Program Structures & Algorithms

Spring 2022

Assignment No. 4

Parallel Sorting

Thomas John
NEU ID: 002933800

Task

Please see the presentation on *Assignment on Parallel Sorting* under the *Exams. etc.* module.

Your task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (*t*) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of *lg t* is reached).

3. An appropriate combination of these.

There is a *Main* class and the *ParSort* class in the *sort.par* package of the INFO6205 repository. The *Main* class can be used as is but the *ParSort* class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].

Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.

You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of a parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.

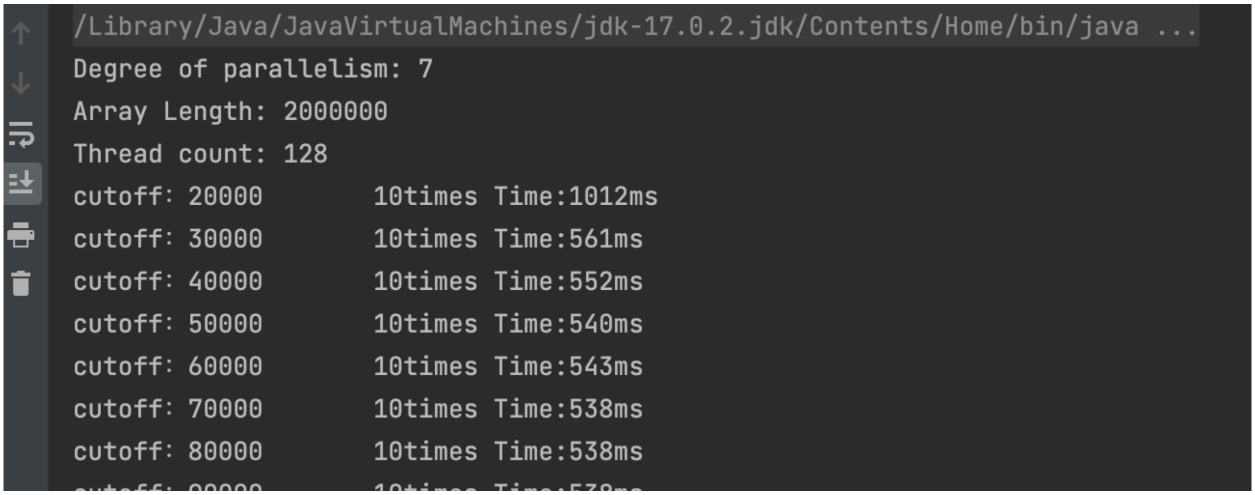
For varying the number of threads available, you might want to consult the following resources:

- https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-pool
- (Links to an external site.)
-
- https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable
- (Links to an external site.)
-

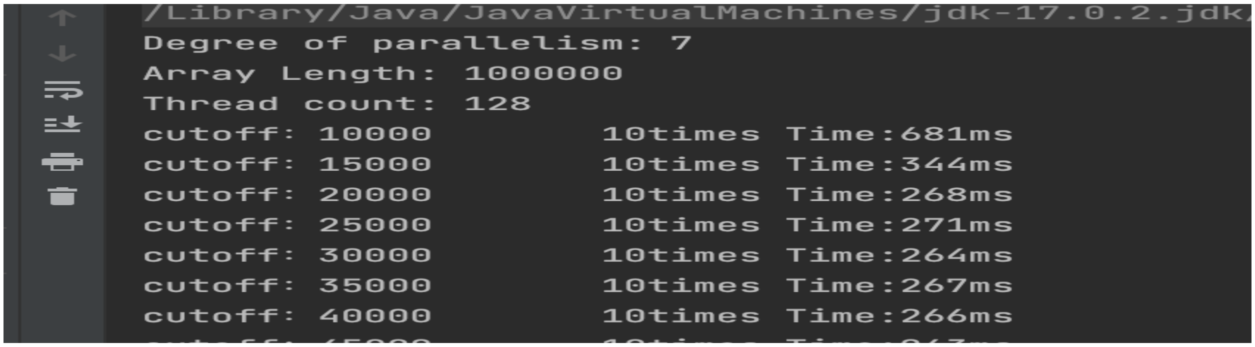
Good luck and enjoy.

