Difeng Li (NUID: 001050353)

INFO 6205

Program Structures & Algorithms

Fall 2020

Assignment No.5

- **Task:**
  The task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel.
  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.
  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).
- **Output:** The following is my test results, which are based on the ratio of different cutoff values to size values. (confirm when thread is 10)

Difeng Li (NUID: 001050353)

Degree of parallelism: 10		
Array Size 50000		
250	39	0.005
500	20	0.01
1000	8	0.02
2000	16	0.04
4000	15	0.08
8000	27	0.16
16000	31	0.32
32000	27	0.64
64000	41	1.28
128000	32	2.56
256000	35	5.12

Degree of parallelism: 10		
Array Size 100000		
1000	48	0.01
2000	24	0.02
4000	26	0.04
8000	30	0.08
16000	49	0.16
32000	28	0.32
64000	64	0.64
128000	89	1.28
256000	83	2.56
512000	73	5.12

Degree of parallelism: 10		
Array Size 400000		
1000	111	0.0025
2000	74	0.01
4000	69	0.02
8000	62	0.04
16000	86	0.08
32000	98	0.16
64000	71	0.32
128000	131	0.64
256000	157	1.28
512000	328	2.56
1024000	329	5.12

Degree of parallelism: 10		
Array Size 800000		
1000	190	0.00125
2000	158	0.0025
4000	130	0.005
8000	167	0.01
16000	114	0.02
32000	138	0.04
64000	141	0.08
128000	141	0.16
256000	187	0.32
512000	307	0.64
1024000	680	1.28
2048000	676	2.56
4096000	678	5.12

Here's an experiment to test the optimal number of threads. The ratio of cutoff to size is 0.2.

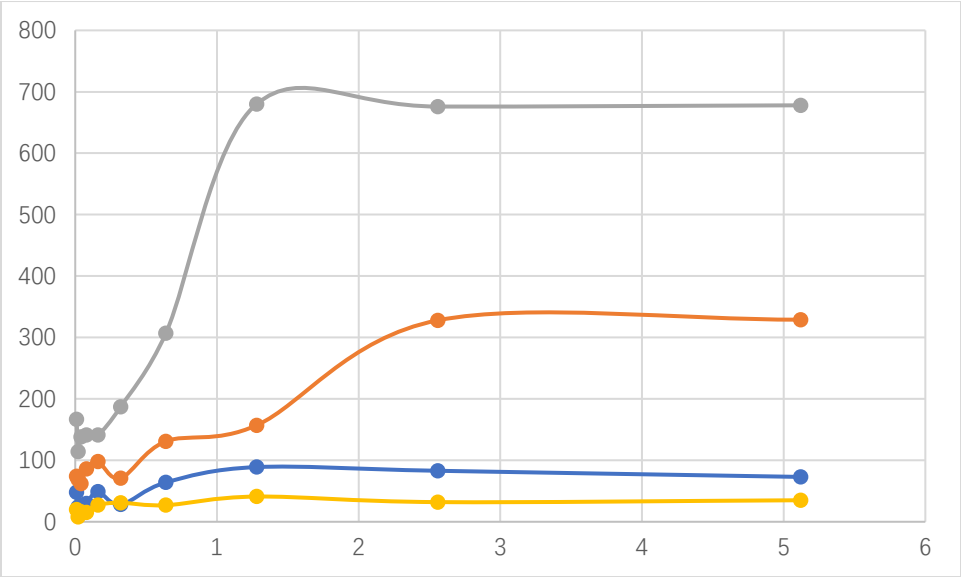
Array Size 100000	
Cutoff 20000	
Threads	Time
1	99
2	76
3	73
4	68
5	67
6	68
7	71
8	70
9	72
10	73
20	75
30	74
50	76
100	76

