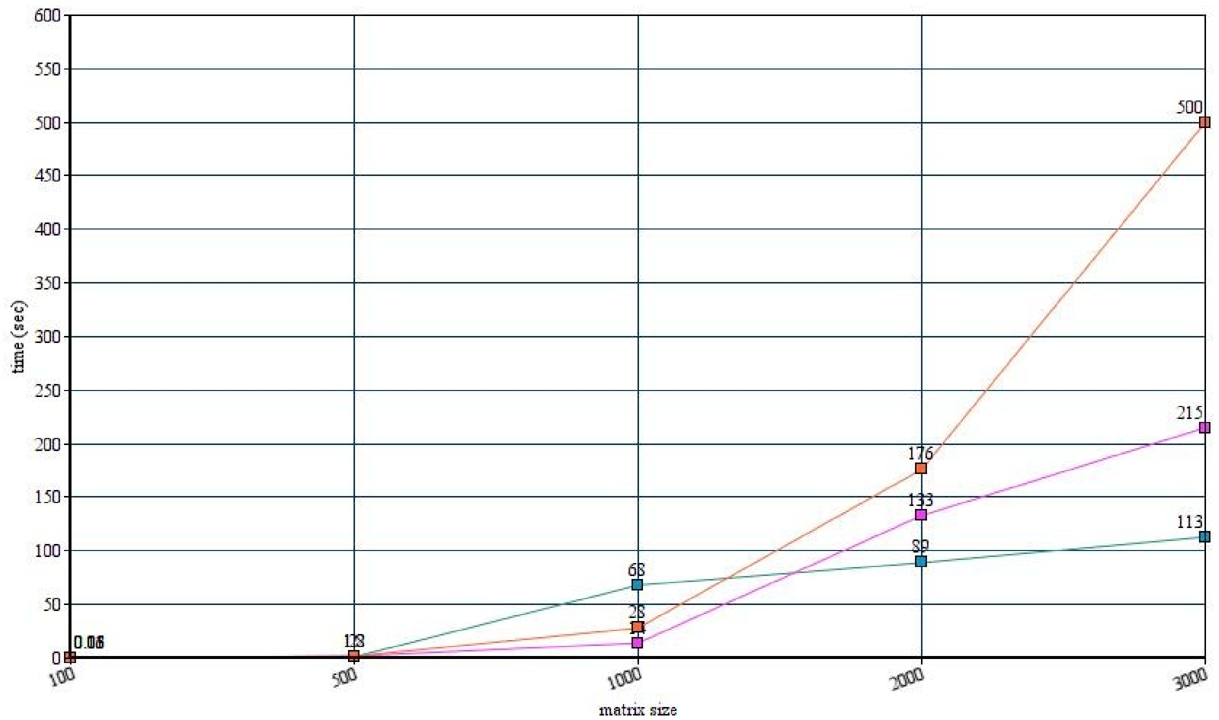
ANALYSIS of Sequential , block , block-thread matrix multiplication

