High Performance Computing Report

Amarnath Karthi
Chahak Mehta

Dhirubhai Ambani Institute of Information and Communication Technology
201501005@daiict.ac.in
201501422@daiict.ac.in

1 Implementation Details (Basic matrix multiplication)

1(a) Brief and clear description about the Serial implementation

The serial implementation involves basic matrix multiplication using 3 nested loops.

$$C_{ij} = \sum A_{ik} B_{kj} \tag{1}$$

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

The parallelization is achieved by load distribution of the middle loop over several processors, using the OpenMP library. We will be using the **omp parallel for directive**.

2 Complexity and Analysis

2(a) Complexity of serial code

Cubic time complexity : $O(n^3)$. This is because of the 3 nested loops, each having an $O(n)$ runtime complexity by itself.

2(b) Theoretical Speedup (using asymptotic analysis, etc.)

For an n core system, the theoretical speedup is approximately n.

3 Curve Based Analysis

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

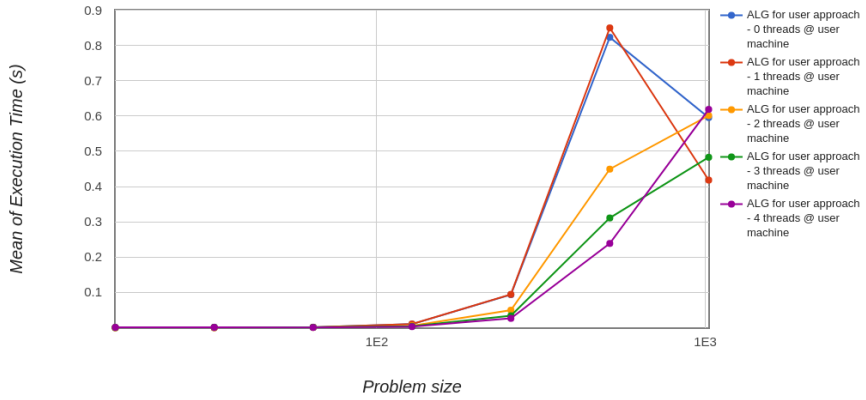

Fig. 1. Execution time for simple matrix multiplication

For small problem sizes, the overhead of adding 1 extra core to the solution outweighs the advantages given by it. Thus for small input sizes, the execution time is higher for a higher number of cores. As the problem size increases, the overhead becomes insignificant in front of the compute time, which decreases drastically when a larger number of cores are used. This is quite natural because the work is shared equally amongst all processors.

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

The speedup is bound by "n", the number of cores. Therefore, irrespective of the problem size, the speedup is always lesser than n for n cores. For small problem sizes, the speedup is less than one if we use more cores. This again indicates that the load is too small to be parallelized efficiently. For very large problem sizes we see almost a constant speedup. This indicates that saturation has been achieved. For 4 cores and a problem size of 512, we get a speedup of approximately 3.2, whereas for 2 cores we get a speedup of 1.78.

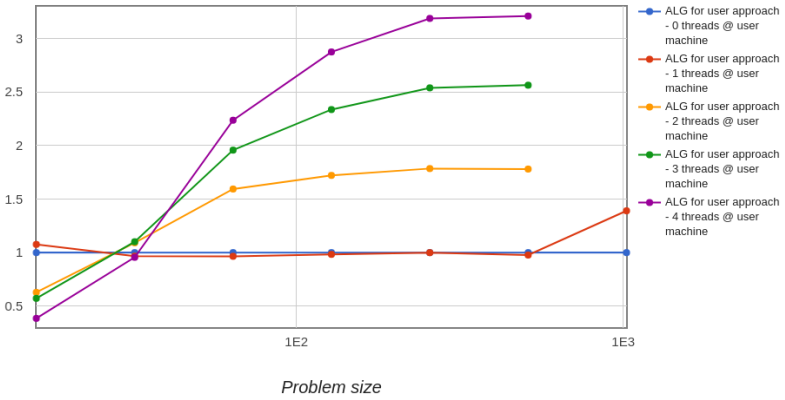

Fig. 2. Speedup vs problem size

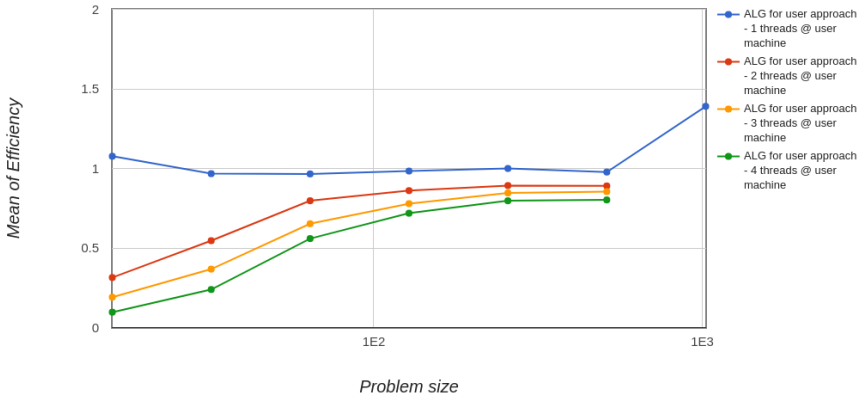

Fig. 3. Efficiency vs problem size

4 Implementation Details(block matrix multiplication)

4(a) Brief and clear description about the Serial implementation

This is a divide and conquer approach on traditional matrix multiplication. We recursively divide each matrix into smaller and smaller blocks, and perform multiplication and summation operations on them.

