Área Académica de Ingeniería en Computadores
CE-4302 Arquitectura de Computadores II
Estudiante:
Juan Peña Rostrán - 2018080231

Ronald García Fernández
Taller 1: OpenMP

1. Ejercicios

1.1. Ejemplo de uso

1. Compile ambos códigos y compare sus tiempos de ejecución.

   La figura 1 muestra el tiempo de ejecución en segundos de ambas implementaciones. En el caso de el programa *single threaded* completa el cálculo en $0{,}428s$ mientras que la implementación *multi-threaded* lo logra en menos tiempo en todos los casos, pero con especial diferencia en el caso que tiene más de 2 hilos, pues para un hilo completa el cálculo en $0{,}383s$, sin embargo, de 2 a 4 hilos su tiempo de ejecución está entre $0{,}195s \leq t \leq 0{,}223s$ aproximadamente. Este caso demuestra cómo la distribución de la carga en el programa al paralelizarlo hace que se aproveche más el uso del procesador. Es importante considerar que, para esta evaluación de cálculo de la integral que da como resultado π, se pueden generar diferentes resultados debido a la concurrencia que generan actualizar el dato *sum* por parte de múltiples hilos, lo cual se mide con un porcentaje de error, que en esta prueba no fue mayor a $6{,}997x10^{12}$.

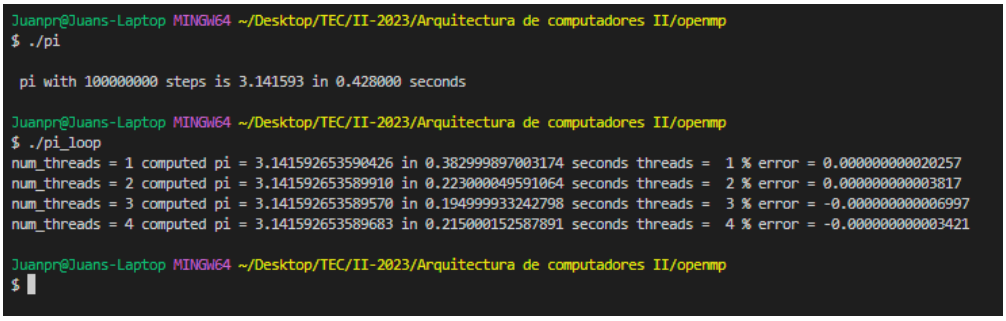

Figura 1: Tiempos de ejecución para la implementación *single threaded* y *multi-threaded*

2. Modifique pi_loop.c de forma que puede ejecutar el numero de threads disponible en su sistema.

   La modificación realizada al archivo `pi_loop.c` es la siguiente:

- Línea 37: `int threads_available = omp_get_num_procs();` para obtener el número de hilos del sistema, luego se utiliza como condición de parada en el ciclo for de la línea 46 que da inicio al contador de tiempo y la suma.

3. Ejecute pi_loop.c múltiples veces (100) y almacene los resultados en un directorio results por ejemplo pi_loop_01.txt, ..., pi_loop_100.txt

   El script `run_pi.sh` permite realizar esta solicitud.

```
#!/bin/bash
for i in {1..100}
do
    ./bin/pi_loop > results_pi/pi_loop_$i.txt
done
rm -r bin/*
```

4. Utilice algún script o herramienta para graficar: tiempo de ejecución vs numero de threads, porcentaje de error vs numero de threads, y mejora (speedup) vs numero de threads, considere el speedup como el tiempo de ejecución con un solo thread entre el tiempo de ejecución con múltiples threads puede referirse al ejemplo proporcionado plotter.py

   La figura 2 muestra que al aumentar el número de hilos pasa lo siguiente: se reduce el runtime de ejecución, el porcentaje de error y aumenta el *speedup*. Aquí es donde se refleja el paralelismo en el cálculo de este problema en particular. Al distribuir la carga de trabajo en el procesador se permite utilizar de manera más eficiente los recursos para un proceso de cálculo intensivo y alcanzar un objetivo en menor tiempo. También se debe considerar que el speedup debido al paralelismo no siempre es creciente, en este caso para 4 hilos se observa un aumento casi lineal, sin embargo, esto no significa que al añadir más hilos se mantenga este crecimiento.

