INFO6205 PROGRAM STRUCTURE AND ALGORITHMS ASSIGNMENT NO. 5

PARALLEL SORTING

Task: To implement a parallel sorting algorithm such that each array partition is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

Problem Explanation (Parallel Sorting): The parallel sorting process involves distributing sorting tasks among several processors or computing nodes to speed up the sorting process. By utilising parallelism, parallel sorting algorithms divide the sorting task into smaller subtasks, sort them independently, and then merge their results to obtain the final sorted result.

The input data is divided into smaller chunks and distributed among multiple processing units in a parallel sorting algorithm. Each processing unit sorts its portion of the data based on its own sorting algorithm. In a series of steps, the sorted chunks are merged until the entire dataset is sorted. In the process of merging, two sorted sublists are compared and merged at a time until the entire dataset is sorted.

CODE SNAPSHOTS:

Main.java × ParSort.java ×

Source History

```java
public class Main {

    public static void main(String[] args) {
        processArgs(args);
        int thread = 2, arraySize = 600000;
        while (thread < 65) {
            ForkJoinPool pool = new ForkJoinPool( parallelism:thread);
            System.out.println("Degree of parallelism: " + pool.getParallelism());
            Random random = new Random();
            int[] array = new int[arraySize];
            ArrayList<Long> timeList = new ArrayList<>();
            for (int j = 0; j < 10; j++) {
                ParSort.cutoff = 8000 * (j + 1);
                // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
                long time;
                long startTime = System.currentTimeMillis();
                for (int t = 0; t < 10; t++) {
                    for (int i = 0; i < array.length; i++) array[i] = random.nextInt( bound:10000000);
                    ParSort.sort(array,  from:0,  to:array.length,pool);//array, 0, array.length, pool);
                }
                long endTime = System.currentTimeMillis();
                time = (endTime - startTime);
                timeList.add( e:time);

                System.out.println("cutoff : " + (ParSort.cutoff) + "\t\t10 times Time:" + time + "ms");
            }
            try {
                FileOutputStream fis = new FileOutputStream("./src/result.csv" + arraySize + "_thread_" + thread + '
                OutputStreamWriter isr = new OutputStreamWriter( out:fis);
                BufferedWriter bw = new BufferedWriter( out:isr);
                int j = 0;
                for (long i : timeList) {
                    String content = (double) ParSort.cutoff * (j + 1) / arraySize + "," + (double) i / 10 + "\n";
                    j++;
                    bw.write( str:content);
                    bw.flush();
                }
```

Main.java × ParSort.java ×

Source History

```java
                e.printStackTrace();
            }
            thread *= 2;
        }
    }


    private static void processArgs(String[] args) {
        String[] xs = args;
        while (xs.length > 0)
            if (xs[0].startsWith( prefix:"-")) xs = processArg(xs);
    }

    private static String[] processArg(String[] xs) {
        String[] result = new String[0];
        System.arraycopy( src:xs,  srcPos:2,  dest:result,  destPos:0, xs.length - 2);
        processCommand(xs[0], xs[1]);
        return result;
    }

    private static void processCommand(String x, String y) {
        if (x.equalsIgnoreCase( anotherString:"N")) setConfig(x,  i:Integer.parseInt( s:y));
        else
            // TODO sort this out
            if (x.equalsIgnoreCase( anotherString:"P")) //noinspection ResultOfMethodCallIgnored
                ForkJoinPool.getCommonPoolParallelism();
    }

    private static void setConfig(String x, int i) {
        configuration.put( key:x,  value:i);
    }

    @SuppressWarnings("MismatchedQueryAndUpdateOfCollection")
    private static final Map<String, Integer> configuration = new HashMap<>();


}
```

Main.java × ParSort.java ×

Source History

```java
package edu.neu.coe.info6205.sort.par;

import java.util.Arrays;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ForkJoinPool;

class ParSort {

    public static int cutoff = 10000;

    public static void sort(int[] array, int from, int to, ForkJoinPool pool) {
        if (to - from < cutoff) {
            Arrays.sort( a:array,  fromIndex:from,  toIndex:to);
        } else {
            // FIXME next few lines should be removed from public repo.
            CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2, pool); // TO IMP
            CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to, pool); // TO IMPLE
            CompletableFuture<int[]> parsort = parsort1.thenCombine( other:parsort2, (xs1, xs2) -> {
                int[] result = new int[xs1.length + xs2.length];
                // TO IMPLEMENT
                int i = 0;
                int j = 0;
                for (int k = 0; k < result.length; k++) {
                    if (i >= xs1.length) {
                        result[k] = xs2[j++];
                    } else if (j >= xs2.length) {
                        result[k] = xs1[i++];
                    } else if (xs2[j] < xs1[i]) {
                        result[k] = xs2[j++];
                    } else {
                        result[k] = xs1[i++];
                    }
                }
                return result;
            });

            parsort.whenComplete((result, throwable) -> System.arraycopy( src:result,  srcPos:0,  dest:array,  destI
//              System.out.println("# threads: "+ ForkJoinPool.commonPool().getRunningThreadCount());
```

