High Performance Computing Report

Author 1 (Kunal Jani)
Author 2 (Pratik Ghosh)

Dhirubhai Ambani Institute of Information and Communication Technology
201601444@daiict.ac.in
201721010@daiict.ac.in

Question 1:

Implementation Details

Brief and clear description about the Serial implementation

the serial implementation of the trapezoidal method for the calculation of the value of pi, initially taking the sum to be zero and running an iteration from zero to the total number of steps, the value of x has been incremented by 0.5 and then it is divided by the total number of steps. This value of x is used to calculate the sum of the series which has been initially set to zero. Finally the value of pi is obtained by multiplying the obtained sum by the number of steps.

Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

The algorithm has been split into four threads. Four different textbfs of the array are assigned to four different threads. The sum of the different textbfs are calculated in these different threads and are stored in an array of 4 elements. The final value of pi can be calculated by adding the adding the values in the array of 4 elements.

Complexity and Analysis Related

Complexity of serial code No of operations = n
Time taken = n
Time complexity = O(n)

Complexity of parallel code (split as needed into work, step, etc.)

No of operations = n
Time taken = n/p
Time complexity = O(n)

Cost of Parallel Algorithm

Cost of the algorithm = O(n).

Theoretical Speedup (using asymptotic analysis, etc.)

Serial execution time=pt
Parallel execution time=t
$Speedup = \frac{Serialtime}{Paralleltime} = p$

Number of memory accesses

Serial algorithm:

No of memory accesses outside loop=5
No of memory accesses inside loop=9
Total memory accesses=9n+5

Parallel algorithm:

No of memory accesses outside loop=8+7p
No of memory accesses inside loop=5p+14n
Total memory accesses=14n+12p+8

Number of computations

Serial algorithm:

No of computations outside loop=0
No of computations inside loop=8n
Total no of computations=8n

Parallel algorithm:

No of computations outside loop=1+p
No of computations inside loop=8n
Total no of computations=8n+p+1

Curve Based Analysis

Time Curve related analysis (as no. of processor increases)

The execution time increases with the greater number of instructions to be exe-

cuted. The execution time will increase with greater number of processors since a large amount of work is divided among a lot of processors. For problem sizes less than 10000, the cost due to execution of instructions will dominate because the problem size is not so large that the effect of division of a large problem into smaller parts will overpower the effect of the greater number of computations required in the parallel algorithm.For problem sizes around 10000, both the effects of intense computations in the parallel algorithm and the burden of work being greater on the lesser number of threads. For problem sizes greater than 10000, the execution time will increase with an increase in the number of processors and the effect of dividing a large number of problem into equal parts will come into dominance.

Time Curve related analysis (as problem size increases, also for serial)

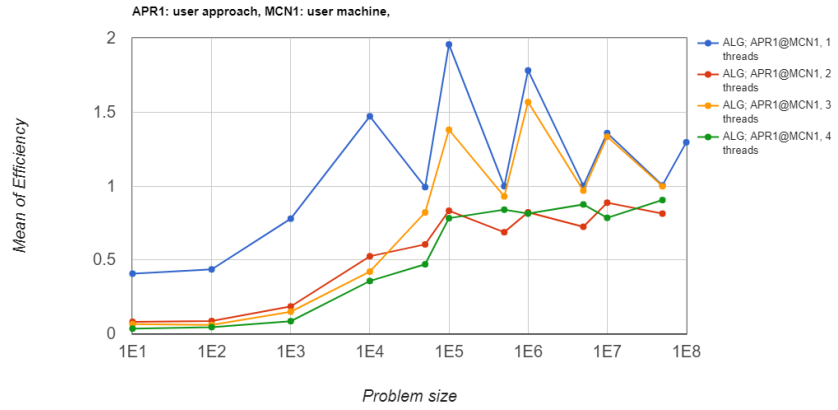
If there is are no threads and the algorithm is purely serial, the execution time will increase the most quickly. As the number of threads keep on increasing, the execution time will increase slowly with an increase in the problem size. This is because for a greater number of threads, there will be a greater number of processors so the division of work will take place thus reducing the execution time significantly.

