Proyecto 1: Open MP

1st Isabel Ortiz
Dept. Computer Science
Universidad del Valle de Guatemala
Guatemala, Guatemala
ort18176@uvg.edu.gt

2nd Luis Urbina
Dept. Computer Science
Universidad del Valle de Guatemala
Guatemala, Guatemala
urb18473@uvg.edu.gt

***Abstract*—Ever since parallel programming became a common practice in computer science, engineers have searched for different problems that can be solved by it. One of the problems that turns out to be ideal for this purpose is the simulation of heat dissipation. Heat dissipation is calculated using the heat equation, a partial differential equation known as the one-dimensional heat PDE. The main objective of this project is to determine the difference in performance between the sequential and parallel version (using Open MP) of a program written in C++ that allows simulating the heat dissipation for a metal bar given certain parameters: the temperature at the boundary left, the temperature at the right boundary, and the number of discrete distance intervals to be evaluated. Two sets of tests were used, varying the parameters mentioned above every 10 tests. For the first, 5,000 distance intervals, 20°C temperature on the left border and 100°C temperature on the right border were used. For the second, 5,000 distance intervals, 30°C temperature on the left border and 150°C temperature on the right border were used. As a result, a speed up of 1.5-folds was obtained for the first set of tests and a speed up of 1.6-folds for the second set. It was concluded that the parallel version of the program had a faster execution time than the sequential version, for both sets of tests.**

***Index Terms*—Heat equation, parallelism, Open MP, C++, speed up**

I. Introducción

Considere una barra metálica delgada (ancho = 0) que tiene una masa constante y densidad constante. Para simular la disipación de calor en la barra anteriormente mencionada se utiliza una ecuación diferencial parcial conocida como EDP de calor unidimensional, esta ecuación se obtiene a partir de la ley de Fourier y el principio de conservación de la energía. Partiendo de la ley de Fourier, que indica que el cambio de flujo de energía calórica por unidad de área a través de una superficie es proporcional al gradiente de temperatura negativo en la superficie [1]. Esta ecuación integra múltiples campos de la física, entre ellos: dinámica de fluidos, termodinámica, energía y estrés mecánico.

El objetivo principal de este este proyecto es determinar la diferencia en rendimiento entre la versión secuencial y paralela (utilizando Open MP) de un programa escrito en C++ que permita simular la disipación de calor para una barra metálica dados ciertos parámetros: la temperatura en la frontera izquierda, la temperatura en la frontera derecha, y el número de intervalos discretos de distancia que se evaularán. Las métrica que se utilizó para determinar la mejora o caída en rendimiento entre el algoritmo secuencial y paralelo fue el speed up calculado con los promedios de 20 mediciones de cada una de las versiones del programa.

II. Marco Teórico

Para alcanzar el objetivo del proyecto se partió de la ya mencionada ecuación diferencial parcial de calor unidimensional, a continuación se presenta el desarrollo de la misma así como una explicación de los parámetros relevantes. Adicionalmente, se presenta una descripción de la librería utilizada para la paralelización del código y la explicación del cálculo de nuestra métrica de relevancia, el speed up.

A. EDP de calor unidimensional

Para plantear la ecuación diferencial de calor para una barra metálica unidimensional podemos partir de la ley de Fourier, que indica que la velocidad de conducción de calor a través de un cuerpo por unidad de sección transversal, es proporcional al gradiente de temperatura que existe en el cuerpo [1].

$$q = -k\nabla u$$

donde k es la conductividad térmica y u es la temperatura, en una dimensión el gradiente es una derivada espacial ordinaria, por lo que la ley de Fourier se puede expresar como:

$$q = -k\frac{\partial u}{\partial x}$$

En ausencia de trabajo realizado, un cambio en la energía interna por unidad de volumen en el material, Q es proporcional al cambio de la temperatura, u. Esta relación se puede representar como:

$$\Delta Q = c_p p \Delta u$$

Donde c_p es la capacidad calórica específica y p es la densidad de masa del material.

