Name : Swetha Shree Byllahali Ananthaswamy

Course : Principle of Comp & Info Tech Arch (2022 Spring)

Assignment : Lab2

Professor : Dr. Gennaro De Luca

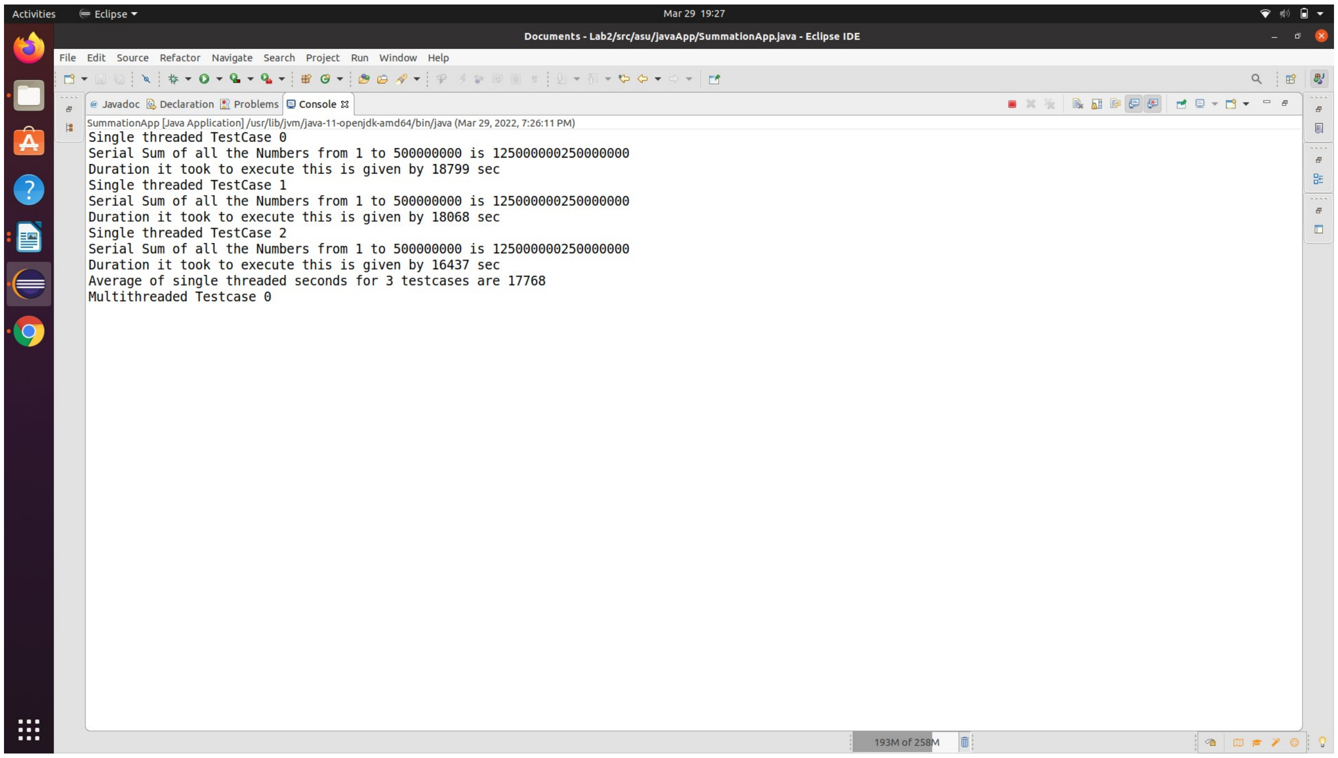
Report

In this lab, you will implement a multi-threaded sum application in a language that supports parallelism (not Python).

Your program will sum up all the numbers from 1 to 500,000,000. The result should be 125,000,000,250,000,000.

You should define a function for each of the following approaches. Each function should be timed so that you can print out how long the operation took. Sample code to do so is here: https://stackoverflow.com/a/7370824

Approach 1: Single threaded sum

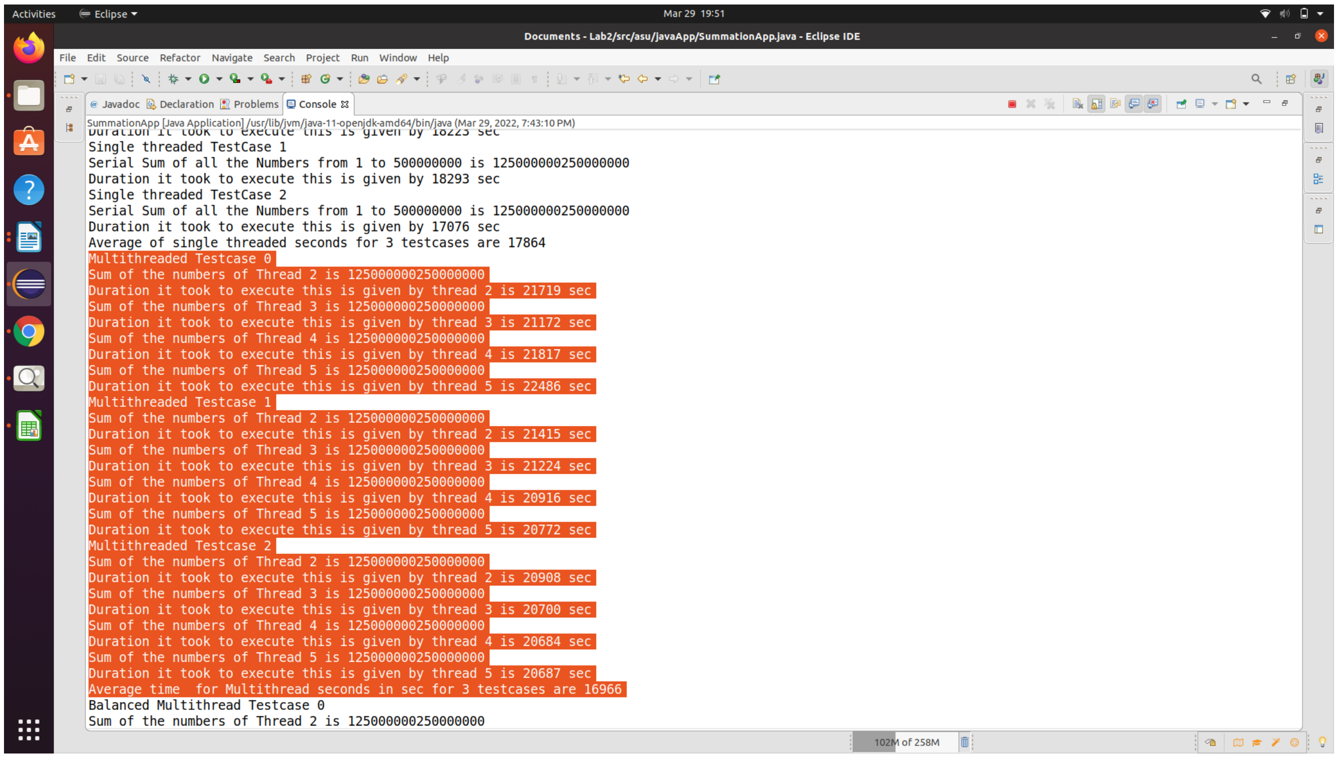

Activities
Eclipse
Mar 29 19:27
Documents - Lab2/src/asu/JavaApp/SummationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Javadoc
Declaration
Problems
Console
193M of 258M

```
SummationApp [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 29, 2022, 7:26:11 PM)
Single threaded TestCase 0
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 18799 sec
Single threaded TestCase 1
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 18068 sec
Single threaded TestCase 2
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 16437 sec
Average of single threaded seconds for 3 testcases are 17768
Multithreaded Testcase 0
```