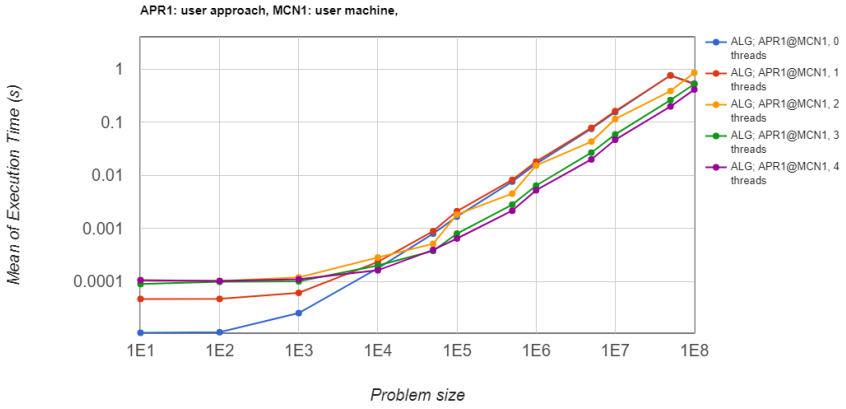
Speedup Curve related analysis (as problem size and no. of processors increase)

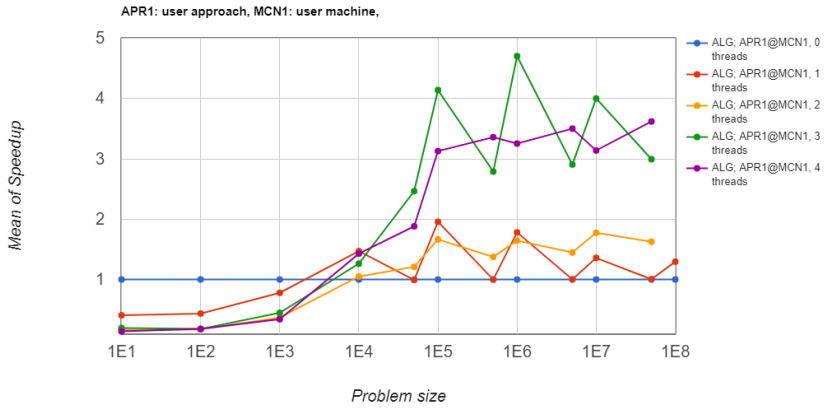
With a low problem size, the greater number of computations per unit time in the parallel algorithm dominates over the effect of dividing the entire task into a number of sub tasks for a low problem size. This the speedup is less than 1 for a lower problem size. With a higher problem size, the increase in the speed due to dividing of a task into a higher number of threads will dominate aver the greater number of computations in the parallel algorithm, thus giving a speedup greater than 1.

Efficiency Curve related analysis

For a lower problem size, the efficiency is the greatest for the serial algorithm as the number of processors used are the least in this algorithm and the speedup is therefore the maximum for this case. As the problem size goes on increasing, the efficiency of the serial algorithm will reduce due to a reduced speedup with an increase in the problem size. For a greater number of threads, the efficiency will slowly increase as the work done is distributed on a greater number of processors and thus the speedup will increase while the number of processors remain constant for a given number of threads.


APR1: user approach, MCN1: user machine,
Mean of Efficiency
2
1.5
1
0.5
0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
1
0.1
0.01
0.001
0.0001
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads



Question 2:

Implementation Details

Brief and clear description about the Serial implementation

The formula which has been used to calculate the value of pi is:

$$\pi = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} \tag{1}$$

itially, the value of π is set to zero. Upon increasing the value of k, the value of π is iteratively added. An if condition is used to check whether the value of $(-1)^k$ is positive or negative. Finally, the result is multiplied by 4 to obtain the value of π. A very large value of the problem size can be assumed to infinity.

Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

Each and every thread number j will calculate the value of the sum of the series by calculating the value of the series from the range $\frac{(j-1)n}{4}$ to $\frac{jn}{4}$. The sum of the four threads are obtained and finding the sun of the values from the four threads can be used to calculate the final sum which is π.

Complexity and Analysis Related

Complexity of serial code

No of operations = n

Time taken = n
Time complexity = O(n)

Complexity of parallel code (split as needed into work, step, etc.)

No of operations = n
Time taken = n/p
Time complexity = O(n)

Cost of Parallel Algorithm

The cost of the algorithm is O(n).

Theoretical Speedup (using asymptotic analysis, etc.)