1.2. Implementación: aproximación de e

1. Un programa que permita emplear múltiples threads para el cálculo de la constante e mediante el uso de series de Taylor con diferentes términos n (justifique cuantos términos va emplear).

   Para este experimento se define un valor de $n = 10000000$ tomando en cuenta los recursos computacionales del sistema con el fin de obtener un porcentaje de error relativamente bajo $\%error <= 0{,}00025$, esta ejecución se realizó utilizando desde 1 a 4 hilos prograsivamente.

2. Un programa que implemente la operación DAXPY de forma single thread y multithread (openMP), Dicho programa deberá generar vectores aleatorios de tamaño definido N, considere las capacidades de su sistema para definir los valores de prueba (al menos 4 diferentes valores).

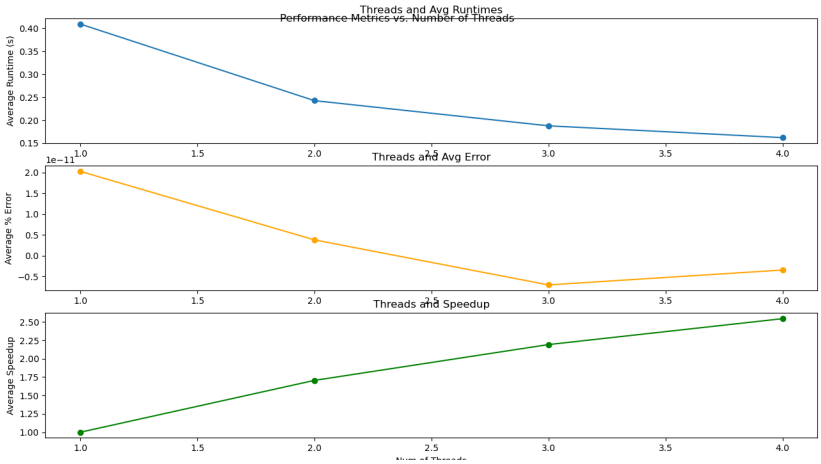

Figura 2: Resultados de los distintos parámetros de evaluación de resultados: cantidad de hilos vs. tiempo de ejecución promedio, cantidad de hilos vs. porcentaje de error, cantidad de hilos vs. speedup.

Para este caso, los valores de N elegidos son:

```
N0 12500000
N1 13500000
N2 17000000
N3 30000000
```

Además, el experimento se planteó de a siguiente manera: dado que el sistema tiene 4 hilos, la ejecución *multithread* ejecuta la operación *daxpy* para una carga distribuida entre estos, y para que la información fuera comparable con *singlethread* se definió que la cantidad de ejecuciones por cada valor de N en *daxpy singlethread* y *multithread* fuera de 100.

3. Un programa/script que permita graficar los resultados después de ejecutar el programa implementado en el punto a y b, de forma que se puede visualizar tiempo de ejecución, speedup, y porcentaje error (para el caso a).

   En los scripts `plot_result.py` se pueden graficar los resultados mostrados en 2 y 3, mientras que en `plot_daxpy.py` se grafica lo mostrado en 5 y 4.

   Con respecto a la figura 3 se encuentran los siguientes hallazgos:

   - La disminución del tiempo de ejecución conforme aumenta el número de threads se debe a que cada thread realiza una parte del cálculo, lo que permite una mayor concurrencia y un uso más eficiente de los recursos de CPU.

- El aumento de % de error conforme aumenta el número de threads podría ser provocado por la paralización de los cálculos ya que podrían provocar condiciones de carrera manejadas por *OpenMP*.
- Similar a los resultados de la figura 2 el speedup aumenta conforme crece el número de threads, sin embargo, esto no es un comportamiento fijo para un problema de paralelización, ya que esta tiene un límite.

