

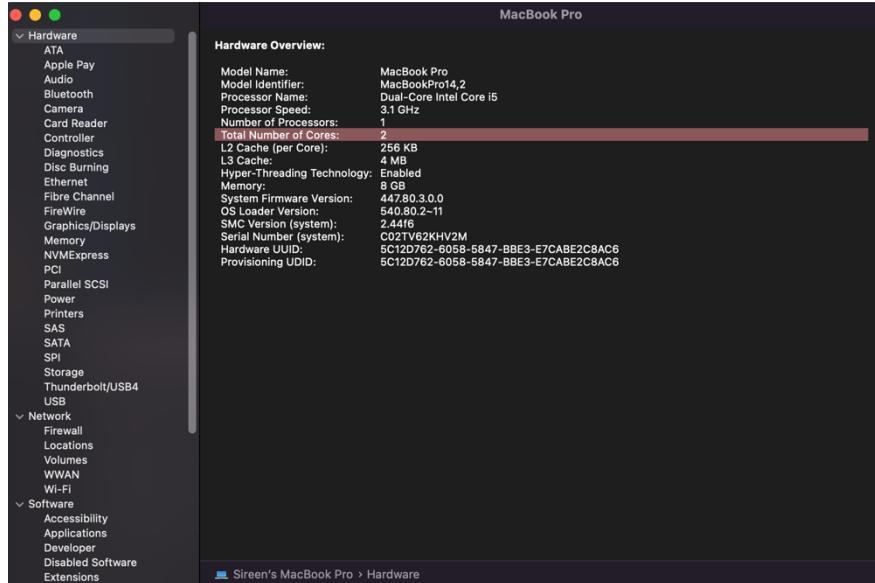
EE463 HW4

Operating Systems

Sereen Omar Bahdad
ID-1806392
Section A

In this HW a multithreaded program was developed to sort an 1D array using the theory of merge sorting. Throughout this report the HW questions were answered, and the results obtained in each case was recoded and analyzed.

1. Available device hardware information



Dual processors and each have two cores this means total of 4 cores, and it was calculated using the java API functions

Case 1 when N=24, K=5

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like ParallelMerge.java, Lab1, Lab2, Lab3, Lab4, Lab5, Lab6, Lab7, and OSMultithreaded.
- Code Editor:** Displays the code for ParallelMerge.java, which implements a parallel merge sort algorithm.
- Console Output:** Shows the execution log and the sorted array output.

```

import java.util.*;
public class ParallelMerge extends Thread {
    //N list size
    //K number of threads
    static int K, N;
    //global variable Array
    static int[] A;
    //temporary 2d matrix to save the subarrays after sorting
    static int[][] output;
    static int[][] indec;
    //start and end index for the sub sets
    int sr;
    int end;
    ParallelMerge(int[] A) {
        this.A = A;
    }
}

Original Array
172 67 109 148 185 182 125 86 3 61 176 168 190 153 207 167 145 185 90 215 67 29
size:22 Threads:5
Thread ID: 13
From index:0 To Index: 4
Thread ID: 14
From index:5 To Index: 9
Thread ID: 15
From index:10 To Index: 13
Thread ID: 16
From index:14 To Index: 17
Thread ID: 17
From index:18 To Index: 21
Now megering from A subarray from indx 0 To 4
Now megering from A subarray from indx 5 To 9
Now megering from A subarray from indx 10 To 13
Now megering from A subarray from indx 14 To 17
Now megering from A subarray from indx 18 To 21
New Array
3 29 61 67 67 86 90 109 125 145 148 153 167 168 172 176 182 185 185 190 207 215

