Program Structures and Algorithms

Name: Chakradhar Grandhi Durga. **NEU ID**: 002933727

ASSIGNMENT-5

5.1 PARALLEL SORT CODE WITH THREAD IMPLEMENTATION

```
class ParSort {

    public static int cutoff = 1000;
    public static ForkJoinPool mypool;
    public static void sort(int[] array, int from, int to) {
        if (to - from < cutoff) Arrays.sort(array, from, to);
        else {
            // FIXME next few lines should be removed from public
repo.
            CompletableFuture<int[]> parsort1 = parsort(array, from,
from + (to - from) / 2); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array, from
+ (to - from) / 2, to); // TO IMPLEMENT
            CompletableFuture<int[]> parsort =
parsort1.thenCombine(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(result, 0, array, from, result.length));
//            System.out.println("# threads: "+
ForkJoinPool.commonPool().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);
                    sort(result, 0, to - from);
                    return result;
                },mypool
        );
    }
}
```

In the above code, I have used the my pool which is a custom declared ForkJoinPool where I am setting a custom number of threads to perform the parallel sort.

5.2 MAIN CODE

```
public class Main {

    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " +
ForkJoinPool.getCommonPoolParallelism());
        Random random = new Random();
        for(int arr_range = 2000000; arr_range<5000000;
arr_range+=1000000){
            for(int threads = 8; threads <1024; threads*=2 ){
                int[] array = new int[arr_range];
                ArrayList<Long> timeList = new ArrayList<>();
                ParSort.mypool = new ForkJoinPool(threads);
                System.out.println("The number of threads is: " +
threads);
                for (int j = 50; j < 100; j++) {
                    ParSort.cutoff = 10000 * (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(10000000);
                        ParSort.sort(array, 0, array.length);
                    }
                    long endTime = System.currentTimeMillis();
                    time = (endTime - startTime);
                    timeList.add(time);


                    System.out.println("cutoff:" + (ParSort.cutoff)
```

```
+ "\t\t10times Time:" + time + "ms");

                }
                try {
                    FileOutputStream fis = new
FileOutputStream("./src/result.csv");
                    OutputStreamWriter isr = new
OutputStreamWriter(fis);
                    BufferedWriter bw = new BufferedWriter(isr);
                    int j = 0;
                    for (long i : timeList) {
                        String content = (double) 10000 * (j + 1) /
2000000 + "," + (double) i / 10 + "\n";
                        j++;
                        bw.write(content);
                        bw.flush();
                    }
                    bw.close();

                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

    }

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

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

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

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

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

```
}
```

I have modified the main program by adding some additions to the code.

- Setting an iterator(**arr_range**) to increase my array size for me to test and compare the results.
- Used the mypool (ForkJoinPool that I created in Parsort.java) with another iterator(**threads**) to compare the statistics created by different thread count to get the values.

5.3 OBSERVATIONS AND CONCLUSIONS

The observations of the data is too large as the cutoff has multiple values and I have conducted experiments for different number of threads (**ranging from 8 -> 512**). So, I am attaching the entire data of the excel file below.

Statistical Data.xlsx

From the above staticstical data my conclusion states that

- After a particular range of threads, the performance does not show a significant increase which creates a wastage of system resources. In my experiments I have seen that in an average case, performance increases when the number of threads increase, and the peak performance average is observed when the thread count is **256.**
- When it comes to the cutoff range, after a particular limit of cutoff I see that the performance time on most of the cases often increase. This gives me an indication that after a limit on the cutoff, it is not feasible to use the parallel sort and it would be much better if the generic system sort is used.