UNIVERSIDAD DEL VALLE DE GUATEMALA
Computación Paralela y Distribuida
Sección 10
Juan José Celada

Proyecto 1:
Programación Paralela con Openmp

Andrea Elías, 17048
Diego Estrada, 18540
Sara Zavala 18893

Guatemala, marzo 2022

Introducción **3**

Antecedentes **4**

Ecuación unidimensional de disipación de calor 4

OpenMP 5

PCAM 6

Conclusiones **7**

Recomendaciones **7**

Referencias bibliográficas **8**

I. Introducción

El presente proyecto, tiene como objetivo la implementación de la ecuación unidimensional de disipación de calor, resolviendo de forma tanto secuencial como paralela. Para esto, se hizo uso de OpenMP para poder paralelizar ciertos bloques de código con el objetivo de obtener un mejor rendimiento comparado a la versión secuencial del mismo.

Este rendimiento será corroborado por medio del cálculo de speedup, el cual se obtendrá mediante la realización de 20 mediciones de tiempo recabadas de ejecuciones secuenciales y luego paralelas.

II. Antecedentes

1. Ecuación unidimensional de disipación de calor

La ecuación de calor es una ecuación diferencial en derivadas parciales tipo parabólica que describe la distribución del calor o variaciones de la temperatura en una región a lo largo del paso del tiempo. Esta ecuación se obtiene de la ley de Fourier y del principio de conservación de la energía. Según lo establecido en la ley de Fourier, la razón por la que hay un cambio en el flujo de la energía calorífica por unidad de área a través de una superficie es proporcional al gradiente de temperatura negativo en la superficie (Trobec, 2018).

A continuación se describe la forma de la EDP:

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

en donde $T(x,t)$ es una temperatura desconocida en la posición x y en tiempo t, y c se refiere a la constante de difusividad térmica, cuyos valores típicos para metales es de aproximadamente $10^{-5} m^2/s$. Dicho modelo menciona que la primera derivada de la temperatura T en t equivale a la segunda derivada de T en x. con el fin de determinar la solución a la EDP es preciso tener la temperatura inicial de la barra, que puede ser vista como una constante T_0, y queda de la siguiente forma $T(x,0) = T_0$. Luego, las temperaturas que son constantes, es decir, independientes del tiempo, en los dos extremos de la barra, se expresan como $T(0) = T_L$ y $T(L) = T_R$ (Trobec, 2018).

Para poder hallar una solución numérica, el dominio debe ser discretizado en el espacio con $j = 2 \ldots (N-1)$ puntos. Simplificando el problema, estos puntos son equidistantes, de manera que $x_{j+1} - x_j = \Delta t$. es una constante. Las temperaturas discretizadas $T_j(t)$ para los $j = 2 \ldots (N-1)$ valores de solución en puntos internos de $T_l = T_L$ y $T_N = T_R$ son barreras con condiciones de barrera constantes.

Con la aproximación de diferencia finita para las derivadas de tiempo y espacio se obtiene

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{T_{j-1}(t_i) - 2T_j(t_i) + T_{j+1}(t_i)}{(\Delta x)^2},$$

$$\frac{\partial T(x_j,t)}{\partial t} = \frac{T_j(t_{i+1}) - T_j(t_i)}{\Delta t}$$

como reemplazo a las derivadas en la EDP continua, se provee una ecuación lineal para cada punto x_j. Acto seguido, usando el método explícito de Euler para la integración en tiempo, se obtiene un algoritmo más simple para el cálculo de nuevas temperaturas $T_j(t_i + 1)$ a partir de temperaturas anteriores (Trobec, 2018).

$$T_j(t_{i+1}) = T_j(t_i) + \frac{c\,\Delta t}{(\Delta x)^2}\left(T_{j-1}(t_i) - 2T_j(t_i) + T_{j+1}(t_i)\right).$$

2. OpenMP

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

Es un modelo de programación el cual resulta portable y escalable, proporcionando así a los programadores una interfaz la cual es simple y flexible para el desarrollo de aplicaciones paralelas para plataformas desde desktop hasta supercomputadoras.

Al incluir una directiva de compilador OpenMP, esto implica que se incluye una sincronización obligatoria en todo el bloque. Es decir el bloque de código se marcará como paralelo y se lanzan hilos según las características dadas a la directiva, teniendo al final de esta una barrera para la sincronización de los diferentes hilos. A este tipo de operación se le llama fork-join.

3. PCAM

Para el diseño de programas paralelos, Foster propone un modelo con tasks que interactúan unas con otras por medio de comunicación por canales. Obteniendo así lo siguiente:

- Un task es un programa, el cual tiene memoria local y su comunicación inports y onports
- Un canal conecta los inports de una task a los outports de otra
- Los canales están amortiguados. El envío es de manera asíncrona mientras el proceso de recibir es síncrono (El recibimiento de tasks se encuentra bloqueado hasta que el mensaje esperado llega.)