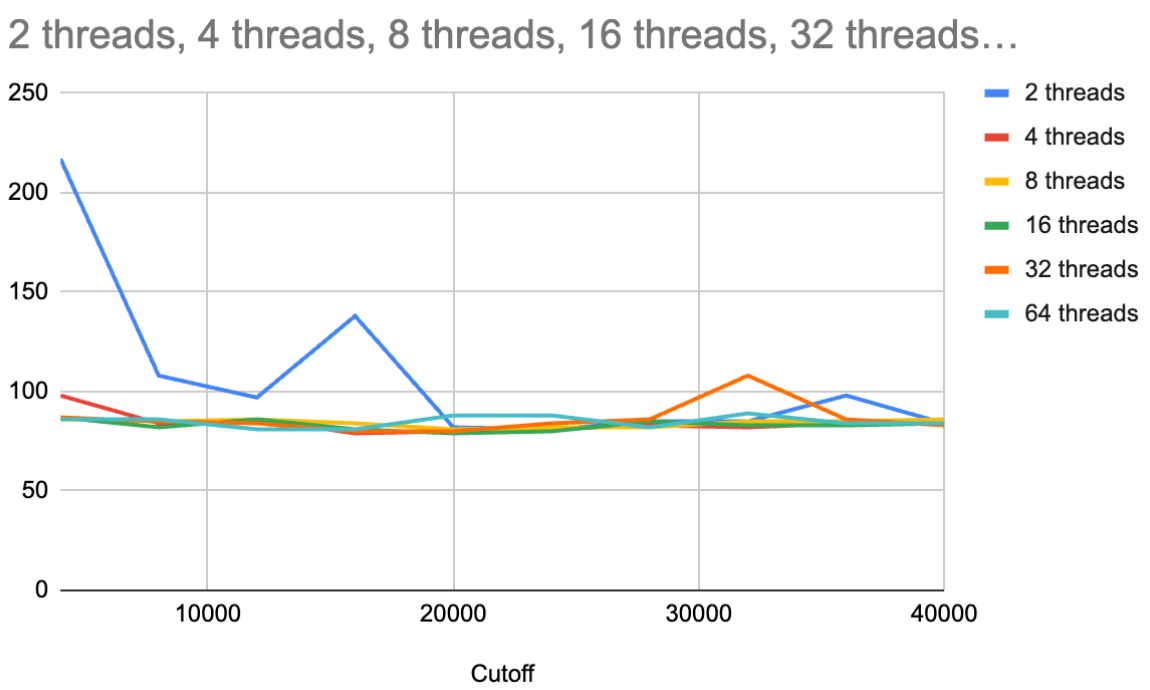
```java
            parsort.whenComplete((result, throwable) -> System.arraycopy( src: result,  srcPos: 0,  dest: array,  destPos: frc
//            System.out.println("# threads: "+ ForkJoinPool.commonPool().getRunningThreadCount());
            parsort.join();
        }
    }

    private static CompletableFuture<int[]> parsort(int[] array, int from, int to, ForkJoinPool pool) {
        return CompletableFuture.supplyAsync(
                () -> {
                    int[] result = new int[to - from];
                    // TO IMPLEMENT
                    System.arraycopy( src: array,  srcPos: from,  dest: result,  destPos: 0,  length: result.length);
                    sort( array: result,  from: 0, to - from,pool);
                    return result;
                }
        );
    }
}
```

REQUIRED OUTPUT:

WITH Array Size = 2,00,000 Cutoff= 4000

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
4000	217	98	86	87	87	86
8000	108	84	85	82	85	86
12000	97	86	86	86	84	81
16000	138	79	84	81	80	81
20000	82	80	81	79	80	88
24000	81	81	82	80	84	88
28000	85	83	82	85	86	82
32000	85	82	85	83	108	89
36000	98	84	85	83	86	84
40000	84	84	86	84	83	84



