Sri Hari Malla - CS19BTECH11039
Prof : Dr. Sathya Peri
Parallel and Concurrent Programming
Indian Institute of Technology Hyderabad

REPORT

Sri Hari Malla - CS19BTECH11039

January 15, 2022

1 Design and Overview:

1.1 Goal:

As per the instructions in problem statement, the following should be implemented:

- Wrapper for Time Calculation
- Race free Thread architecture
- DAM, SAM methods along with a method to check if a number is prime.

1.2 Design:

1.2.1 Time calculation:

The time should be calculated to measure the performance of each method.

- This can be done using `std::chrono` and related `timepoint` casting.
- Wrapping this in a class as a method helps us in saving time writing multiple lines of code again and again.

1.2.2 Threading:

To achieve the parallelism, we need to use threads.

- As `C++` thread library is a wrapper implementation of threads available on various architectures and is platform independent,`std::thread` could be used instead of `pthread` as we've scope to produce some high level code.
- For dealing with race conditions we could use `std::mutex` to acquire and release locks in threads.

1.2.3 DAM method:

Let's see the DAM method in a nut shell:

- The global counter will be dynamically incremented in various threads.
- The increment of the counter is a race condition, so locks are to be used.
- The counter will be sent to prime checker function and it returns a boolean.
- This seems to produce but couldn't guarantee in the ordered set of primes as one thread can go faster than the other thread as it depends on the OS allocation of cpu based on the load.

1.2.4 SAM1 method:

Let's see the SAM1 method in a nut shell:

- Each thread runs a loop of numbers until it reaches the upper bound.
- No locks are to be used as there are no race conditions and everything is thread safe.
- This clearly produces primes with no relative order as different threads work on different ranges of values at each time.

1.2.5 SAM2 method:

Let's see the SAM2 method in a nut shell:

- It is entirely same that of SAM1 method but has a little tweak
- The SAM1 method loops for even numbers also, but in this method the increment is set in the way that no even number comes to the iterator.
- This saves minimal time but when a large limit is considered, it's contribution can be significant.

1.3 Complications:

- The out stream in `C++` is not properly synchronized, so need to use locks while printing it out.
- For measuring the time, `std::chrono` is used and needed to refer to the official documentation to know about proper data types and proper type casting.

1.4 Program Flow:

The flow of the program will be as followed:

- Program starts at main
- Reads the input from the file
- Calculates the number of threads and Upper bound
- Instantiates the timer class which records the time stamp
- Creates threads corresponding to the input for each method(DAM/SAM)
- The methods DAM and SAM will be executed one after the other
- The timer class will be stopped after execution i.e after all the threads are joined.

2 Increment method in DAM:

- While incrementing the counter in DAM, extra care is to be taken as it is a critical section and is thread unsafe. For this one can use `atomic int` and a normal counter variable wrapped with locks.
- Here in this program, I used a variable wrapped with locks because if offers a better performance than the `atomic int`.
- If atomic int is used, then there is no need of wrapping locks because it is thread safe.
- The bench marking is referred to the internet.

3 Observations and Report:

3.1 Graph1 : Change in N vs Change in Time (constant m set to 10)

As instructed, this is the graph plotted with points observed while running the program with changing the N and recording the time whilst the number of threads is set to 10.

- One can clearly see that the SAM1 and SAM2 methods are outperforming DAM method.
- It is also seen that from 3 to 6, all the methods are taking almost the same time. But its only visual perception that is because of the scaling of the entire picture.