```

To Record the time taken please comment all the output messages in the code

Case 2 when N=1000,000 K=1

The screenshot shows the Eclipse IDE interface with the file `ParallelMerge.java` open. The code implements a parallel merge sort algorithm. A comment in the code specifies `N = 1000000` and `K = 1`. The `main` method calculates the number of cores available using `Runtime.getRuntime().availableProcessors()`. The console output shows the execution took approximately 675ms.

```
34     Arrays.sort(this.A, this.sr, this.end);
35 }
36 }
37 // To create the array of random int of size N
38 public static int[] start(int N) {
39     Random rand = new Random(System.currentTimeMillis());
40     int[] s = new int[N];
41     for (int i = 0; i < N; i++) {
42         s[i] = rand.nextInt(10*N);
43         //System.out.print(s[i] + " ");
44     }
45     return s;
46 }
47 }
48
49 public static void main(String[] args) throws InterruptedException {
50     // TODO Auto-generated method stub
51     // Take line argument as input for K , N values
52     // N=Integer.parseInt(args[0]);
53     // K= Integer.parseInt(args[1]);
54     N = 1000000;
55     K = 1;
56     //To calculate the number of cores available
57     int cores = Runtime.getRuntime().availableProcessors();
58     //System.out.println("The Available Cores are: " + cores);
59     //declaring the temporary matrix size
60     output = new int[K][];
61     //indicates whiter the sub array will all have equal sizes
62     int fch1 = N % K;// 1
63 }
```

Console output:

```
<terminated> ParallelMerge [Java Application] /Library/Java/JavaVirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java (Mar 21, 2022, 1:10:16 PM – 1:10:17 PM)
Took----->675ms
```

Case 2 when N=1000,000 K=2

The screenshot shows the Eclipse IDE interface with the file `ParallelMerge.java` open. The code is identical to the previous version but with `K = 2`. The `main` method calculates the number of cores available using `Runtime.getRuntime().availableProcessors()`. The console output shows the execution took approximately 490ms.

```
34     Arrays.sort(this.A, this.sr, this.end);
35 }
36 }
37 // To create the array of random int of size N
38 public static int[] start(int N) {
39     Random rand = new Random(System.currentTimeMillis());
40     int[] s = new int[N];
41     for (int i = 0; i < N; i++) {
42         s[i] = rand.nextInt(10*N);
43         //System.out.print(s[i] + " ");
44     }
45     return s;
46 }
47 }
48
49 public static void main(String[] args) throws InterruptedException {
50     // TODO Auto-generated method stub
51     // Take line argument as input for K , N values
52     // N=Integer.parseInt(args[0]);
53     // K= Integer.parseInt(args[1]);
54     N = 1000000;
55     K = 2;
56     //To calculate the number of cores available
57     int cores = Runtime.getRuntime().availableProcessors();
58     //System.out.println("The Available Cores are: " + cores);
59     //declaring the temporary matrix size
60     output = new int[K][];
61     //indicates whiter the sub array will all have equal sizes
62     int fch1 = N % K;// 1
63 }
```

Console output:

```
<terminated> ParallelMerge [Java Application] /Library/Java/JavaVirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java (Mar 21, 2022, 1:10:01 PM – 1:10:04 PM)
Took----->490ms
```

Case 2 when N=1000,000 K=3

The screenshot shows the Eclipse IDE interface on a Mac OS X system. The title bar reads "eclipse-workspace - OSMultithreaded/src/ParallelMerge.java - Eclipse IDE". The left sidebar displays a file tree with various projects and files, including "Arch7", "BinaryTree", "Ch3_Lab_practice", "EE367Project", "erwc", "HammingcodeE", "Lab2A", "Lab3", "Lab4", "Lab5", "LabIntoGAG", "OSHW4", "OShw4part1", "OShw4part2", "OSMultithreaded", "JRE System", and "src" containing "ParallelMerge.java". The main editor window shows the Java code for "ParallelMerge.java". The code defines a class "ParallelMerge" with methods for sorting subarrays and running the parallel merge sort algorithm. The "Console" tab at the bottom shows the output of the application's execution, indicating it took 450ms.

```
static int[] A;
//temporary 2d matrix to save the subarrays after sorting
static int[][] output;
//start and end index for the sub sets
int sr;
int end;

ParallelMerge(int[] A) {
    this.A = A;
}

ParallelMerge() {
    this.A = A;
}

ParallelMerge(int sr, int end) {
    this.sr = sr;
    this.end = end;
}

public void run() {

    Arrays.sort(this.A, this.sr, this.end);
}

// To create the array of random int of size N
public static int[] start(int N) {
}
```