OUTPUT SCREENSHOT

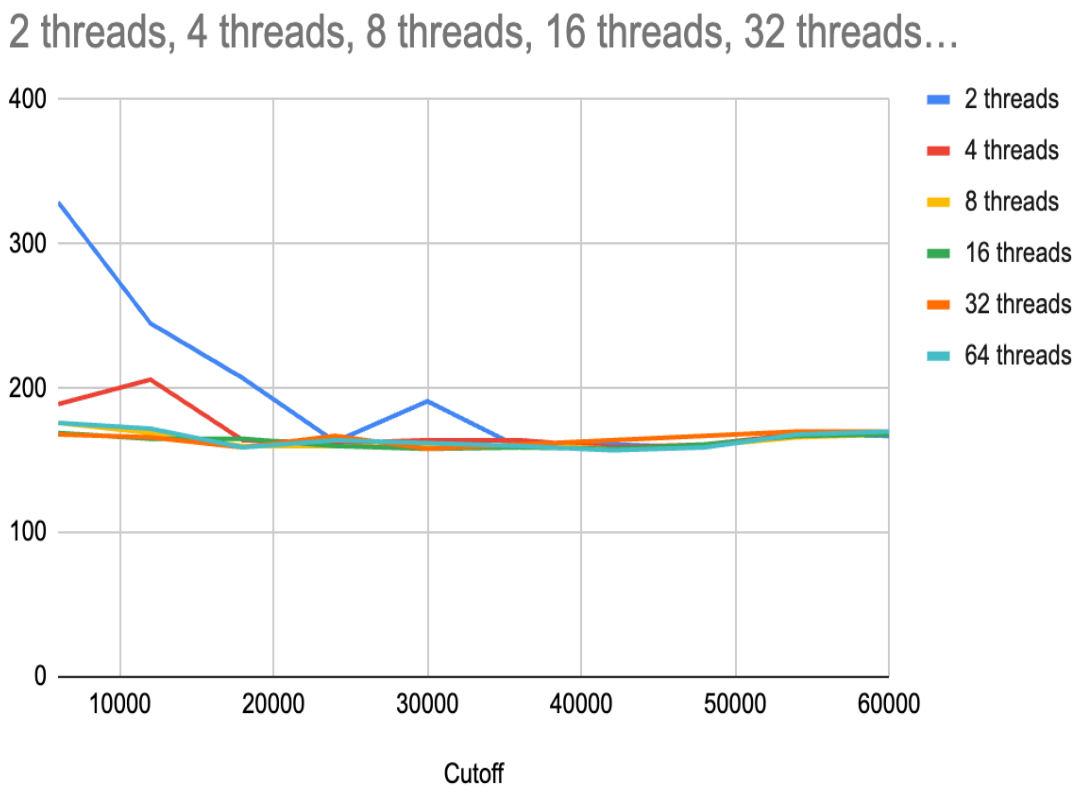
```
--------------------< edu.neu.coe.mgen:INFO6205 >----------------------
Building INFO6205 1
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ INFO6205 ---
Degree of parallelism: 2
cutoff: 4000            10 times Time:210ms
cutoff: 8000            10 times Time:133ms
cutoff: 12000           10 times Time:97ms
cutoff: 16000           10 times Time:82ms
cutoff: 20000           10 times Time:134ms
cutoff: 24000           10 times Time:83ms
cutoff: 28000           10 times Time:85ms
cutoff: 32000           10 times Time:83ms
cutoff: 36000           10 times Time:83ms
cutoff: 40000           10 times Time:81ms
Degree of parallelism: 4
cutoff: 4000            10 times Time:93ms
cutoff: 8000            10 times Time:85ms
cutoff: 12000           10 times Time:82ms
cutoff: 16000           10 times Time:78ms
cutoff: 20000           10 times Time:79ms
cutoff: 24000           10 times Time:84ms
cutoff: 28000           10 times Time:83ms
cutoff: 32000           10 times Time:82ms
cutoff: 36000           10 times Time:82ms
cutoff: 40000           10 times Time:82ms
Degree of parallelism: 8
cutoff: 4000            10 times Time:85ms
cutoff: 8000            10 times Time:81ms
cutoff: 12000           10 times Time:81ms
cutoff: 16000           10 times Time:81ms
cutoff: 20000           10 times Time:79ms
cutoff: 24000           10 times Time:81ms
cutoff: 28000           10 times Time:81ms
cutoff: 32000           10 times Time:82ms
cutoff: 36000           10 times Time:82ms
cutoff: 40000           10 times Time:83ms
Degree of parallelism: 16
```

```
Degree of parallelism: 16
cutoff: 4000            10 times Time:84ms
cutoff: 8000            10 times Time:81ms
cutoff: 12000           10 times Time:82ms
cutoff: 16000           10 times Time:80ms
cutoff: 20000           10 times Time:79ms
cutoff: 24000           10 times Time:78ms
cutoff: 28000           10 times Time:83ms
cutoff: 32000           10 times Time:82ms
cutoff: 36000           10 times Time:82ms
cutoff: 40000           10 times Time:81ms
Degree of parallelism: 32
cutoff: 4000            10 times Time:86ms
cutoff: 8000            10 times Time:81ms
cutoff: 12000           10 times Time:80ms
cutoff: 16000           10 times Time:79ms
cutoff: 20000           10 times Time:80ms
cutoff: 24000           10 times Time:80ms
cutoff: 28000           10 times Time:82ms
cutoff: 32000           10 times Time:81ms
cutoff: 36000           10 times Time:81ms
cutoff: 40000           10 times Time:82ms
Degree of parallelism: 64
cutoff: 4000            10 times Time:87ms
cutoff: 8000            10 times Time:83ms
cutoff: 12000           10 times Time:83ms
cutoff: 16000           10 times Time:78ms
cutoff: 20000           10 times Time:78ms
cutoff: 24000           10 times Time:80ms
cutoff: 28000           10 times Time:81ms
cutoff: 32000           10 times Time:87ms
cutoff: 36000           10 times Time:104ms
cutoff: 40000           10 times Time:187ms
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  5.791 s
Finished at: 2023-02-18T22:47:21-05:00
------------------------------------------------------------------------
```

