CS5300 - Parallel & Concurrent Programming
Spring 2022

Comparing Different Parallel Implementations for
Identifying Prime Numbers
Assignment 1

Vibhanshu Jain
CS19B1027

Program Design

- We define some global variables which are helpful for the implantation. These are:
  - mtx: the mutex lock
  - inputFile: for the input from the file
  - outputFile: for adding the output to the file
  - Time: for adding the time taken details
- All the three functions are implemented separately, DAM, SAM1, and SAM2.
- In the main function, we are first taking the input from the file
- Then creating new M threads variable using pthead_t
- We are using a two-time variable which is time0 and time1 for storing the start and end time of each event
- Assigning the task to all the threads followed by merging them.
- We are getting the time details using the function, gettimeofday, and storing it a text file, followed by closing the file output.
- A simple isPrime function to check whether the given integer is prime or not

DAM

- Created a Counter class so that we can assign the tasks accordingly
- Created a new object counter and initialized it to 1
- Call each thread one by one
- Assigning a particular to be searched at a time by a particular thread at a time
- Uses the mutex lock whenever we are incrementing the counter

SAM1

- Created a stuck threadData which contains the field to store the data easily. It can store three integers which will contain the starting number, increment step, and the maximum limit
- Calling each thread and passing an object data which contains the data.
- Each thread will get the data like i,m, maximum limit from which thread i can itself calculate which many numbers it needed to be checked.

SAM2

- Similar to SAM1, the difference is in the data values which we are passing. In SAM2 these are 2*i + 1, 2*m, the maximum limit, which helps to achieve our goals.

Problems faced:

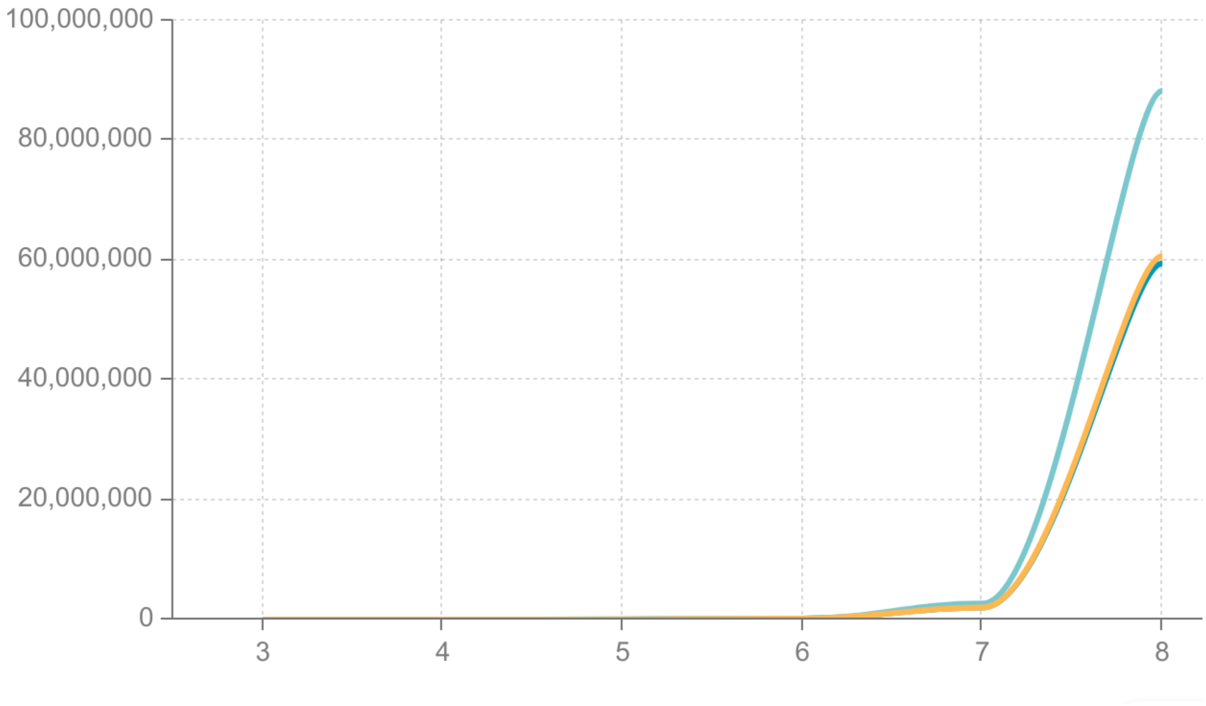
- The file IO is not atomic, because of which it was printing irregularly. To solve this I use mutex lock around all the printing statements.
- Some prime numbers are missing when I kept the counter class public because in that case, the threads are changing the values randomly even the some other is changing it.