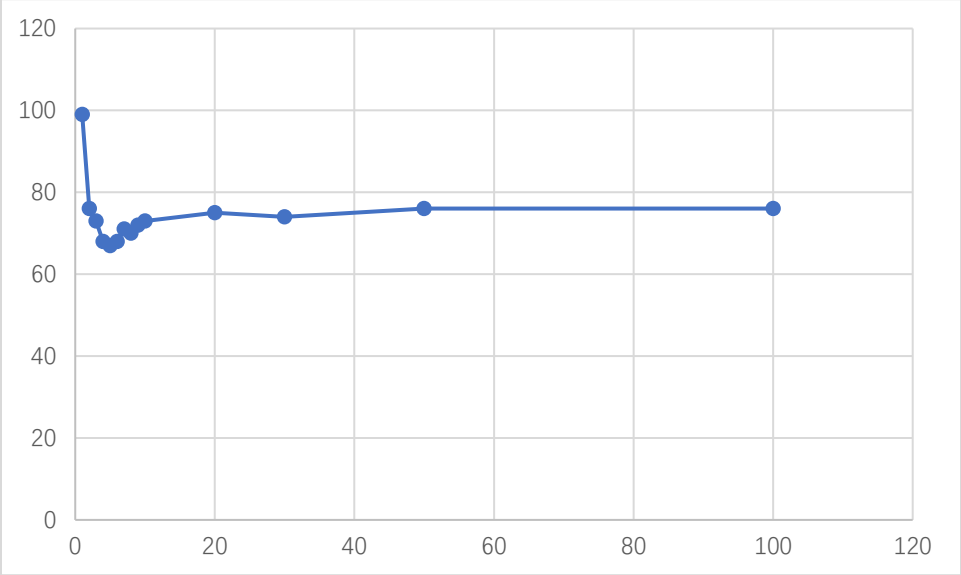
- **Relationship conclusion**

The best ratio of cutoff to size is 0.16, and the optimal ratio is 500001000000400000080000O.

For the optimal number of threads, the experimental results show that when the number of experimental threads is more than 10, the later time is almost stable in an interval. When the number of threads is 5, there is the least time, so the optimal number of threads is 5, and the time will not change with the number of threads.

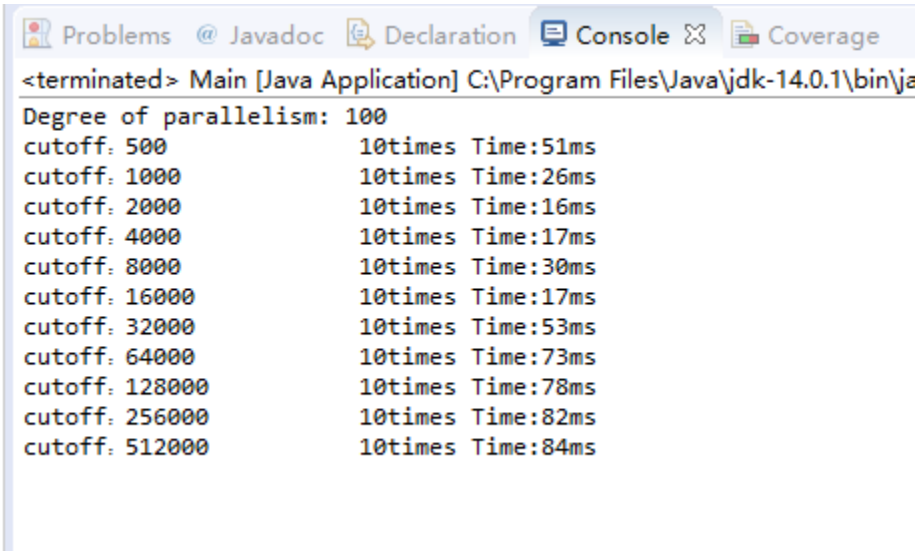
- **Evidence to support relationship**





Difeng Li (NUID: 001050353)

- **Screenshot of Unit test passing**




```
class ParSort {

    public static int cutoff = 1000;

    public static void sort(int[] array, int from, int to) {
        if (to - from < cutoff) Arrays.sort(array, from, to);
        else {
            CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2); // TO IM
            CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to); // TO IMPL
            CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) -> {
                int[] result = new int[xs1.length + xs2.length];
                int i = 0, j = 0;
                for(int a = 0; a < result.length; a++) {
                    if(i >= xs1.length) result[a] = xs2[j++];
                    else if(j >= xs2.length) result[a] = xs1[i++];
                    else if(xs1[i] > xs2[j]) result[a] = xs2[j++];
                    else  result[a] = xs1[i++];
                }
                // TO IMPLEMENT
                return result;
            });

            parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array, from, resul
     //      System.out.println("# threads: "+ Main.p.getRunningThreadCount());
            parsort.join();
        }
    }

    private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
        return CompletableFuture.supplyAsync(
                () -> {
                    int[] result = new int[to - from];
                    // TO IMPLEMENT
                    System.arraycopy(array, from, result, 0, result.length);
                    int mid = (to -from)/2;
                    if(array[mid] < array[mid+1]) return result;
                    else
                    sort(result, 0, to - from);
                    return result;
                }, Main.p
        );
    }
}
```