Github URL:
https://github.com/thomasjohn-neu/INFO6205/commit/544e25c8f7f26e4ebdb92be90310076fbd109a73

Output

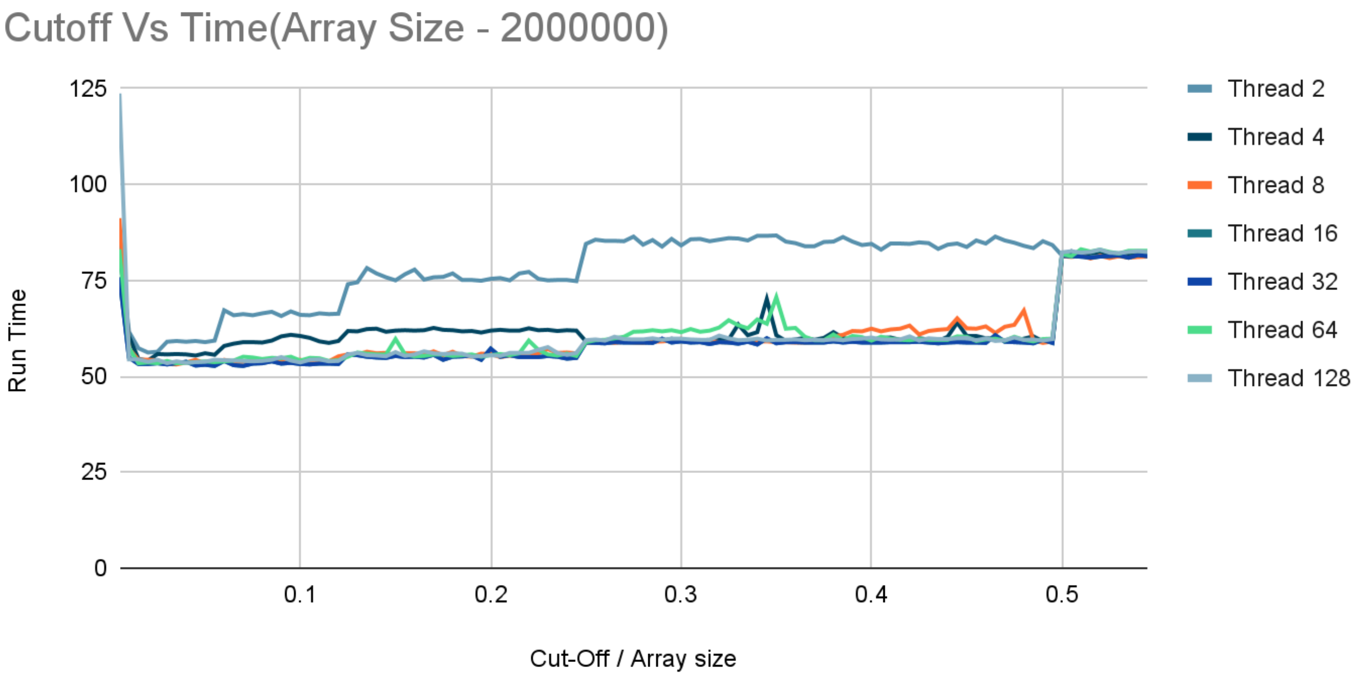


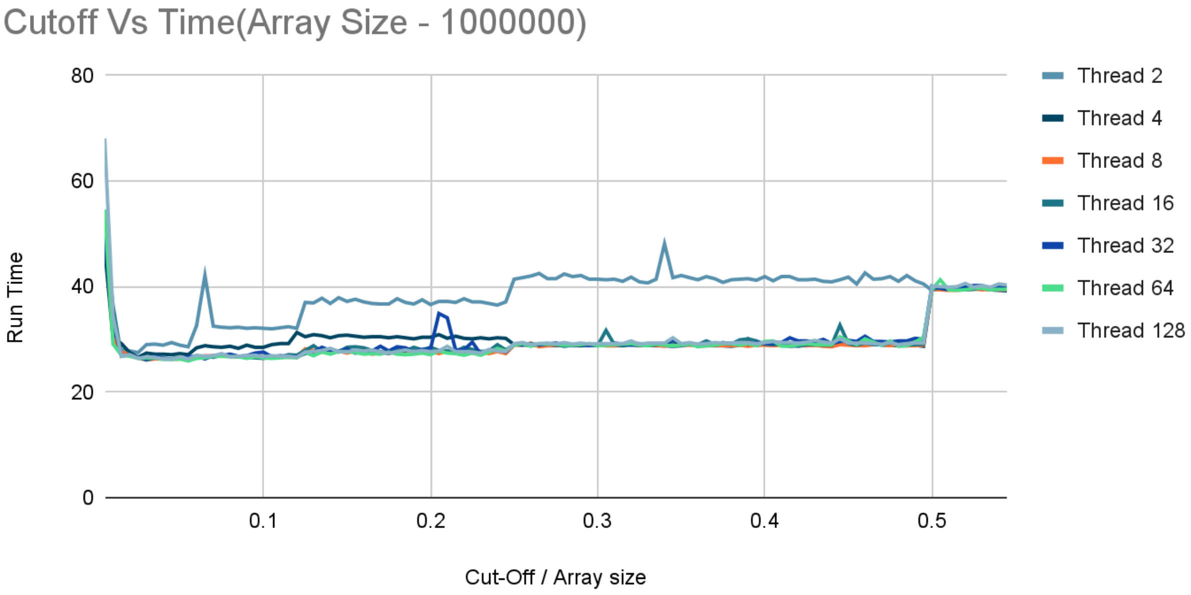
```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
Degree of parallelism: 7
Array Length: 2000000
Thread count: 128
cutoff: 20000       10times Time:1012ms
cutoff: 30000       10times Time:561ms
cutoff: 40000       10times Time:552ms
cutoff: 50000       10times Time:540ms
cutoff: 60000       10times Time:543ms
cutoff: 70000       10times Time:538ms
cutoff: 80000       10times Time:538ms
cutoff: 90000       10times Time:538ms
```



```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/
Degree of parallelism: 7
Array Length: 1000000
Thread count: 128
cutoff: 10000       10times Time:681ms
cutoff: 15000       10times Time:344ms
cutoff: 20000       10times Time:268ms
cutoff: 25000       10times Time:271ms
cutoff: 30000       10times Time:264ms
cutoff: 35000       10times Time:267ms
cutoff: 40000       10times Time:266ms
cutoff: 45000       10times Time:263ms
```

Degree of parallelism: 7
Array Length: 2000000, 1000000
Thread count: 2, 4, 8, 16, 32, 64, 128
Number of cores in test machine: 8


Cutoff Vs Time(Array Size - 2000000)
Run Time
125
100
75
50
25
0
0.1
0.2
0.3
0.4
0.5
Cut-Off / Array size
Thread 2
Thread 4
Thread 8
Thread 16
Thread 32
Thread 64
Thread 128


Cutoff Vs Time(Array Size - 1000000)
Run Time
80
60
40
20
0
0.1
0.2
0.3
0.4
0.5
Cut-Off / Array size
Thread 2
Thread 4
Thread 8
Thread 16
Thread 32
Thread 64
Thread 128

Conclusion

1. Any number of threads for any sized array, after 0.5 cut-offs of array size, runtime remains similar.
2. Below the 0.5 cut-offs, the more the threads increase the sorting efficiency.
3. Threads are dependent on the core, more cores, support more threads.

Code Repository

https://github.com/thomasjohn-neu/INFO6205/commit/451d41a5b7152fdb9eeebc9b2adf09eebe207fc8