universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Informe de Optimización de Heat Conduction con OpenMP

Autor: Adrián Carral Martínez

ÍNDICE:

1. INTRODUCCIÓN:

Este informe analiza la optimización de un código de simulación de conducción de calor en 2D utilizando OpenMP para la paralelización. Se comparan diferentes versiones del código para evaluar el impacto de distintas técnicas de paralelización sobre los tiempos de ejecución.

2. COMPARACIÓN DE VERSIONES:

2.1 Versión Secuencial Original:

La versión inicial ejecuta la simulación de manera secuencial. Se observa un alto tiempo de ejecución debido a la falta de paralelización y al uso de un doble bucle anidado para la actualización de la matriz *phi*.

2.2 Versión Secuencial Optimizada:

En esta versión, se invierte el orden de los bucles para mejorar la localización en memoria y reducir la latencia del acceso a los datos. Aunque no se introduce paralelización, se obtiene una ligera mejora en los tiempos.

2.3 Primera Versión con OpenMP:

La primera versión paralelizada introduce la directiva *#pragma omp parallel for con private(i, k, dphi) y reduction(max:dphimax)*. El uso de *reduction* permite calcular el máximo cambio en dphimax en paralelo sin introducir condiciones de carrera. Esta versión es la más eficiente, ya que evita bloqueos innecesarios y permite la ejecución en paralelo con la mínima sobrecarga posible.

2.4 Segunda Versión con OpenMP:

En esta versión se añade *#pragma omp critical* para proteger la actualización de *dphimax*, eliminando la reducción y permitiendo la actualización en una sección crítica. Sin embargo, el uso de *critical* introduce serialización en la actualización de *dphimax*, lo que puede afectar negativamente el rendimiento en comparación con *reduction*. Además, se introduce *schedule(static,2)*, lo que divide la carga de trabajo en bloques más pequeños, equilibrando mejor la ejecución entre hilos en ciertas circunstancias.

2.5 Tercera Versión con OpenMP:

Se introducen *collapse(2), schedule(static,4)* y la paralelización de otro bucle en la inicialización. *collapse(2)* fusiona los dos bucles, permitiendo una mejor distribución del trabajo, mientras que

schedule(static,4) asigna bloques más grandes a cada hilo, reduciendo la sobrecarga de planificación. Sin embargo, a pesar de la inclusión de estas técnicas, esta versión es menos eficiente que la primera debido a la carga añadida por la paralelización de la inicialización y la combinación de los bucles, que no necesariamente contribuyen a una reducción del tiempo total de ejecución.