Speedup=p/1=p

Number of memory accesses

Serial algorithm:
No of memory accesses outside loop=3
No of memory accesses inside loop=11n
Total memory accesses=11n+3
Parallel algorithm:
No of memory accesses outside loop=3
No of memory accesses inside loop=11n
Total memory accesses=11n+3

Number of computations

Serial algorithm:
No of computations outside loop=1
No of computations outside loop=11n
Total no of computations=11n+1
Parallel algorithm:
No of memory accesses outside loop=1
No of memory accesses inside loop=11n
Total memory accesses=11n+1

Curve Based Analysis

Time Curve related analysis (as no. of processor increases)

As the number of processes increases with an increase in the number of threads,

the greater number of computations required in a parallel algorithm will dominate compared to the serial algorithm. For greater problem sizes, the lower execution time due to separation of work into equal slots will come into effect.

Time Curve related analysis (as problem size increases, also for serial)

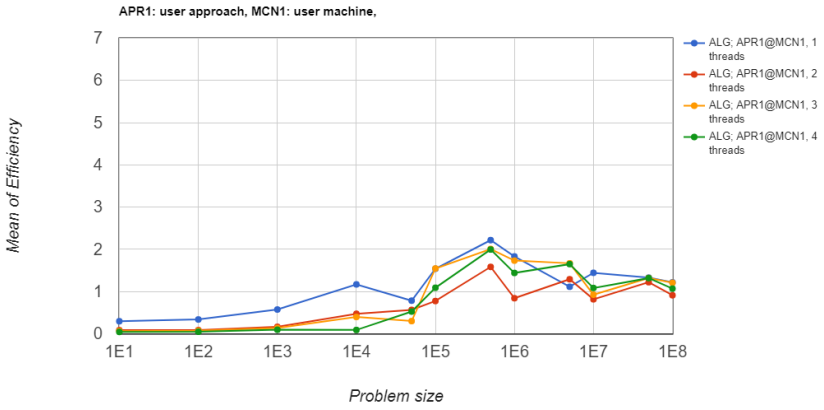
For lower number of threads, the rate of increase of the execution time will be much greater compared to the execution time for higher number of threads because for a greater number of threads, work is distributed on a lager number of processors and thus the execution will be faster.

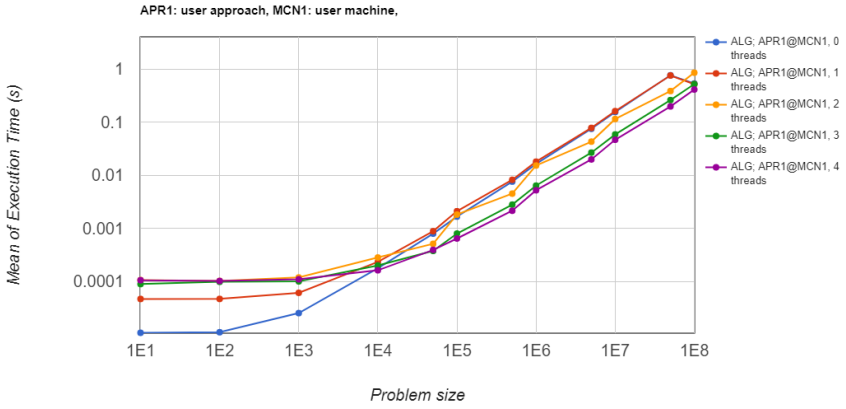
Speedup Curve related analysis (as problem size and no. of processors increase)

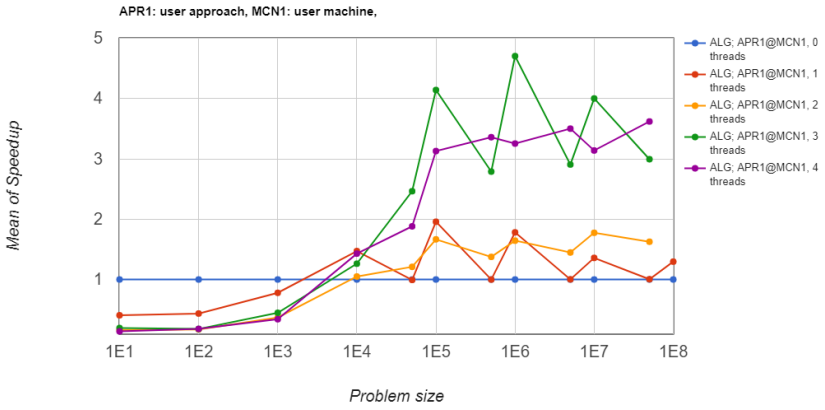
For a smaller process size, the speedup is greater for a smaller number of threads because the number of computations required in a parallel algorithm is much greater compared to the number of computations required in a serial algorithm, nut for a larger problem size, the speedup is larger for a parallel algorithm as the large amount of work id divided equally among a larger number of processors.

