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

Three matrices A, B and R are stored in memory as two dimensional arrays in memory. A and B are the input matrices and R is the output matrices. All the indices in R are initially assigned a value of 0.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & a_{d3} & \dots & a_{dn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{d1} & b_{d2} & b_{d3} & \dots & b_{dn} \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

For element in r, we sum up all the horizontal elements of the A matrix and the vertical elements of the B matrix.

$r_{ij} = \Sigma a_{ik} b_{kj}$

Thus for the serial implementation, the first loop is created for running from 0 to n, where n is the matrix size, then one inner loop is created again for 0 to n and again a third loop is created for 0 to n to calculate the sum for each and

every element.

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

Using the 'pragma omp parallel for' command in programming, the equal division of work is possible in any of the three iterative loops that have been created. The global and private variables are supposed to be declared by the user.

Complexity and Analysis Related

Complexity of serial code No of operations = n^3
Time taken = n^3
Time complexity = O(n^3)

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

No of operations = n^3
Time taken = n^3/p
Time complexity = O(n^3)

Cost of Parallel Algorithm

Cost of the algorithm = O(n^3).

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=3
No of memory accesses inside loop=$4n^3 + 7n^2 + 7n + 8$
Total memory accesses=$4n^3 + 7n^2 + 7n + 11$

Parallel algorithm:

No of memory accesses outside loop=3
No of memory accesses inside loop=$4n^3 + 7n^2 + 7n + 8$
Total memory accesses=$4n^3 + 7n^2 + 7n + 11$

Number of computations

Serial algorithm:

No of computations outside loop=0
No of computations inside loop=$2n^3$
Total no of computations=$2n^3$

Parallel algorithm:

No of computations outside loop=0
No of computations inside loop=$2n^3$
Total no of computations=$2n^3$

Curve Based Analysis

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

For a lower number of processors for the outermost loop, the execution time will be much higher because the division of work is lower. The time spent in the creation ad finishing of threads is minimal and so the proper work division to reduce the execution time will dominate over the time spent over the creation and finishing of threads even when the number of threads increase. Total threads created and finished are only p.

the case of the middle loop parallelization, the time spent in the creation and destruction of threads is much higher because for each and every value for i from 0 to n, threads have to be created as well as destroyed. The number of times thread will be created and finished is p*p. However, the execution time will still be greater for large problem sizes because the division of work for large problem sizes will still provide a compensating effect.

the final case of innermost loop parallelization, the innermost loop will be parallelized. For this case the thread will be created as well as finished p*p*p times and thus even for larger problem sizes when the algorithm is supposed to work well, the time spent in creating and finishing threads is much greater compared to the time saved in the division of work among the threads, so the overall execution time of the algorithm will increase.

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

As the problem size will increase, execution time will initially remain the same because for a lower problem size, the time taken in the number of computations is much greater for a small problem size, but as the problem size is larger, a larger amount of work will get divided among the threads thus giving the same execution time. As the problem size further increases, the execution time will increase because the time taken for computation is much greater for a larger problem size.

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

For the outermost loop parallelization, the speedup will be the maximum because the time which had been reduced in the division of work among the threads will be much more effective in the time lost in the creation of threads, so there is an overall increase in the speedup.

For the middle loop parallelization, the speedup will reduce because the time that has been spent in the creation of threads will increase because p*p threads have been created.

There will be a speedup which is less than in the case of the innermost loop parallelization because too much time is spent in the creation and finishing the threads p*p*p times. Thus in this case, the division of work among threads is ineffective.