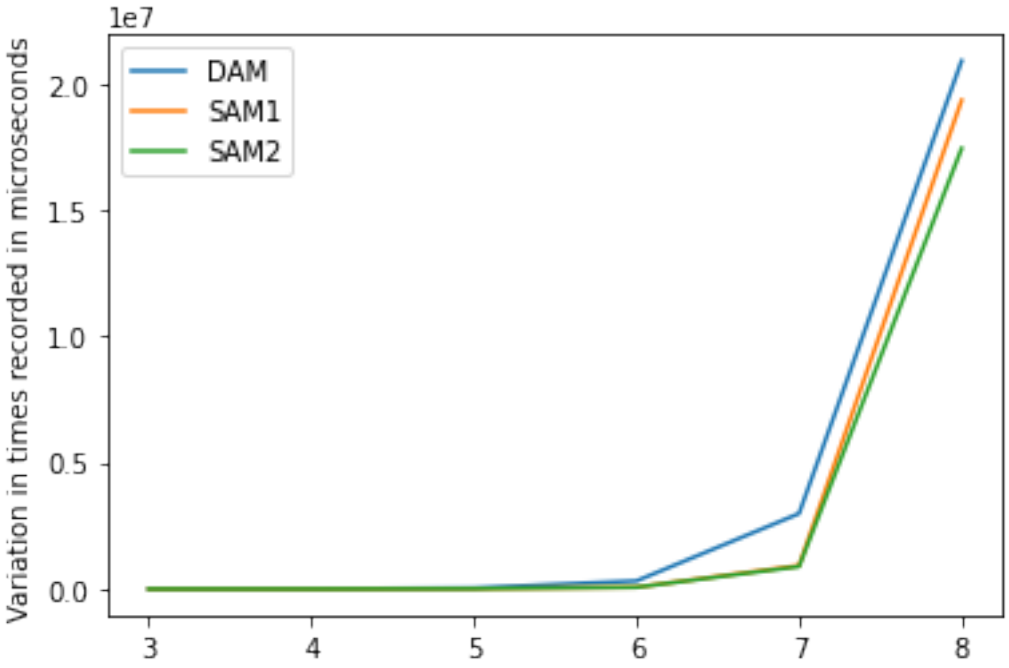

Figure 1: Change in N vs Change in Time taken

- To the reference the picture with variation of n from 3 to 6 is also attached here.

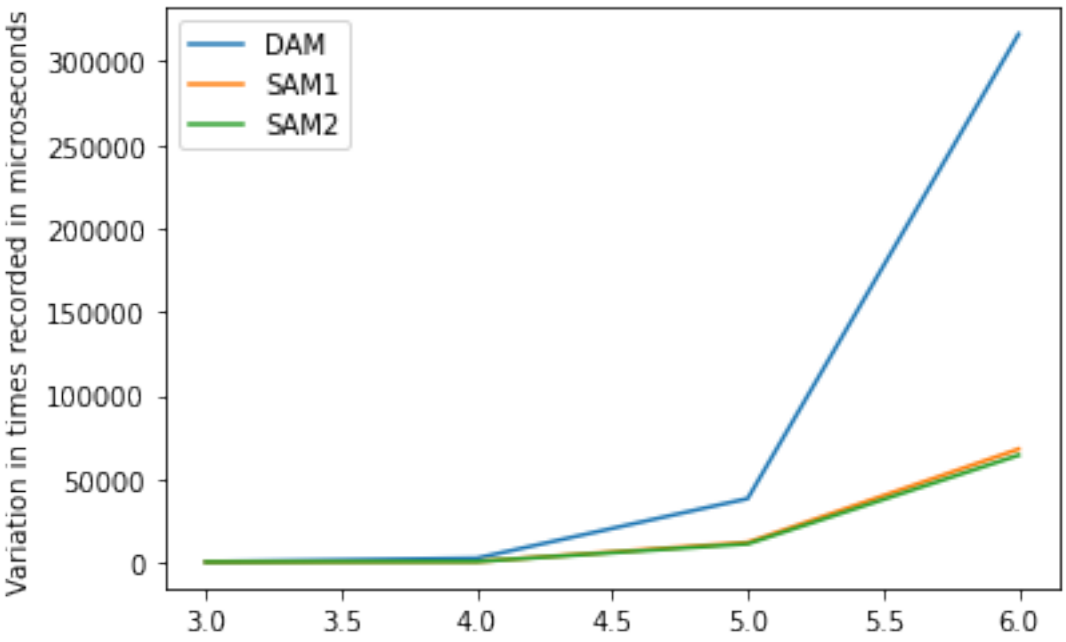

Figure 2: Change in N vs Change in Time taken(smaller n)