Pero en este caso, definiendo que la energía es nula cuando la temperatura es el cero absoluto, podemos reescribir la ecuación anterior como:

$$Q = c_p p u$$

Haciendo uso del teorema fundamental del cálculo, se puede calcular el incremento de energía interna en una subregión de

$$k\int_{t-\Delta t}^{t+\Delta t}\left[\frac{\partial u}{\partial x}(x+\Delta x,\tau)-\frac{\partial u}{\partial x}(x-\Delta x,\tau)\right]d\tau = k\int_{t-\Delta t}^{t+\Delta t}\int_{x-\Delta x}^{x+\Delta x}\frac{\partial^2 u}{\partial \xi^2}\,d\xi\,d\tau$$

distancia: $x-\Delta x \leq \xi \leq x+\Delta x$ del material sobre un lapso de tiempo: $t-\Delta t \leq \tau \leq t+\Delta t$ como:

Si se aplica a la ecuación anterior el teorema fundamental del cálculo se obtiene la siguiente ecuación:

$$\int_{t+\Delta t}^{t-\Delta t}\int_{x+\Delta x}^{x-\Delta x}\left[c_p p u_\tau - k u_\xi \xi\right]d\xi\,d\tau$$

Luego de aplicar el teorema fundamental del cálculo variacional para cualquier región rectangular $[t-\Delta t, t+\Delta t][x-\Delta x, x+\Delta x]$, la región a integrar se puede igualar a 0, teniendo:

$$c_p p u_t - k u_{xx} = 0$$

Que puede reescribirse en la forma de la ecuación de disipación de calor convencional [2]:

$$\frac{\partial u}{\partial t} = \frac{k}{c_p p}\frac{\partial^2 u}{\partial x^2}$$

Para que el lector pueda comprender mejor el código se puede reescribir la ecuación anterior como:

$$\frac{\partial T(x,t)}{\partial t} = c\frac{\partial^2 T(x,t)}{\partial x^2}$$

Donde $T(x,t)$ es la temperatura desconocida en la distancia x en tiempo t, c es la difusividad térmica del material, con valor típico de $10^-5 m^2/s$ para metales.

B. Open MP

OpenMp es un proyecto open source que ofrece un API públic compatible con la programación paralela de memoria compartida multiplataforma en C/C++ y Fortran. La API de OpenMP define un modelo escalable y portátil con una interfaz simple y flexible para desarrollar aplicaciones paralelas en todo tipo de plataformas [3].

C. Speed up

El speed up de un algoritmo paralelo sobre un algoritmo secuencial correspondiente es la relación entre el tiempo de cálculo del algoritmo secuencial y el tiempo del algoritmo paralelo [4]. El speed up está dado por la siguiente ecuación:

$$speedup = \frac{T_s}{T_p}$$

Donde T_s es el tiempo de cálculo del algoritmo secuencial y T_p es el tiempo de cálculo del algoritmo paralelo. Si el speedup es n, entonces decimos que tenemos una speedup de n-folds. Por ejemplo, si un algoritmo secuencial requiere 10 minutos de tiempo de cálculo y el algoritmo paralelo correspondiente requiere 2 minutos de tiempo de cálculo, decimos que hay un speedup de 5-folds [4].

III. Metodología

A. Desarrollo de los programas

Como se mencionó anteriormente, el objetivo del proyecto fue desarrollar dos programas que permitieran simular la disipación de calor en una barra metálica unidimensional dados ciertos parámetros. Para ello, primero se desarrolló el programa secuencial en C++, así como una función main() para invocar a la función que calcula la EDP de calor unidimensional (calcularHeat). Posteriormente, haciendo uso de la librería Open MP se paralelizó una parte de este código en otro archivo que contiene otra función main() para invocar a la versión paralelizada del código (calculatHeatParalelo). A continuación se presentan los diagramas de flujo para que el lector pueda comprender los algoritmos en su totalidad.