Screenshort 1:

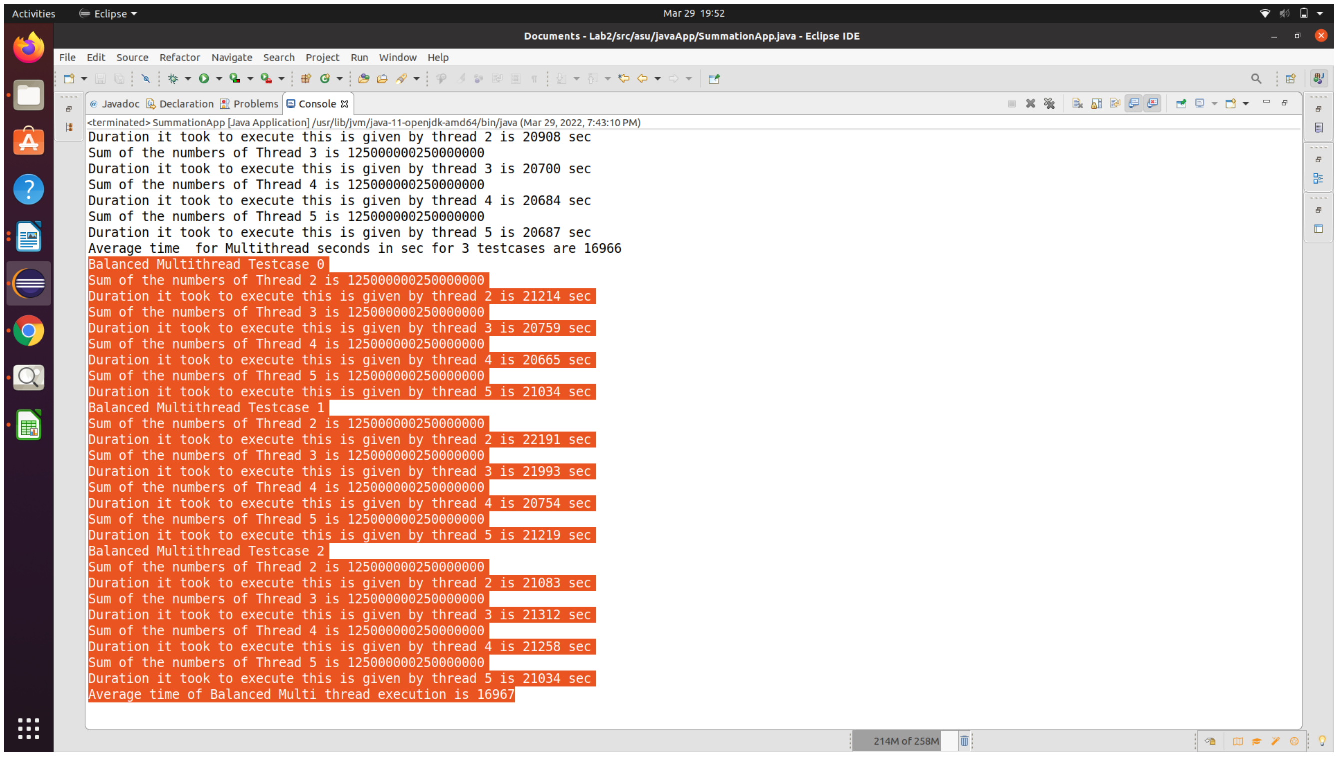
Approach 2: Multi-threaded sum. For this approach, you should test with a variable number of threads ranging from 2 to (at least) 10. The work should be split in numerical order. For example, if you have 2 threads, thread 1 should sum up the first half of the numbers and thread 2 should handle the second half.

Screenshot-2


Activities
Eclipse
Mar 29 19:51
Documents - Lab2/src/asu/javaApp/SummationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Javadoc Declaration Problems Console
102M of 258M

```
SummationApp [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 29, 2022, 7:43:10 PM)
Duration it took to execute this is given by 18223 sec
Single threaded TestCase 1
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 18293 sec
Single threaded TestCase 2
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 17076 sec
Average of single threaded seconds for 3 testcases are 17864
Multithreaded Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21719 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21172 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 21817 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 22486 sec
Multithreaded Testcase 1
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21415 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21224 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 20916 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 20772 sec
Multithreaded Testcase 2
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 20908 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 20700 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 20684 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 20687 sec
Average time  for Multithread seconds in sec for 3 testcases are 16966
Balanced Multithread Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
```

Approach 3: Balanced multi-threaded sum. For this approach, you should test with a variable number of threads ranging from 2 to (at least) 10. The work should be split in numerical order. For example, if you have 2 threads, thread 1 should sum up the odd numbers and thread 2 should handle the evens. With 3 threads, thread 1 would handle 1, 4, 7, ..., thread 2 would handle 2, 5, 8, ..., and thread 3 would handle 3, 6, 9, ..., etc.


Activities
Eclipse
Mar 29 19:52
Documents - Lab2/src/asu/JavaApp/SummationApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Javadoc Declaration Problems Console
214M of 258M

```
<terminated> SummationApp [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 29, 2022, 7:43:10 PM)
Duration it took to execute this is given by thread 2 is 20908 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 20700 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 20684 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 20687 sec
Average time  for Multithread seconds in sec for 3 testcases are 16966
Balanced Multithread Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21214 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 20759 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 20665 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21034 sec
Balanced Multithread Testcase 1
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 22191 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21993 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 20754 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21219 sec
Balanced Multithread Testcase 2
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21083 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21312 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 21258 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21034 sec
Average time of Balanced Multi thread execution is 16967
```

Screenshort 3.

2. Your CPU and the number of cores + threads

CPU(Processor) :
Intel® Core™ i5-6200U CPU @ 2.30GHz × 4

OS Name :
Ubuntu 20.04.2 LTS

In both Approaches I have used 16 threads for testing purpose.