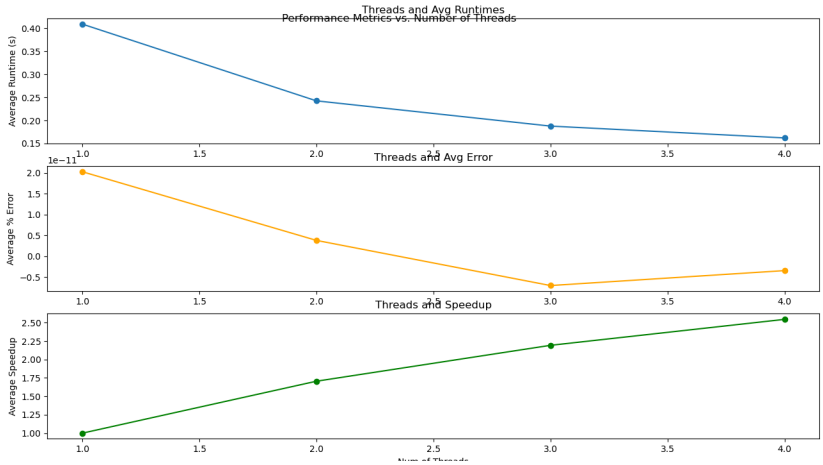

Figura 3: Resultados de los distintos parámetros de evaluación de resultados: cantidad de hilos vs. tiempo de ejecución promedio, cantidad de hilos vs. porcentaje de error, cantidad de hilos vs. speedup. al aproximar la constante e

Por otro lado, comportamiento mostrado en las figuras 4 y 5 con respecto al tiempo de ejecución promedio evidencia de maneja intuitiva que aumentar el tamaño del array de datos aumenta el tiempo que se dura ejecutando la operación *daxpy*, por otro lado la figura 4, de manera interesante, muestra que el tamaño de array que se vio más beneficiado fue $N = 17000000$

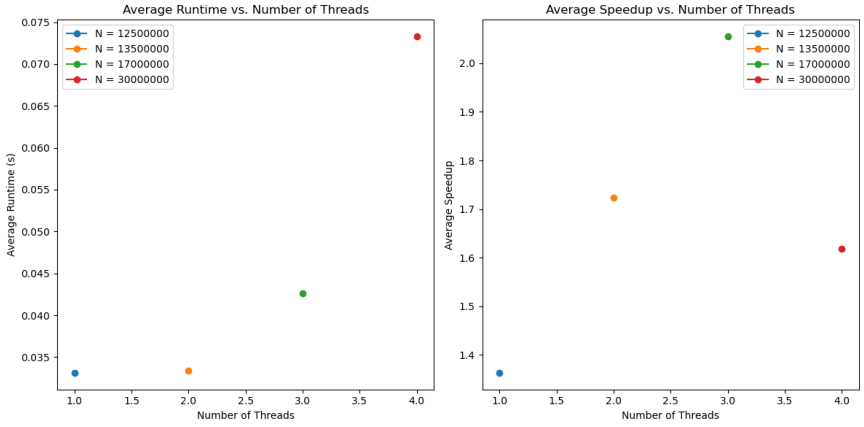

Figura 4: Cantidad de hilos vs. tiempo de ejecución promedio, cantidad de hilos vs. speedup. al ejecutar la función de prueba *daxpy* con *multithreading*.

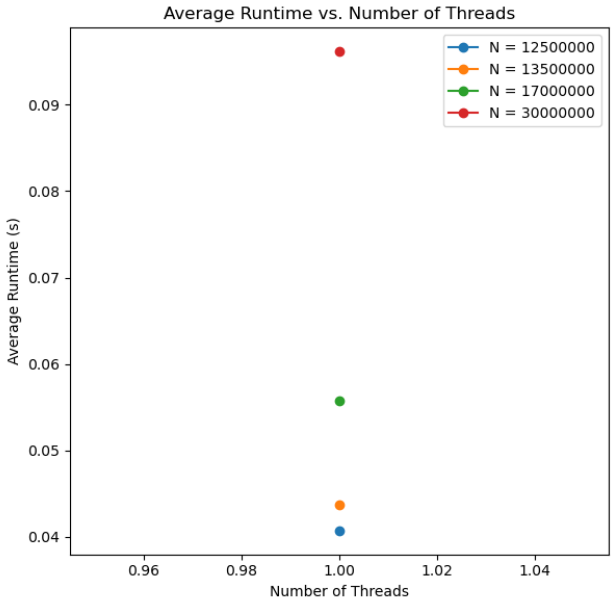

Figura 5: Cantidad de hilos vs. tiempo de ejecución promedio, al ejecutar la función de prueba *daxpy singlethread*.