<terminated> ParallelMerge [Java Application] /Library/Java/JavaVirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java (Mar 21, 2022, 1:28:58 PM – 1:28:58 PM)
Took-->450ms

Case 2 when N=1000,000 K=4

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Word, File, Edit, View, Insert, Format, Tools, Table, Window, Help.
- Title Bar:** eclipse-workspace - OSMultithreaded/src/ParallelMerge.java - Eclipse IDE
- Left Sidebar:** Shows a file tree with various projects and files, including "Arch7", "Binarytree", "Ch3_lab_practi", "EE367Project", "erwc", "HammingcodeE", "Lab2A", "Lab3", "Lab4", "Lab5", "LabintoGAG", "OSHW4", "OSHW4part1", "OSMultithreaded", "JRE System", "src", and "ParallelMerge".
- Central Area:** A code editor window titled "ParallelMerge.java" containing Java code for parallel merging. The code includes comments explaining the creation of a random array, the start of a timer, and the implementation of a parallel merge algorithm using a for-each loop and nested loops to handle subarray sizes.
- Bottom Area:** A "Console" window showing the output of the application's execution. It displays the command run, the Java environment path, and the execution time: "Took-->---->458ms".

Case 2 when N=1000,000 K=8

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a project named "OSMultithreded" containing a source folder "src" with a file "ParallelMerge.java".
- Code Editor:** Displays the "ParallelMerge.java" code. The code implements a parallel merge sort algorithm. It starts by declaring temporary matrices and indices, then prints the original array. It then creates a random array and initializes thread objects. The main loop handles cases where the sub arrays do not have the same size, calculating subarray sizes and counts. The code uses `Math.ceil` to determine the number of sub arrays of size `sbsize - 1`.
- Console:** Shows the output of the run command: "Took----->418ms".

Case 2 when N=1000,000 K=12

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a project named "OSMultithreded" containing a source folder "src" with a file "ParallelMerge.java".
- Code Editor:** Displays the "ParallelMerge.java" code. This version of the code is more concise, using `Arrays.sort` to sort the array within the `run` method.
- Console:** Shows the output of the run command: "Took----->408ms".

Case 2 when N=1000,000 K=12

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows various Java projects like Arch7, BinaryTree, Ch3_lab_practic, EE367Project, errwc, HammingcodeE, lab2A, Lab3, Lab4, lab5, LabintoGAG, OSHW4, OShw4part1, and OSMultithreded.
- Code Editor:** The active file is ParallelMerge.java. The code implements a parallel merge sort algorithm for a 2D matrix A. It includes methods for initializing the matrix, setting start and end indices, and performing the sort. The code uses the Arrays.sort() method.
- Console:** Displays the output of the application execution. It shows the time taken for execution: "Took----->408ms".
- Status Bar:** Shows the date and time as Mon 21 Mar 13:35.

Case 2 when N=1000,000 K=16

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the same set of Java projects as the previous screenshot.
- Code Editor:** The active file is ParallelMerge.java. This version of the code includes additional logic for calculating subarray sizes and thread counts. It uses System.out.println() statements to log information about cores, array size, and thread creation.
- Console:** Displays the output of the application execution. It shows the time taken for execution: "Took----->339ms".
- Status Bar:** Shows the date and time as Mon 21 Mar 16:25.