WITH Array Size = 4,00,000 Cutoff= 6000

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
6000	329	189	176	169	168	176
12000	245	206	169	165	166	172
18000	207	164	160	165	159	159
24000	163	162	160	160	167	164
30000	191	164	159	158	158	162
36000	159	164	161	159	161	160
42000	161	159	158	158	164	157
48000	159	161	160	161	167	159
54000	168	168	166	167	170	168
60000	167	168	168	168	170	170



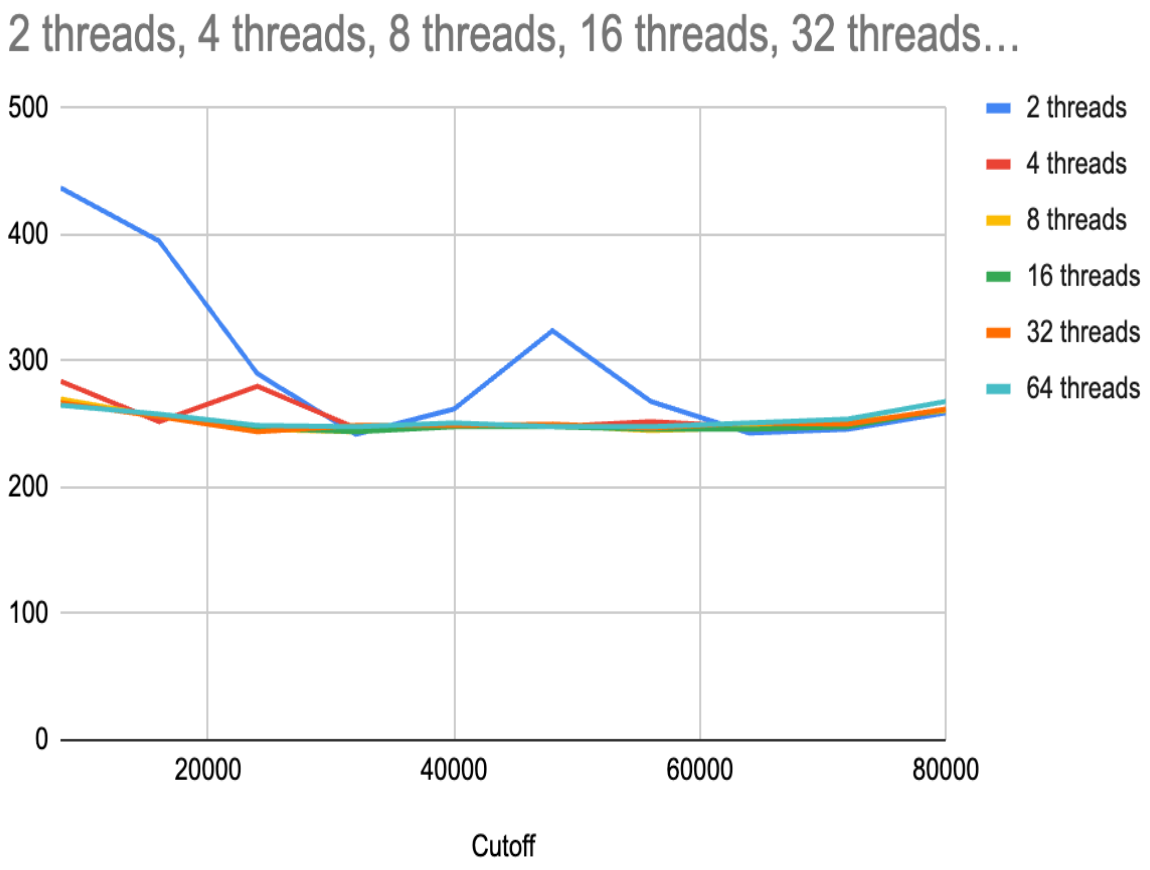
OUTPUT SCREENSHOTS

```
Degree of parallelism: 2
cutoff: 6000            10 times Time:346ms
cutoff: 12000           10 times Time:212ms
cutoff: 18000           10 times Time:169ms
cutoff: 24000           10 times Time:182ms
cutoff: 30000           10 times Time:220ms
cutoff: 36000           10 times Time:163ms
cutoff: 42000           10 times Time:161ms
cutoff: 48000           10 times Time:157ms
cutoff: 54000           10 times Time:167ms
cutoff: 60000           10 times Time:168ms
Degree of parallelism: 4
cutoff: 6000            10 times Time:187ms
cutoff: 12000           10 times Time:165ms
cutoff: 18000           10 times Time:160ms
cutoff: 24000           10 times Time:157ms
cutoff: 30000           10 times Time:158ms
cutoff: 36000           10 times Time:158ms
cutoff: 42000           10 times Time:159ms
cutoff: 48000           10 times Time:159ms
cutoff: 54000           10 times Time:167ms
cutoff: 60000           10 times Time:167ms
Degree of parallelism: 8
cutoff: 6000            10 times Time:171ms
cutoff: 12000           10 times Time:165ms
cutoff: 18000           10 times Time:156ms
cutoff: 24000           10 times Time:159ms
cutoff: 30000           10 times Time:162ms
cutoff: 36000           10 times Time:160ms
cutoff: 42000           10 times Time:158ms
cutoff: 48000           10 times Time:158ms
