Assignment 1 Report - CS5300 Comparing Different Parallel Implementations for Identifying Prime Numbers

Thupten Dukpa - ES20BTECH11029

DAM1 Implementation

In DAM1 or Dynamic Allocation Method-1, to find all the prime numbers less than $10^n = N$, there is a global counter and each thread takes the value of the counter then increments it and checks if the value is prime or not. It uses the standard *thread* library for multi-threading.

The *Counter* class stores a variable *value* which is shared among all the threads. The *getAndIncrement*() function returns the value and then increments it. To ensure that at a time only one thread accesses and increments the *value* variable, i.e, there is no race condition, there is a *mutex* lock and *lock_guard* which acquires the lock and then releases it when it goes out of scope, i.e, when the value has been returned and incremented by one thread only at a time.

The *checkPrime*() function simply takes in a value as a parameter, returns true if it is prime else it returns false.

The inputs n and m are read from the 'inp-params.txt' file. An object of *Counter* class called *counter* is initialized. Then m threads are created and each thread executes the *thread_function* method. Each thread repeatedly gets a number using the *getAndIncrement*() function and checks whether it is prime until it reaches the limit N. *lock_guard* is also used when writing the output so that only one thread writes to the file at a time.

Lastly, the execution time is calculated(in microseconds) and all the prime numbers are printed in 'Primes-DAM1.txt' file.

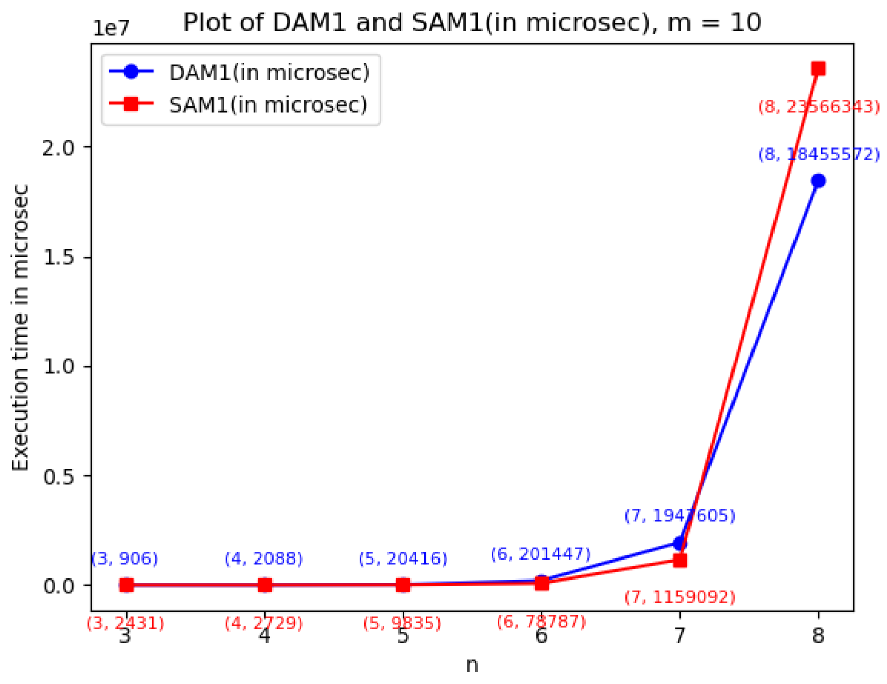
SAM1 Implementation

In SAM1 or Static Allocation Method-1, each thread is preassigned some numbers to check for prime. Numbers 1, m + 1, 2 m + 1, ... will be assigned to thread 0;
Numbers 2, m + 2, 2 m + 2, ... will be assigned to thread 1 and so on.

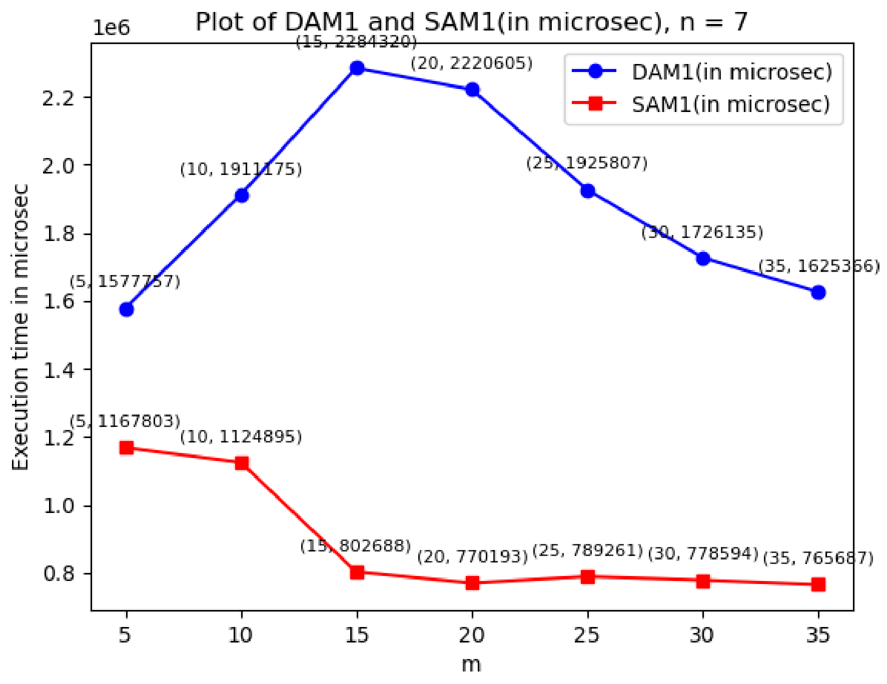
Similar to DAM1, the *checkPrime*() function takes a value as a parameter and returns true if it is a prime else it returns false. The inputs n and m are read from the 'inp-params.txt' file. m threads are created and each thread executes the *thread_function* method. For each thread, the numbers for which it needs to check for prime are found out using the thread number. Thread number is passed from the *main*() function. For example if there are 5 threads: thread 0 checks 1, 6, 11 and so on; thread 1 checks 2, 7, 12 and so on; thread 2 checks 3, 8 and so on.... The *lock_guard* is used to write output in a thread-safe manner.

Lastly, the execution time is calculated(in microseconds) and all the prime numbers are printed in 'Primes-SAM1.txt' file.

Time(in microseconds) vs Size, N, $m = 10$



Time(in microseconds) vs Number of threads, m, $n = 7$


Plot of DAM1 and SAM1(in microsec), n = 7
1e6
DAM1(in microsec)
SAM1(in microsec)
Execution time in microsec
m
(5, 1577757)
(10, 1911175)
(15, 2284320)
(20, 2220605)
(25, 1925807)
(30, 1726135)
(35, 1625366)
(5, 1167803)
(10, 1124895)
(15, 802688)
(20, 770193)
(25, 789261)
(30, 778594)
(35, 765687)