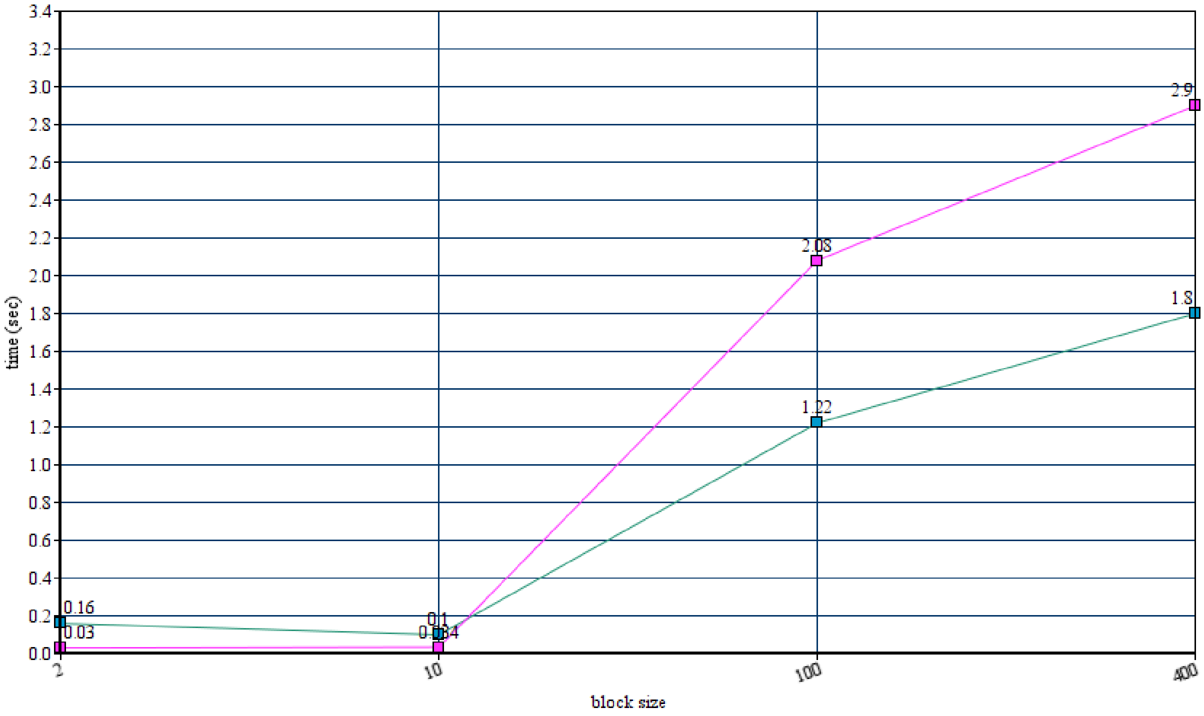
ADITYA BHARDWAJ
CI17M01
DOT ,PUNE

red -> sequential
purple -> block
green -> block+thread

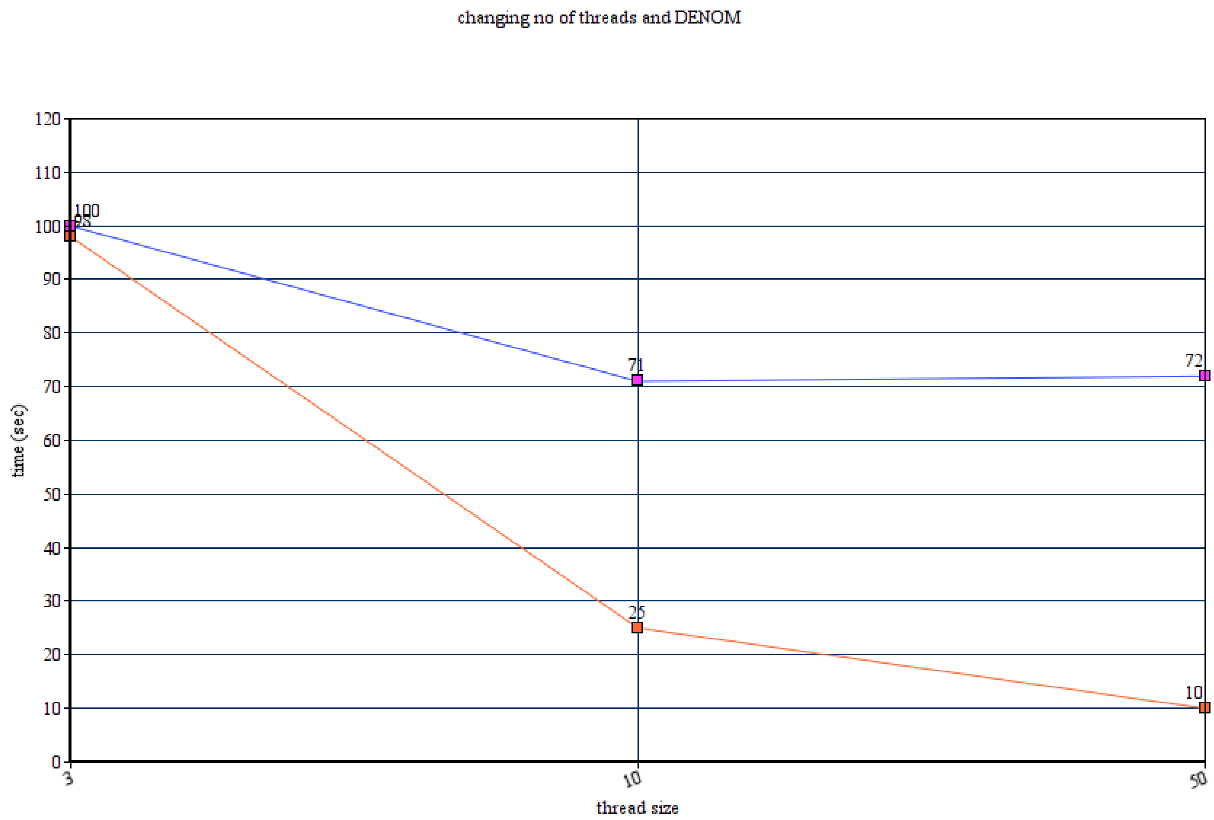


Note : Block size and thread size was kept constant

It can be concluded that block and thread algorithms outperform the sequential algo for large matrices. And hence are scalable and effecient for large matrics and real world problems.