Case 2 when N=1000,000 K=24

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure with a file named "ParallelMerge.java" selected.
- Code Editor (center):** Displays the Java code for "ParallelMerge.java". The code implements a parallel merge sort algorithm using threads. It includes comments explaining the creation of a random array, starting threads, and calculating subarray sizes.
- Console (bottom):** Shows the output of the application's execution. It indicates the application is a Java application running on Java Virtual Machine (JDK 15.0.2). The execution took 282ms.

```
//create the random array
A = start(N);
//System.out.println("");
//System.out.println("size:" + N + " Threads:" + K);
//From here start counting the time
Long inputstat = System.nanoTime();
// Thread object declaration
ParallelMerge r1 = new ParallelMerge();

if (fsbl != 0) {
    // if the sub arrays does not have the same size
    //sblsize = the size of array with big partitions
    int sblsize = (int) Math.ceil((float) N / (float) K);
    //resthbl the number of sub arrays with size of sblsize -1
    int resthbl = K - fsbl; // 2

    int resblsize = sblsize - 1; // 3
    int en = 0;
    int s = 0;
    for (int i = 0; i < fsbl; i++) {
        for (int j = 0; j < sblsize; j++) {
            en++;
        }
        //creating thread for every sub array
        indec[i][0] = s;
        indec[i][1] = en-1;
        r1 = new ParallelMerge(s, en);
    }
}
```

Case 2 when N=1000,000 K=32

The screenshot shows the Eclipse IDE interface with the following details:

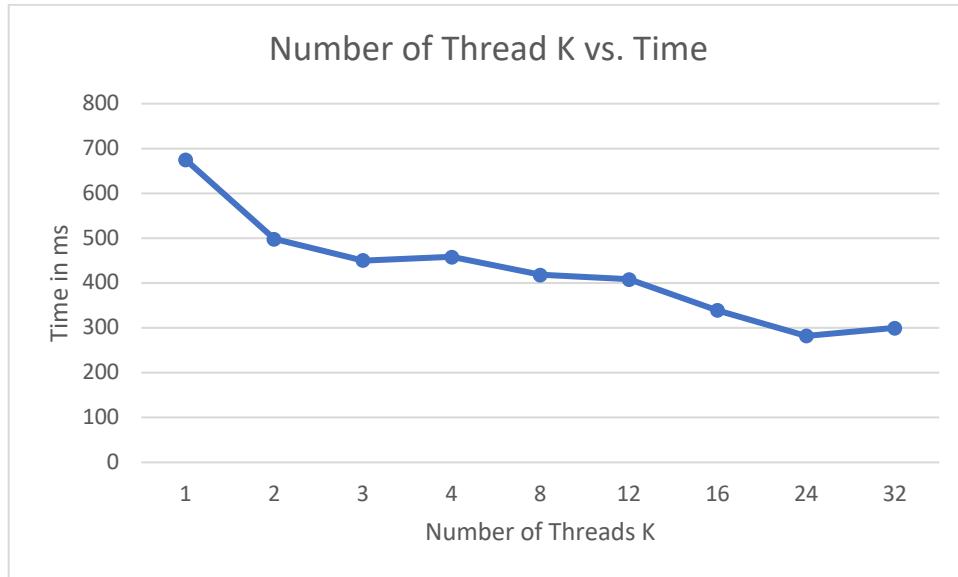
- Toolbar:** Eclipse, File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Top Status Bar:** eclipse-workspace - OSMultithreaded/src/ParallelMerge.java - Eclipse IDE, U.S., Mon 21 Mar 16:27
- Left Sidebar (Project Explorer):** Shows various Java projects and files, including Arch7, BinaryTree, Ch3_Lab_practic, EE367Project, erwc, HammingcodeE, lab2A, Lab3, Lab4, Lab5, LabintoGAG, OSHW4, OSHW4part1, OSMultithreaded, OSRE System, ParallelMerge, Prac_q3, practice_Lab4, Practice2, and tuwb.
- Central Editor Area:** Displays the Java code for `ParallelMerge.java`. The code implements a parallel merge algorithm for an array of size N across K threads. It handles cases where N is not divisible by K by creating additional sub-arrays of size 1. The code uses `System.nanoTime()` to measure execution time.
- Bottom Status Bar:** Writable, Smart Insert, 57 : 15 : 1201.

```
//System.out.println("The Available Cores are: " + cores);
//declaring the temporary matrix size
output = new int[K][];
indec= new int[K][2];
//locates wheter the sub array will all have equal sizes
int fsbl = N % K;// 1
//System.out.println("Original Array");
//create the random array
A = start(N);
//System.out.println("");
//System.out.println("size:" + N + " Threads:" + K);
//From here start counting the time
Long inputstat = System.nanoTime();
// Thread object declaration
ParallelMerge r1 = new ParallelMerge();
if (fsbl != 0) {
    // if the sub arrays does not have the same size
    //shlsize the size of array with big partitions
    int shlsize = (int) Math.ceil((float) N / (float) K);
    // resthbl the number of sub arrays with size of shlsize -1
    int resthbl = K - fsbl;// 2
    int resblsize = shlsize - 1;// 3
    int en = 0;
    int s = 0;
    for (int i = 0; i < fsbl; i++) s
```