Performance Comparison:

<u>Time vs Size:</u>
'm' is fixed to 10.

n	DAM	SAM	SAM1
3	76	64	106
4	3332	2442	2567
5	21144	15619	17181
6	156689	165814	159412
7	1985369	2687037	2009533
8	59508553	88302056	60683357

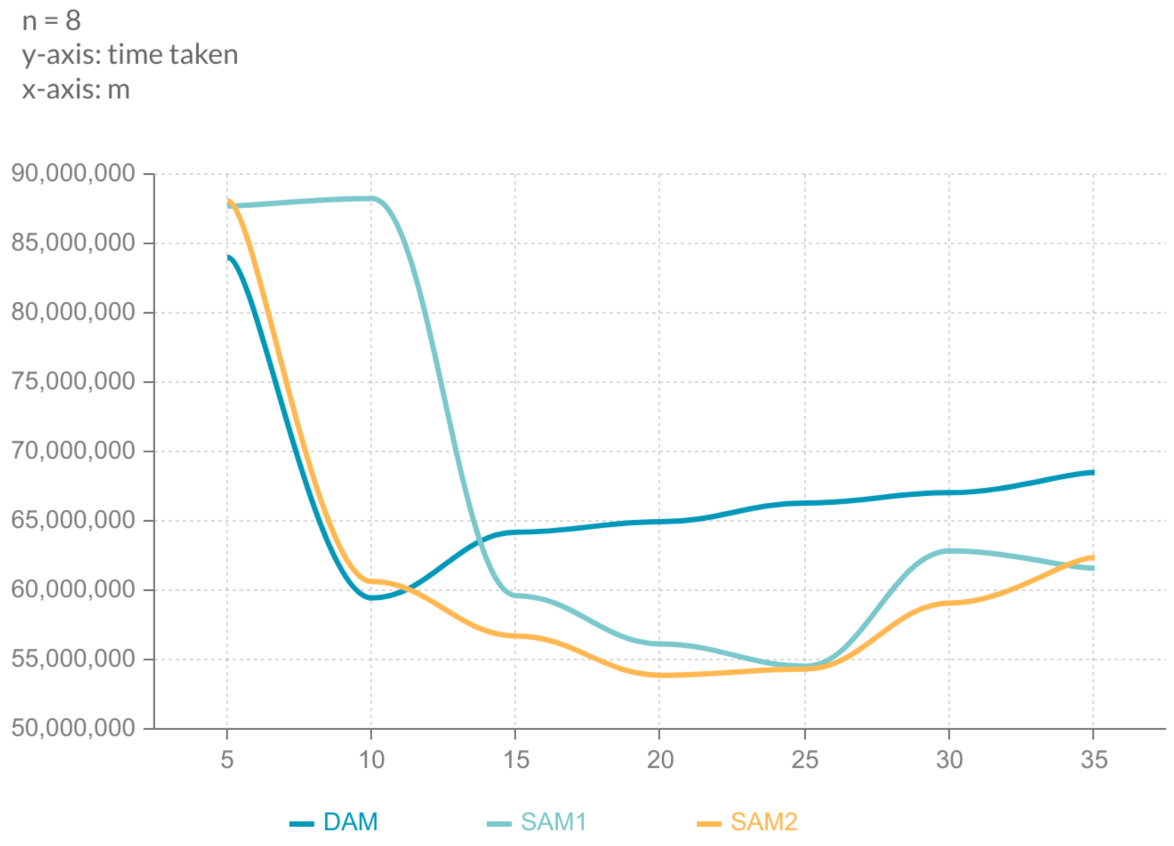
Graph:



Time vs Number of threads:
n = 8

m	DAM	SAM	SAM1
5	84063301	87762049	88144324
10	59508553	88302056	60683357
15	64247300	59668216	56758260
20	64999655	56184767	53928873
25	66344334	54588781	54371968
30	67092327	62903735	59132571
35	68561709	61646539	62420486

Graph:



Conclusion:

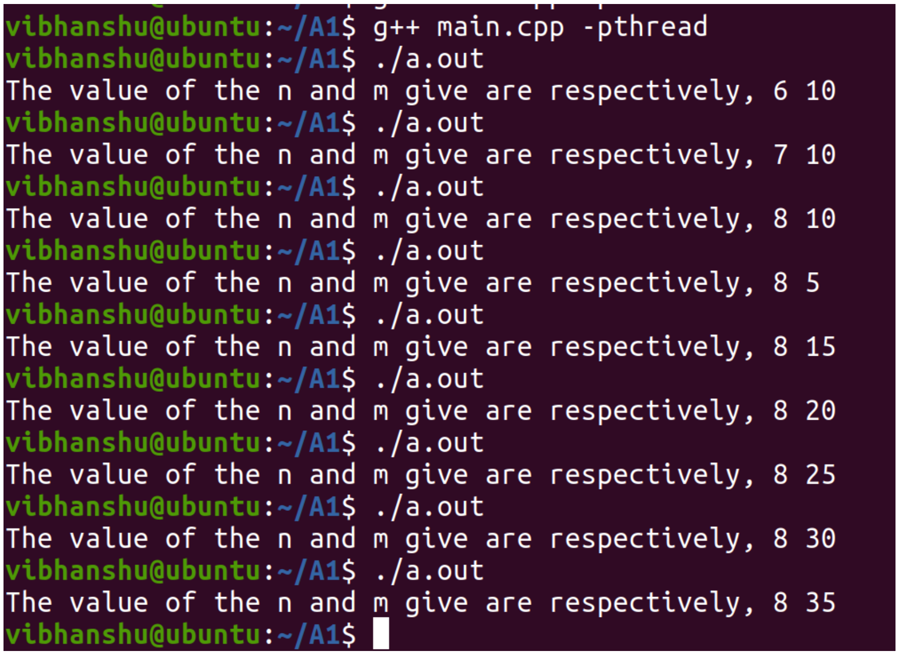
It can be clearly seen that, as we increase ‘n’ keeping the thread number constant the time of execution increases. It increases very quickly as we are increasing the number by the exponential.
For the execution time for some initial values like 3, the execution time is negligible as compared to the 7 or 8.

The decrease as we are increasing the number of threads, which can also be seen easily as more threads more distributed work, and less time.
But the number of threads is very very less than the number of integers, so the curve decreases but not quickly.

Also, it can be seen that the curve is lowest at 25 for both SAM1 and SAM2, and even lesser than DAM. After this point, the time of execution again started increasing.

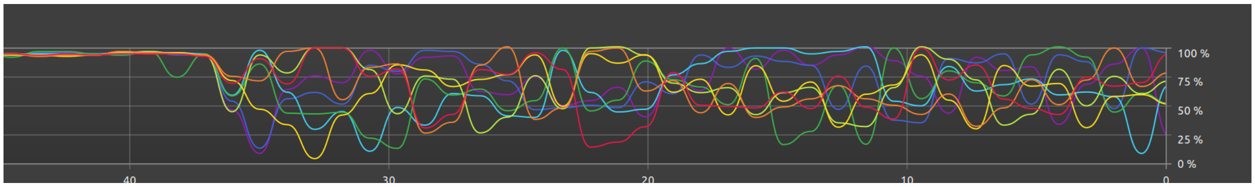
The system I used:
AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz
8.00 GB DDR4

Some screenshots of the system while coding execution.



```
vibhanshu@ubuntu:~/A1$ g++ main.cpp -pthread
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 6 10
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 7 10
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 10
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 5
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 15
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 20
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 25
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 30
vibhanshu@ubuntu:~/A1$ ./a.out
The value of the n and m give are respectively, 8 35
vibhanshu@ubuntu:~/A1$ 
```

System Usage: (Multithreaded process)


100 %
75 %
50 %
25 %
0 %
40
30
20
10
0