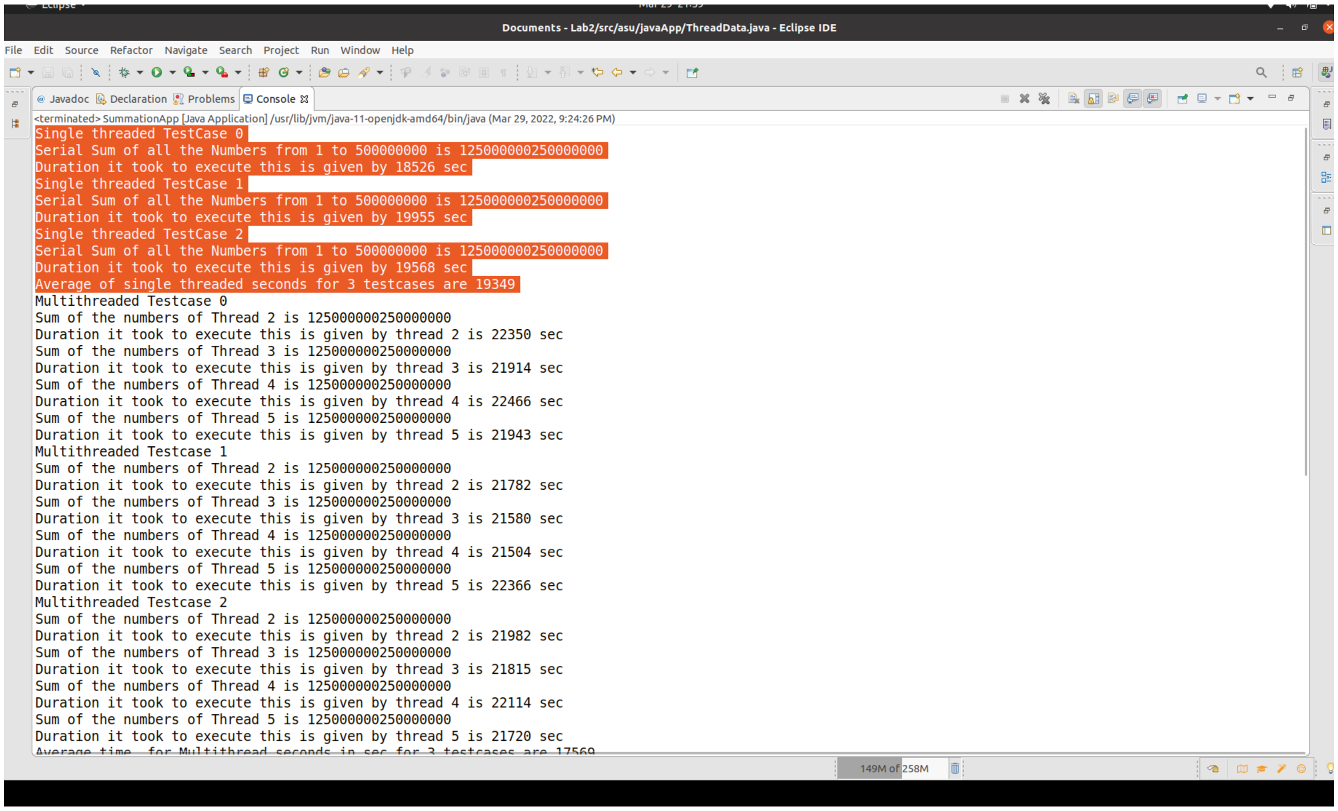
3. The results from each test. You should perform each test at least 3 times and report all results and average results for each test.

Test case 1 :

Approach 1 :

Single threaded TestCase 0
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by **18526 sec**
Single threaded TestCase 1
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by **19955 sec**
Single threaded TestCase 2
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by **19568 sec**

Average of single threaded seconds for 3 testcases are **19349 sec**


Documents - Lab2/src/asu/javaApp/ThreadData.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Javadoc Declaration Problems Console
149M of 258M

```
<terminated> SummationApp [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 29, 2022, 9:24:26 PM)
Single threaded TestCase 0
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 18526 sec
Single threaded TestCase 1
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 19955 sec
Single threaded TestCase 2
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 19568 sec
Average of single threaded seconds for 3 testcases are 19349
Multithreaded Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 22350 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21914 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22466 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21943 sec
Multithreaded Testcase 1
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21782 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21580 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 21504 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 22366 sec
Multithreaded Testcase 2
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21982 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21815 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22114 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21720 sec
Average time  for Multithread seconds in sec for 3 testcases are 17569
```

Approach 2 :

Multithreaded Testcase 0

```
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 22350 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21914 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22466 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21943 sec
```

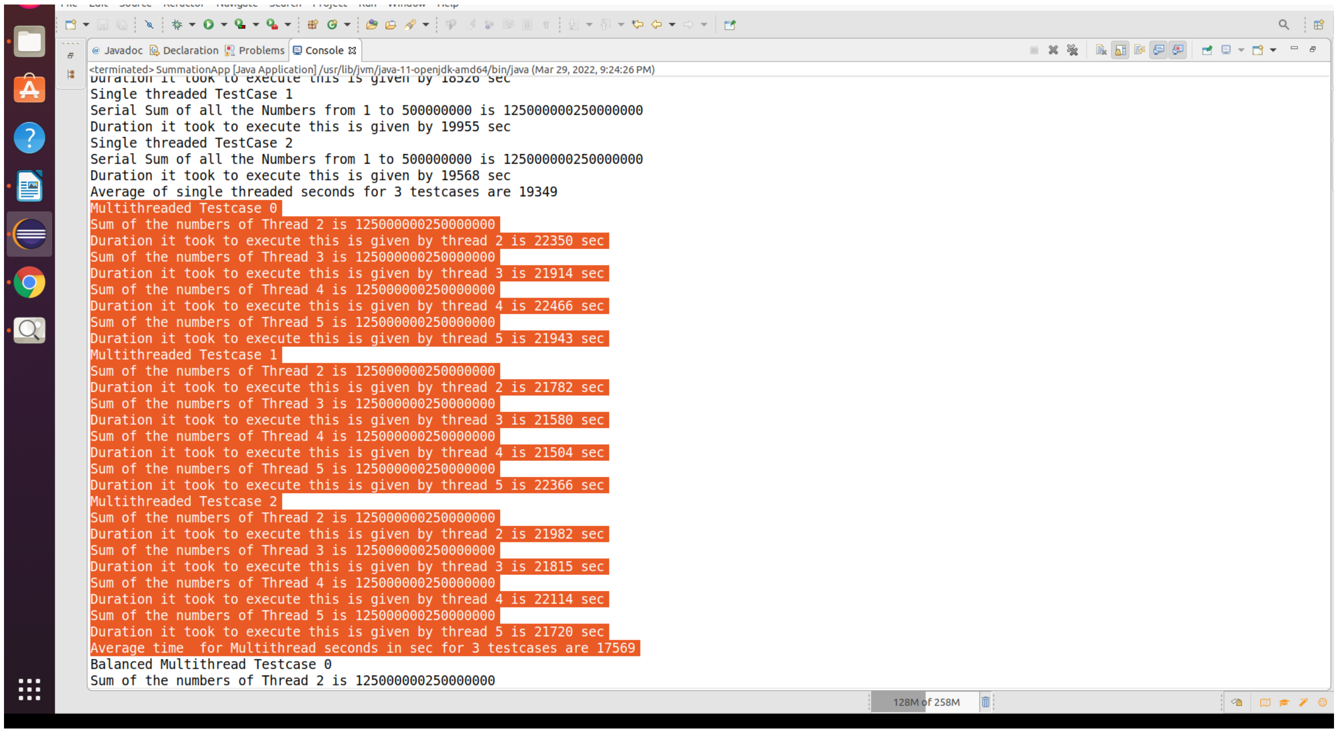
Multithreaded Testcase 1

```
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21782 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21580 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 21504 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 22366 sec
```