B. Diagramas de flujo de ambas versiones

Para la versión secuencial del código, que se encuentra en el archivo pdeSecuencial.cpp, se tienen dos funciones: (1) calcularHeat, que simula la disipación de calor en una barra metálica dados ciertos parámetros y (2) main que se encarga de invocar a la función anteriormente mencionada y estimar su tiempo de ejecución. El diagrama de flujo de estas funciones se presenta en las figuras 1 y 2 de la sección de anexos, subsección C.

Para la versión paralela del código, que se encuentra en el archivo pdeParalela.cpp, se tienen dos funciones: (1) calcularHeatParalelo, que simula la disipación de calor en una barra metálica dados ciertos parámetros haciendo uso de Open MP y (2) main que se encarga de invocar a la función anteriormente mencionada y estimar su tiempo de ejecución. El diagrama de flujo de estas funciones se presenta en las figuras 3 y 4 de la sección de anexos, subsección C.

C. Pruebas y medición de speed up

Luego de validar el funcionamiento de ambas versiones del algoritmo, se realizaron 20 pruebas de la siguiente forma:

Primero, 10 mediciones con 5,000 intervalos de distancia, 20°C de temperatura en el extremo izquierdo y 100°C de temperatura en el extremo derecho, a temperatura inicial constante. Esto para ambas versiones del algoritmo.

Luego, 10 mediciones con 5,000 intervalos de distancia, 30°C de temperatura en el extremo izquierdo y 150°C de temperatura en el extremo derecho, a temperatura inicial constante. Esto para ambas versiones del algoritmo.

Para ambos sets de pruebas, se calculó el speed up del programa paralelo respecto al programa secuencial tomando T_s como el promedio de las 10 mediciones del tiempo secuencial y T_p como el promedio de las 10 mediciones del tiempo de ejecución paralelo. Por lo que se tienen dos mediciones de speed ups distintas, una para cada set de pruebas.

IV. Resultados

Para la versión secuencial del programa, con el primer set de pruebas (con 5,000 intervalos de distancia, 20°C de temperatura en el extremo izquierdo y 100°C de temparatura en el extremo derecho) se obtuvieron los siguientes resultados en milisegundos: 1637, 1789, 2210, 2315, 2210, 2315, 2304, 2251, 1821, 1895. El promedio de estas mediciones fue de: 2074 milisegundos. Para la versión paralela del programa, con el primer set de pruebas se obtuvieron los siguientes resultados en milisegundos: 1447, 1400, 1452, 1411, 1419, 1452, 1279, 1319, 1286, 1400. El promedio de estas mediciones fue de 1380 milisegundos. Esto resulta en un speedup de 1.5-folds.

Para la versión secuencial del programa, con el segundo set de pruebas (con 5,000 intervalos de distancia, 30°C de temperatura en el extremo izquierdo y 150°C de temparatura en el extremo derecho) se obtuvieron los siguientes resultados en milisegundos: 2231, 2261, 2409, 2335, 2326, 2409, 1969, 2063, 2150, 2261. El promedio de estas mediciones fue de: 2241 milisegundos. Para la versión paralela del programa, con el primer set de pruebas se obtuvieron los siguientes resultados en milisegundos: 1194, 1246, 1458, 1518, 1458, 1518, 1459, 1427, 1321, 1205. El promedio de estas mediciones fue de 1386.5 milisegundos. Esto resulta en un speedup de 1.6-folds.

V. Conclusiones

- La paralelización del algoritmo resultó en una mejora en el tiempo de ejecución para ambos sets de prueba resultando en un speed up de 1.5-folds para el primer set y 1.6-folds para el segundo set.
- Los valores de los speed ups indican que el algoritmo paralelizado es más veloz que el algoritmo secuencial incluso con valores de frontera distintos.