There is a thershold for the block sizes as it can be inferred that for block size 100 -200 the block and thread algorithm are effecient. Increasing the blocksize doesnt improve time but can also degrade it as there will redudancy in computations and cache misses.



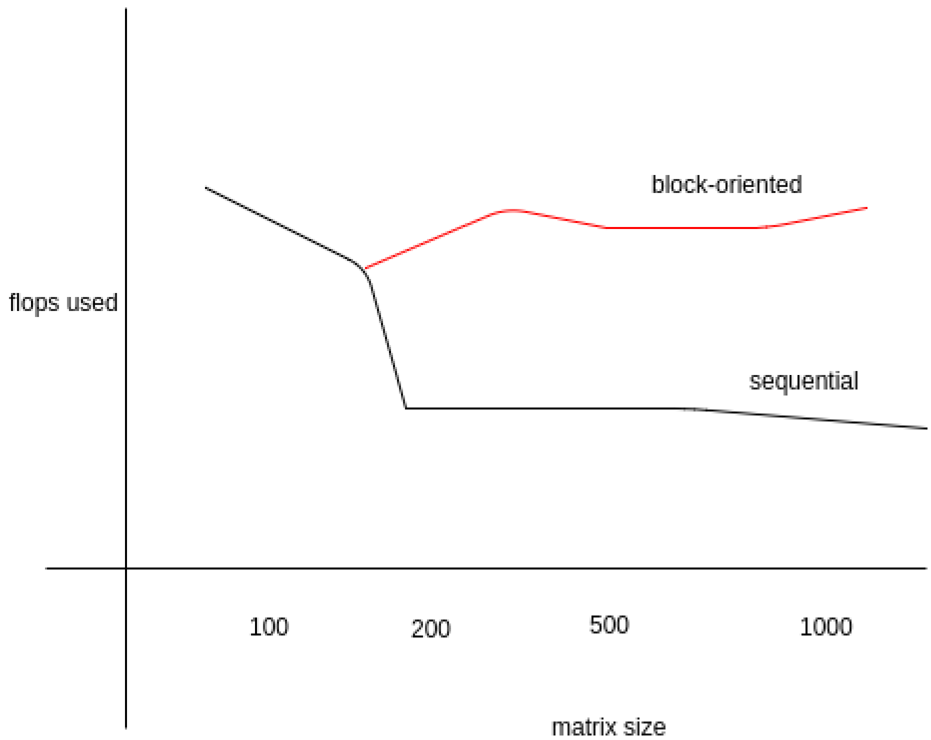
red-> size 1000
blue -> size 2000

Increasing the no of threads does decrease the computation time massively. But again it too has a threshold value beyond which no performance upgradation is seen.
Besides , the DENOM value i.e. computation per thread , if we increase this ratio in the code it also improves the result by a good margin.

Assuming A * B and C = result matrix (all are N*N)

1) The sequential algorithm
During each iteration , each process multiplies (N/p *N/p) block of A by (N/p * N) portion of B . It then adds (N/p * N) matrix to C . Therefore , computation time for each iteration is N^3/p^2. Hence , T(n) = O(n^3)

2) In block matrix algo ,
The total computation come around n^3 , but the path/steps taken is O(log^2n) and hence in the overall complexity is appx O(n^2) . Performs better for large scale matrix.



USING MPI :

1) 1000* 1000 matrix
exec time -> 18sec

2) 2k*2k matrix
exec time -> 180sec

3) For mpi execution (multi-core) use :

```
mpicc filename.c -o filename -lm
mpiexec -n 5 filename argv1 argv2
```

CONCLUSION :

In this we use different matrix multiplication algo-rithms on different layers to show how performance will beaffected in mixed mode programming without a good cachealgorithm, even when the work load is perfectly balanced.Since the core of parallel computations are still sequential computations, to improve the overall performance, not only do we need a model to utilize memory on every layer, butalso good sequential core algorithms to achieve high per-formance.If thecomputations is divided into many stages,and each stages only works on small data size, improving distributed algorithms improve the performance since cache misses do not matter much on computing small data size.On the other hand, if the computation has to work on largechucks of data, it is important to combine a good cache al-gorithms with an increase in the number of processors. The sat-uration of the thread space beyond the total number of com-puting threads equal to the number of available processors provide a modest performance enhancement.