# HPC Mini Project Synopsis

**Project Title:** Parallel Quick-Sort using OpenMP on different size of Datasets.

**Team Members:** 1.Yoganand Kanhed (402077)

2.Vaibhav Gole (402062)

3.Rishikesh Kakad (402075)

**Project Domain:** High Performance Computing.

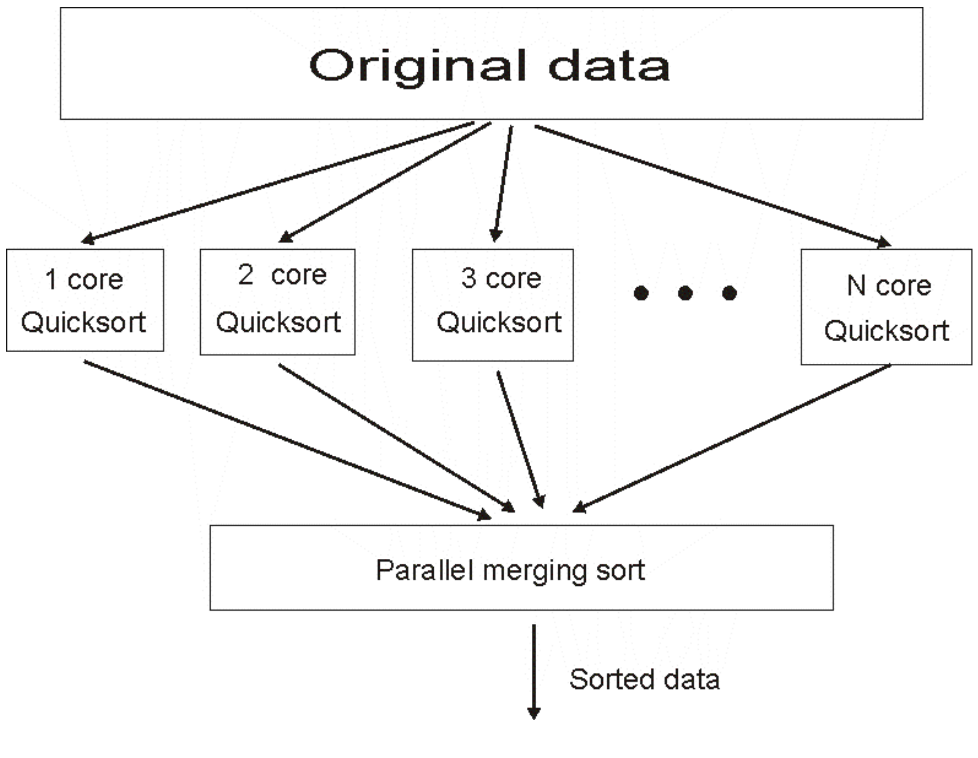
## Project Description:

Sorting is an important part of high-performance multiprocessing. We implemented the parallel Quick sort. This project is parallelizing the quicksort algorithm using openMP According to avoid the initial partitioning of data and merging step in all the processes, we could get a performance improvement for this implementation

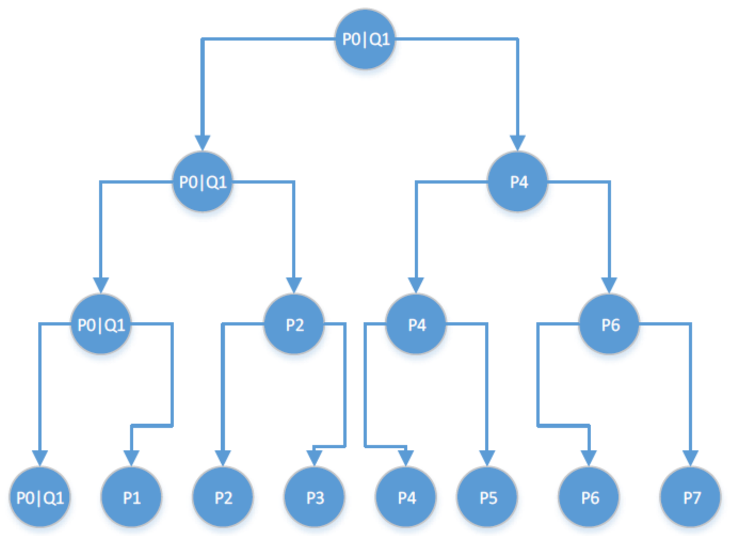
**Input**: Different size of Datasets.

**Output**: The parallel quicksort have advantage with big datasets with the size of data set growing, it performance will be better and better.

## High Level Design:



## Low Level Design:



## Algorithm steps:

1. Pick an element, called a pivot, from the array.

2. Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

3. Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

## Implementation Steps:

1. Perform an initial data partition and give the two subsets to two available processes.

2. Each process performs the data partition if there is any available process. Then send a sub part of data to the available process.

3. Perform the ordinary quick sort on the sub data if no more process is available.

4. Send the sorted sub data to its sender.

5. Get the exact sorted data set

**Results:**

We sorted 2MB,4MB and 8MB dataset in this project. We implement a java code for generating integer dataset. Take a 200000 integers dataset as 2M, 400000 as 4M and so on.

| **Size of dataset:** | **Parallel execution time(second):** | **Serial execution time(second) :** |
|---|---|---|
| 2 | 0.250631 | 0.155379 |
| 4 | 0.477811 | 0.588359 |
| 8 | 0.92485 | 2.17711 |

**Running time**

On average, the serial quicksort algorithm takes ***O (n log n)*** comparisons to sort n items. Analysis the running time of parallel quicksort according this. Assume that each partition creates two equal sub sets,
For 2 processes,
***T(n) = (n/2) log(n/2).***

For 3 processes,
***T(n) = (n/2) log(n/2) + (n/4) log(n/4) = (n/2) log(n/2).***

Actually, in our implementation, we consider log2(2) equals to log2(3).
So, for 3 processes,
***T(n) = (n/2) log(n/2)***, indeed.

For 4 processes,
***T(n) = (n/4) log(n/4)***

Thus, for $p$ processes,
***T(n) = (n/k) log(n/k) where*** $2^{k-1}<p\leq 2^k$

## Software Requirements:

CUDA, OpenMP libraries, Eclipse.

## Hardware Requirements:

Intel Quad Core i5 or i7 processor

4GB RAM

Nvidia GTX or higher for optimal performance.

## Conclusion:

The parallel quicksort using OpenMP, we got a chance to have a further understanding of OpenMP and parallel algorithm, especially sorting algorithm. We wondering if there is some better strategy of the pivot choice also better load balance. Generally speaking, parallel algorithm is powerful in dealing with big data. Our world today is a big data world. We should bring more parallel algorithm into our real life, to improve the life quality.