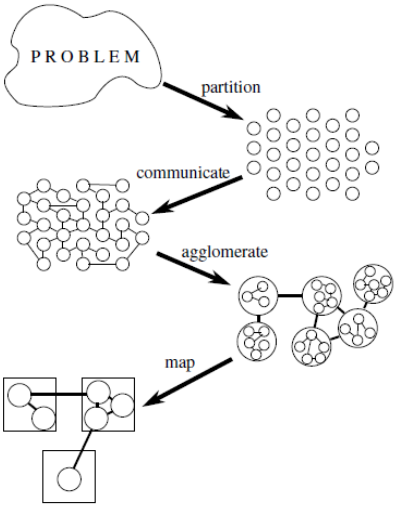
Es una metodología de apoyo para el desarrollo de programas paralelos, el cual se basa en:

- Particionar
  Se refiere a la descomposición de la información y exhibir aquellas oportunidades de paralelismo por medio de la creación de varias tasks pequeñas.

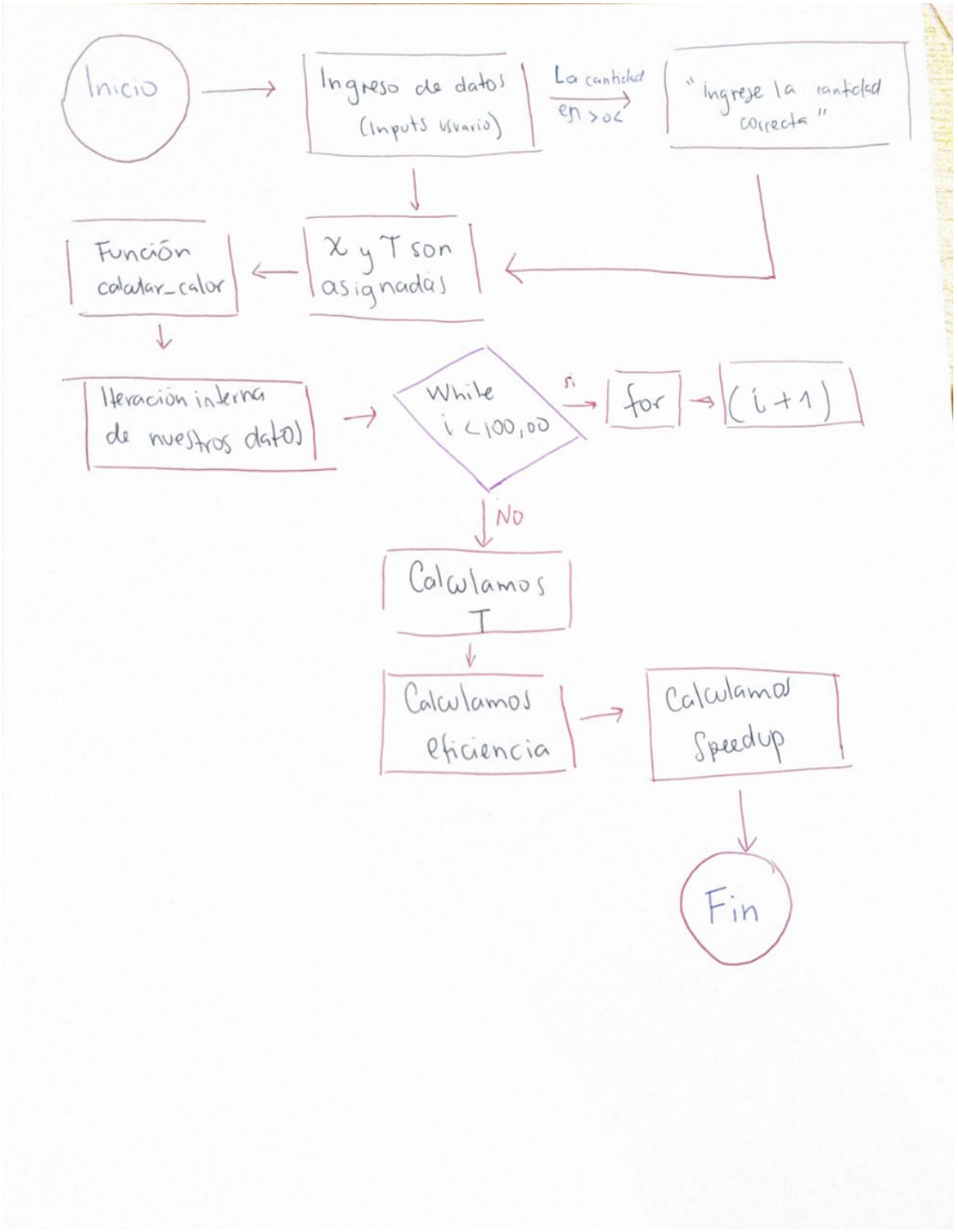
- Comunicación
  Analiza los datos y su dependencia, con el objetivo de determinar una estructura de comunicación y coordinación.

