LABORATORIO 2

Francisco Castillo 21562
Andrés Montoya 21552

ENLACE AL REPOSITORIO

https://github.com/Montoya086/Parallel-Number-Classification

MODIFICACIONES AL PROGRAMA SECUENCIAL

Para poder realizar una versión que sea mejor que la paralela fue necesario encontrar las modificaciones que no introdujeran mucho *overhead* a la ejecución del código. Por lo que se realizaron las siguientes modificaciones

PARALELIZACIÓN *PER SE*

La primera modificación (y la más obvia) fue la introducción de OpenMP; con el fin de poder utilizar las herramientas de computación paralela que nos brinda. El objetivo era paralelizar la implementación del algoritmo de **Quicksort**, por lo que las primeras directivas utilizadas fueron para indicar la zona paralela y crear un equipo de hilos que permita optimizar el trabajo. Además, se utilizó una directiva adicional que obliga a que, al inicio, el algoritmo sea ejecutado solamente por un hilo para evitar que múltiples hilos hagan el proceso de ordenamiento.

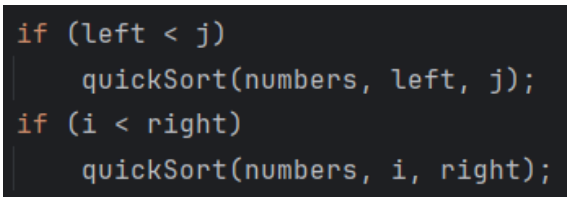
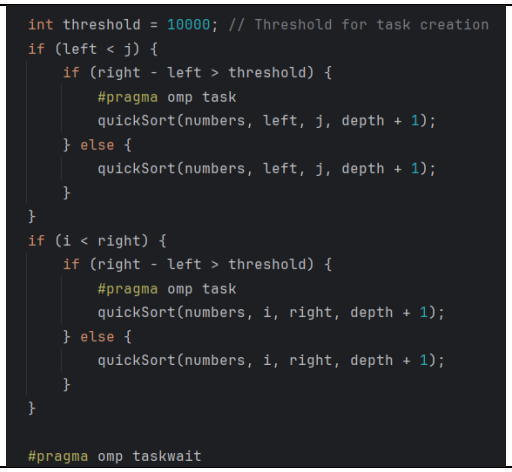
Secuencial	Paralelo

`quickSort(numbers, 0, numbers.size() - 1);`	`#pragma omp parallel`
	`{`
	`  #pragma omp single`
	`  quickSort(numbers, 0, numbers.size() - 1);`
	`}`

UTILIZACIÓN DE *TASKS*

Gracias a la naturaleza del algoritmo, después de hacer la partición, se crea un problema que es resuelto de manera recursiva y, por lo tanto, de manera independiente. Para poder declarar dicha independencia utilizamos *Tasks*, los cuáles nos permiten aprovechar de mejor manera los múltiples procesadores del CPU y tener un mejor balance de carga.

Adicionalmente, se agregó un límite (*threshold*) para la cantidad de *tasks* creados pues, si no se limitan, el *overhead* inducido por la declaración llega a ser perjudicial.

Secuencial	Paralelo
	

```
if (left < j)
    quickSort(numbers, left, j);
if (i < right)
    quickSort(numbers, i, right);
```

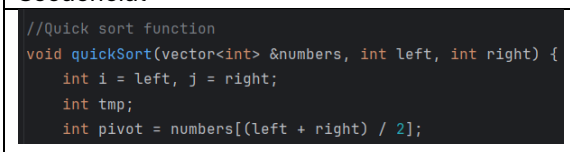
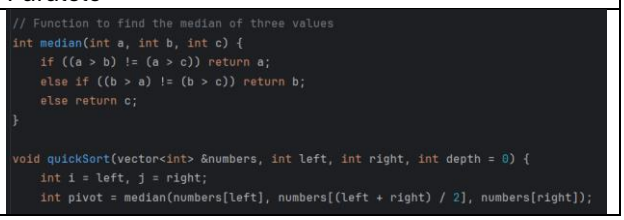
```
int threshold = 10000; // Threshold for task creation
if (left < j) {
    if (right - left > threshold) {
        #pragma omp task
        quickSort(numbers, left, j, depth + 1);
    } else {
        quickSort(numbers, left, j, depth + 1);
    }
}
if (i < right) {
    if (right - left > threshold) {
        #pragma omp task
        quickSort(numbers, i, right, depth + 1);
    } else {
        quickSort(numbers, i, right, depth + 1);
    }
}

#pragma omp taskwait
```

El realizar esto provocaba que algunas tareas terminaran antes, por lo que se agregó una directiva para esperar a que las demás terminen su ejecución. Cuando una tarea llega a dicha barrera libera los recursos que, en caso de que el *threshold* haya sido alcanzado, permite la creación de nuevas tareas.

OTRAS OPTIMIZACIONES

Investigamos diferentes optimizaciones que se pueden realizar al algoritmo (Comput, 2001) y aplicamos el método de la mediana-de-tres. Esto permite que exista una mejor selección del pivot y una reducción considerable de apariciones del peor caso para el algoritmo. Además, introduce un efecto de ordenamiento parcial; con los tres elementos ya comparados y ordenados, se realizan menos comparaciones.

Secuencial	Paralelo
	

```
//Quick sort function
void quickSort(vector<int> &numbers, int left, int right) {
    int i = left, j = right;
    int tmp;
    int pivot = numbers[(left + right) / 2];
```

```
// Function to find the median of three values
int median(int a, int b, int c) {
    if ((a > b) != (a > c)) return a;
    else if ((b > a) != (b > c)) return b;
    else return c;
}

void quickSort(vector<int> &numbers, int left, int right, int depth = 0) {
    int i = left, j = right;
    int pivot = median(numbers[left], numbers[(left + right) / 2], numbers[right]);
```

REFERENCIAS

Comput, S. 2001. *OPTIMAL SAMPLING STRATEGIES IN QUICKSORT AND QUICKSELECT*. https://www.cs.upc.edu/~conrado/research/papers/sicomp-mr01.pdf