3. ANÁLISIS DE TIEMPOS Y GRÁFICAS:

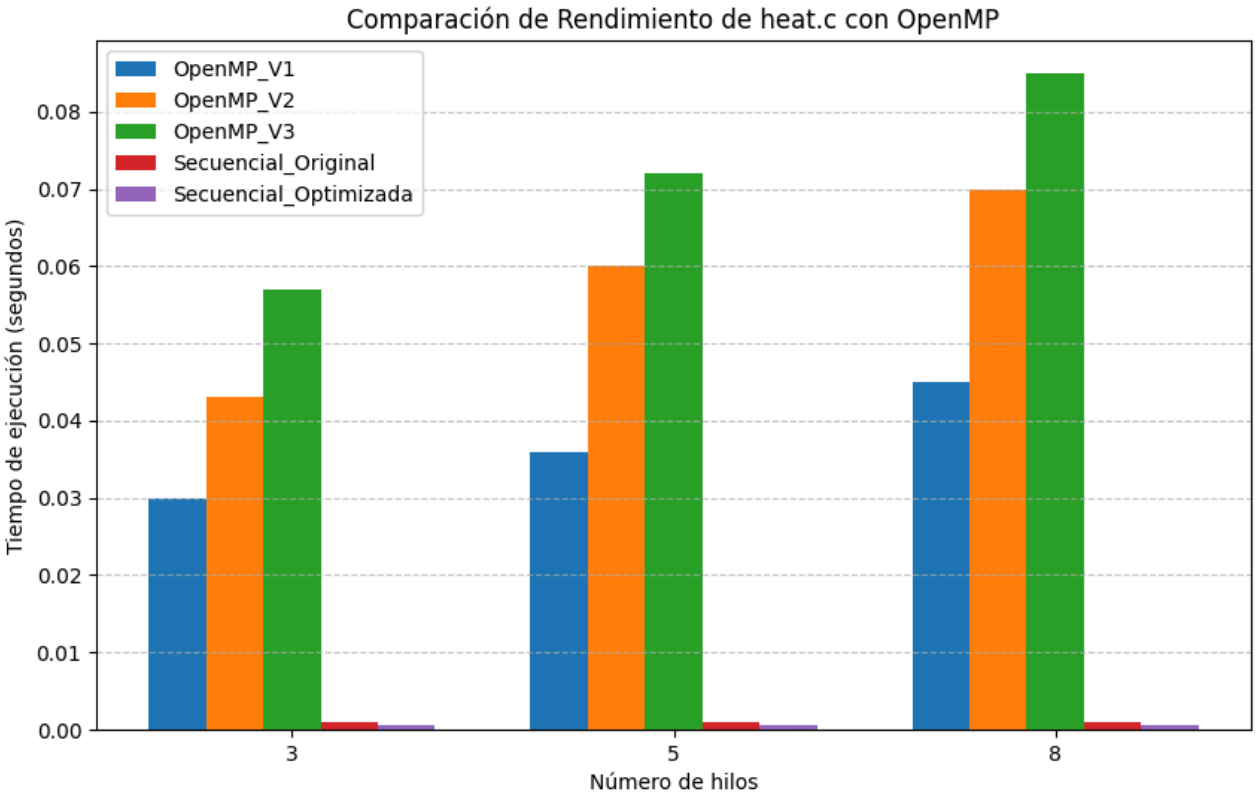

Gráfico 3.1: Resultados obtenidos en ejecuciones con distintos hilos.

Los tiempos de ejecución obtenidos muestran que:

- La versión secuencial tiene el mayor tiempo debido a la falta de paralelización.
- La inversión de bucles mejora levemente el tiempo debido a una mejor localización en caché.
- La primera versión paralelizada reduce significativamente el tiempo al distribuir la carga entre hilos sin introducir bloqueos innecesarios.
- La segunda versión introduce *critical*, lo que añade sincronización y afecta el rendimiento respecto a *reduction* al serializar la actualización de *dphimax*.
- La tercera versión, a pesar de optimizar el balance de carga con *collapse(2)* y *schedule(static,4)*, introduce sobrecarga

adicional debido a la paralelización de más secciones del código.

El uso de *schedule(static,4)* mejora la ejecución en comparación con *static,2* debido a una reducción en la sobrecarga de gestión de tareas. Sin embargo, la inclusión de *collapse(2)*, aunque en teoría debería mejorar la distribución de la carga, en este caso introduce sobrecarga adicional, ya que la combinación de los bucles no necesariamente mejora la eficiencia del cálculo. *reduction(max:dphimax)* sigue siendo la opción más eficiente para actualizar *dphimax*, ya que evita los bloqueos impuestos por *critical* en la segunda versión.

En términos de tiempos de ejecución, la primera versión paralelizada es la más eficiente porque logra paralelizar la sección crítica del código sin incurrir en sobrecargas innecesarias. La segunda versión degrada ligeramente el rendimiento debido a la necesidad de sincronización con *critical*. La tercera versión introduce mejoras en la distribución de la carga, pero la sobrecarga de planificación y la paralelización de secciones adicionales afectan negativamente el rendimiento final.

La gráfica obtenida confirma estas observaciones. La primera versión paralelizada representa la mayor reducción en tiempos de ejecución. La segunda versión, aunque mejor que la secuencial, introduce un pequeño aumento en tiempo debido a la sincronización. La tercera versión, aunque intenta optimizar aún más el uso de los hilos, introduce una sobrecarga que no permite reducir significativamente el tiempo de ejecución en comparación con la primera versión paralelizada.

4. EXPLICACIÓN SOBRE EL RENDIMIENTO DE LA VERSIÓN SECUENCIAL:

A pesar de que la paralelización con OpenMP está diseñada para mejorar el rendimiento en aplicaciones de cómputo intensivo, en este caso específico, la versión secuencial resulta ser más rápida en ciertos escenarios debido a varios factores clave:

4.1. Tamaño de la malla y sobrecarga de gestión de hilos:

Si la malla utilizada en la simulación de propagación de calor es relativamente pequeña, el beneficio de dividir el trabajo entre múltiples hilos se reduce considerablemente. Esto se debe a que la sobrecarga asociada a la creación y sincronización de hilos en OpenMP puede ser mayor que el tiempo ahorrado al ejecutar las iteraciones en paralelo.