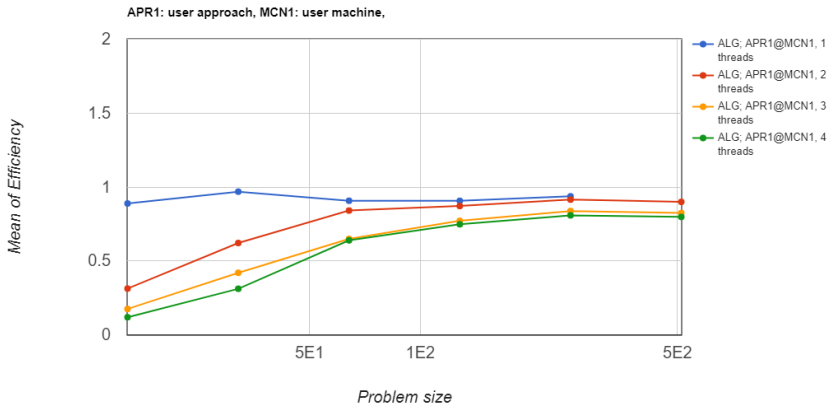
Efficiency Curve related analysis

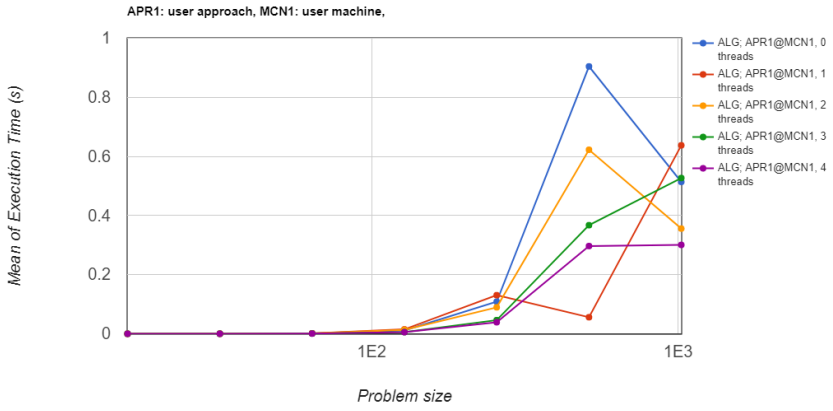
For the parallelization of the outermost loop, the efficiency will increase with an increase in the problem size as the speedup will be greater due to division of work among the threads. However for a greater number of threads and a fixed problem size, the efficiency will reduce due to an increase in the number of processors.

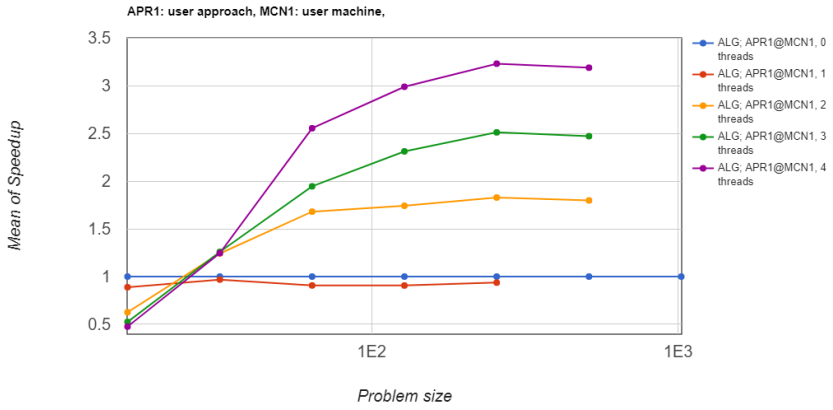
For the parallelization of the middle loop, the efficiency will not be as great as the case for the outermost loop parallelization. This is because of a reduction in the speedup. Similarly like the above case the efficiency of the parallel algorithm is much less when the number of threads are greater due to the increase in the number of processors running concurrently.

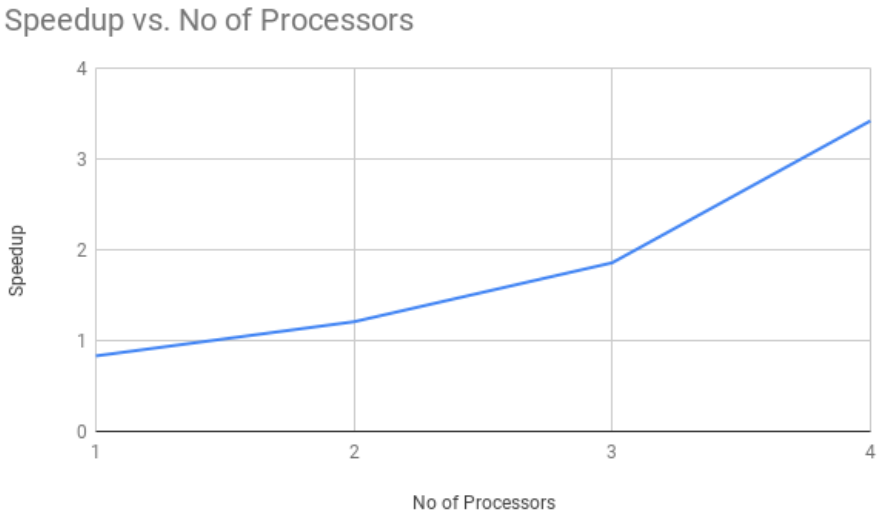
the case of the innermost loop, the efficiency is the least due to the least speedup due to too much time spent in creating and finishing threads. Similarly, as discussed in the above 2 cases, the efficiency is much lower for a larger number of threads as the increase in speedup is not much compared to the increase in number of threads.