Multithreaded Testcase 2

```
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21982 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21815 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22114 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21720 sec
```

Average time for Multithread seconds in sec for 3 testcases are **17569 sec**



```
<terminated> SummationApp [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 29, 2022, 9:24:26 PM)
Duration it took to execute this is given by 18526 sec
Single threaded TestCase 1
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 19955 sec
Single threaded TestCase 2
Serial Sum of all the Numbers from 1 to 500000000 is 125000000250000000
Duration it took to execute this is given by 19568 sec
Average of single threaded seconds for 3 testcases are 19349
Multithreaded Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 22350 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21914 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22466 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21943 sec
Multithreaded Testcase 1
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21782 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21580 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 21504 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 22366 sec
Multithreaded Testcase 2
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21982 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 21815 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22114 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21720 sec
Average time  for Multithread seconds in sec for 3 testcases are 17569
Balanced Multithread Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
```

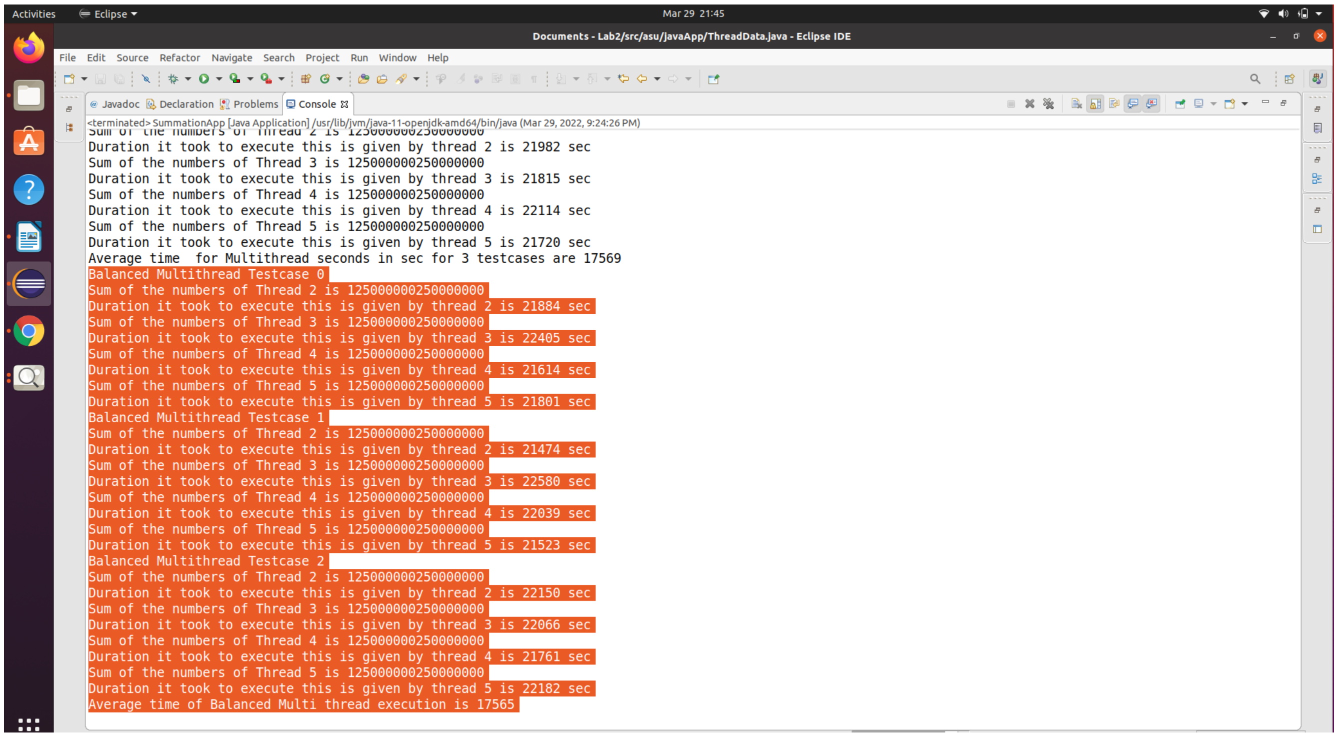
Approach 3 :

```
Balanced Multithread Testcase 0
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21884 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 22405 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 21614 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21801 sec

Balanced Multithread Testcase 1
Sum of the numbers of Thread 2 is 125000000250000000
Duration it took to execute this is given by thread 2 is 21474 sec
Sum of the numbers of Thread 3 is 125000000250000000
Duration it took to execute this is given by thread 3 is 22580 sec
Sum of the numbers of Thread 4 is 125000000250000000
Duration it took to execute this is given by thread 4 is 22039 sec
Sum of the numbers of Thread 5 is 125000000250000000
Duration it took to execute this is given by thread 5 is 21523 sec

Balanced Multithread Testcase 2
```

```
Sum of the numbers of Thread 2 is 1250000000250000000
Duration it took to execute this is given by thread 2 is **22150 sec**
Sum of the numbers of Thread 3 is 1250000000250000000
Duration it took to execute this is given by thread 3 is **22066 sec**
Sum of the numbers of Thread 4 is 1250000000250000000
Duration it took to execute this is given by thread 4 is **21761 sec**
Sum of the numbers of Thread 5 is 1250000000250000000
Duration it took to execute this is given by thread 5 is **22182 sec**
Average time of Balanced Multi thread execution is **17565 sec**
```


