

Venkteshprasad Maya Rao (001087357)

## **Program Structures & Algorithms**

**Fall 2021**

### **Assignment No. 5(Parallel Sorting)**

- ◉ **Task (List down the tasks performed in the Assignment)**
- ◉ **Relationship Conclusion: (For ex :  $z = a * b$ )**
- ◉ **Evidence to support the conclusion:**
  1. **Output (Snapshot of Code output in the terminal)**
  2. **Graphical Representation(Observations from experiments should be tabulated and analyzed by plotting graphs(usually in excel) to arrive on the relationship conclusion)**
- ◉ **Unit tests result:(Snapshot of successful unit test run)**

Parallel sorting algorithm has been implemented such that each partition is sorted in parallel.

[https://github.com/venkteshgm/INFO6205-Assignments/blob/Fall2021/src/main/java/edu/neu/coe/info6205/union\\_find/UF\\_HWQUPC.java](https://github.com/venkteshgm/INFO6205-Assignments/blob/Fall2021/src/main/java/edu/neu/coe/info6205/union_find/UF_HWQUPC.java)

The system that the experiment is being run on has 12 processor units.

Program was run for array lengths from 1 million to 4 million, with 1 million increments, totalling 4 runs per array length.

Thread counts were varied from 2 to 32 in powers of 2(2,4,8,16,32).

Cutoff times were varied from 50,000 to 2,000,000 in increments of 100,000.

Minimum time taken for each array length and thread count, and its corresponding cutoff values have been recorded, as follows.