Analysis when serial code is compared with the outermost loop parallelization.

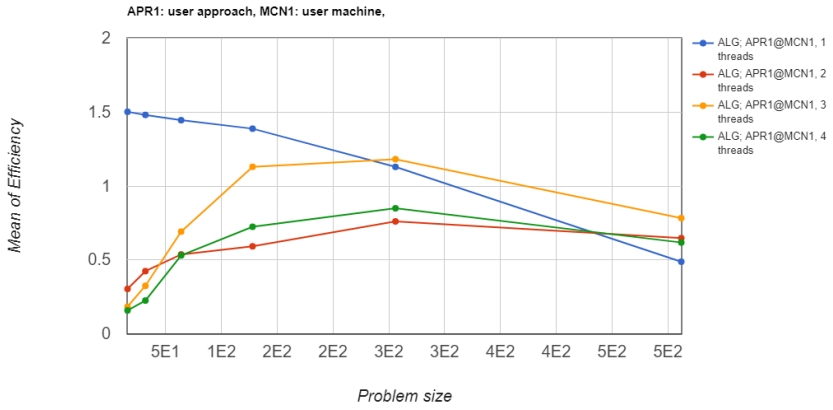


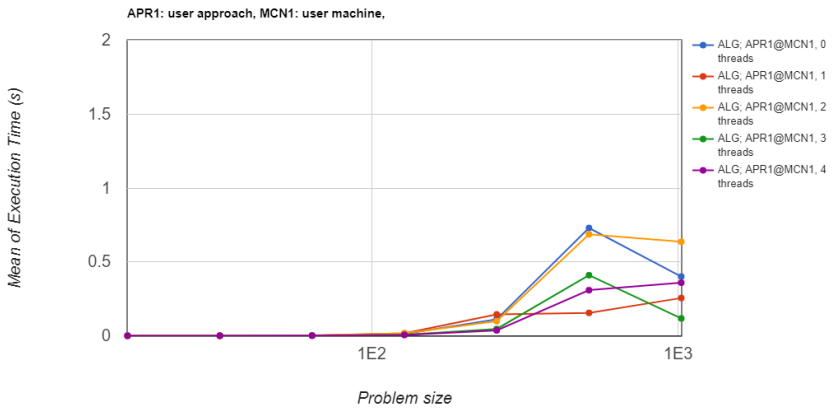


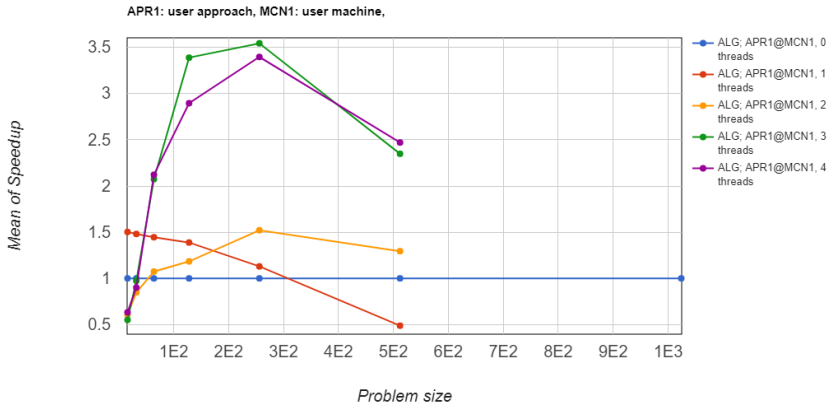


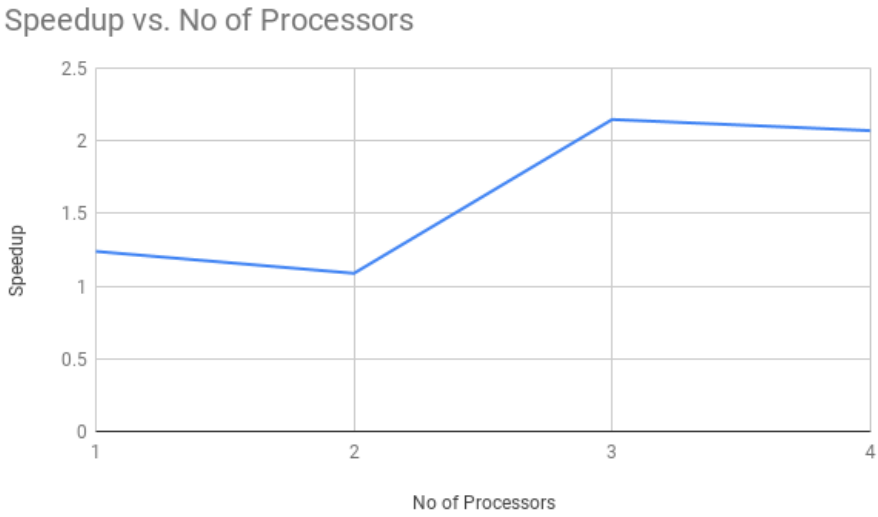


Analysis when serial code is compared with the middle loop parallelization.

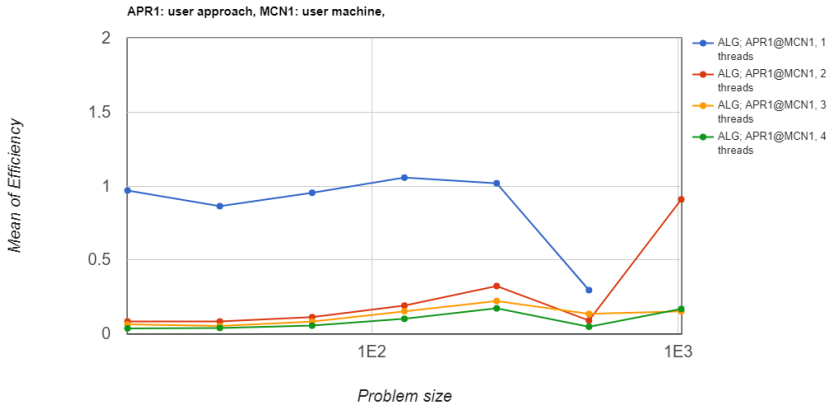

