Universidad del Valle de Guatemala
Computación paralela y distribuida
Sección 20

Laboratorio No. 2

Carlos Edgardo López Barrera 21666

Brian Anthony Carrillo Monzon - 21108

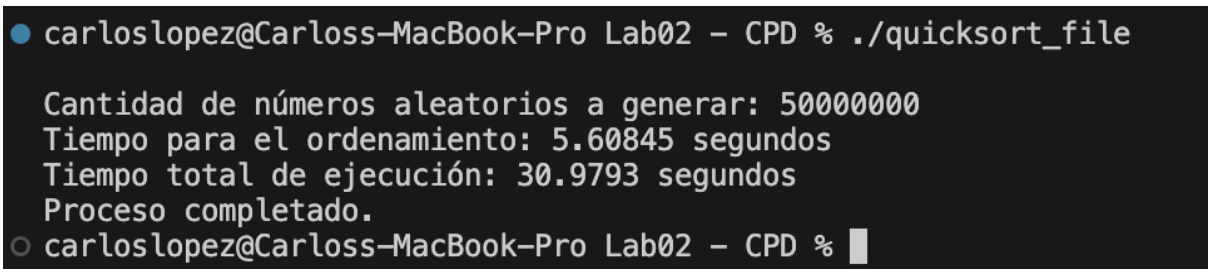
Guatemala, 02 de septiembre del 2024

Speedup

Pruebas realizadas con 50,000,000 de números aleatorios a ordenar.

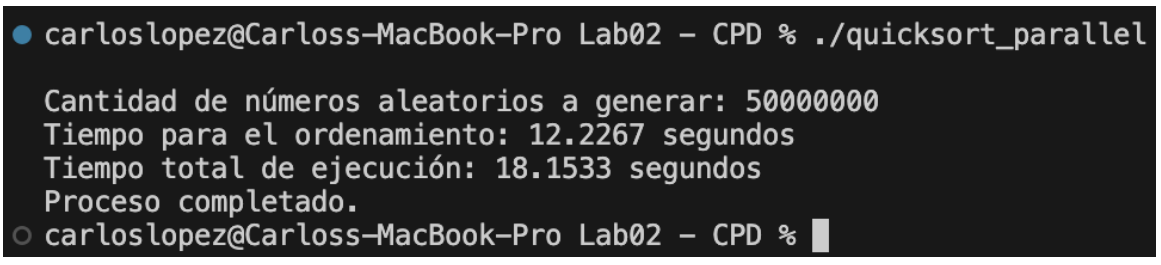
Iteración	Tiempo secuencial (s)	Tiempo paralelo (s)	Speedup	Cambios realizados
1	30.9793	18.1533	1.7065	8 threads
2	30.9793	18.1453	1.7072	Reducción del umbral para crear tareas en #pragma omp task.
3	30.9793	22.6749	1.3662	Paralelización de la lectura de números desde el archivo CSV
4	30.9793	17.8481	1.7357	Utilización todos los threads disponibles del sistema

