CS5300 - Parallel & Concurrent Programming

J SAI NISHITH
CS18BTECH11018

REPORT : Comparing Different Parallel Implementations for Identifying Prime Numbers

Two methods have been used to compute parallel numbers until a given power of ten, namely dynamic allocation method (DAM) and static allocation method-one (SAM).

Dynamic allocation method (DAM) :
In this method, each thread gets a dynamically determined number of numbers to test. We just create ‘m’ number of threads and call the function ‘**primePrintDAM**’ on each of them.

Each thread lies in the critical section in some part of the function where it takes the number that it needs to compute and increments the total number of numbers computed till that point. The function ends once the required limit is achieved.

I have used a vector to store all primes calculated in that process instead of printing directly to avoid the inclusion of I/O time during the actual computation part.

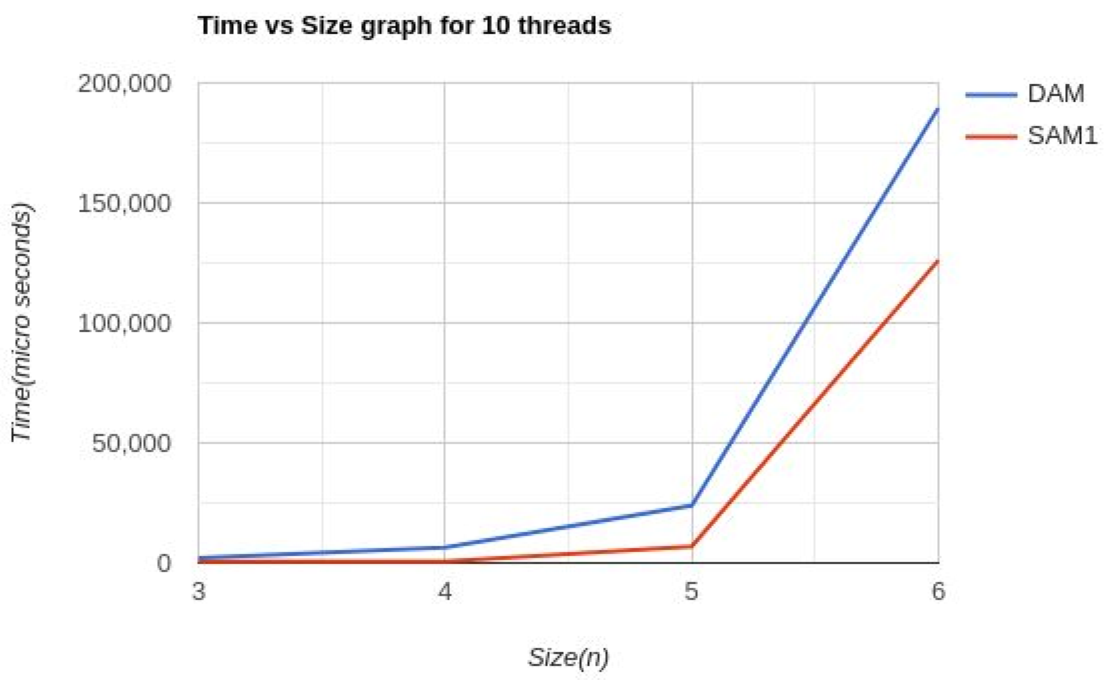
Static allocation method-one (SAM1) :
In this method, each thread receives some pre-assigned set of numbers to be computed.
First thread receives the numbers 1, m+1, 2*m+1, …
iTh thread receives the numbers i, m+i, 2*m+i, …
Where m is the number of threads.

Each thread is created and passed in to function ‘**primePrintSAM**’ with thread number as argument i.e. for the 1st thread we pass 1 as the argument.

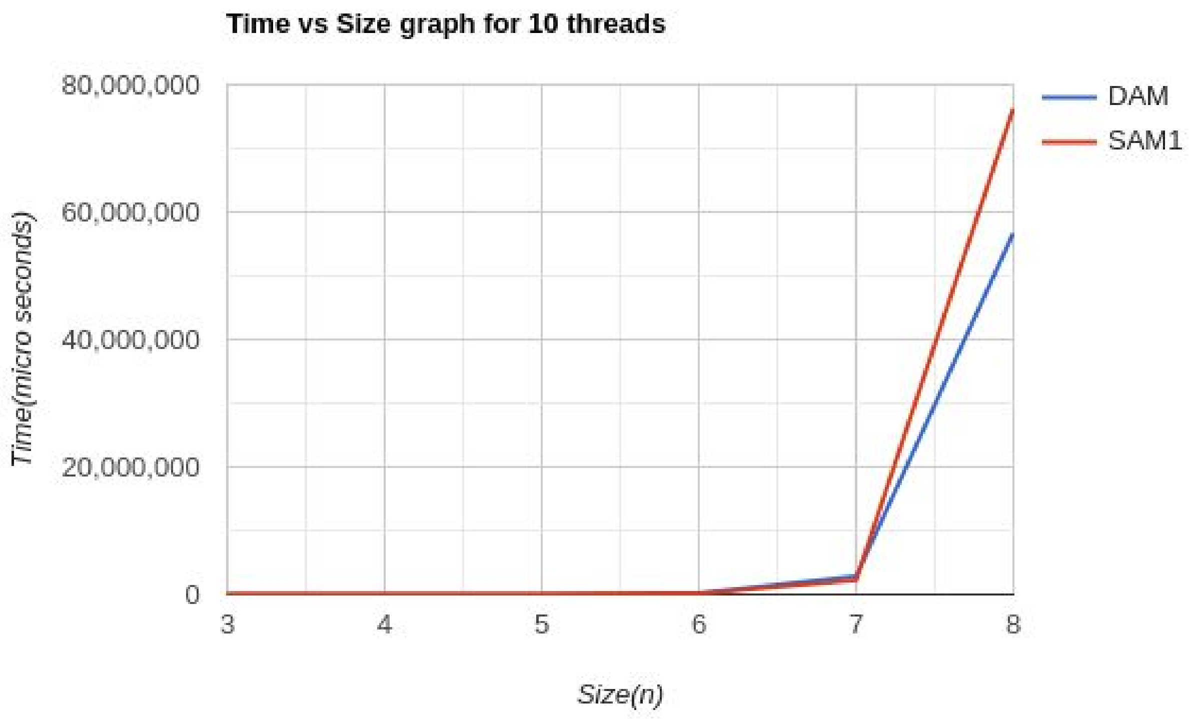
We add ‘m’ to a number after it is computed and we iterate until we compute all numbers till the given range. I have used a vector to store all primes calculated in that process instead of printing directly to avoid the inclusion of I/O time during the actual computation part.