APR1: user approach, MCN1: user machine,
Mean of Efficiency
2
1.5
1
0.5
0
5E1
1E2
2E2
2E2
3E2
3E2
4E2
4E2
5E2
5E2
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads

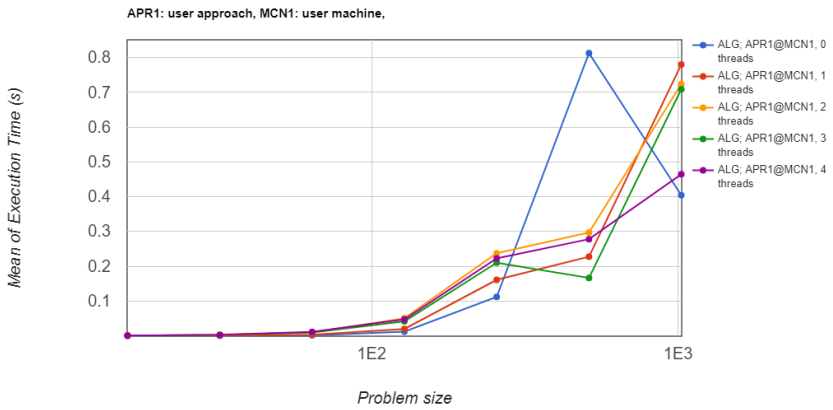

APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
2
1.5
1
0.5
0
1E2
1E3
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads

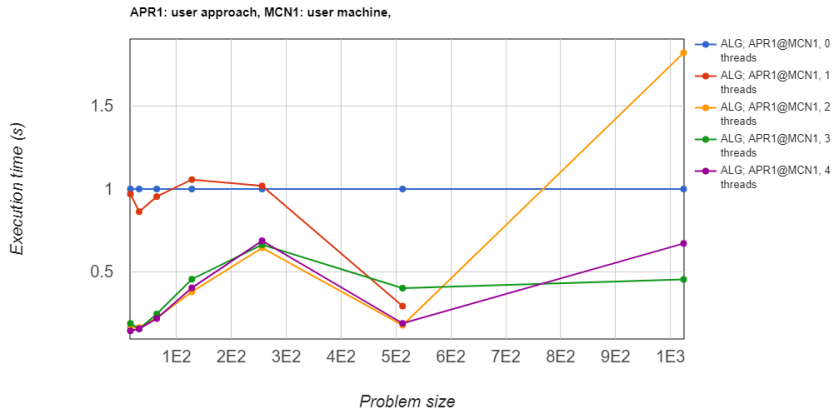


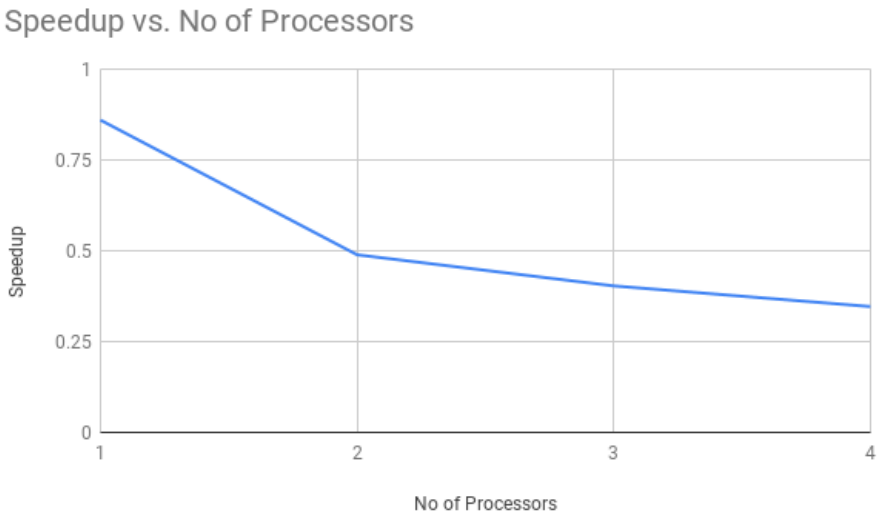


Analysis when serial code is compared with the innermost loop parallelization.


APR1: user approach, MCN1: user machine,
Mean of Efficiency
2
1.5
1
0.5
0
1E2
1E3
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)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1E2
1E3
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

Three matrices A, B and R are divided into blocks of equal size. Each block is treated as an individual matrix. Each block is assigned an index number. The blocks having the same index number i.e. the same row number and column number are multiplied with each other and the addition of the blocks is taken out in the same way it is done for regular element addition.

For element in r, we sum up all the horizontal elements of the A matrix and the vertical elements of the B matrix.

$r_{ij} = \Sigma a_{ik} b_{kj}$

Thus for the serial implementation, the first loop is created for running from 0 to n, where n is the matrix size, then one inner loop is created again for 0 to n and again a third loop is created for 0 to n to calculate the sum for each and every element.

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

Using the 'pragma omp parallel for' command in programming, the equal division of work is possible in any of the three iterative loops that have been created. The global and private variables are supposed to be declared by the user.

When the serial code is to be compared with the block multiplication:

Number of blocks=Problem size/No of threads

Thus it will be possible to analyze the effects of a block implementation of matrix multiplication replacing the serial implementation of matrix multiplication.

Complexity and Analysis Related

Complexity of serial code No of operations = n^3
Time taken = n^3
Time complexity = O(n^3)

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

No of operations = n^3
Time taken = n^3/p
Time complexity = O(n^3)

Cost of Parallel Algorithm

Cost of the algorithm = O(n^3).

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=3
No of memory accesses inside loop=$5n^5 + 7n^4 + 4n^3 + 11n^2 + 11n + 8$
Total memory accesses=$5n^5 + 7n^4 + 4n^3 + 11n^2 + 11n + 11$

Parallel algorithm:

No of memory accesses outside loop=3
No of memory accesses inside loop=$5n^5 + 7n^4 + 4n^3 + 11n^2 + 11n + 8$
Total memory accesses=$5n^5 + 7n^4 + 4n^3 + 11n^2 + 11n + 11$

Number of computations

Serial algorithm:

No of computations outside loop=0
No of computations inside loop=$4 + 4n + 2n^2 + 5n^3 + 4n^4 + 4n^5$
Total no of computations=$4 + 4n + 2n^2 + 5n^3 + 4n^4 + 4n^5$

Parallel algorithm:

No of computations outside loop=0
No of computations inside loop=$4 + 4n + 2n^2 + 5n^3 + 4n^4 + 4n^5$
Total no of computations=$4 + 4n + 2n^2 + 5n^3 + 4n^4 + 4n^5$

Curve Based Analysis

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

The execution time increases with the greater number of instructions to be executed. 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.