- Aglomeración
  Se trata de combinar tasks pequeños a manera de crear tasks más grandes para mejorar el performance de la ejecución en computadoras.

- Mapeo
  Se trata de asignar tasks a los procesadores a manera de maximizar la utilización y minimizar la comunicación.



III. Diagrama de flujo



IV. Resultados

Al momento de la implementación de este programa, realizamos un total de 25 ejecuciones con diversidad de datos. El resultado de todas estas ejecuciones siempre fue el tiempo secuencia, paralelo, speedup y eficiencia de cada una de las veces que corría el programa. En la siguiente tabla podremos observar el top 10 de nuestras ejecuciones.

#	Tiempo Secuencial (µs)	Tiempo Paralelo (µs)
1	69452648	5986
2	70956650	52737
3	69764922	63895
4	71839604	56429
5	69345762	55497
6	71601972	51604
7	68496713	48377
8	69374155	49399
9	68417455	50266
10	69040247	50311

V. Bitácora de pruebas

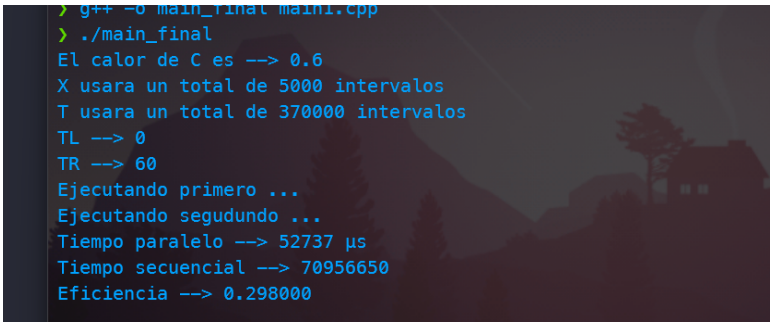
A continuación, podemos observar algunas de las pruebas que se realizaron al momento de ejecutar el programa. Se utilizaron tuplas con datos distintos (tr, tl). Los tiempos de ejecución paralelos son bastante similares entre sí, esto nos ayuda a concluir acerca de su consistencia y la de los rangos de resultados. En la siguiente sección podremos ver las capturas de pantalla donde se aprecian pas resultados.

TR	TL	#1
0	40	TS: 69485225 TP: 59869
0	60	TS: 70956650 TP: 52737
50	600	TS: 69374155 TP: 49399
100	300	TS: 69040247 TP: 50311
0	100	TS: 69764922 TP: 63895

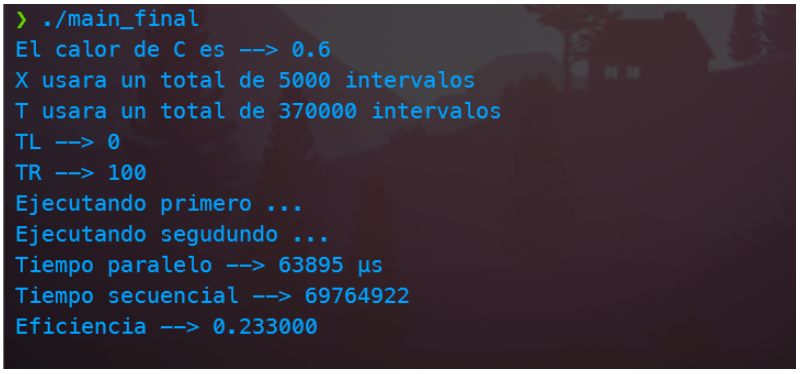
VI. Capturas de Pantalla

1.

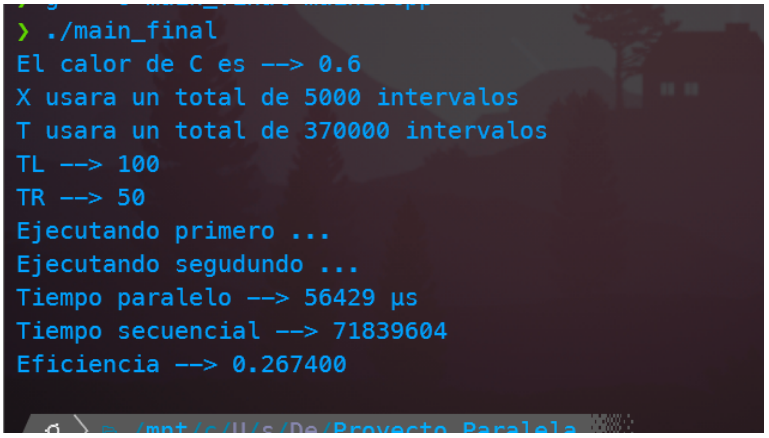
2.



