

Zach Serna  
Nicole Serna  
CPSC 479  
Professor Doina Bein

CPSC 479  
Project 2: Parallel Quicksort Using OpenMP

## Group Members:

Zach Serna: [serna648@csu.fullerton.edu](mailto:serna648@csu.fullerton.edu)  
Nicole Serna: [NicoleSerna@csu.fullerton.edu](mailto:NicoleSerna@csu.fullerton.edu)

## Language Used: C++

## How to Compile:

1. add the file to your directory
2. compile the .cpp file using: `g++ -fopenmp (filename).cpp -o (desiredexecname)`
3. run the exe with: `./ (desiredexecname)`
4. Note: Numbers being sorted can be modified under the void run() function. Simply change the numbers in int arr[].

## Description:

Parallel implementation of the QuickSort Algorithm utilizing C++ and OpenMP. Quicksort uses a pivot point to partition a list of numbers into sub-arrays, with numbers greater than the pivot put into one array and values less than the pivot in the other; the sub-arrays are then sorted recursively. The pivot point in our program is chosen using the Median of Three method. This method compares the first, middle, and last number of an array and then selects the median of the three values as the pivot.

To make this process parallel, the program uses `#pragma omp parallel sections` and `#pragma omp section` to divide Quicksort's sorting and partitioning process among parallel running sections. `#pragma omp parallel sections` combines `Omp Parallel` and `Omp Sections`, which are commands that tell the compiler to parallelize the following lines of code within the block and divide the parallelized block of code among the threads to be executed, respectively.

Our motivation behind this project was to make a sorting algorithm that could more efficiently sort large amounts of data. Quicksort's divide-and-conquer formulation makes it an ideal candidate for parallelization.

## Issues Encountered:

We encountered issues when trying to print the outputs of our sorts. The output statements would be mixed up with one another, likely due to the threads fighting over the cout stream. We mitigated this by assigning the outputs to a string, which would then be printed all at once instead of in parts.

## Shortcomings:

None!

## Pseudocode:

//Project 479 PseudoCode:

//Class ParallelQSort

//variables. . .

Numthread = 0

//functions

**partition(array, first, last)**

{

    //pivot point is chosen by using median of three method

    //after function is ran first element is used as pivot

    //iterate through list using last value and second value

    //loop continues until new parition/pivot is found

    while (true)

    {

        //move values to appropriate side based on pivot

        //less than go to the left, greater than go to right

    }

**MedianofThree()**

{

```

//an ideal pivot point is the median of a list
//selecting the first element as the pivot increases the chance of the worst
possible time complexity
//however going through the entire list for the median is not efficient
//taking all that into consideration we used the Median of Three Method to choose a
pivot

//we compare the first, middle, and last elements in the list
//and use the median element as the pivot

//the elements are swapped so that
//middle < first < last
//because our partition function uses the first element as the pivot
//we want the first element to be the median of the three
}

```

### **QuickSort(array, first, last)**

```

{
    //variables. . .
    Partionpoint = partition(array, first, last)

    //pragma omp parallel sections
    {
        // #pragma omp section
        {
            //threadnum = threadnum + 1;
            //Recursive call QuickSort(arr, first, partitionPoint-1);
        }
        // #pragma omp section
        {
            //threadnum = threadnum + 1;
            // Recursive call quickSort(arr, partitionPoint+1, last);
        }

        //array setup will be done by user prior to compile
Run()
    {
        array = [whatever numbers the user desires]
        size = size of array
        QuickSort(array, 0, size - 1)
    }
}

```

```

//Main
{
    //variables. . .
    //declare parallel quicksort object

    //Object will call the Run() function
    //Print Output after being sorted
    //return 0; DONE
}

```

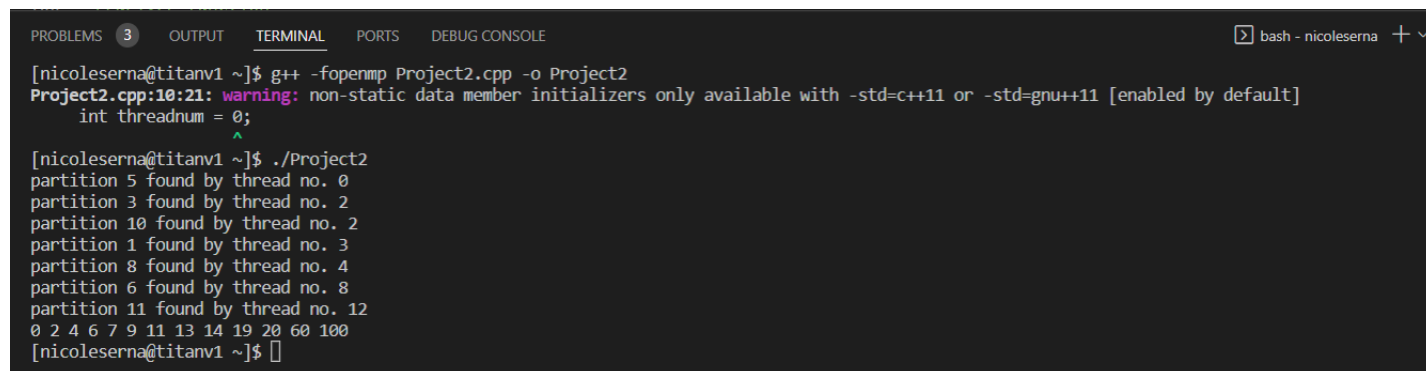
## Screenshots:

## GROUP MEMBERS:

Zach Serna [serna648@csu.fullerton.edu](mailto:serna648@csu.fullerton.edu)

Nicole Serna [NicoleSerna@csu.fullerton.edu](mailto:NicoleSerna@csu.fullerton.edu)

## Test Run 1:



```

PROBLEMS 3 OUTPUT TERMINAL PORTS DEBUG CONSOLE
[nicoleserna@titanv1 ~]$ g++ -fopenmp Project2.cpp -o Project2
Project2.cpp:10:21: warning: non-static data member initializers only available with -std=c++11 or -std=gnu++11 [enabled by default]
    int threadnum = 0;
    ^
[nicoleserna@titanv1 ~]$ ./Project2
partition 5 found by thread no. 0
partition 3 found by thread no. 2
partition 10 found by thread no. 2
partition 1 found by thread no. 3
partition 8 found by thread no. 4
partition 6 found by thread no. 8
partition 11 found by thread no. 12
0 2 4 6 7 9 11 13 14 19 20 60 100
[nicoleserna@titanv1 ~]$

```

## Test Run 2:

```
[nicoleserna@titanv1 ~]$ g++ -fopenmp Project2.cpp -o Project2
Project2.cpp:10:21: warning: non-static data member initializers only available with -std=c++11 or -std=gnu++11 [enabled by default]
    int threadnum = 0;
                    ^
```

```
[nicoleserna@titanv1 ~]$ ./Project2
partition 6 found by thread no. 0
partition 3 found by thread no. 2
partition 12 found by thread no. 2
partition 1 found by thread no. 3
partition 10 found by thread no. 4
partition 4 found by thread no. 7
partition 8 found by thread no. 8
partition 17 found by thread no. 14
partition 14 found by thread no. 15
partition 15 found by thread no. 17
partition 24 found by thread no. 20
partition 21 found by thread no. 21
partition 19 found by thread no. 22
partition 22 found by thread no. 25
partition 26 found by thread no. 28
0 1 2 4 6 7 9 11 13 14 15 17 19 20 23 24 38 40 57 60 64 71 78 85 90 100 315 417
[nicoleserna@titanv1 ~]$
```