When the serial code is compared with the block implementation of matrix multiplication, the execution time will not reduce as quickly as the reduction of execution time in the case of parallelizing the outermost loop. However the execution time be greater if a smaller number of blocks are created but will reduce drastically if a larger number of blocks are created because it is easier to compute the product of smaller blocks compared to larger blocks. This is because the time taken to access memory from cache is much greater compared to the time taken to access memory from RAM. When the data is divided into blocks, each and every cache is fitted in the cache which is of a limited size. For the serial algorithm, one huge block of data wouldn't be fitting inside the cache and thus time taken to access this data would be much greater.

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

When the serial algorithm is compared with the parallel algorithm, is can be observed that the execution time for the parallel algorithm will initially remain same for a lower value of the problem size due to the effects of time spent in thread creation and the effect of increasing time reduced due to work division. After the problem size reaches a creation value, the time spent in creation of the threads will remain same and the division of blocks will take place, but the computations for a larger problem size would be much greater. When the serial and parallel implementation of the block algorithm takes place, the execution time will increase with the increase in problem size for very large values of the problem size due to the sheer greater number of computations and the execution time will remain almost constant for a lower problem size due to the interplay between the greater number of computations in threads and the division of work.

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

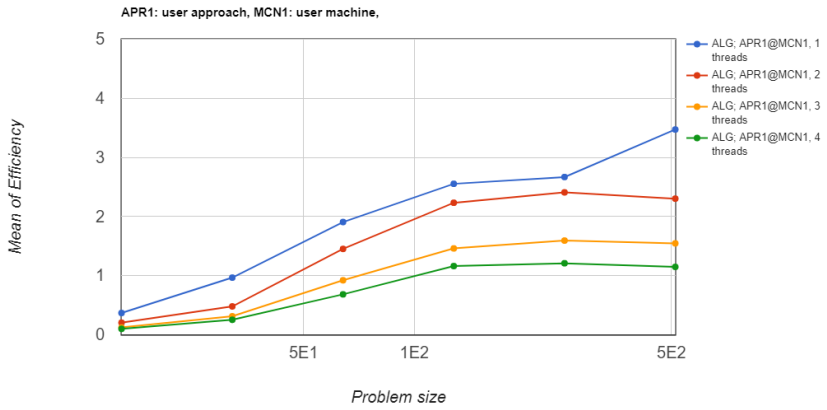
With a low problem size, the greater number of computations per unit time in the parallel algorithm dominates over the effect of dividing the entire task into a number of sub tasks for a low problem size. This the speedup is less than 1 for a lower problem size. With a higher problem size, the increase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

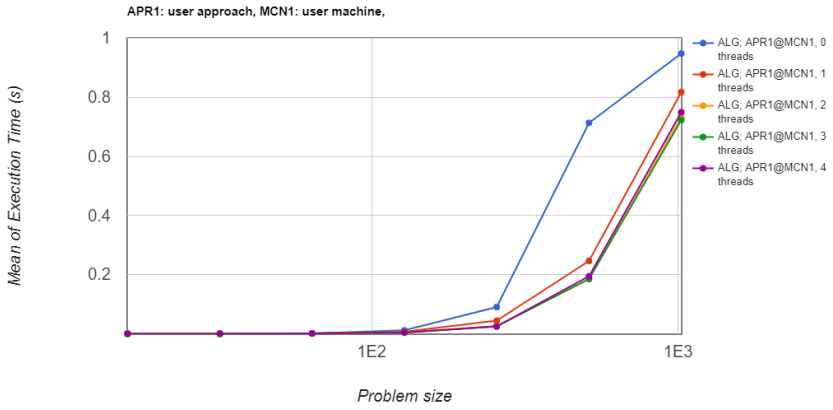
When the serial algorithm is compared with the block algorithm, the number of computations will be greater due to the need of creation of blocks in the code and thus, the speedup will not be as significant compared to the speedup when the 'pragma omp parallel' for command is used. For greater problem sizes, the speedup will even reduce because a lot of time is wasted in the creation of blocks and it will also reduce with the increase in the number of threads. The speedup which is greater than 1 will be due to access from cache memory which is much faster compared to access from RAM.