3.



4.



5.

```
El calor de C es --> 0.6
X usara un total de 5000 intervalos
T usara un total de 370000 intervalos
TL --> 100
TR --> 50
Ejecutando primero ...
Ejecutando segudundo ...
Tiempo paralelo --> 55497 µs
Tiempo secuencial --> 69345762
Eficiencia --> 0.219600
```

6.

```
❯ ./main_final
El calor de C es --> 0.6
X usara un total de 5000 intervalos
T usara un total de 370000 intervalos
TL --> 100
TR --> 50
Ejecutando primero ...
Ejecutando segudundo ...
```

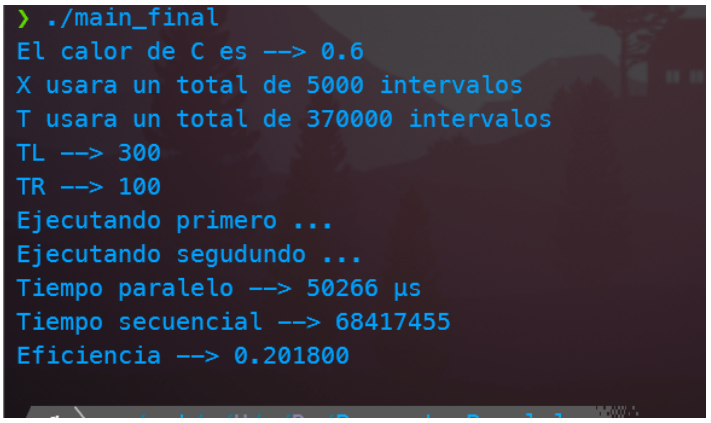
7.

```
❯ ./main_final
El calor de C es --> 0.6
X usara un total de 5000 intervalos
T usara un total de 370000 intervalos
TL --> 600
TR --> 50
Ejecutando primero ...
Ejecutando segudundo ...
Tiempo paralelo --> 48377 µs
Tiempo secuencial --> 68496713
Eficiencia --> 0.257600
```

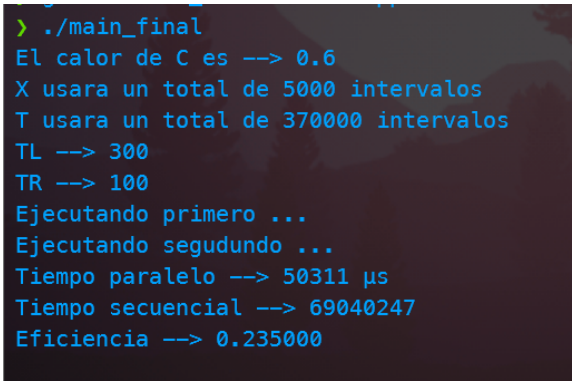
8.

```
❯ ./main_final
El calor de C es --> 0.6
X usara un total de 5000 intervalos
T usara un total de 370000 intervalos
TL --> 600
TR --> 50
Ejecutando primero ...
Ejecutando segudundo ...
Tiempo paralelo --> 49399 µs
Tiempo secuencial --> 69374155
Eficiencia --> 0.250000
```

9.



10.



VII. Conclusión

- El tiempo del algoritmo paralelo terminó siendo mucho más tardado, frente al algoritmo secuencial. Esto nos da a entender que el algoritmo paralelo

VIII. Recomendaciones

- Se propone utilizar directivas diferentes que permitan la paralelización sin dependencia de información de cálculos anteriores.
- Repetir múltiples veces las ejecuciones del algoritmo para poder obtener mejores resultados y mejores comparaciones.

IX. Referencias bibliográficas

- OpenMP Architecture Review Board. «OpenMP Compilers» (en inglés). Consultado el 18 de marzo de 2022

- Trobec, R., Slivnik, B., Bulić, P., & Robič, B. (2018). *Introduction to parallel computing: from algorithms to programming on state-of-the-art platforms*. Springer.
- Ibarra, María del Carmen (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. Consultado el 18 de marzo de 2022
- Asanovic, Krste, et al. (December 18, 2006). The Landscape of Parallel Computing Research: A View from Berkeley (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183.