Activities
Eclipse
Mar 29 21:45
Documents - Lab2/src/asu/javaApp/ThreadData.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Javadoc Declaration Problems Console

```
<terminated> SummationApp [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 29, 2022, 9:24:26 PM)
Sum of the numbers of Thread 2 is 1250000000250000000
Duration it took to execute this is given by thread 2 is 21982 sec
Sum of the numbers of Thread 3 is 1250000000250000000
Duration it took to execute this is given by thread 3 is 21815 sec
Sum of the numbers of Thread 4 is 1250000000250000000
Duration it took to execute this is given by thread 4 is 22114 sec
Sum of the numbers of Thread 5 is 1250000000250000000
Duration it took to execute this is given by thread 5 is 21720 sec
Average time  for Multithread seconds in sec for 3 testcases are 17569
Balanced Multithread Testcase 0
Sum of the numbers of Thread 2 is 1250000000250000000
Duration it took to execute this is given by thread 2 is 21884 sec
Sum of the numbers of Thread 3 is 1250000000250000000
Duration it took to execute this is given by thread 3 is 22405 sec
Sum of the numbers of Thread 4 is 1250000000250000000
Duration it took to execute this is given by thread 4 is 21614 sec
Sum of the numbers of Thread 5 is 1250000000250000000
Duration it took to execute this is given by thread 5 is 21801 sec
Balanced Multithread Testcase 1
Sum of the numbers of Thread 2 is 1250000000250000000
Duration it took to execute this is given by thread 2 is 21474 sec
Sum of the numbers of Thread 3 is 1250000000250000000
Duration it took to execute this is given by thread 3 is 22580 sec
Sum of the numbers of Thread 4 is 1250000000250000000
Duration it took to execute this is given by thread 4 is 22039 sec
Sum of the numbers of Thread 5 is 1250000000250000000
Duration it took to execute this is given by thread 5 is 21523 sec
Balanced Multithread Testcase 2
Sum of the numbers of Thread 2 is 1250000000250000000
Duration it took to execute this is given by thread 2 is 22150 sec
Sum of the numbers of Thread 3 is 1250000000250000000
Duration it took to execute this is given by thread 3 is 22066 sec
Sum of the numbers of Thread 4 is 1250000000250000000
Duration it took to execute this is given by thread 4 is 21761 sec
Sum of the numbers of Thread 5 is 1250000000250000000
Duration it took to execute this is given by thread 5 is 22182 sec
Average time of Balanced Multi thread execution is 17565
```

4. Provide a detailed explanation for all of the results you received (this should probably be the longest section). Why did you receive these results? Are they what you expected? Etc.

Approch 1 :
Serial Approch :
main : 20561 sec

Example TestCase1 :

Approach 2:
In the Equal Split :
Thread 2 : 20491 sec
Thread 3 : 20336 sec
Thread 4 : 20259 sec

Approach 3:
In the Balanced Split :
Thread 2 : 20032 sec
Thread 3 : 20052 sec
Thread 4 : 20055 sec

In case of Approach 1, serial execution by the main thread, In this case no threads were actually created externally.
though the JVM creates the thread named main when it starts running the program.
so approach 1 took 20561 sec for serial execution as shown in Test case 1.

In case of Approach 2 , threads are divided equally to handle huge number. like for example if there are two threads,
one will handle half of the numbers addition (sum) and other thread handle next half of the numbers.
In this case , we are externally creating the threads and calculating the time it takes execute addition functionality when multiple threads involved.
In Test case shown above , 2 threads took 20491 sec to do the assigned operation. In this case, Thread0 handled 1st half of the numbers addition and thread1 handled next half of the functionality.
In case of 3 Threads, the numbers are equally divided for all three of the threads and functionality got executed.
Thread 3 takes lower time to execute.
Thread 4 took even less time to execute.
as the number of threads increases, the work should be done in the fastest possible way.

In case of Approach 3 , threads are divided in the balanced way. like for Example , If there are two Threads, thread0 will handle the sum of the odd numbers
up to the max. and that of thread2 handle the even number addition.
If In case of two Threads, it takes 20032 sec to complete the execution process.
if the 3 threads starts(In the balanced way, thread0 starts with execution 1 ,4, 7...summation ,
thread1 2,5,8... thread2 is 3,6,9..summation.) ,it took 20052 sec to complete the execution process.
if the 4 threads starts ,it took 20055 sec to complete the execution process.
as the number of the Threads Increases each time , the execution will become faster(task in hand will complete soon).

Various factors are involved when the thread execution starts and all threads are got spited there work equally. Each thread should completes its work in order to calculate the time of execution.

When I executed the threads each time, some time threads were taking much longer time than single approach. Usually, 2 Threads takes a bit longer time to execute than 16Threads.

Analyzing the performance of the balanced multi thread approach, we can see that it follows a similar trend when compared to the approach 2 with performance peak at 7 threads. But comparing the averages of approaches 2 and 3, shows us that both of them are more or less at the same level of performance, with approach 2 performing better in some cases and approach 3 performing better in the other.

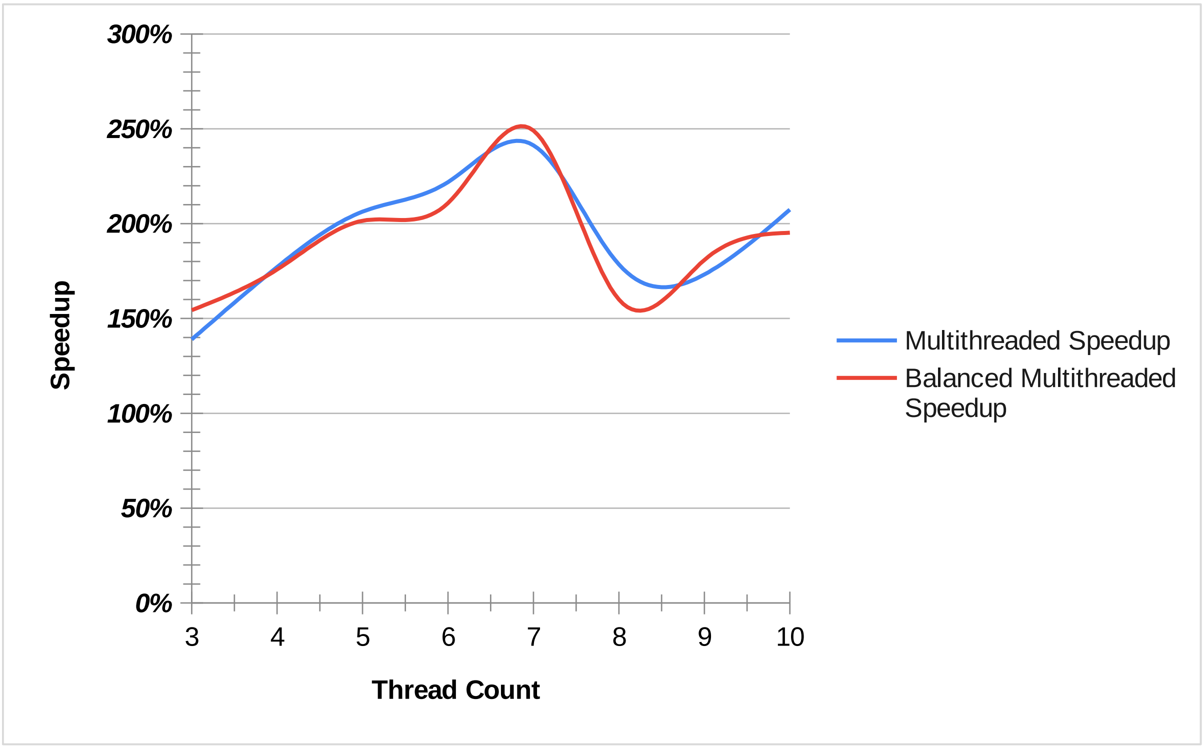
Are they what you expected?

