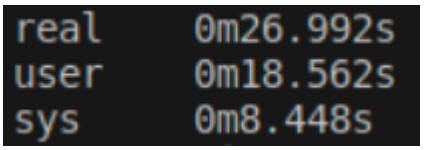
Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Docente: Sebastián Galindo
Semestre 2, 2023

Paola De León, 20361
Paola Contreras, 20213

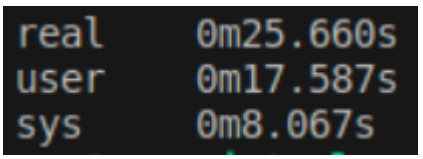
Laboratorio #2

Pruebas

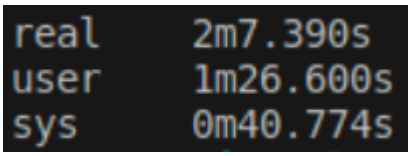
1. Prueba con 50000 datos:
    - Paralelizado

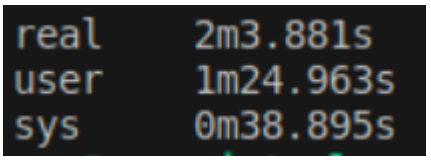
    - Secuencial

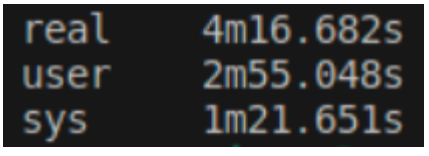
2. Prueba con 100000 datos:
    - Paralelizado

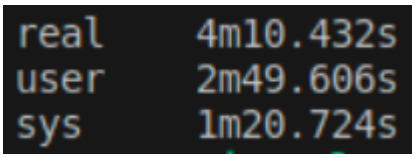
    - Secuencial

3. Prueba con 150000 datos:
    - Paralelizado:

    - Secuencial

Directivas utilizadas

1. *#pragma omp sections*
   Directiva para indicar la sección de código que será paralizada en secciones.
2. *#pragma omp section*
   Directiva para indicar que la sección indicada será ejecutada por 1 hilo de forma paralela con las demás secciones.

Discusión

En la paralelización del programa secuencial del algoritmo sort, se buscó aprovechar las directivas que ofrece OpenMP y así ejecutar el código mediante múltiples cores para agilizar el ordenamiento de los datos generados aleatoriamente. Para este caso, decidimos utilizar #pragma omp sections porque nos permite dividir la tarea de ordenación en dos secciones independientes, cada una de las cuales puede ser ejecutada en paralelo por diferentes hilos.

A pesar de esta intención de mejorar el rendimiento a través de la paralelización, en las pruebas realizadas con diferentes tamaños de datos (100,000, 150,000, etc.), no se obtuvo una mejora significativa en el tiempo de ejecución en todos los casos en comparación con la versión secuencial del algoritmo. Esta falta de mejora podría atribuirse a una serie de factores.

En primer lugar, el overhead introducido por la creación y gestión de hilos podría haber tenido un impacto negativo en el tiempo total de ejecución. Aunque OpenMP facilita la paralelización, la administración de hilos y la sincronización entre ellos conllevan ciertos costos adicionales, lo que puede contrarrestar los beneficios esperados en algoritmos más pequeños o situaciones donde la cantidad de trabajo paralelo es limitada.

Además, es importante considerar la granularidad de las tareas paralelas. Si las tareas son demasiado pequeñas, el overhead de paralelización puede superar el beneficio de la ejecución paralela. En el contexto de un algoritmo de ordenación, como el QuickSort, dividir la tarea en dos secciones puede no ser lo suficientemente granular para aprovechar plenamente los recursos de la CPU.

Adicionalmente, la cantidad de datos puede ser un factor limitante para la paralelización. A medida que la cantidad de datos aumenta, es posible que la capacidad de la memoria y la administración de los hilos se vuelvan un cuello de botella. Si la memoria no es suficiente para manejar múltiples conjuntos de datos parciales en paralelo, podría conducir a tiempos de ejecución más largos debido a la necesidad de cargar y descargar datos repetidamente.