Tecnológico de Monterrey

Aproximación de Pi mediante la serie de Nilakantha utilizando métodos Secuenciales, Paralelos y CUDA
Diseño de sistemas embebidos avanzados

Alumnos:
Ricardo Sierra Roa A01709887

Profesor:
Pedro Oscar Pérez Murueta

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Querétaro

Fecha de entrega:
22 de noviembre de 2024

Índice

Índice **2**
Resumen **3**
Introdcción **3**
Desarrollo **3**
Método Secuencial 3
Método Paralelo 3
Método CUDA 4
Comparaión de Resultados 4
Conclusión **5**

Resumen

Este trabajo compara tres métodos de aproximación de la constante Pi mediante la serie de Nilakantha: secuencial, paralelo (utilizando múltiples núcleos) y CUDA (para procesamiento en la GPU). Se miden los tiempos de ejecución de cada enfoque, para lograr calcular su aceleración (speed up) y eficiencia en función del tiempo medio de cálculo. Los resultados muestran diferencias significativas en la eficiencia y velocidad entre los métodos.

Introdcción

La serie de Nilakantha es una fórmula de convergencia relativamente rápida que permite aproximar el valor de Pi mediante la suma de términos alternantes. En este proyecto, se utiliza esta serie para calcular Pi utilizando tres métodos diferentes:

1) Método Secuencial: Calcula Pi de manera secuencial en un solo hilo de CPU.
2) Método Paralelo: Divide el cálculo en múltiples hilos para aprovechar la paralelización en la CPU.
3) Método CUDA: Implementa el cálculo en una GPU utilizando la arquitectura CUDA para lograr una aceleración considerable.

Cada método será evaluado en términos de tiempo de ejecución, aceleración y eficiencia, permitiendo identificar el más adecuado para el cálculo de esta constante en contextos de alta precisión y velocidad.

Desarrollo

Método Secuencial

- Este método calcula Pi iterando a través de la serie de Nilakantha de manera continua en un solo hilo de CPU. El tiempo de ejecución promedio fue de 70.635 ms.
- Este tiempo sirve de base para comparar la eficiencia de los métodos paralelos.

Método Paralelo

- Utilizando hilos de CPU, el cálculo de Pi se divide en bloques asignados a diferentes hilos para realizar las sumas parciales de la serie. En este caso, se utilizó la capacidad de múltiples hilos de la CPU para distribuir el trabajo y acelerar el cálculo.
- Resultados: Tiempo promedio de 19.144 ms, con una aceleración de 3.69x y una eficiencia de 0.4612.
- La eficiencia es menor a 1, lo que sugiere que la sobrecarga de administración de hilos y la sincronización afecta el rendimiento.

Método CUDA

- Para la implementación en CUDA, se utilizaron 512 hilos y 16 bloques en la GPU. Cada hilo calcula una parte de la serie, y los resultados parciales se reducen en cada bloque para obtener el valor final de Pi.
- Resultados: Tiempo promedio de 9.561 ms, con una aceleración de 3.88x y una eficiencia muy baja de 0.00047.
- A pesar del tiempo reducido, la eficiencia es extremadamente baja, lo que indica que la implementación en CUDA es limitada por la arquitectura de memoria y la necesidad de sincronización en la GPU para este tipo de cálculo.

Comparaión de Resultados

Secuencial VS Paralelo			
Procesadores		8	
Secuencial	70.635 ms	Speed Up	3.689667781
Paralelo	19.144 ms	Eficiencia	0.4612084726

Tabla 1. "Secuencial vs Paralelo"

Secuencial VS CUDA			
THREADS	512	BLOCKS	16
Secuencial	37.098 ms	Speed Up	3.880138061
Paralelo	9.561 ms	Eficiencia	0.0004736496656

Tabla 2. "Secuencial vs CUDA"

Conclusión

La implementación más eficiente en términos de tiempo de ejecución es la versión CUDA, ya que proporciona el menor tiempo promedio de cálculo. Sin embargo, la eficiencia extremadamente baja sugiere que el paralelismo en la GPU no es completamente aprovechado para este tipo de cálculo debido a la necesidad de sincronización y a la naturaleza del cálculo secuencial de la serie de Nilakantha. El método paralelo en CPU, aunque más lento que CUDA, presenta una eficiencia considerablemente mayor, lo que lo hace más adecuado si se busca un balance entre eficiencia y velocidad en arquitecturas de CPU multinúcleo. La elección del método dependerá de los recursos disponibles y de los requisitos de eficiencia y tiempo de procesamiento del sistema.