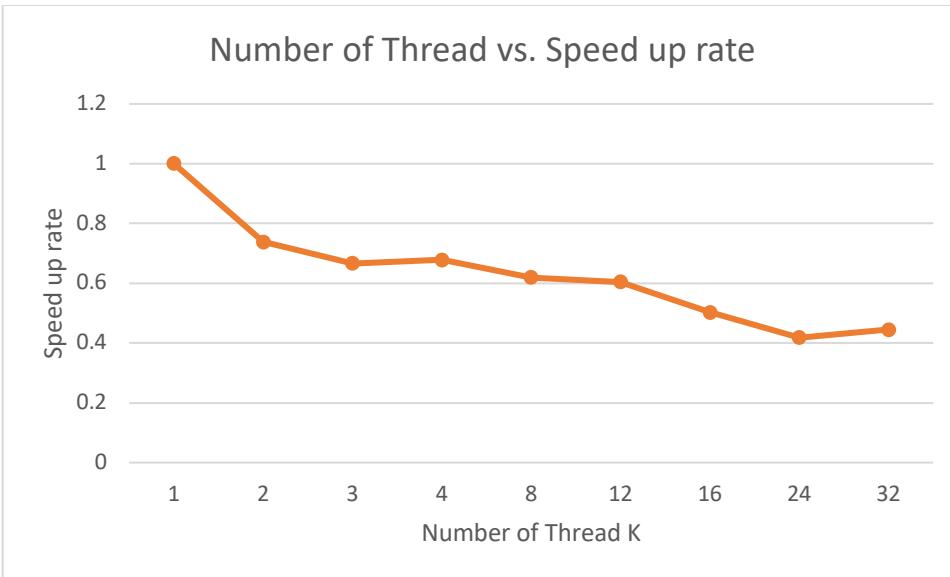
<terminated> ParallelMerge [Java Application] /Library/Java/JavaVirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java (Mar 21, 2022, 4:27:32 PM - 4:27:33 PM)
Took-->300ms

Table 1- Where $N = 1,000,000$ K Varies

T	675	498	450	458	418	408	339	282	300
N	1,000,000								
K	1	2	3	4	8	12	16	24	32
Speed Up Rate	1	0.737	0.667	0.678	0.6192	0.6044	0.5022	0.4178	0.444