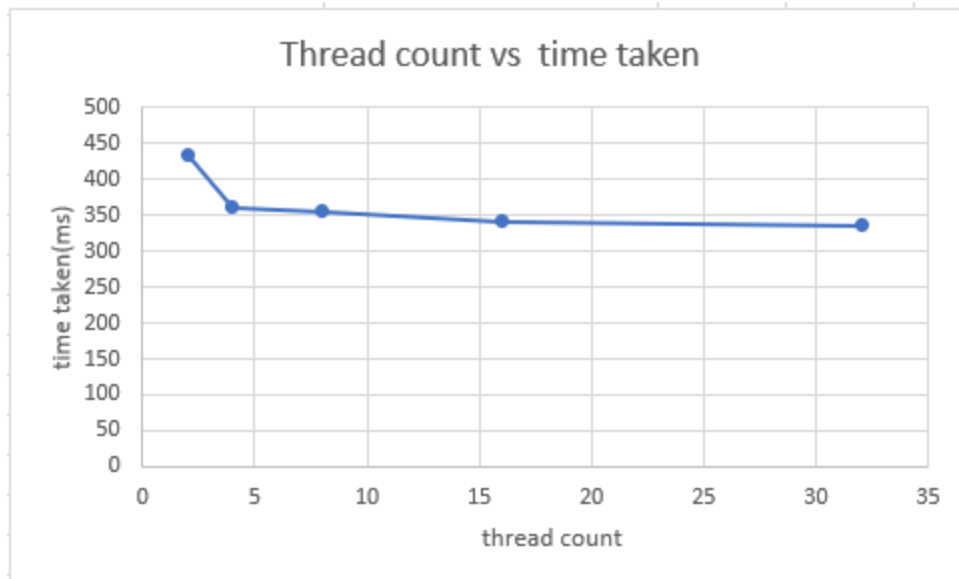
Main output:

```

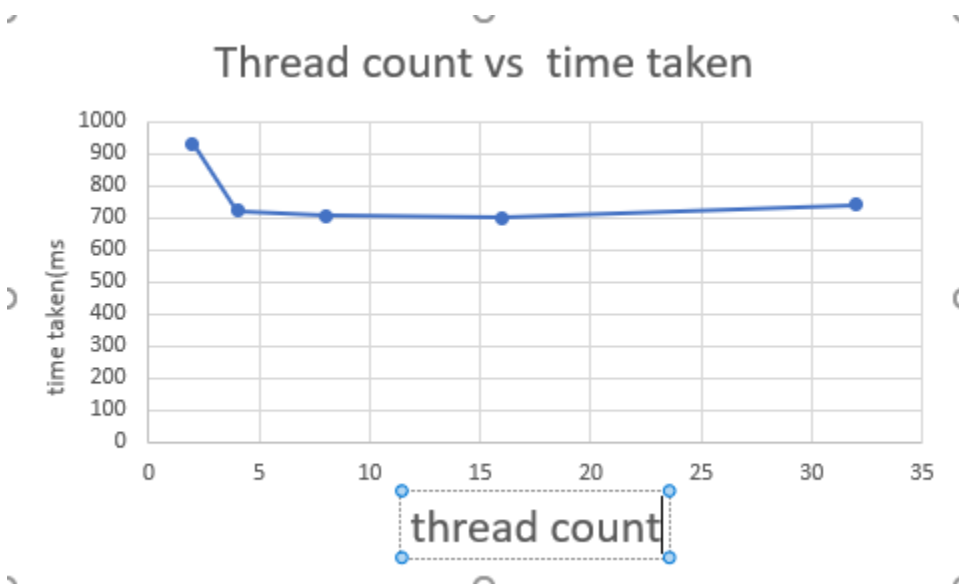
Main X
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" ...
Degree of parallelism: 11
processor count: 12
for arrayLen: 1000000 threadCount: 2 threads, meanTime: 575 minTime: 435 optimal cutoff: 250000
for arrayLen: 1000000 threadCount: 4 threads, meanTime: 539 minTime: 361 optimal cutoff: 350000
for arrayLen: 1000000 threadCount: 8 threads, meanTime: 539 minTime: 355 optimal cutoff: 250000
for arrayLen: 1000000 threadCount: 16 threads, meanTime: 540 minTime: 341 optimal cutoff: 150000
for arrayLen: 1000000 threadCount: 32 threads, meanTime: 535 minTime: 336 optimal cutoff: 250000
for arrayLen: 2000000 threadCount: 2 threads, meanTime: 974 minTime: 932 optimal cutoff: 1450000
for arrayLen: 2000000 threadCount: 4 threads, meanTime: 859 minTime: 724 optimal cutoff: 150000
for arrayLen: 2000000 threadCount: 8 threads, meanTime: 864 minTime: 706 optimal cutoff: 950000
for arrayLen: 2000000 threadCount: 16 threads, meanTime: 864 minTime: 702 optimal cutoff: 850000
for arrayLen: 2000000 threadCount: 32 threads, meanTime: 877 minTime: 741 optimal cutoff: 850000
for arrayLen: 3000000 threadCount: 2 threads, meanTime: 1542 minTime: 1383 optimal cutoff: 1950000
for arrayLen: 3000000 threadCount: 4 threads, meanTime: 1296 minTime: 1113 optimal cutoff: 950000
for arrayLen: 3000000 threadCount: 8 threads, meanTime: 1266 minTime: 1081 optimal cutoff: 550000
for arrayLen: 3000000 threadCount: 16 threads, meanTime: 1262 minTime: 1108 optimal cutoff: 850000
for arrayLen: 3000000 threadCount: 32 threads, meanTime: 1238 minTime: 1121 optimal cutoff: 1150000
for arrayLen: 4000000 threadCount: 2 threads, meanTime: 2122 minTime: 1937 optimal cutoff: 150000
for arrayLen: 4000000 threadCount: 4 threads, meanTime: 1625 minTime: 1406 optimal cutoff: 1150000
for arrayLen: 4000000 threadCount: 8 threads, meanTime: 1574 minTime: 1405 optimal cutoff: 850000
for arrayLen: 4000000 threadCount: 16 threads, meanTime: 1599 minTime: 1436 optimal cutoff: 1850000
for arrayLen: 4000000 threadCount: 32 threads, meanTime: 1582 minTime: 1419 optimal cutoff: 1850000

Process finished with exit code 0
```

From the results, the following graphs have been plotted of thread count vs time taken



For array length of 1,000,000



For array length of 2,000,000

Similar results can be found for array lengths of 3 and 4 million as well, as seen from the output.

The performance seems to be maximum around the 8-16 thread count, which is the closest to the processor count of the system.

Having all processors running and not having them remain idle, while not leaving any threads waiting for processors to pick it up, results in the best performance.

In other words, most processor engagement and minimal context switching between threads is present when the number of thread executions are the same or closest to the number of processors available.

8-16 threads is closest to the processor count of 12.

Cutoff determines the number of subtasks created for the different threads to pick up and execute. The lower the cutoff value, the greater the number of subtasks.

Hence, performance improvements can be seen when the cutoff value results in number of subtasks nearing number of threads of the experiment. It is not optimal in all cases, but in the results posted above, for array length 1,000,000, the optimal cutoff is 150,000. The ratio is 6.67 which is closest to 8 among the thread counts.

Thus, having the thread count be the closest to the processor count, and choosing a cutoff time  $C$  for array length  $A$ ,  
Such that  $A/C = \text{thread count } T$  leads to optimal results.