VI. Recomendaciones

- Como primera recomendación, se tiene el explorar la posibilidad de guardar otra variable concerniente al algoritmo en la memoria compartida que provee Open MP.
- Además, si se cuenta con una computadora con más recursos, se recomienda probar con parámetros más grandes (intervalos de distancia, temperatura en la frontera izquierda, temperatura en la frontera derecha).
- Finalmente, se recomienda probar con diferentes configuraciones de Open MP, variar el tipo de calendarización puede resultar en una mejora de tiempo para el programa paralelo.

References

[1] Serway, R. y Jewett, J. (2004). Physics for scientists and engineers. 9th ed. Peason Education.

[2] Ibarra, M.d.C (2012). La ecuación de calor de Fourier: resolución mediante métodos de análisis en variable real y en variable compleja. UTN Facultad Regional Resistencia: II Jornadas de Investigación en Ingeniería del NEA y Países Limítrofes.

[3] Open MP. (2022). About Us. Extraído de: https://www.openmp.org/about/about-us/.

[4] Schreiner, W. (s.f.). Performance of Parallel Programs. Johannes Kepler University. Extraído de: https://www3.risc.jku.at/education/oldmoodle/file.php/12/theory/slides-main.pdf

VII. Anexos

A. Catálogo de funciones

El catálogo de funcines se encuentra en el repositorio de Github, puede acceder a él a través del siguiente link.

B. Bitácora de pruebas (con 20 pruebas totales)

La bitácora de pruebas se encuentra en el repositorio de Github, puede acceder a ella a través del siguiente link.

