Identifying Prime Numbers

Aaryan, CO21BTECH11001

- **Program Design:**

1. In both of the files, m threads are created after reading the input. Running the program will generate two files, one containing the prime numbers and other containing the time taken.

2. A function named `isPrime` is used to check if a number is a prime number. This function takes $O(\sqrt{n})$ time to find if a number n is a prime number.

3. In DAM, a global variable named `counter` is initialized to 3. A thread accesses the value of `counter` and checks if it is prime or not. Also, the value of `counter` is increased by 2 every time (even numbers are avoided). Mutual exclusion is ensured by using a binary semaphore named `counter_mutex`.

   If the value of `counter` becomes more than N, then the thread exits the loop and terminates.

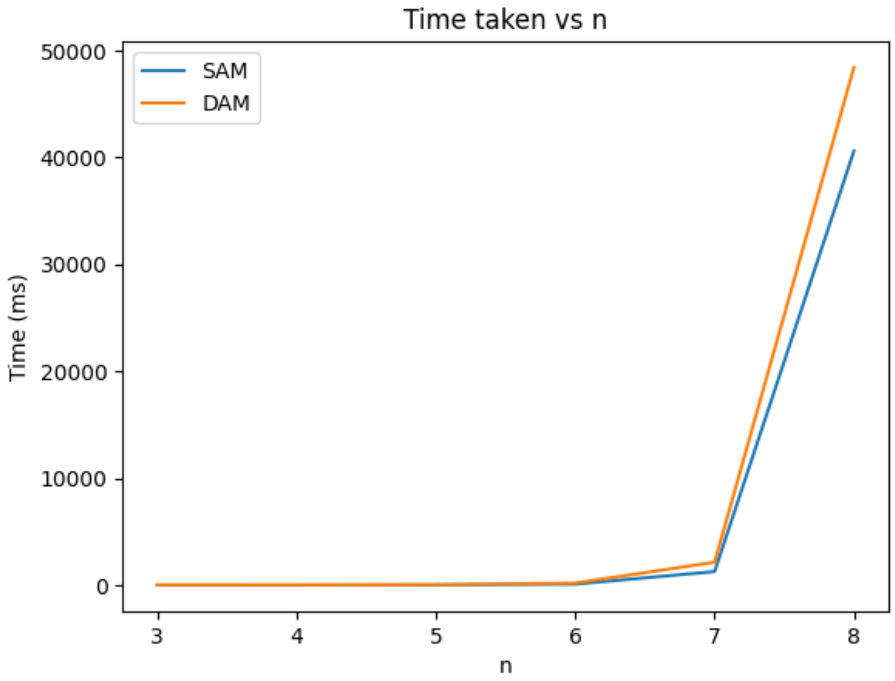
4. In SAM, a thread with thread id equal to i will check the numbers $[2i + 1,\ (2i + 1) + 2m,\ (2i + 1) + 4m,\ \ldots]$. For example if m is 5 and thread id is 0, then the thread will check the numbers $[1, 11, 21, 31, \ldots]$. In this way, all the odd numbers are checked by all the threads cumulatively.

5. In both of the codes, if a number turns out to be prime, it is pushed into an array named `primes`, which contains the number 2 beforehand. Mutual exclusion is ensured by using a binary semaphore named `primes_mutex`.

6. At the end of the code, the array `primes` is sorted and the numbers are written in an output file.

7. Time taken since the creation of threads till their completion is printed in another output file.

- **Time vs Size graph:**

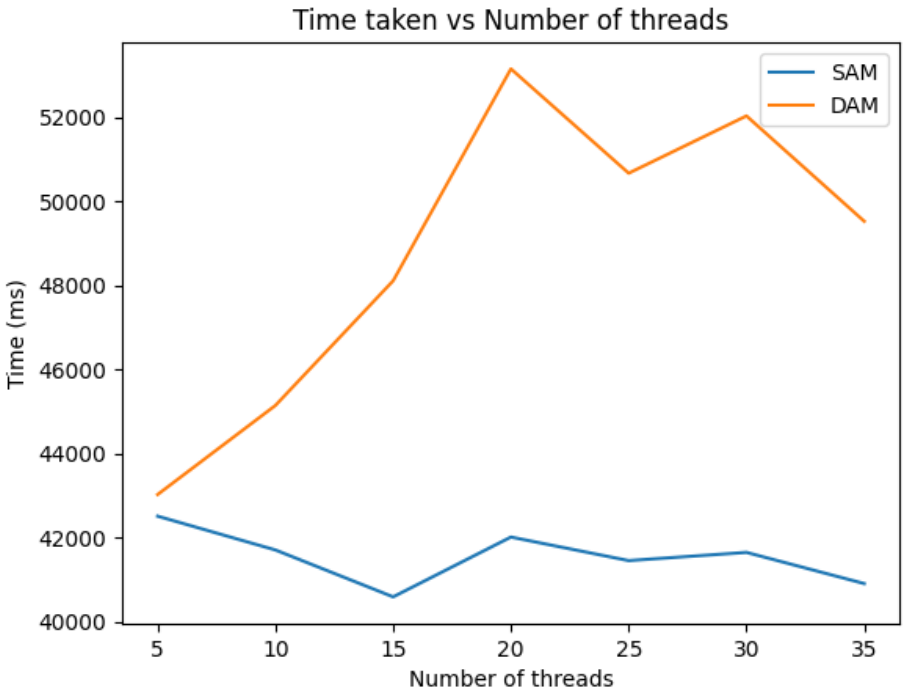


Here are the observations from the above figure:

1. Time taken to find all the prime numbers in the range of 1 to N increases when N is increased, keeping m constant.

2. Time taken by DAM is more than SAM when N and m are kept constant. The reason is that in DAM, threads spend more time waiting to access the value of the variable `counter`, whereas in SAM, there is no such waiting time. Waiting for accessing the `primes` array is common for both of the methods.

- **Time vs Number of threads:**



Here are the observations from the above figure:

1. My machine contains 4 cores. Therefore, making more than 8 threads will not increase the efficiency of the program and may even decrease it.

2. In DAM, since the threads are spending more time just waiting to access the value of the variable counter, the overhead of this waiting time increases as we increase the number of threads. Therefore, we can see a significant increase in time taken when the number of threads increases from 5 to 10, 10 to 15 and 15 to 20.

3. In SAM, time taken decreases slightly when the number of threads are increased from 5 to 10 and 10 to 15. This is because the threads are working independently on the numbers and waiting only when they have to push a prime number into the array.

4. The time taken doesn't change significantly after the number of threads are more than 15 because of the management overhead on the machine.