Efficiency Curve related analysis

For small values of the problem size, the efficiency is the highest for the serial algorithm as the speedup due to lower number of operations to be performed is good and the number of processors is only one. For greater values of the problem size, efficiency will be greater for a larger number of processors as the speedup is much greater in this case.


APR1: user approach, MCN1: user machine,
Mean of Efficiency
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads
0
1
2
3
4
5
6
7
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8


APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads
0.0001
0.001
0.01
0.1
1
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8



Question 3:

Implementation Details

Brief and clear description about the Serial implementation Two vectors A and B consisting of four elements are taken and the the sum both the elements from the vectors A and B are calculated one by one using an iteration.

Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads) The entire algorithm is distributed into four threads. No iteration is needed to calculate the sum of the values of the four individual elements. Each thread will calculate the sum of each and every individual element.

Complexity and Analysis Related

Complexity of serial code

No of operations = n

Time taken = n

Time complexity = O(n)

Complexity of parallel code (split as needed into work, step, etc.)

No of operations = n
Time taken = 1

Time complexity = O(1)

Cost of Parallel Algorithm

The cost of the algorithm is O(n).
Theoretical Speedup (using asymptotic analysis, etc.) Speedup=p/1=p

Number of memory accesses

Serial algorithm:

No of computations outside loop=3
No of computations outside loop=7n
Total no of computations=7n+3

Parallel algorithm:

No of memory accesses outside loop=3
No of memory accesses inside loop=7p+3
Total memory accesses=7p+3

Number of computations

Serial algorithm:

No of computations outside loop=1
No of computations outside loop=3n
Total no of computations=3n+1

Parallel algorithm:

No of memory accesses outside loop=1
No of memory accesses inside loop=3p
Total memory accesses=3p+1

Curve Based Analysis

Time Curve related analysis (as no. of processor increases)

the case of the addition of two vectors which are represented by arrays, the parallel algorithm will work the worst when there is one thread only as the number of computations required are much greater compared to a serial algorithm.

Morevoer all that intense work is done only on a single processor.

Time Curve related analysis (as problem size increases, also for serial)

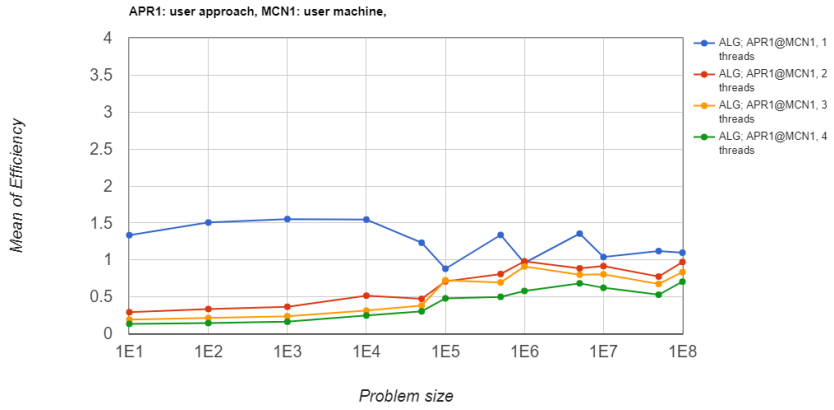
With an increase in the problem size, the execution time will initially not change much because both the effects of higher number of computations due to the implementation of the parallel algorithm and the time reduced due to division of work among processors will cancel out each other, but with a greater problem size, the execution time will increase as a lot of work will have to be done.