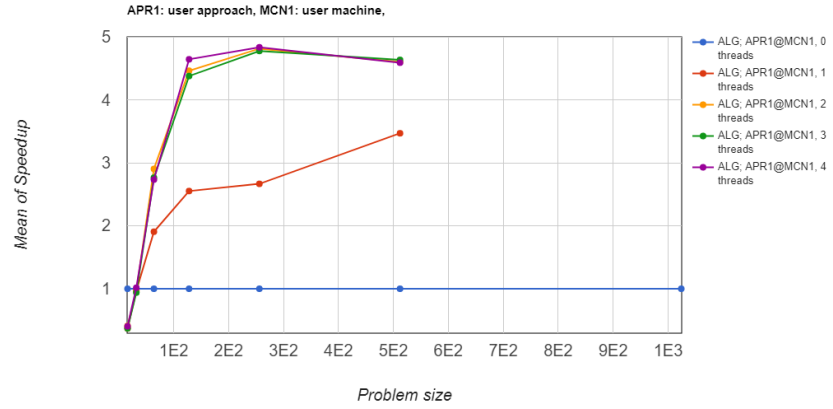
Efficiency Curve related analysis

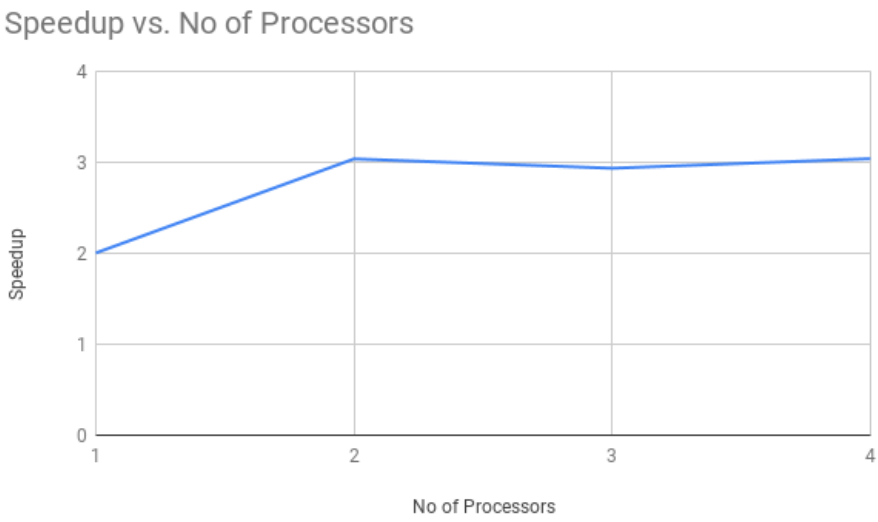
For a lower problem size, the efficiency is the greatest for the serial algorithm as the number of processors used are the least in this algorithm and the speedup is therefore the maximum for this case. As the problem size goes on increasing, the efficiency of the serial algorithm will reduce due to 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.

Analysis when serial code is compared with the outermost loop parallelization.