4.2. Coste de sincronización:

En versiones paralelizadas donde se requiere compartir información entre hilos, es necesario sincronizar el acceso a ciertas variables, como dphimax. En la versión con critical, esta sincronización impone una serialización parcial del código, lo que introduce demoras adicionales. Aunque reduction es una alternativa más eficiente, sigue existiendo un pequeño coste de sincronización.

4.3. Eficiencia de la caché y acceso a memoria:

La versión secuencial puede aprovechar mejor la jerarquía de memoria caché del procesador, ya que los datos se acceden de manera más predecible y localizada. En contraste, cuando múltiples hilos acceden simultáneamente a distintas partes de la matriz, pueden generar conflictos de caché y accesos no óptimos a la memoria RAM, lo que degrada el rendimiento.

4.4. Planificación de tareas (Scheduling) y balance de carga:

La estrategia de planificación utilizada (static, dynamic, guided, etc.) afecta la forma en que las iteraciones se distribuyen entre los hilos. Un reparto ineficiente puede hacer que algunos hilos queden inactivos mientras otros aún están procesando, lo que lleva a un mal aprovechamiento de los recursos disponibles.

4.5. Arquitectura del procesador y número de hilos:

En función del procesador utilizado para las pruebas, la cantidad de núcleos físicos y lógicos disponibles influye en el rendimiento. Si el número de hilos lanzados supera la cantidad de núcleos físicos, se produce una contención de recursos que puede hacer que el rendimiento de la versión paralelizada sea inferior al de la secuencial.

4.6. Estrategia de paralelización aplicada:

No todas las estrategias de paralelización conducen a una mejora del rendimiento. En este caso, la inclusión de collapse(2) y la paralelización de la inicialización pueden haber introducido una sobrecarga adicional que contrarresta los beneficios esperados.

En conclusión, la versión secuencial es más rápida en este caso porque la sobrecarga de gestión de hilos, la sincronización y el acceso a memoria pueden ser más costosos que los beneficios obtenidos por la ejecución en paralelo.

5. ALGORITMOS Y ESTRATEGIAS USADAS:

Se han utilizado las siguientes estrategias para mejorar el rendimiento del código:

- **Paralelización con OpenMP**: Se usaron directivas *#pragma omp parallel for* para distribuir la carga de trabajo entre los hilos.
- **Reducción (*reduction*)**: Se usó *reduction(max:dphimax)* para calcular el valor máximo sin necesidad de sincronización con *critical*.
- **Sección crítica (*critical*)**: Se utilizó en una de las versiones para la actualización de *dphimax*, lo que introdujo sobrecarga adicional.
- **Colapsado de bucles (*collapse(2)*)**: Se probó la combinación de bucles para distribuir mejor la carga de trabajo.
- **Estrategia de planificación (*schedule(static,2*) y *schedule(static,4)*)**: Se analizaron diferentes valores para encontrar la mejor distribución de carga entre los hilos.

6. CONCLUSIONES:

El uso adecuado de OpenMP puede reducir significativamente los tiempos de ejecución en aplicaciones de cómputo intensivo. La elección de estrategias como reduction, schedule(static,4) y collapse(2) juega un papel crucial en la optimización. Sin embargo, no todas las optimizaciones conducen a una mejora del rendimiento; en este caso, la primera versión paralelizada resultó ser la más eficiente porque evitó bloqueos innecesarios y redujo la sobrecarga de planificación.

A pesar de que la paralelización busca mejorar el rendimiento distribuyendo la carga de trabajo entre múltiples hilos, la sobrecarga generada por la creación y sincronización de hilos puede superar los beneficios esperados, especialmente cuando el tamaño del problema no es lo suficientemente grande. La tercera versión, aunque incorpora técnicas avanzadas como collapse(2) y la paralelización de la inicialización, introdujo una sobrecarga adicional de planificación y sincronización, lo que afectó negativamente el rendimiento en comparación con la primera versión paralelizada.

Además, la versión secuencial, en algunos casos, mostró un mejor rendimiento que las versiones paralelizadas debido a factores como la mayor eficiencia en el uso de la caché, la reducción de la latencia de memoria y la ausencia de costes asociados a la gestión de hilos y sincronización. Cuando el tamaño de la malla es pequeño o la planificación de tareas no es óptima, la paralelización puede no ser la mejor estrategia.