Diego Andrés Alonzo Medinilla 20172

Laboratorio 2

Modificaciones al programa

En primer lugar, se paralelizaron los ciclos for, de manera que se paralelizó la generación de los números aleatorios ya que esta parte podría ser independiente el una de la otra, aunque sí se debió de utilizar un omp critical para que funcionara como un mutex y así manejar la memoria compartida que sería el archivo. Así mismo, se paralelizó la lectura del archivo al momento de generar el array de números aleatorios para poder ordenarlos, finalmente se paralelizó el último ciclo for de manera que se paralelizó la escritura y se utilizó un omp critical para asegurar que no ocurran condiciones fronteras. Sin embargo, también nos dimos cuenta de que se podía paralelizar el mismo Quicksort, debido a que los mini arreglos generados durante la ejecución eran independientes unos de los otros, de manera que se podían paralelizar debido a que los mini arreglos y los mini Quicksort podían paralelizarse unos con otros.

Resultados

Secuencial

```
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio1
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio1.cpp -o Ejercicio1
root@a766ec7c71b8:/parallel_computing# ./Ejercicio1
Ingrese el número de elementos: 1000
Tiempo del programa: 0.702222 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio1
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio1.cpp -o Ejercicio1
root@a766ec7c71b8:/parallel_computing# ./Ejercicio1
Ingrese el número de elementos: 10000
Tiempo del programa: 3.3422 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio1
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio1.cpp -o Ejercicio1
root@a766ec7c71b8:/parallel_computing# ./Ejercicio1
Ingrese el número de elementos: 100000
Tiempo del programa: 39.488 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio1
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio1.cpp -o Ejercicio1
root@a766ec7c71b8:/parallel_computing# ./Ejercicio1
Ingrese el número de elementos: 1000000
Tiempo del programa: 1204.21 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio1
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio1.cpp -o Ejercicio1
root@a766ec7c71b8:/parallel_computing# ./Ejercicio1
Ingrese el número de elementos: 10000000
Tiempo del programa: 95549.7 ms
root@a766ec7c71b8:/parallel_computing#
```

Paralelo (optimizado solo en ciclos for's)

```
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 1000
Tiempo del programa: 15.4151 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 10000
Tiempo del programa: 107.267 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 100000
Tiempo del programa: 9442.36 ms
root@a766ec7c71b8:/parallel_computing#
```

Paralelo (optimizado solo en Quicksort)

```
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 1000
Tiempo del programa: 11.4688 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# 10000
bash: 10000: command not found
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 10000
Tiempo del programa: 28.3354 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 100000
Tiempo del programa: 109.336 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 1000000
Tiempo del programa: 998.596 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 10000000
Tiempo del programa: 15558.8 ms
root@a766ec7c71b8:/parallel_computing#
```

Paralelización de todo

```
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
^[[Aroot@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 1000
Tiempo del programa: 13.9171 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
^[[Aroot@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 10000
Tiempo del programa: 149.564 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
^[[Aroot@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 100000
Tiempo del programa: 11983.7 ms
root@a766ec7c71b8:/parallel_computing#
```

Comparación

Con no alcanza se refiere a que no logró concretarse la operación y se quedó esperando

N	Tiempo Secuencial (ms)	Tiempo Paralelo solo for's (ms)	Tiempo Paralelo solo con la paralelización de Quicksort (ms)	Tiempo paralelo con todo (ms)
1000	0.702222	15.4151	11.4688	10.6939
10000	3.3422	107.267	28.3354	142.407
100000	39.488	9442.36	109.336	12236.9
1000000	1204.21	No alcanza	998.596	No alcanza
10000000	95549.7	No alcanza	15558.8	No alcanza

```
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 1000
Tiempo del programa: 15.4151 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 10000
Tiempo del programa: 107.267 ms
root@a766ec7c71b8:/parallel_computing# make clean
rm -f numeros.csv numeros_ordenados.csv
rm -f Ejercicio2
root@a766ec7c71b8:/parallel_computing# make all
g++ -fopenmp -Wall Ejercicio2.cpp -o Ejercicio2
root@a766ec7c71b8:/parallel_computing# ./Ejercicio2
Ingrese el número de elementos: 100000
Tiempo del programa: 9442.36 ms
root@a766ec7c71b8:/parallel_computing#
```

Speed up

N	Speed up for's (ms)	Speedup Quicksort (ms)	Speed up todo (ms)
1000	0.0456	0.0612	0.0656
10000	0.0312	0.1179	0.0235
100000	0.0042	0.3611	0.0032
1000000	No alcanza	1.206	No alcanza
10000000	No alcanza	6.139	No alcanza

Conclusión

Durante este laboratorio se hicieron pruebas de paralelización al momento de modificar código que tenía un algoritmo de divide and conquer además de modificación de archivos. Se obtuvieron los tiempos y se paralelizaron ciertas partes del código, intentando paralelizar ciclos for con el cuál se leía y se escribían los archivos en cuestión, así como paralelización de las divisiones de memoria producto del divide and conquer. Sin embargo, se obtuvieron resultados en los cuáles la paralelización del archivo for era ineficiente esto tiene sentido ya que al haber tantos hilos

realizando diferentes tareas a la vez se haría una cola y un mutex dado que podría dar raise conditions, es por ello que no era en absoluto eficiente. Por el contrario, paralelizar el árbol generado por el divide and conquer si fue eficiente entre más datos se manejaran, es por ello que se logró obtener un speed up del 6.139 a comparación del volumen de datos mínimo donde claramente el secuencial actúa mejor.