- It is observed that DAM is taking nearly 1.5-2 times of that of SAM1 method is taking and interestingly SAM2 is taking the 0.95-1 times that of SAM1.
- The reason for this is possibly all the threads in DAM are taking actually much time than the SAM because of the overhead of the locks used.
- Although the number of numbers determined in each threads is not the same in DAM, it is more or less same assuming ideal balance on each thread which we can't control. With this analogy DAM needs to perform better but SAM is performing better because of thread safe code properties.
- So clearly SAM methods are performing a way better than the DAM as expected.

3.2 Graph2 : Change in m vs Change in Time (constant n set to 8)

As instructed, this is the graph plotted with points observed while running the program with changing the M and recording the time whilst the number of upper bound is set to 10e8.

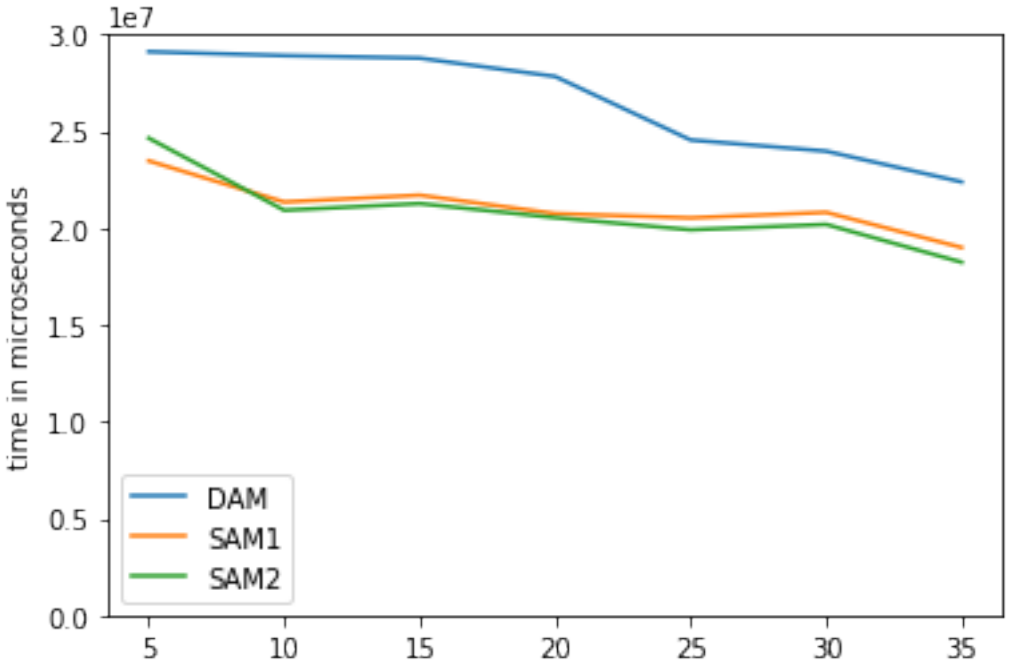

Figure 3: Change in M vs Change in Time taken

- One can see clearly that once again the SAM methods are outperforming DAM.
- With the amount by which the times is varying is analogous to that of the previous graph, there is another property also to spot on.

- With more the number of threads, the work is balanced and the load is reduced on each thread thus results in faster computation in lesser time.
- This property can be clearly seen in the graph, with increase in number of threads, the time taken is decreasing by a significant amount.
- Coming to the SAM2 and SAM1 methods, in the beginning SAM1 is performing better than the SAM2 and also at a later point both the methods are performing as same and in the end, SAM2 is performing better than SAM1.
- This can be accounted as for less number of threads, the overhead is increasing for SAM2 and as number of threads is increasing, the overhead is balanced with respect to the computation.

Note: The observations recorded are the average of the observations which were run over 50 times using a shell script to avoid any type of inference and uncertainty from one run.

LaTeX generated document

THE END