Program Structures and Algorithms Spring 2023(SEC –1)
Assignment-5

Name: Naga Venkata Nishanth Sayana

NU ID: 002930970

Task:

Our Task here is perform parallel sort and switch between parallel sort and system sort based on the cut-off values. We need to make a note of times taken while providing different number of threads for performing parallel sort.

Code Screen Shots:

ParaSort:

```
3 usages  xiaohuanlin *
class ParSort {

    public static int cutoff = 1000;

    xiaohuanlin *
    public static void sort(int[] array, int from, int to,ForkJoinPool forkJoinPool) {
        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,  to: from + (to - from) / 2,forkJoinPool); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array,  from: from + (to - from) / 2, to,forkJoinPool); // 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,  srcPos: 0, array, from, result.length));
//            System.out.println("# threads: "+ ForkJoinPool.commonPool().getRunningThreadCount());
            parsort.join();
        }
```

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

Main Method:

```
public class Main {

    xiaohuanlin *
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());

        int[] arrLengths = {524288,1048576,2097152,4194304};
        ForkJoinPool forkPool;

        int[] threadNum = {2, 4, 8, 16,32};
        Random random = new Random();
        int[] array;
        ArrayList<Long> timeList = new ArrayList<>();

        for(int l=0;l<arrLengths.length;l++) {
            int length=arrLengths[l];
            array = new int[length];

            int[] cutoffLen = {length / 1024 + 1, length / 512 + 1, length / 256 + 1, length / 128 + 1,
                    length / 64 + 1, length / 32 + 1, length / 16 + 1, length / 8 + 1, length / 4 + 1,
                    length / 2 + 1, length + 1};

            System.out.println("Length of array is " + arrLengths[l]);
            System.out.println();
```

```
            for (int n = 0; n < threadNum.length; n++) {
                for (int c = 0; c < cutoffLen.length; c++) {
                    ParSort.cutoff = cutoffLen[c];

                    forkPool = new ForkJoinPool(threadNum[n]);
                    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, array.length, forkPool);
                    }
                    long endTime = System.currentTimeMillis();
                    time = (endTime - startTime);
                    timeList.add(time);


                    System.out.println("cutoff: " + (ParSort.cutoff) + " Thread Count: " + threadNum[n] + "\t\t10times Time:" + time + "ms");}
```

Output:

```
C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
Degree of parallelism: 19
Length of array is 524288

cutoff: 513 Thread Count: 2      10times Time:685ms
cutoff: 1025 Thread Count: 2         10times Time:354ms
cutoff: 2049 Thread Count: 2         10times Time:317ms
cutoff: 4097 Thread Count: 2         10times Time:290ms
cutoff: 8193 Thread Count: 2         10times Time:350ms
cutoff: 16385 Thread Count: 2        10times Time:315ms
cutoff: 32769 Thread Count: 2        10times Time:301ms
cutoff: 65537 Thread Count: 2        10times Time:352ms
cutoff: 131073 Thread Count: 2       10times Time:394ms
cutoff: 262145 Thread Count: 2       10times Time:275ms
cutoff: 524289 Thread Count: 2       10times Time:419ms
cutoff: 513 Thread Count: 4      10times Time:516ms
cutoff: 1025 Thread Count: 4         10times Time:297ms
cutoff: 2049 Thread Count: 4         10times Time:256ms
cutoff: 4097 Thread Count: 4         10times Time:253ms
cutoff: 8193 Thread Count: 4         10times Time:269ms
cutoff: 16385 Thread Count: 4        10times Time:281ms
cutoff: 32769 Thread Count: 4        10times Time:313ms
cutoff: 65537 Thread Count: 4        10times Time:350ms
cutoff: 131073 Thread Count: 4       10times Time:312ms
cutoff: 262145 Thread Count: 4       10times Time:276ms
cutoff: 524289 Thread Count: 4       10times Time:419ms
cutoff: 513 Thread Count: 8      10times Time:511ms
cutoff: 1025 Thread Count: 8         10times Time:304ms
cutoff: 2049 Thread Count: 8         10times Time:277ms
cutoff: 4097 Thread Count: 8         10times Time:235ms
cutoff: 8193 Thread Count: 8         10times Time:255ms
cutoff: 16385 Thread Count: 8        10times Time:290ms
cutoff: 32769 Thread Count: 8        10times Time:292ms
```

```
cutoff: 513 Thread Count: 16         10times Time:504ms
cutoff: 1025 Thread Count: 16        10times Time:286ms
cutoff: 2049 Thread Count: 16        10times Time:273ms
cutoff: 4097 Thread Count: 16        10times Time:257ms
cutoff: 8193 Thread Count: 16        10times Time:245ms
cutoff: 16385 Thread Count: 16       10times Time:249ms
cutoff: 32769 Thread Count: 16       10times Time:216ms
cutoff: 65537 Thread Count: 16       10times Time:211ms
cutoff: 131073 Thread Count: 16      10times Time:219ms
cutoff: 262145 Thread Count: 16      10times Time:280ms
cutoff: 524289 Thread Count: 16      10times Time:432ms
cutoff: 513 Thread Count: 32         10times Time:525ms
cutoff: 1025 Thread Count: 32        10times Time:304ms
cutoff: 2049 Thread Count: 32        10times Time:262ms
cutoff: 4097 Thread Count: 32        10times Time:257ms
cutoff: 8193 Thread Count: 32        10times Time:240ms
cutoff: 16385 Thread Count: 32       10times Time:218ms
cutoff: 32769 Thread Count: 32       10times Time:213ms
cutoff: 65537 Thread Count: 32       10times Time:215ms
cutoff: 131073 Thread Count: 32      10times Time:214ms
cutoff: 262145 Thread Count: 32      10times Time:274ms
cutoff: 524289 Thread Count: 32      10times Time:435ms
```