No, the performance of the actual received outcomes falls far short of the expectations. Although, in terms of performance, approach 2 – multi-threaded sum outperforms approach 1 – single threaded sum. Similarly, approach 3 – balanced multi-threaded sum outperforms approach 2 – multi-threaded sum by a little margin.

5. What is the theoretical maximum speedup you expected, according to Amdahl's Law? Detail this number as both a percent speedup and an expected time (in seconds) according to your average sequential results. How do these values compare to what you received?

Amdal's Law states that :

MaximumSpeedup = 1 / (1- (P/100)).



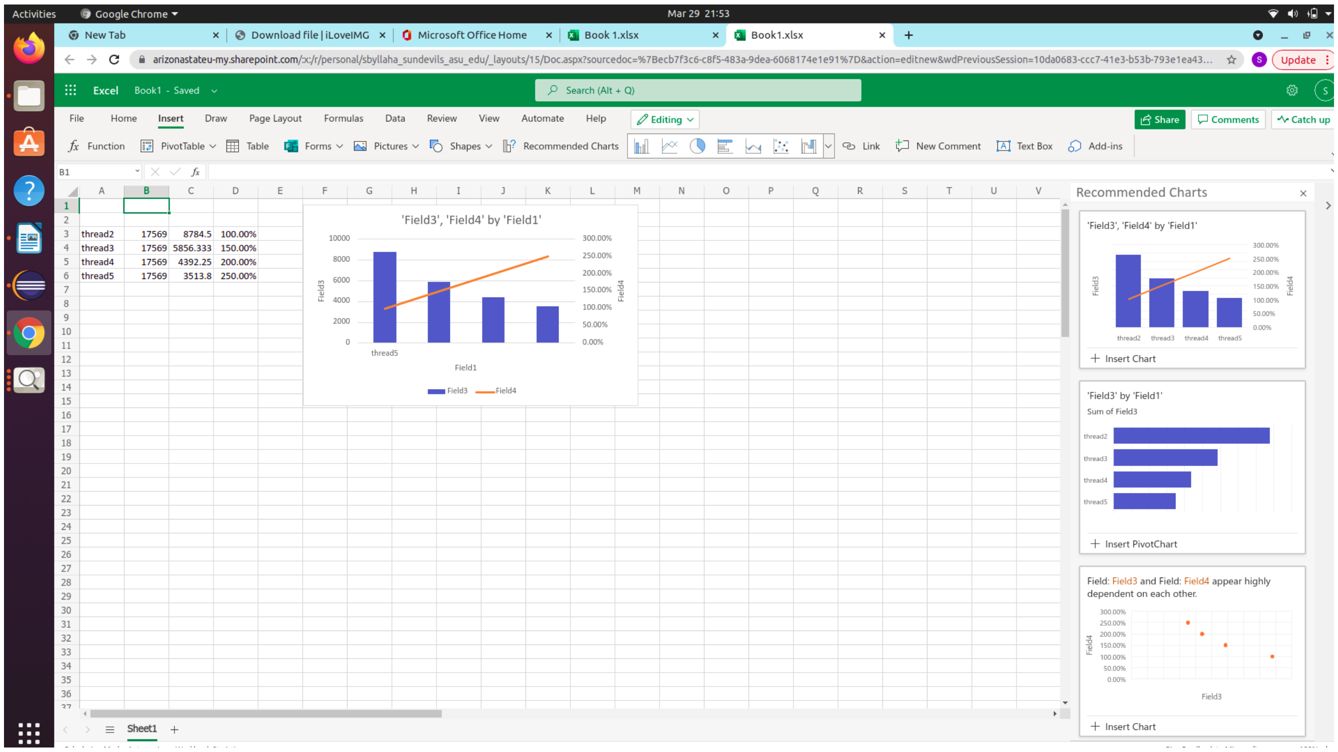
How do these values compare to what you received?

Between the theoretical maximum speedup expected and the actual speedup numbers received,there is a significant variance. For example, for a multi-threaded sum of 3 threads, the maximum speedup should be 300 %, but the maximum speedup received is only 139 %. Maximum speedup for balanced multi-threaded total of 3 thread count should be 300 %, while the actual received maximum speedup is only 154 %. When the theoretical maximum speedup expected values are compared to the actual maximum speedup values received, there is a significant disparity between the two.

Approach For 5 Threads:

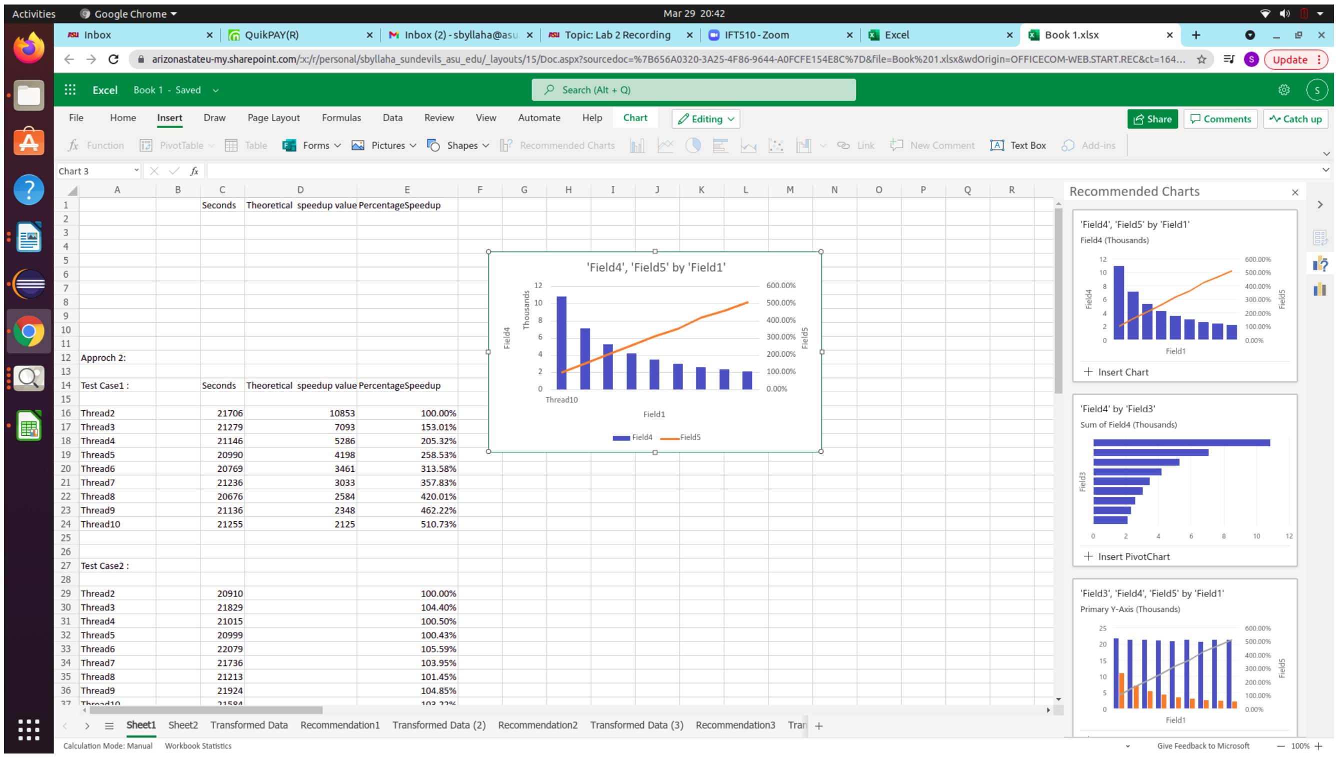
as shown in screenshots below:

Threads	3	4	5
Multi-threaded	153%	205%	258%
Balanced-MultiThreaded	146%	197%	246%



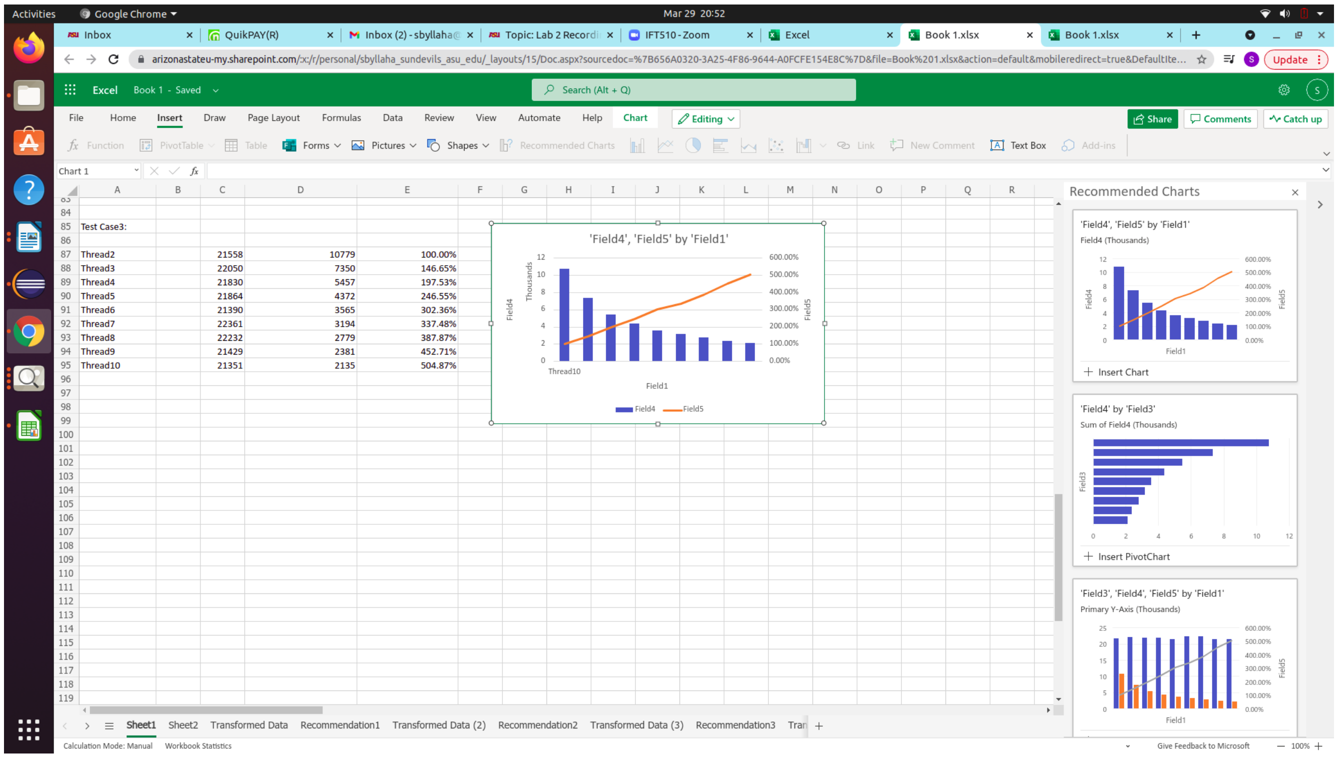
	A	B	C	D
3	thread2	17569	8784.5	100.00%
4	thread3	17569	5856.333	150.00%
5	thread4	17569	4392.25	200.00%
6	thread5	17569	3513.8	250.00%

Approach 2 for 10 threads:-



	A	B	C	D	E
1			Seconds	Theoretical speedup value	PercentageSpeedup
12	Approch 2:				
14	Test Case1 :		Seconds	Theoretical speedup value	PercentageSpeedup
16	Thread2		21706	10853	100.00%
17	Thread3		21279	7093	153.01%
18	Thread4		21146	5286	205.32%
19	Thread5		20990	4198	258.53%
20	Thread6		20769	3461	313.58%
21	Thread7		21236	3033	357.83%
22	Thread8		20676	2584	420.01%
23	Thread9		21136	2348	462.22%
24	Thread10		21255	2125	510.73%
27	Test Case2 :				
29	Thread2		20910		100.00%
30	Thread3		21829		104.40%
31	Thread4		21015		100.50%
32	Thread5		20999		100.43%
33	Thread6		22079		105.59%
34	Thread7		21736		103.95%
35	Thread8		21213		101.45%
36	Thread9		21924		104.85%

Approch3 for 10 Threads:-



Test Case3:

Thread2		21558	10779	100.00%
Thread3		22050	7350	146.65%
Thread4		21830	5457	197.53%
Thread5		21864	4372	246.55%
Thread6		21390	3565	302.36%
Thread7		22361	3194	337.48%
Thread8		22232	2779	387.87%
Thread9		21429	2381	452.71%
Thread10		21351	2135	504.87%

