

	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 está probando parece ser que el multithreading no lo maneja bien nuevamente (como en el trabajo anterior) 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 de mejor manera.

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.



Conclusiones

1. Este trabajo podría decir que el uso de OpenMP e 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. Cabe aclarar que estos resultados son totalmente diferentes en otras máquinas, por lo que claramente no se puede generalizar.

2. Esta claro que para cualquier trabajo de optimización es importante probar diferentes métodos y escoger el mejor para esa maquina o ese ambiente de trabajo porque los resultados podrían no se los esperados.

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++