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

Initially we distribute a different set of blocks to each thread. In the combine step of a level, the results of 2 threads are added into one matrix, and one thread hands over its entire data to another and dies. This step goes on until there is one thread surviving. This thread will hold the final result of the matrix multiplication.

5 Complexity and Analysis

5(a) Complexity of serial code

Cubic complexity $O(n^3)$

5(b) Theoretical Speedup (using asymptotic analysis, etc.)

n, where n is the total number of processors.

6 Curve Based Analysis

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

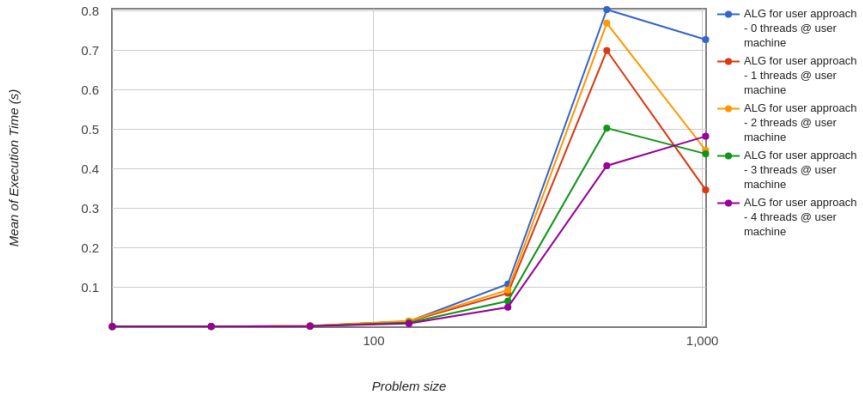

Fig. 4. Execution time for simple matrix multiplication

For small problem sizes, the overhead of adding 1 extra core to the solution outweighs the advantages given by it. Thus for small input sizes, the execution time is higher for a higher number of cores. As the problem size increases, the overhead becomes insignificant in front of the compute time, which decreases drastically when a larger number of cores are used. This is quite natural because the work is shared equally amongst all processors.

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

The speedup is bound by "n", the number of cores. Therefore, irrespective of the problem size, the speedup is always lesser than n for n cores. For small problem sizes, the speedup is less than one if we use more cores. This again indicates that the load is too small to be parallelized efficiently. For very large problem sizes we see almost a constant speedup. This indicates that saturation has been achieved. For 4 cores and a problem size of 512, we get a speedup of approximately 2.09, whereas for 2 cores we get a speedup of 1.28.

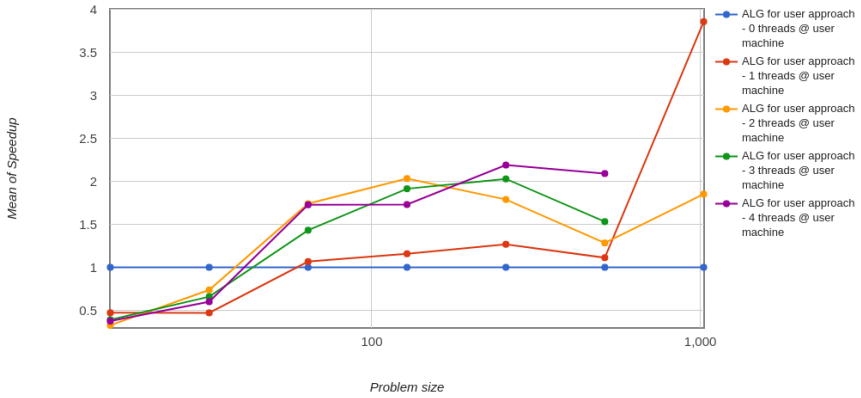

Fig. 5. Speedup vs problem size

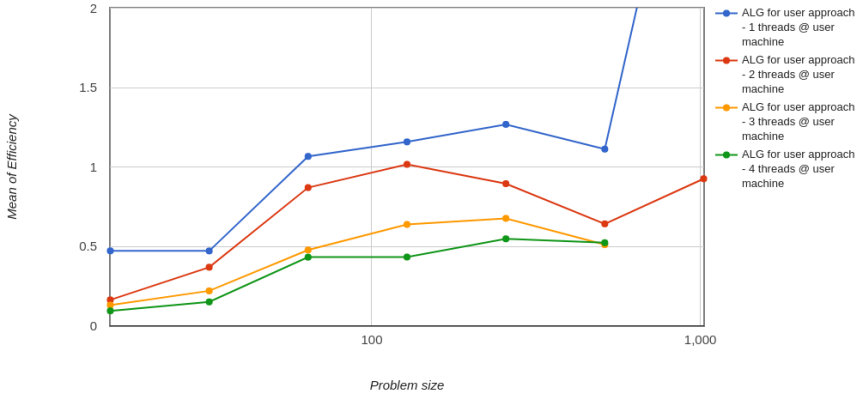

Fig. 6. Efficiency vs problem size