cutoff: 54000           10 times Time:168ms
cutoff: 60000           10 times Time:167ms
Degree of parallelism: 16
cutoff: 6000            10 times Time:170ms
cutoff: 12000           10 times Time:162ms
cutoff: 18000           10 times Time:157ms
cutoff: 24000           10 times Time:158ms
cutoff: 30000           10 times Time:163ms
```

```
Degree of parallelism: 16
cutoff: 6000            10 times Time:170ms
cutoff: 12000           10 times Time:162ms
cutoff: 18000           10 times Time:157ms
cutoff: 24000           10 times Time:158ms
cutoff: 30000           10 times Time:163ms
cutoff: 36000           10 times Time:158ms
cutoff: 42000           10 times Time:159ms
cutoff: 48000           10 times Time:159ms
cutoff: 54000           10 times Time:168ms
cutoff: 60000           10 times Time:167ms
Degree of parallelism: 32
cutoff: 6000            10 times Time:170ms
cutoff: 12000           10 times Time:169ms
cutoff: 18000           10 times Time:158ms
cutoff: 24000           10 times Time:156ms
cutoff: 30000           10 times Time:158ms
cutoff: 36000           10 times Time:160ms
cutoff: 42000           10 times Time:159ms
cutoff: 48000           10 times Time:158ms
cutoff: 54000           10 times Time:168ms
cutoff: 60000           10 times Time:169ms
Degree of parallelism: 64
cutoff: 6000            10 times Time:173ms
cutoff: 12000           10 times Time:163ms
cutoff: 18000           10 times Time:158ms
cutoff: 24000           10 times Time:156ms
cutoff: 30000           10 times Time:158ms
cutoff: 36000           10 times Time:159ms
cutoff: 42000           10 times Time:159ms
cutoff: 48000           10 times Time:158ms
cutoff: 54000           10 times Time:169ms
cutoff: 60000           10 times Time:167ms
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  10.579 s
```