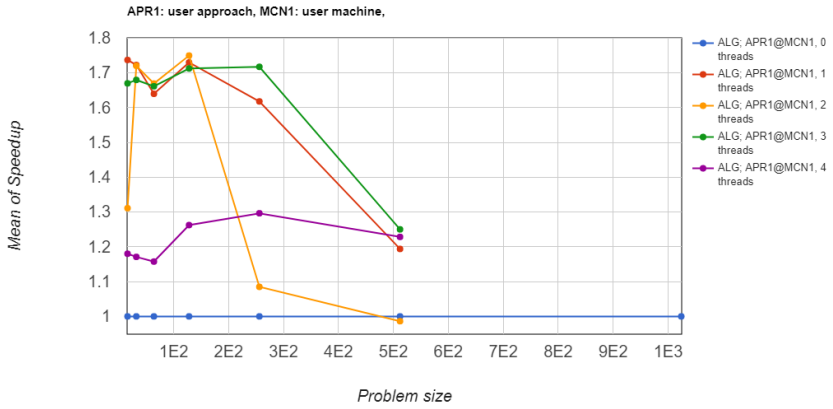


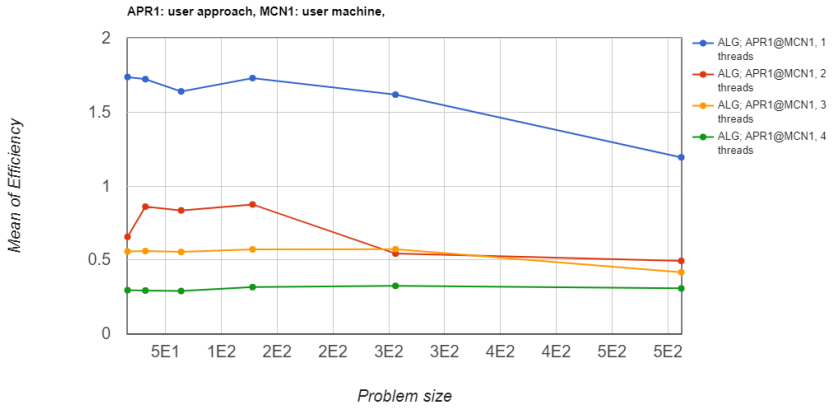

APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
1
0.8
0.6
0.4
0.2
1E2
1E3
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads

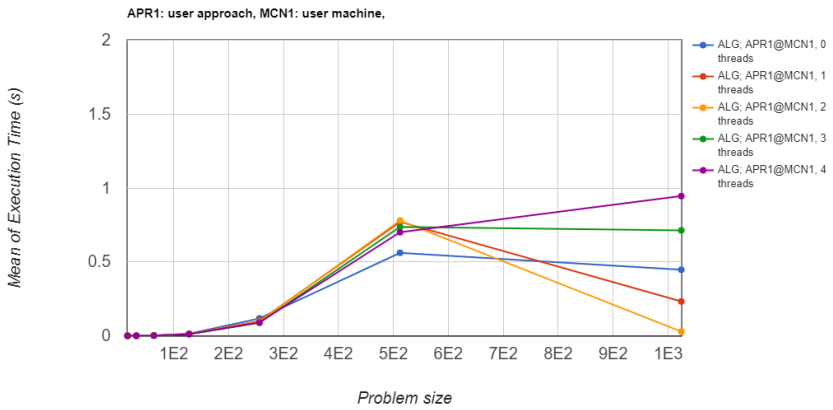

APR1: user approach, MCN1: user machine,
Mean of Speedup
5
4
3
2
1
1E2
2E2
3E2
4E2
5E2
6E2
7E2
8E2
9E2
1E3
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads

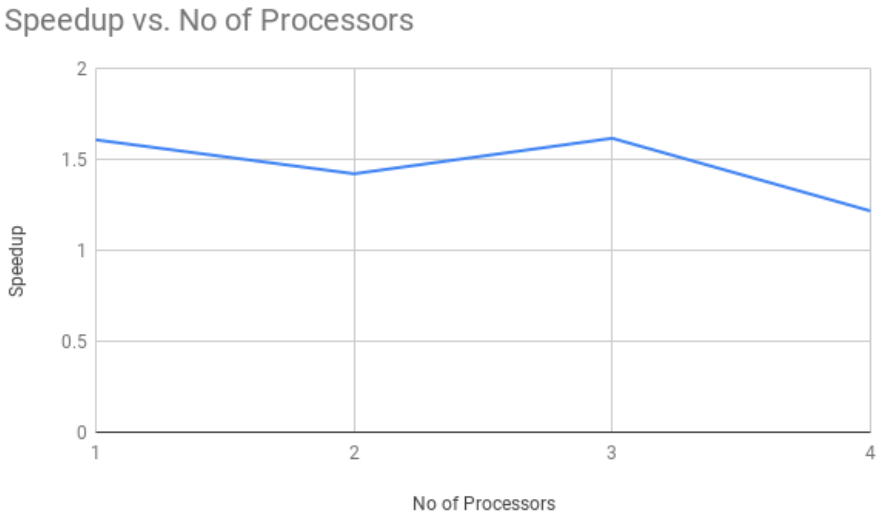


Analysis when regular matrix multiplication is compared with block matrix multiplication.




APR1: user approach, MCN1: user machine,
Mean of Efficiency
2
1.5
1
0.5
0
5E1
1E2
2E2
2E2
3E2
3E2
4E2
4E2
5E2
5E2
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)
2
1.5
1
0.5
0
1E2
2E2
3E2
4E2
5E2
6E2
7E2
8E2
9E2
1E3
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


Speedup vs. No of Processors
Speedup
2
1.5
1
0.5
0
1
2
3
4
No of Processors