C. Diagramas de flujo de funciones utilizadas

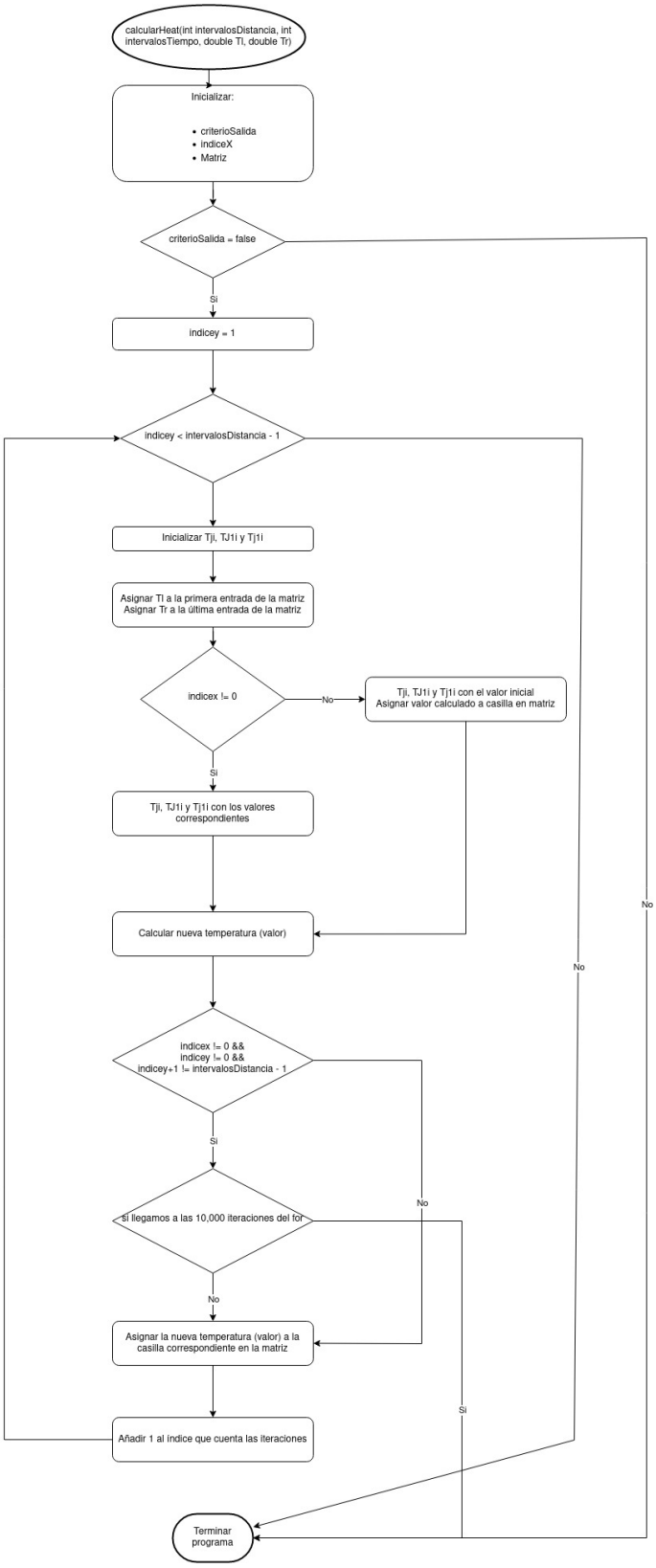

Fig. 1. Diagrama de flujo de función calculateHeat secuencial

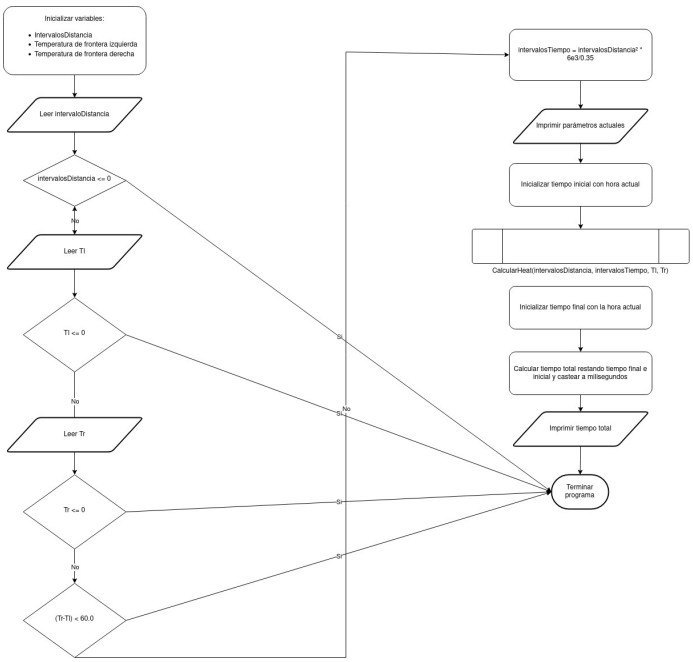

Fig. 2. Diagrama de flujo de función main para programa secuencial