WITH Array Size = 6,00,000 Cutoff= 8000

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
8000	437	284	270	266	267	265
16000	395	252	256	257	256	258
24000	290	280	246	247	244	249
32000	242	247	244	244	249	248
40000	262	250	248	248	249	251
48000	324	248	249	248	250	248
56000	268	252	245	246	247	248
64000	243	248	247	246	251	251
72000	246	250	251	248	250	254
80000	259	261	261	262	262	268



OUTPUT SNAPSHOTS:

```
Degree of parallelism: 2
cutoff: 8000            10 times Time:508ms
cutoff: 16000           10 times Time:322ms
cutoff: 24000           10 times Time:276ms
cutoff: 32000           10 times Time:317ms
cutoff: 40000           10 times Time:277ms
cutoff: 48000           10 times Time:259ms
cutoff: 56000           10 times Time:244ms
cutoff: 64000           10 times Time:261ms
cutoff: 72000           10 times Time:269ms
cutoff: 80000           10 times Time:281ms
Degree of parallelism: 4
cutoff: 8000            10 times Time:269ms
cutoff: 16000           10 times Time:258ms
cutoff: 24000           10 times Time:261ms
cutoff: 32000           10 times Time:253ms
cutoff: 40000           10 times Time:250ms
cutoff: 48000           10 times Time:242ms
cutoff: 56000           10 times Time:245ms
cutoff: 64000           10 times Time:248ms
cutoff: 72000           10 times Time:241ms
cutoff: 80000           10 times Time:260ms
Degree of parallelism: 8
cutoff: 8000            10 times Time:281ms
cutoff: 16000           10 times Time:264ms
cutoff: 24000           10 times Time:250ms
cutoff: 32000           10 times Time:242ms
cutoff: 40000           10 times Time:246ms
cutoff: 48000           10 times Time:246ms
cutoff: 56000           10 times Time:240ms
cutoff: 64000           10 times Time:246ms
cutoff: 72000           10 times Time:243ms
cutoff: 80000           10 times Time:263ms
Degree of parallelism: 16
cutoff: 8000            10 times Time:268ms
cutoff: 16000           10 times Time:255ms
cutoff: 24000           10 times Time:250ms
cutoff: 32000           10 times Time:255ms
cutoff: 40000           10 times Time:250ms
```

```
Degree of parallelism: 16
cutoff: 8000            10 times Time:268ms
cutoff: 16000           10 times Time:255ms
cutoff: 24000           10 times Time:250ms
cutoff: 32000           10 times Time:255ms
cutoff: 40000           10 times Time:250ms
cutoff: 48000           10 times Time:247ms
cutoff: 56000           10 times Time:251ms
cutoff: 64000           10 times Time:247ms
cutoff: 72000           10 times Time:243ms
cutoff: 80000           10 times Time:263ms
Degree of parallelism: 32
cutoff: 8000            10 times Time:266ms
cutoff: 16000           10 times Time:254ms
cutoff: 24000           10 times Time:247ms
cutoff: 32000           10 times Time:250ms
cutoff: 40000           10 times Time:252ms
cutoff: 48000           10 times Time:249ms
cutoff: 56000           10 times Time:246ms
cutoff: 64000           10 times Time:246ms
cutoff: 72000           10 times Time:246ms
cutoff: 80000           10 times Time:260ms
Degree of parallelism: 64
cutoff: 8000            10 times Time:283ms
cutoff: 16000           10 times Time:264ms
cutoff: 24000           10 times Time:244ms
cutoff: 32000           10 times Time:243ms
cutoff: 40000           10 times Time:245ms
cutoff: 48000           10 times Time:249ms
cutoff: 56000           10 times Time:248ms
cutoff: 64000           10 times Time:249ms
cutoff: 72000           10 times Time:244ms
cutoff: 80000           10 times Time:263ms
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

CONCLUSION:

When the cutoff value is small then the input dataset is divided into many small subproblems, which increases the amount of parallelism but also increases the communication overhead required to merge the results. This can result in reduced performance, as the time required to merge the results may outweigh the benefits of parallelism.

On the other hand,when the cutoff value is large, the input dataset is divided into fewer, larger subproblems, which reduces the communication overhead but also reduces the amount of parallelism. This can also result in reduced performance, as the processing units may be underutilised if some subproblems take longer to process than others.