Code :

```java
package asu.javaApp;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;


public class SummationApp {

    public SummationApp() {
        // TODO Auto-generated constructor stub
    }

    public static BigInteger noOfThreads;

    public static void main(String[] args) {

        long average = 0;
        for (int i = 0; i < 3; i++) {

            System.out.println("Single threaded TestCase " + i);
            long startTime = System.currentTimeMillis();
            BigInteger sum = serialCalculation();

            long endTime = System.currentTimeMillis();
            long duration = (endTime - startTime);
            average += duration;
            System.out.println("Serial Sum of all the Numbers from 1
to 500000000 is " + sum);
            System.out.println("Duration it took to execute this is
given by " + duration + " sec");

        }
        System.out.println("Average of single threaded seconds for 3
testcases are " + average / 3);

        average = 0;
        int threadValue = 5;
        for (int i = 0; i < 3; i++) {

            System.out.println("Multithreaded Testcase " + i);
            for (int m = 2; m <= threadValue; m++) {

                long startTime = System.currentTimeMillis();
                BigInteger threadCount = BigInteger.valueOf(m);
                multiThreadCalculation(threadCount);
                long endTime = System.currentTimeMillis();
                long duration = (endTime - startTime);
                average += duration;
                System.out.println("Sum of the numbers of Thread " +
m + " is " + ThreadData.sum);
                System.out.println("Duration it took to execute this
is given by thread " + m + " is " + duration + " sec");
            }
        }
        System.out.println("Average time  for Multithread seconds in
sec for 3 testcases are " + average / (threadValue * 3));

        average = 0;

        for (int i = 0; i < 3; i++) {

            System.out.println("Balanced Multithread Testcase " +
i);
            for (int n = 2; n <= threadValue; n++) {

                long startTime = System.currentTimeMillis();
                noOfThreads = BigInteger.valueOf(n);
                ThreadData.isApproch3 = true;
                try {

multiThreadCalculationUsingApprochThree(noOfThreads);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                long endTime = System.currentTimeMillis();
                long duration = (endTime - startTime);
                average += duration;
                System.out.println("Sum of the numbers of Thread " +
n + " is " +
                    ThreadData.sum);
                System.out.println("Duration it took to execute this
is given by thread " + n +
                    " is " + duration + " sec");

            }
        }
        System.out.println("Average time of Balanced Multi thread
execution is " + average / (threadValue * 3));


    }
    private static void
multiThreadCalculationUsingApprochThree(BigInteger threadCount)
throws InterruptedException {


        BigInteger number = new BigInteger("500000000");

        ThreadData.sum = BigInteger.ZERO;
        List < ThreadData > data = new ArrayList < ThreadData > ();
        List < Thread > threads = new ArrayList < Thread > ();

        int datapointer = 0;
        BigInteger threadDivision =
threadCount.divide(BigInteger.TWO);
        BigInteger remainder = threadCount.subtract(threadDivision);


        for (BigInteger i = BigInteger.ZERO;
i.compareTo(threadCount) < 0; i = i.add(BigInteger.ONE)) {

            BigInteger start;
            if (threadCount.equals(BigInteger.TWO)) {
                start = i.equals(BigInteger.ZERO) ? BigInteger.TWO :
threadCount.subtract(i);

            } else {
                start = threadCount.subtract(i);
            }

            if (!threadDivision.equals(BigInteger.ZERO)) {

                data.add(new ThreadData(number, start, number,
false));
                threads.add(new Thread(data.get(datapointer)));
                threadDivision =
threadDivision.subtract(BigInteger.ONE);
                datapointer++;
            } else if (!remainder.equals(BigInteger.ZERO)) {

                data.add(new ThreadData(number, start, number,
true));
                threads.add(new Thread(data.get(datapointer)));
                remainder = remainder.subtract(BigInteger.ONE);
                datapointer++;
            }
        }
        for (Thread thread: threads) {

            thread.start();
            thread.join();
        }
    }


    private static void multiThreadCalculation(BigInteger
threadCount) {

        BigInteger number = new BigInteger("500000000");

        ThreadData.sum = BigInteger.ZERO;
        List < ThreadData > data = new ArrayList < ThreadData > ();
        List < Thread > threads = new ArrayList < Thread > ();

        int datapointer = 0;

        for (BigInteger i = BigInteger.ZERO;
i.compareTo(threadCount) < 0; i = i.add(BigInteger.ONE)) {


            BigInteger start = i.equals(BigInteger.ZERO) ?
BigInteger.ONE :
i.multiply((number.divide(threadCount))).add(BigInteger.ONE);


            BigInteger stop = i.equals(BigInteger.ZERO) ?
number.divide(threadCount) :

(i.add(BigInteger.ONE)).multiply(number.divide(threadCount));


            if (!number.mod(threadCount).equals(BigInteger.ZERO) &&
i.equals(threadCount.subtract(BigInteger.ONE))) {
                BigInteger x = number.mod(threadCount);
                stop = stop.add(x);

            }
            data.add(new ThreadData(number, start, stop));

            threads.add(new Thread(data.get(datapointer)));

            datapointer++;
        }
        for (Thread thread: threads) {
            try {
                thread.start();
                thread.join();

            } catch (InterruptedException e) { // TODO Auto-
generated catch block
                e.printStackTrace();
            }
        }
    }

    public static BigInteger serialCalculation() {

        BigInteger number = new BigInteger("500000000");
        BigInteger sum = BigInteger.ZERO;
        for (BigInteger i = BigInteger.valueOf(1);
i.compareTo(number) <= 0; i = i.add(BigInteger.ONE)) {

            sum = sum.add(i);
        }
        return sum;

    }

}
```

```
package asu.javaApp;

import java.math.BigInteger;


public class ThreadData implements Runnable {

    public BigInteger number;
    public BigInteger start;
    public BigInteger stop;
    public static BigInteger sum;
    public static Boolean isApproch3 = false;
    public Boolean threadName = null;
    public static Boolean divideOnlyOnce = true;

    public ThreadData(BigInteger number, BigInteger start, BigInteger
stop) {

        this.number = number;
        this.start = start;
        this.stop = stop;
    }


    public ThreadData(BigInteger number, BigInteger start, BigInteger
stop, Boolean threadName) {

        this.number = number;
        this.start = start;
        this.stop = stop;
        this.threadName = threadName;

    }




    @Override
    public void run() {

        if (isApproch3.equals(false)) {
            for (BigInteger i = this.start; i.compareTo(this.stop)
<= 0; i = i.add(BigInteger.ONE)) {
                sum = sum.add(i);
            }
        } else {

            if (this.threadName) {
                sumOfEvenNumber(this.start, this.stop);
            } else {
                sumOfOddNumbers(this.start, this.stop);
            }
        }

    }

    private void sumOfEvenNumber(BigInteger start2, BigInteger
stop2) {
        for (BigInteger i = start2; i.compareTo(stop2) <= 0; i =
i.add(SummationApp.noOfThreads)) {
            sum = sum.add(i);
        }

    }

    private void sumOfOddNumbers(BigInteger start2, BigInteger
stop2) {

        for (BigInteger i = start2; i.compareTo(stop2) <= 0; i =
i.add(SummationApp.noOfThreads)) {
            sum = sum.add(i);
        }

    }

}
```