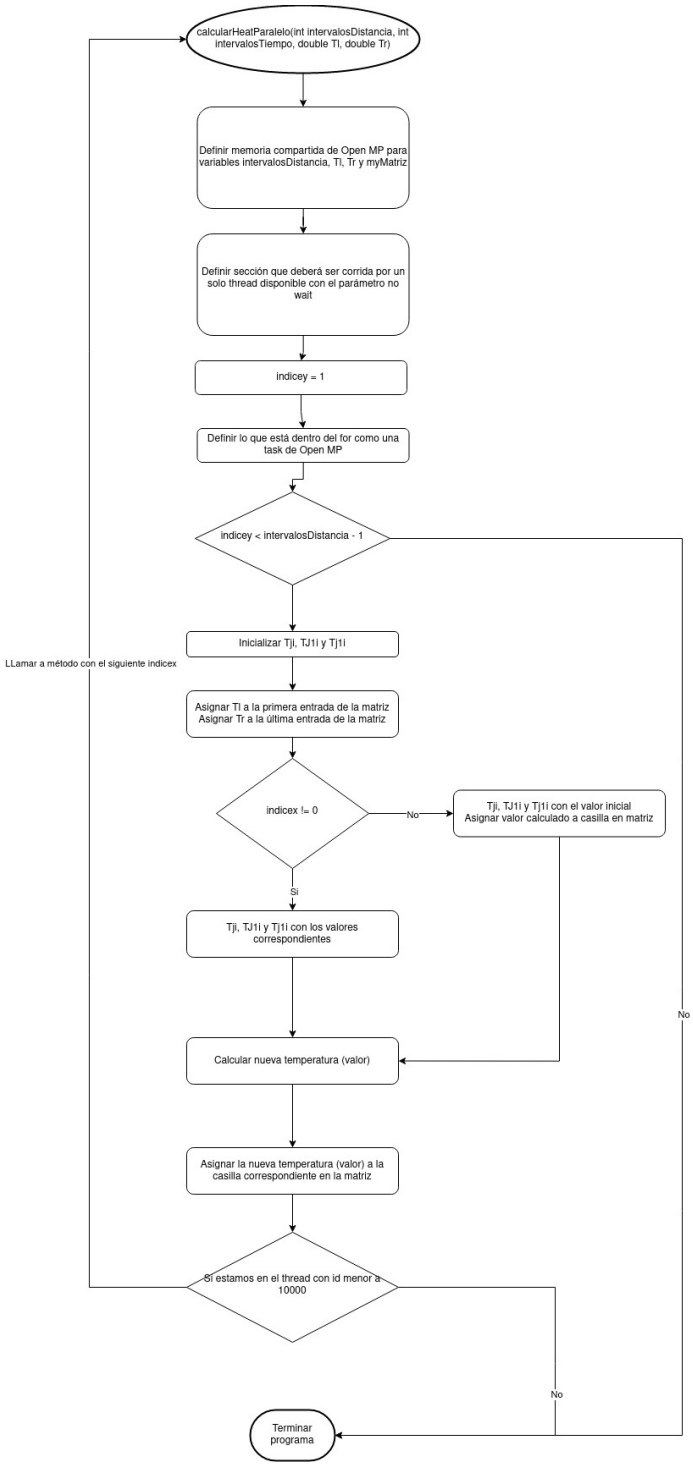

Fig. 3. Diagrama de flujo de función calculateHeatParalelo

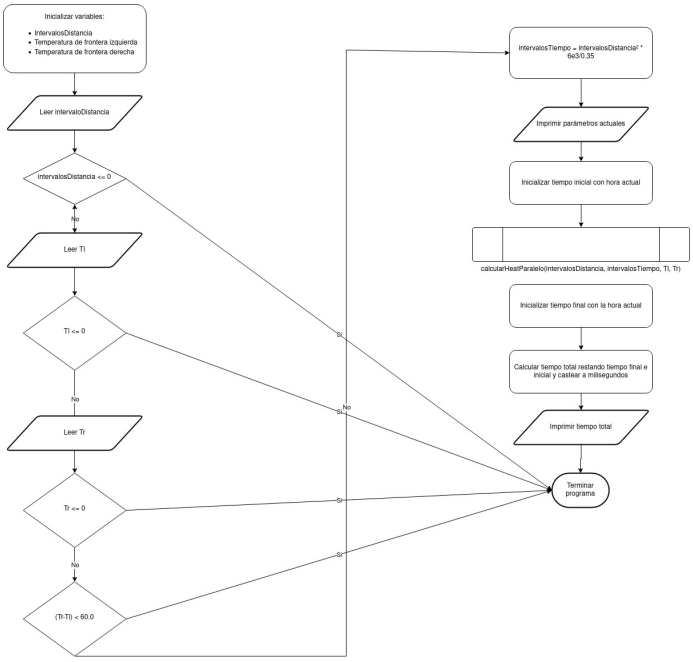

Fig. 4. Diagrama de flujo de función main para programa paralelizado