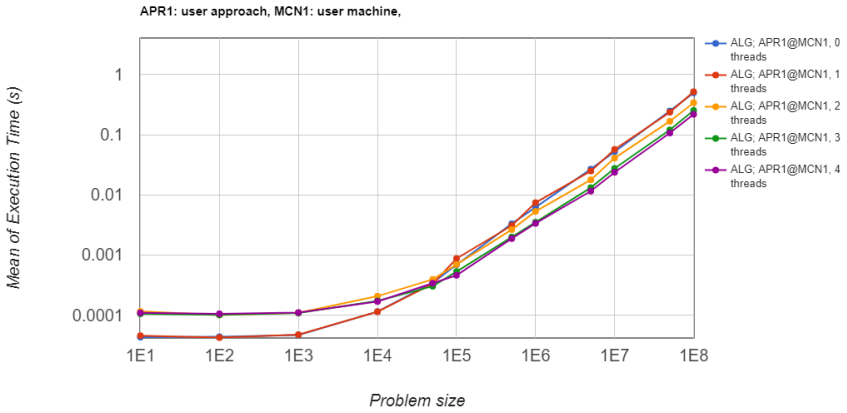
Speedup Curve related analysis (as problem size and no. of processors increase)

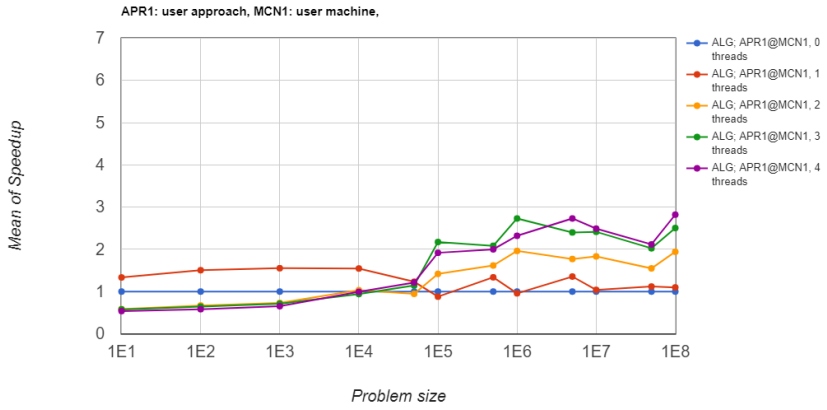
For a lower number of threads, for smaller number of processors, the speedup will be greater, but for larger number of threads, the speedup will be much less.

Efficiency Curve related analysis

The efficiency of the algorithm will be the greatest for a lower problem size because the speedup is the highest, and the number of processors used are less, but for larger problem size, the efficiency of the algorithm will increase for a larger number of threads due to greater speedup by distribution of work and decrease for the serial algorithm as the speedup will reduce because all the work is done on only one processor.


APR1: user approach, MCN1: user machine,
Mean of Efficiency
4
3.5
3
2.5
2
1.5
1
0.5
0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
1
0.1
0.01
0.001
0.0001
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads



Question 4:

Implementation Details

Brief and clear description about the Serial implementation Two vectors A and B consisting of four elements are taken and the the dot product of both the elements from the vectors A and B are calculated one by one using an iteration.

Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

The entire algorithm is distributed into four threads. No iteration is needed to calculate the sum of the values of the four individual elements. Each thread will calculate the sum of each and every individual element.

Complexity and Analysis Related

Complexity of serial code

No of operations = n
Time taken = n
Time complexity = O(n)

Complexity of parallel code (split as needed into work, step, etc.)

No of operations = n
Time taken = 1

Time complexity = O(1)

Cost of Parallel Algorithm

The cost of the algorithm is O(n).

Theoretical Speedup (using asymptotic analysis, etc.)

eedup=p/1=p

Number of memory accesses

Serial algorithm:

No of memory accesses outside loop=0
No of memory accesses inside loop=3n
Total no of computations=3n

Parallel algorithm:

No of memory accesses outside loop=0
No of memory accesses inside loop=3p
Total memory accesses=3p

Number of computations

Serial algorithm:

No of computations outside loop=0
No of computations outside loop=4n
Total no of computations=4n

Parallel algorithm:

No of memory accesses outside loop=0
No of memory accesses inside loop=4p
Total memory accesses=4p

Curve Based Analysis

Time Curve related analysis (as no. of processor increases)

For the case of calculation of the dot product, execution time lower for the serial algorithm in the case of a small problem size due to the domination of high number of computational operations and the execution time is higher as there are a lower number o processors due to equal work distribution among processors.