Observations:

I have taken arrays of different lengths ranging from 524288 to 4194304(Powers of 2), and I have given number of threads as input ranging from 2 to 32(Powers of 2) and varied the cut off values based on the lengths of each array. I picked the cut off value as Cut- off=(length/2^L) +1, as L would be the number of levels of the recursion tree, this way, I could control until which level of the recursion tree, the parallel sort can be performed.

Here are the values that I have tabulated:

a) For Array of length=524288

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
4097	4796ms	2038ms	2163ms	1906ms	1771ms
8193	4578ms	1896ms	1849ms	1897ms	1808ms
16385	4488ms	1883ms	1773ms	1691ms	1681ms
32769	4635ms	1914ms	1931ms	1759ms	1739ms
65537	2198ms	2111ms	1982ms	2218ms	1918ms
131073	2263ms	2239ms	2332ms	2220ms	1655ms
262145	2780ms	2476ms	2492ms	1944ms	1539ms
524289	3001ms	3265ms	2385ms	1574ms	1769ms
1048577	3429ms	2685ms	1885ms	1818ms	1913ms
2097153	2559ms	2488ms	2507ms	2475ms	2502ms
4194305	3950ms	3970ms	3904ms	3957ms	3943ms

b) For Array of length= 1048576

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
2049	2611ms	2514ms	2416ms	2330ms	2310ms
4097	2355ms	2118ms	2057ms	2059ms	2052ms
8193	2271ms	2062ms	1895ms	2059ms	1915ms
16385	2233ms	2008ms	2077ms	1994ms	1966ms
32769	2152ms	2031ms	2083ms	2200ms	1903ms
65537	2344ms	2430ms	2486ms	2237ms	1744ms
131073	2664ms	2913ms	2804ms	1938ms	1600ms
262145	2991ms	3232ms	2429ms	1701ms	1683ms
524289	3718ms	2737ms	1793ms	1806ms	1824ms
1048577	2621ms	2634ms	2687ms	2619ms	2632ms
2097153	4450ms	4379ms	4486ms	4334ms	4358ms

c) For Array of length= 2097152

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
1025	747ms	624ms	662ms	1322ms	1283ms
2049	571ms	526ms	532ms	1161ms	1067ms
4097	595ms	500ms	521ms	1071ms	1051ms
8193	572ms	491ms	528ms	973ms	1089ms
16385	574ms	505ms	1098ms	1015ms	956ms
32769	588ms	588ms	1141ms	1082ms	795ms
65537	628ms	694ms	1133ms	1061ms	835ms
131073	767ms	574ms	1209ms	840ms	757ms
262145	848ms	664ms	869ms	881ms	888ms
524289	585ms	567ms	1258ms	1301ms	1267ms
1048577	905ms	903ms	2046ms	1997ms	2047ms

d) For Array of length= 4194304

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
513	685ms	516ms	511ms	504ms	525ms
1025	354ms	297ms	304ms	286ms	304ms
2049	317ms	256ms	277ms	273ms	262ms
4097	290ms	253ms	235ms	257ms	257ms
8193	350ms	269ms	255ms	245ms	240ms
16385	315ms	281ms	290ms	249ms	218ms
32769	301ms	313ms	292ms	216ms	213ms
65537	352ms	350ms	262ms	211ms	215ms
131073	394ms	312ms	223ms	219ms	214ms
262145	275ms	276ms	279ms	280ms	274ms
524289	419ms	419ms	433ms	432ms	435ms

Based on the above observations, I concluded that,

The performance gains from increasing the number of threads are not consistent across different cutoff values. In some cases, increasing the number of threads leads to a significant reduction in execution time, while in other cases the reduction is relatively small or even negligible.

In some cases, there appears to be significant variability in execution times across multiple runs, as indicated by the standard deviations of the measurements. This could be due to factors such as system load or other processes running on the computer.

Overall, the data suggests that increasing the number of threads can lead to improved performance for lower cutoff values, but there is a diminishing return to using more threads as the cutoff value increases. Therefore, choosing the optimal number of threads for a given task requires careful consideration of factors such as the size of the problem and the available computing resources.