Informe Laboratorio 2

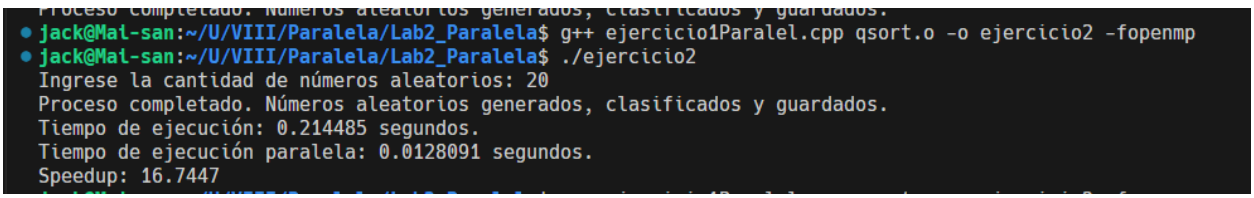
Modificaciones

Primera Modificacion (Generacion de Numeros)

```
#pragma omp parallel for
    for (int i = 0; i < N; ++i)
    {
        int num = rand() % 100; // Generar números entre 0 y 99
#pragma omp critical
        outputFile << num;
        if (i < N - 1)
        {
            outputFile << ",";
        }
    }
    outputFile.close();
```

La primera modificacion fue agregar un omp parallel for a la Generacion de numeros aleatorios a esto le tuvimos que agregar una critical section al guardar el output en un archivo ya que estaba siendo inconsistente a la hora de guardar los datos

SpeedUp de la Modificacion



Esto fue con solo esta seccion en paralelo

Segunda Modificacion (Escribir el archivo)

```
#pragma omp parallel
    {
#pragma omp for
        for (int i = 0; i < N; ++i)
        {
            numbersG[i] = rand() % 100; // Generar números entre 0 y 99
        }

// Nodo maestro escribe en el archivo
#pragma omp master
        {
            for (int i = 0; i < N; ++i)
            {
                outputFile << numbersG[i];
                if (i < N - 1)
                {
                    outputFile << ",";
                }
            }
            outputFile.close();
        }
    }
#pragma omp barrier
```

Al generar los numeros con un paralel for Se escriben en desorden por lo que necesitamos que solo un nodo escriba los datos en el documento para que ordene numero y coma y no pase que se guarden nos numeros seguidos sin la coma

SpeedUp de la Modificacion

```
jack@Mai-san:~/U/VIII/Paralela/Lab2_Paralela$ g++ ejercicio1Paralel.cpp qsort.o -o ejercicio2 -fopenmp
jack@Mai-san:~/U/VIII/Paralela/Lab2_Paralela$ ./ejercicio2
Ingrese la cantidad de números aleatorios: 20
Proceso completado. Números aleatorios generados, clasificados y guardados.
Tiempo de ejecución: 0.419034 segundos.
Tiempo de ejecución paralela: 0.0247015 segundos.
Speedup: 16.9639
jack@Mai-san:~/U/VIII/Paralela/Lab2_Paralela$ ./ejercicio2
```

El speedUp no sufrio un mayor cambio ya que las partes en paralelo siguen siendo las mismas

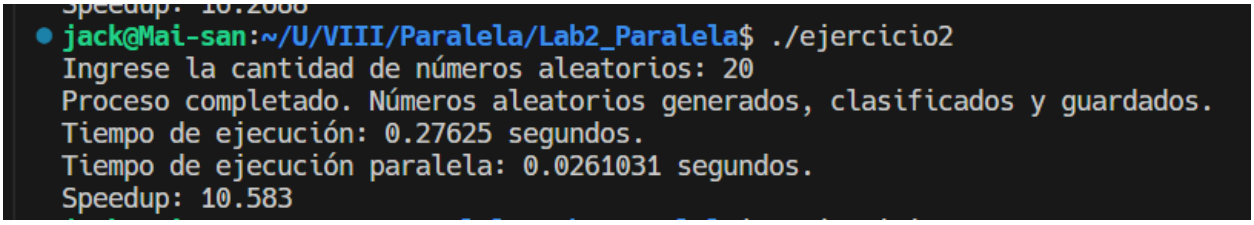
Tercera Modificacion (Lectura del archivo)

```
#pragma omp parallel for // private(comma)
    for (int i = 0; i < N - 1; ++i)
    {
#pragma omp critical
        {
            inputFile >> numbers[i];
            inputFile >> comma;
        }
    }
```

Al leer el archivo de numeros aleatorios no importa mucho el orden que se lean pero si que cada dos

SpeedUp de la Modificacion

El se redujo ya que por todos los mecanismos de sincronia se prodcue mucho overhead



Tercera Modificacion (Escritura de sorted)

```
#pragma omp parallel for
    for (int i = 0; i < N; ++i)
    {
        std::stringstream ss;
        ss << numbers[i];
        if (i < N - 1)
        {
            ss << ',';
        }
        parallelOutput[i] = ss.str(); // Cada hilo escribe su parte en el vector
    }

// Un solo "master thread" se encarga de la escritura
#pragma omp parallel
    {
```

```
#pragma omp master
        {
            for (const auto &str : parallelOutput)
            {
                sortedOutputFile << str;
            }
            sortedOutputFile.close();
        }
    }
```

Este enfoque permite mantener el orden original mientras se aprovecha la paralelización para generar las cadenas que se van a escribir en el archivo. Después de que todos los hilos han terminado, el "master thread" escribe todas estas cadenas en el archivo en el orden correcto.

SpeedUp de la Modificacion

Vemos que el SpeedUp con esta implementacion aumento un poco esto se puede deber un poco a la reduccion del overhead, ademas de que Al utilizar un único "master thread" para la escritura, los datos se escriben de manera más localizada, lo cual podría aprovechar mejor el caché y reducir los fallos de caché