DAM vs SAM1 Comparison :

1.Time vs Size:



Graph1



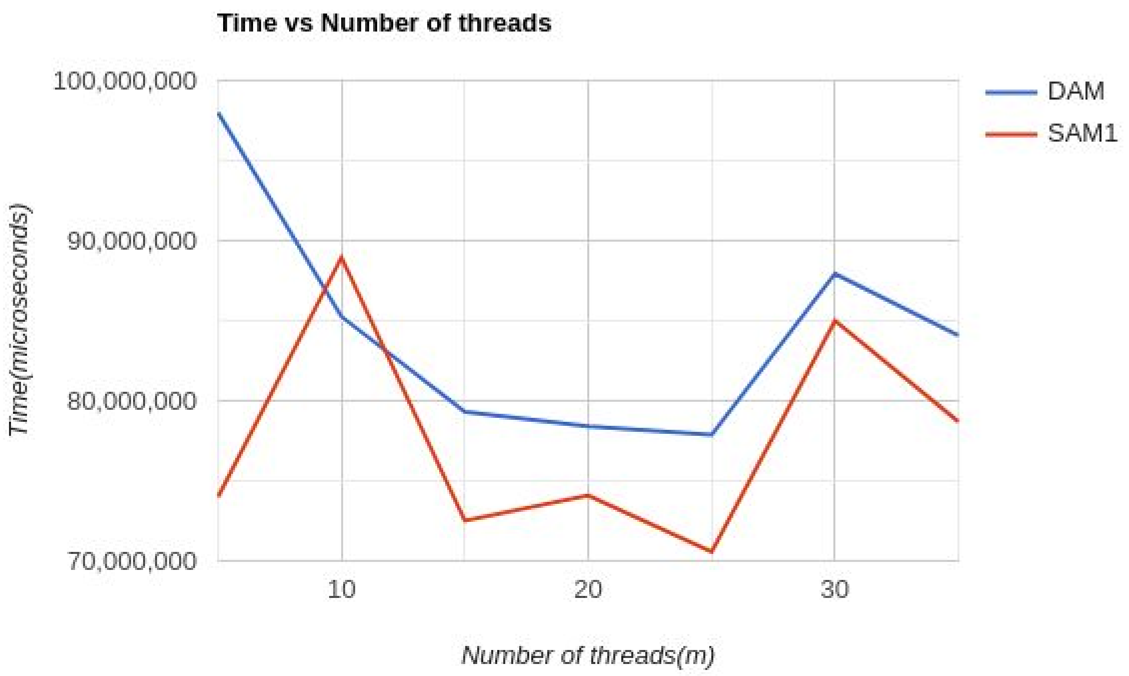
Graph 2

SAM1 is faster in general because there is no shared variable in the case of SAM1 as compared to DAM. So threads need to wait in the DAM method when another thread is in the critical section.

But in case of SAM1, each thread has some set of values which are to be calculated. In graph 1, n is from 3-6, we can see that SAM is always after but slowly as n goes to 8, SAM is slower because when m=10;
In SAM
1st thread computes 1,11,21,31,...
2nd thread computes 2,12,22,32,42,...

We can clearly see that thread 1 does more computation than thread 2 because all numbers in thread 2 are just even numbers. So this is not very optimal because one thread does much more work than another. Here since N=10^8 and the number of threads is just 10, time taken in SAM1 is a lot more. But as the number of threads increases, this effect of even number of threads decreases as there are a lot more threads which compute for odd numbers. So for smaller values of n(3-6) SAM1 was faster but as N increases and the number of threads is fixed, time increases because of bad distribution of numbers in case of even number of threads.

2.Time vs Number of Threads:(N=10^8)



As we can see SAM1 is in general faster than DAM but at m=10, we can see that it is a bit slow compared to DAM as explained above. SAM is faster than DAM in general because there is no waiting for a shared variable. But if the number of threads(m) is even, SAM1 performance degrades because of poor distribution of numbers to threads. Some threads get only odd numbers which take more computation time, then those threads which have only even numbers. So we see that at m=10,20,30 there is a slight increase in time of SAM1. Also increasing the number of threads doesn't always result in an increase in speed because the threads might have to wait more time for shared variables. The increase in time from m=25 to 30, can be accounted to this reason.