Ejecución programa secuencial

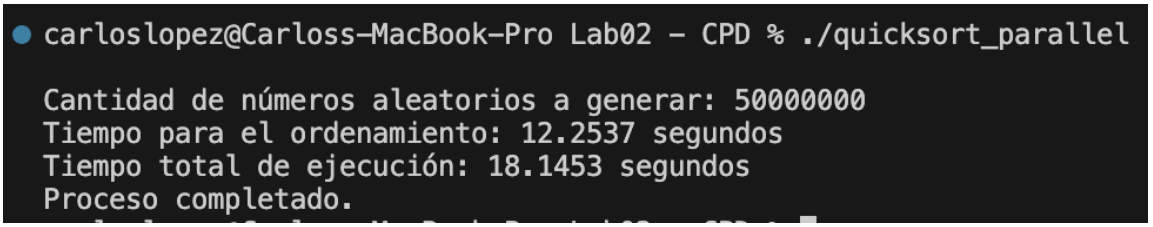


Ejecución programa paralelo

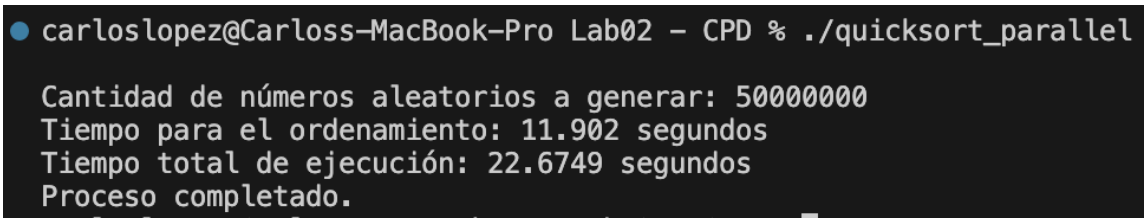
- 8 threads



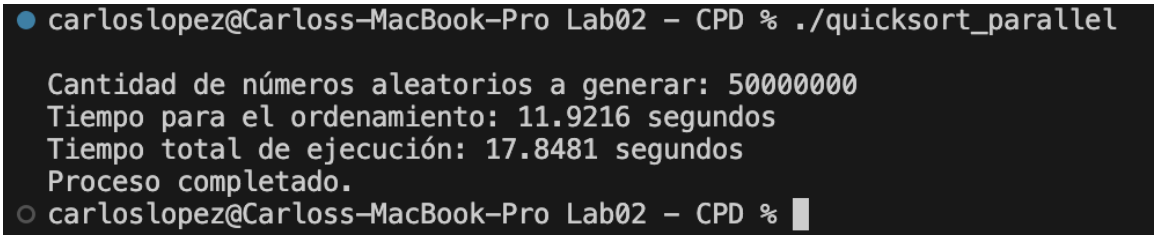
- Reducción del umbral para crear tareas en #pragma omp task.



- Paralelización de la lectura de números desde el archivo CSV



- Utilización de todos los threads disponibles del sistema



Modificaciones finales

- Incorporación de OpenMP
  Se agregó #include <omp.h> para habilitar el uso de OpenMP y así poder paralelizar diferentes secciones del código.

- Paralelización de la Generación de Números Aleatorios
  - Modificación: Se añadió una directiva #pragma omp parallel con #pragma omp for en la generación de números aleatorios.
  - Razonamiento: Como cada número es generado de manera independiente, es una tarea que puede ser distribuida entre múltiples hilos. Con esta modificación permitimos que la tarea sea ejecutada en paralelo, reduciendo el tiempo total de esta operación.

- Paralelización del Algoritmo QuickSort
  - Modificación: Se utilizó la directiva #pragma omp task para paralelizar la ejecución del QuickSort. Se aplicaron tareas en cada partición del arreglo si el tamaño de la partición es suficientemente grande (if(h - lo > 1000) y if(hi - l > 1000)).
  - Razonamiento: QuickSort es un algoritmo recursivo que puede beneficiarse de la paralelización en sus llamadas recursivas. Sin embargo, la creación de demasiadas tareas pequeñas puede llevar a un overhead significativo, por lo que se establece un umbral (1000 en este caso) para crear tareas sólo para particiones grandes, mejorando así la eficiencia del programa.

    - Modificación: Se usó #pragma omp single dentro de una región paralela para asegurar que la primera llamada a quicksort sea realizada por un solo hilo, permitiendo que las tareas recursivas sean paralelizadas.
    - Razonamiento: Usando esta directiva, permitimos que la primera llamada a quicksort se ejecute de manera controlada y luego se paralelice de manera segura en las llamadas recursivas.

- Paralelización de la Lectura de Números desde el Archivo CSV
    - Modificación: Se añadió #pragma omp parallel y #pragma omp for en la lectura de los números desde el archivo CSV.
    - Razonamiento: Al igual que la generación de números, la lectura de datos puede ser paralelizada ya que cada número en el archivo es independiente. Agregando esto, hacemos que se reduzca el tiempo necesario para cargar los datos en el arreglo.

Conclusión

Las modificaciones que hemos introducido han transformado un programa secuencial en una versión paralela, capaz de aprovechar los recursos de hardware disponibles. El uso de OpenMP ha permitido realizar esta transición de manera incremental y controlada, asegurando que solo las partes del programa que realmente se benefician del paralelismo sean modificadas. Con esto hemos logrado un speedup de 1.7357 respecto a la versión secuencial.