Time Curve related analysis (as problem size increases, also for serial)

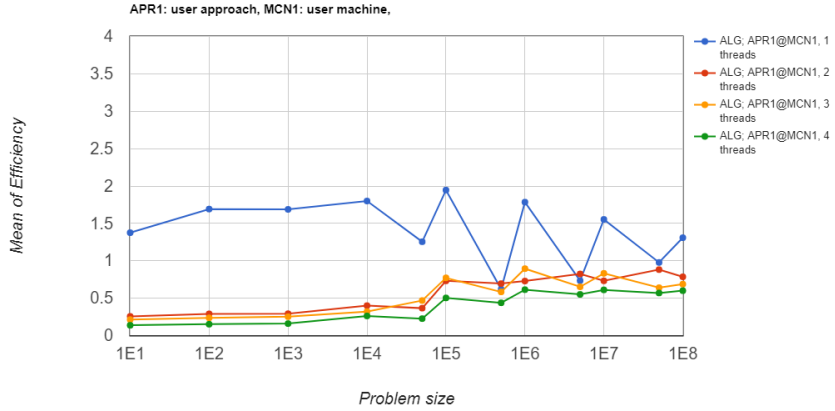
For the case of calculation of the dot product, execution time is almost constant due to the nullifying effects of the large amount of time taken in a parallel algorithm and the effect of work distribution in a parallel algorithm for a small problem size. For a large problem size, the execution time will increase due to a sheer increase in the number of operations required.

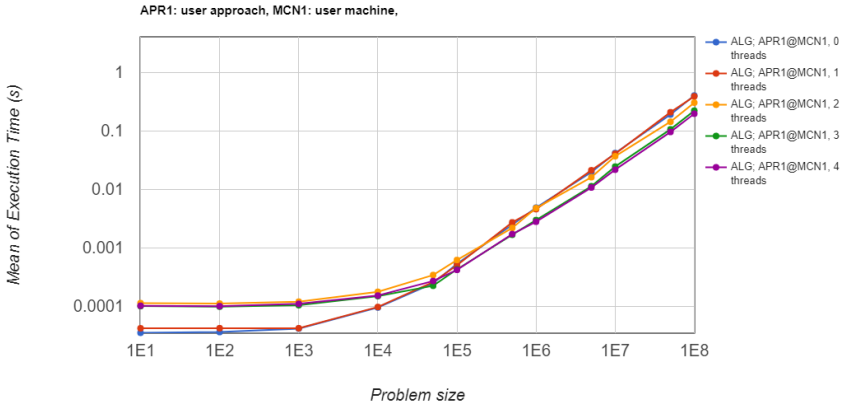
Speedup Curve related analysis (as problem size and no. of processors increase)

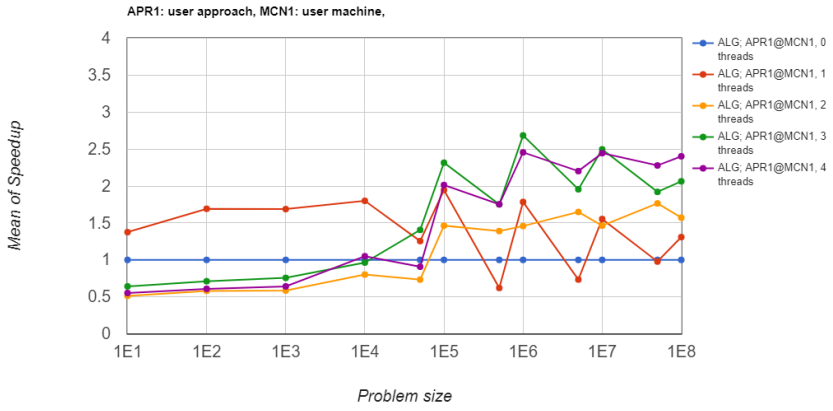
The speedup is the greatest for a small number of processors when the problem size is small due to greater number of operations required in a computations in a parallel algorithm and it will be greater for a large number of processors when the problem size is larger.

Efficiency Curve related analysis

The efficiency of the algorithm will be the greatest for a lower problem size because the speedup is the highest, and a lower number of processors are used, but for larger problem size, the efficiency of the algorithm will be much greater for a larger number of threads due to greater speedup by distribution of work and decrease for the serial algorithm as the speedup will reduce because all the work is done on only one processor.


APR1: user approach, MCN1: user machine,
Mean of Efficiency
4
3.5
3
2.5
2
1.5
1
0.5
0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
1
0.1
0.01
0.001
0.0001
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


APR1: user approach, MCN1: user machine,
Mean of Speedup
4
3.5
3
2.5
2
1.5
1
0.5
0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads