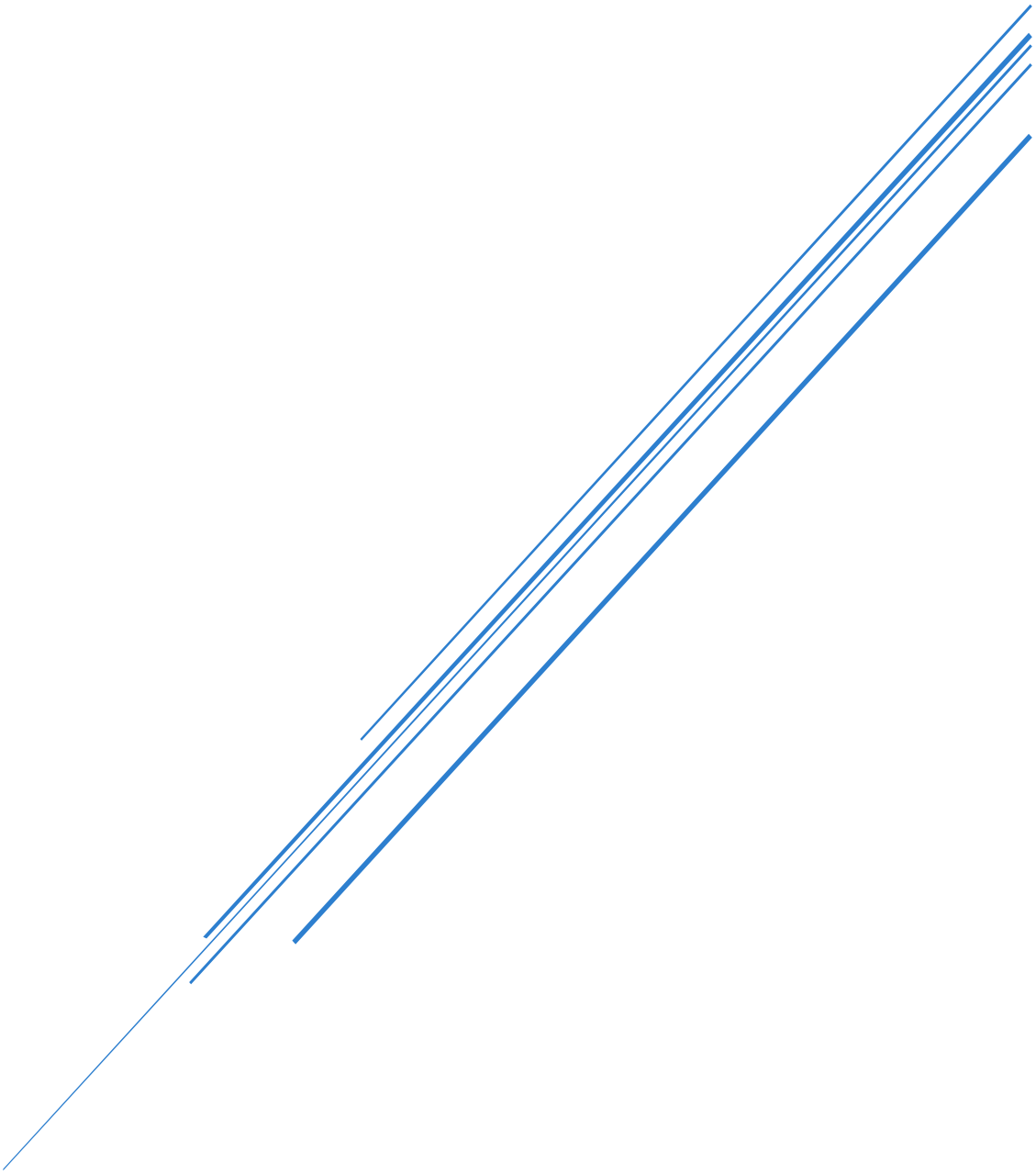
PRÁCTICA FINAL – OPEN MP

GRUPO 9



Universidad Alfonso X El Sabio
Arquitectura de Computadores

INTRODUCCIÓN

Esta práctica tiene como objetivo implementar una simulación de difusión de calor en una matriz bidimensional. Esta simulación pretende comparar el rendimiento entre una versión secuencial y una versión paralelizada utilizando la biblioteca OpenMP. El ejercicio consiste en calcular iterativamente la temperatura de cada celda en una rejilla bidimensional, en función de la temperatura de sus celdas vecinas, hasta alcanzar un estado de equilibrio térmico, definido por un umbral máximo de cambio permitido entre iteraciones.

Este tipo de simulaciones son fundamentales en la modelización de fenómenos físicos y tienen aplicaciones en diversas áreas como la ingeniería térmica, la meteorología, la física computacional y el diseño de sistemas de refrigeración. Además, el estudio del rendimiento entre ejecuciones secuenciales y paralelas es clave en el ámbito de la Arquitectura de Computadores, ya que permite entender los beneficios y limitaciones de la programación paralela en sistemas multicore.

OpenMP (Open Multi-Processing) es una API que facilita la programación paralela en arquitecturas de memoria compartida. Se trata de una herramienta muy utilizada en computación científica e ingeniería, gracias a su facilidad de uso y compatibilidad con lenguajes como C y C++. OpenMP permite aprovechar múltiples núcleos de procesamiento sin necesidad de realizar una gestión explícita de los hilos, lo que lo convierte en una solución eficaz para la paralelización de bucles y otras tareas concurrentes. Las principales características de OpenMP incluyen:

- Modelo de programación basado en hilos.
- Control explícito de paralelismo mediante directivas #pragma.
- Soporte para variables compartidas y privadas entre hilos.
- Facilidades para la sincronización y la reducción de datos.

• DESARROLLO

El programa se ha escrito en lenguaje C y utiliza la biblioteca OpenMP para la paralelización de los bucles. El código fuente se estructura de la siguiente manera:

MÉTODOS INICIALES:

- **calcular_promedio:** calcula el valor promedio de las temperaturas de las celdas adyacentes (arriba, abajo, izquierda, derecha).
- **crear_matriz:** reserva dinámicamente memoria para almacenar una matriz bidimensional de tamaño NxN.
- **inicializar_matriz:** establece las condiciones iniciales de la simulación. Las celdas exteriores se inicializan a 100.0 (ºC) y las celdas interiores se inicializan a 0.0 (ºC).
- **liberar_matriz:** libera la memoria dinámica utilizada por la matriz.

FUNCIÓN PRINCIPAL:

La función **main** recibe los parámetros desde la línea de comandos, verifica que se ingresen los 4 valores requeridos y los convierte a los tipos de datos correspondientes:

- **N**: tamaño de la matriz.
- **umbral**: umbral de cambio térmico para determinar la convergencia.
- **max_iter**: número máximo de iteraciones permitidas.
- **modo**: define si la ejecución será secuencial (0) o paralela (1).

A continuación:

- Se crean dos matrices (actual y siguiente) para almacenar el estado térmico actual y el siguiente.
- Se inicializa la matriz actual y se registra el tiempo de inicio.
- Se ejecuta un bucle *do-while* hasta que el máximo cambio de temperatura sea inferior al umbral o se alcance el número máximo de iteraciones.
- Dentro del bucle, si *modo == 0*, el programa se ejecuta de forma secuencial, mientras que si *modo == 1*, el programa se ejecuta en paralelo.
- Al final de cada iteración, se intercambian los punteros de las matrices actual y siguiente.
- Tras finalizar la simulación, se mide el tiempo total, se imprime la información de salida y se libera la memoria utilizada.

• INSTALACIÓN

Para compilar y ejecutar el programa se utilizó el entorno de desarrollo CLion de JetBrains, que permite integrar el compilador gcc y la biblioteca OpenMP con facilidad. También se puede compilar y ejecutar desde la terminal en sistemas basados en UNIX.

REQUISITOS:

- Sistema operativo: Windows 10 o superior / Linux / macOS
- CLion instalado (o un entorno de desarrollo C compatible con CMake)
- Compilador GCC con soporte para OpenMP (versión 9.0 o superior)
- CMake 3.20 o superior

INSTALACIÓN EN CLION:

- Clonar o descargar el proyecto desde el repositorio correspondiente.
- Abrir el proyecto en CLion.
- Asegurarse de que el compilador utilizado sea GCC y que OpenMP esté habilitado.

ALTERNATIVA PARA LINUX:

- sudo apt update
- sudo apt install build-essential libomp-dev
- gcc -fopenmp -o DifusionCalor difusion_calor.c

• EJECUCIÓN

El programa recibe los siguientes parámetros por consola:

./DifusionCalor <N> <umbral> <max_iter> <modo>

- **<N>:** tamaño de la matriz (por ejemplo, 100 implica una matriz de 100x100)
- **<umbral>:** valor de tolerancia que determina la convergencia
- **<max_iter>:** número máximo de iteraciones permitidas
- **<modo>:** 0 para ejecución secuencial, 1 para ejecución paralela

EJECUCIÓN EN CLION:

Para pasar argumentos al programa desde CLion, es necesario configurar los argumentos del programa dentro de la configuración de ejecución. En CLion, se debe hacer clic en el menú desplegable “DifusionCalor” de la parte superior derecha (al lado del botón de ejecución) y seleccionar “Edit Configurations...”. En el campo “Program arguments”, se deben escribir los parámetros que se quieran establecer separados por espacios. Por ejemplo: 100 2 1000 1. Después, solo hay que guardar la configuración y ejecutar el programa; CLion pasará automáticamente estos argumentos al iniciar la ejecución.

EJEMPLO DE EJECUCIÓN SECUENCIAL:

- ./DifusionCalor 100 2 1000 0

EJEMPLO DE EJECUCIÓN PARALELA:

- ./DifusionCalor 100 2 1000 1

SALIDA DEL PROGRAMA:

El programa muestra información sobre el modo de ejecución, el número de iteraciones realizadas y el tiempo total de ejecución. Por ejemplo:
Modo: Paralelo
Iteraciones: 932
Tiempo: 0.078000 segundos

• RESULTADOS

Tiempos obtenidos:

N	Umbral	Iteraciones	Modo	Tiempo
100	2	1,000	Secuencial	0,092 seg
100	2	1,000	Paralelo	0,106 seg
200	2	2,000	Secuencial	0,554 seg
200	2	2,000	Paralelo	0,334 seg
10,000	4	2,000	Secuencial	> 30 mins
10,000	4	2,000	Paralelo	3,2 mins

Como se puede comprobar, los resultados obtenidos muestran diferencias notables entre los modos de ejecución secuencial y paralelo, especialmente al aumentar el tamaño de la matriz y el número de iteraciones.

TUPLA INICIAL:

- N = 100, Umbral = 2, Iteraciones = 1,000

Para tamaños reducidos, el tiempo de ejecución es muy bajo tanto en la versión secuencial (0,092 s) como en la paralela (0,106 s). De hecho, en este caso el modo secuencial es ligeramente más rápido que el paralelo, lo cual puede deberse a la sobrecarga que supone crear y gestionar los hilos en OpenMP para una carga de trabajo tan pequeña.

TUPLA PRUEBA 1:

- N = 200, Umbral = 2, Iteraciones = 2,000

Al duplicar tanto el tamaño de la matriz como las iteraciones, se observa un incremento del tiempo en ambos modos, pero la ejecución paralela (0,334 s) mejora sustancialmente respecto a la secuencial (0,554 s). En este caso, el uso de múltiples hilos comienza a ser efectivo, superando la sobrecarga inicial y permitiendo una ejecución más eficiente.

TUPLA PRUEBA 2:

- N = 10,000, Umbral = 4, Iteraciones = 2,000

Con esta configuración, la ejecución secuencial resulta inviable, habiéndose superado los 30 minutos sin terminar la ejecución del programa, mientras que la ejecución paralela, aunque considerablemente más rápida, ha requerido más de 3 minutos para completarse. Esto evidencia que el coste computacional crece de forma exponencial con el aumento de N, y aunque la paralelización ayuda, no es suficiente por sí sola para garantizar un rendimiento aceptable a gran escala.

• CONCLUSIONES

La práctica ha permitido observar de forma clara las diferencias de rendimiento entre una ejecución secuencial y una ejecución paralelizada con OpenMP en la simulación de difusión del calor en una matriz bidimensional. A través de la comparación de distintas configuraciones de tamaño de matriz, umbral de convergencia y número de iteraciones, se han extraído varias conclusiones importantes:

1. **Escalabilidad limitada en modo secuencial**: La versión secuencial presenta tiempos de ejecución aceptables únicamente para tamaños reducidos. A medida que el tamaño de la matriz y el número de iteraciones aumentan, el tiempo de ejecución crece de forma exponencial. En la última tupla de prueba (N = 10,000), el tiempo de ejecución supera los 30 minutos, haciendo inviable este enfoque para simulaciones a gran escala.
2. **Ventajas de la paralelización**: La paralelización mediante OpenMP ofrece mejoras notables de rendimiento a partir de cargas de trabajo medias y grandes. En la segunda tupla de prueba (N = 200), el tiempo de ejecución paralelo fue aproximadamente un 40% menor que el secuencial. En la tercera tupla (N = 10,000), aunque el tiempo sigue siendo elevado (más de 3 minutos), la reducción frente a la ejecución secuencial es muy significativa, permitiendo completar una tarea que, de otro modo, sería inviable.
3. **Sobrecarga de gestión de hilos**: En problemas pequeños (N = 100), la ejecución paralela puede ser ligeramente más lenta que la secuencial. Esto se debe a la sobrecarga que implica la creación, sincronización y gestión de los hilos, que en cargas livianas puede superar los beneficios del paralelismo.
4. **Importancia de la paralelización en aplicaciones reales**: Esta práctica refuerza la importancia de aprovechar arquitecturas multicore en aplicaciones intensivas en cálculo, como las simulaciones físicas. Sin paralelización, muchas de estas simulaciones serían computacionalmente inviables, especialmente cuando se requiere alta resolución o tiempos de simulación prolongados.
5. **Límites del paralelismo**: Aunque la paralelización mejora el rendimiento, no lo escala linealmente con el número de hilos ni elimina el crecimiento exponencial del coste computacional en problemas grandes. Por tanto, es fundamental combinar la paralelización con otras técnicas de optimización, como el uso de algoritmos más eficientes o arquitecturas distribuidas, para abordar problemas de mayor envergadura.

En definitiva, la práctica ha demostrado que OpenMP es una herramienta eficaz y relativamente sencilla de implementar para mejorar el rendimiento de programas intensivos en cálculo, y que su uso es clave en el diseño de software eficiente sobre sistemas modernos con múltiples núcleos de procesamiento.

- **PARTICIPANTES**

  - Alberto Valera (142379)
  - Mario Serrano (146935)
  - Samuel Muñoz (146290)

- **LINK AL PROYECTO DE GITHUB**

  - https://github.com/Samuu10/ProyectoOpenMP.git