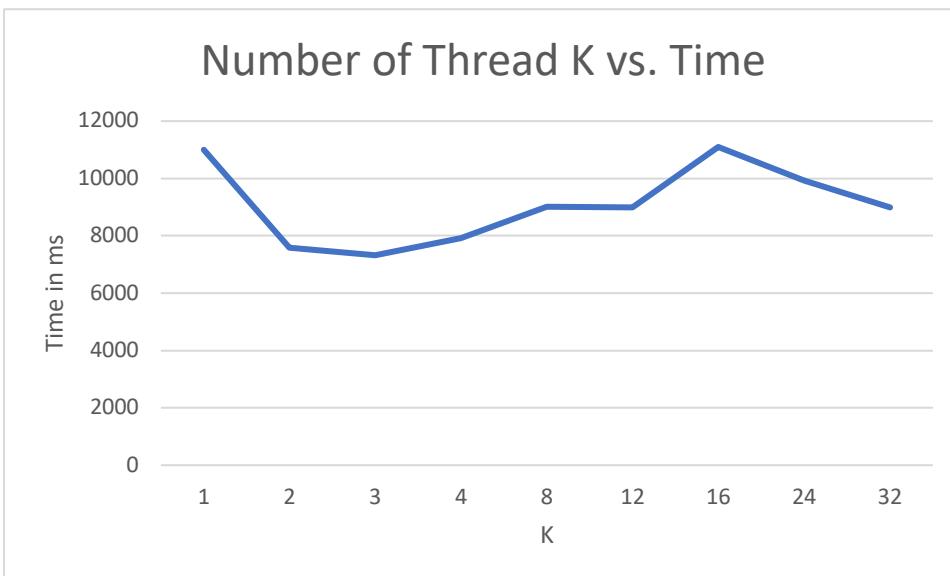
It is presented from the graph of K vs. T that the total time started to decrease with the increasing of the number of threads until K=4, and this highly illustrated the theory of CPU time sharing and parallelism of process and threads. However, when K>4 which represents the available cores for my device (dual processor, 2 cores) the program total time will increase with the increasing of the number of threads as in case K=4 the total number of threads will be 5 along with the main thread that will not be occupy with this device performance but still the results will be better than the serial single process when k=1.



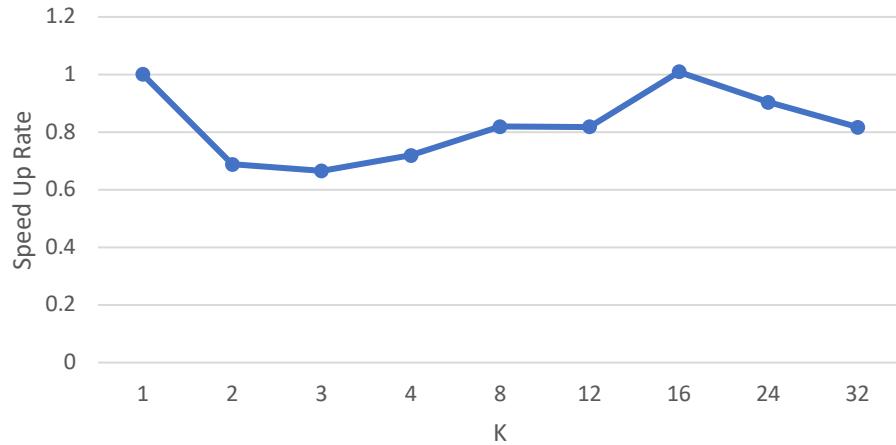
Speed up rate was calculated by dividing the time recorded for each K by the time recorded for K=1 and the result obtained were plotted as well. From the plot its clear that the time taken by the K>1 was less than the serial thread when K=1 and it also indicates the time will decrease with the increment of the thread however the hardware of the device used to run this algorithm needs to be taken into consideration. As in this case the hardware used was had 4 physical cores so from K=4 the response of changing the K and time will be changing.

Table 2- Extra Case Testing When N=100,000,000

T	11003	7578	7323	7911	9011	9000	11100	9939	8993
N	100,000,000								
K	1	2	3	4	8	12	16	24	32
Speed up rate	1	0.737	0.667	0.678	0.6192	0.6044	0.5022	0.4178	0.444



Number of Thread vs. Speed up rate



In Table2, this case the effect of hardware cores is highly illustrated as after $K=4$ the program will take more time executing the algorithm because the number of threads are more the available cores.

Testing the program on macOS cmd using line argument as inputs

