1. **Implementation**
    a. Vector, Matrix Addition: Distributed the elements across ranks/threads uniformly (i.e #elements/thread = #total_elements/#ranks)
    b. Matrix Multiplication: Used the general matrix multiplication algorithm. Distributed the rows of the first matrix(A) uniformly across ranks and broadcasted the second matrix(B) to all ranks. So, each rank computes #A_rows/size of the output.
2. **Graphs**

    a. **OpenMP**

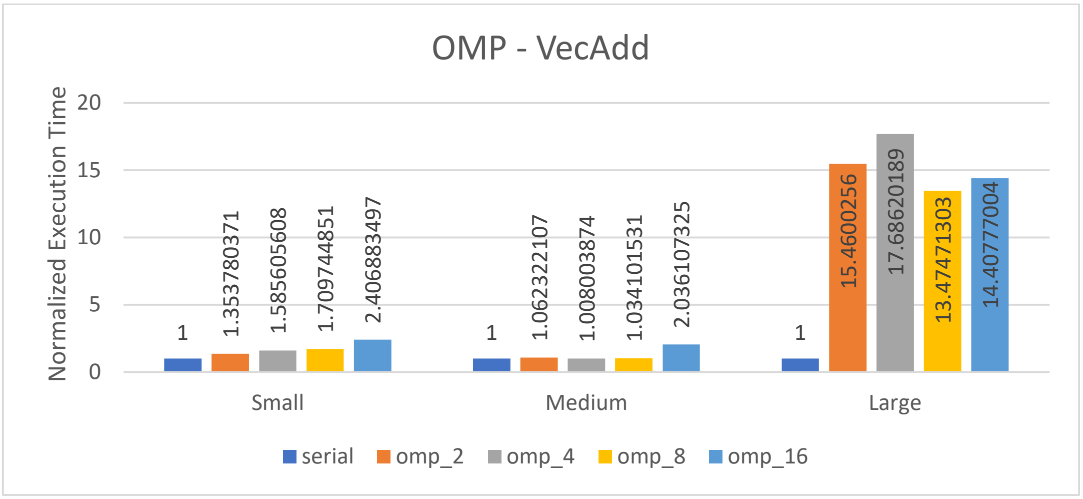

Figure 1: Normalized Execution Time for OpenMp based Vector Addition

For small and medium workloads, omp 2,4,8 performs about the same as serial. However, for omp 16 for small and medium and omp for large datasets, the overhead seems to be higher than performance gained due to parallelism.

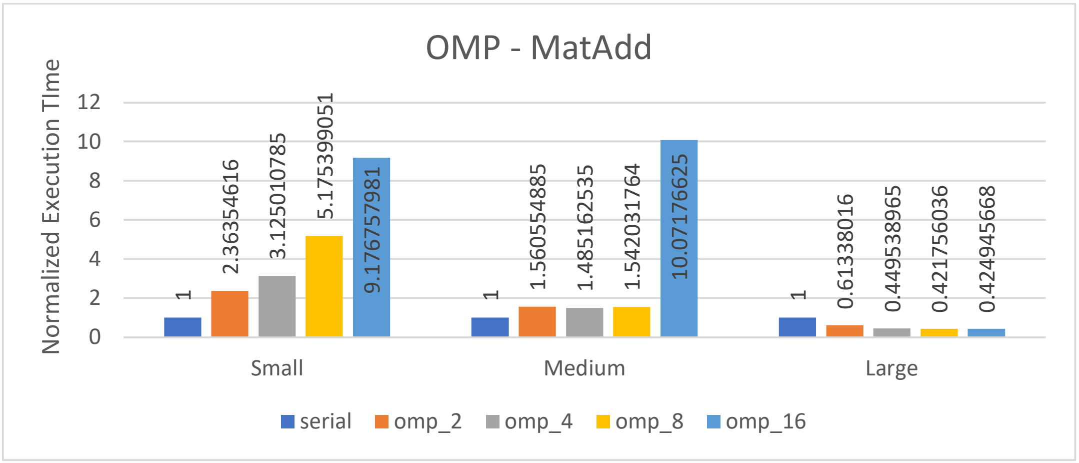

Figure 2: Normalized Execution time for OpenMp based Matrix Addition

For small and medium datasets, overheads due to synchronization seems to outweigh the performance gained due to parallelism. However, for large dataset omp provides significant reduction in execution time as compared to serial execution.

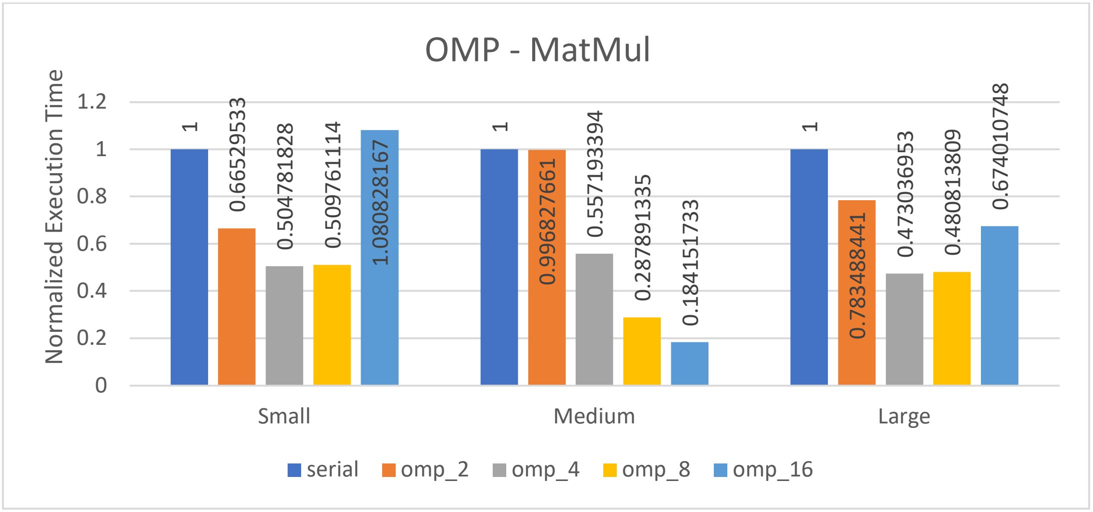

Figure 3: Normalized Execution Time for OpenMp based Matrix Multiplication

OpenMp based execution provides speedup as compared to serial execution for all thread configurations. For small and large datasets, increasing number of threads beyond 4 doesn't seem to provide any performance gain, the 16 threaded configuration seems to be slower than the 4,8 threaded configurations. This is probably because the overheads are higher than the amount of parallelism that can be extracted from the dataset. However, for medium dataset, the performance seems to be improving when number of threads are increased.

b. MPI

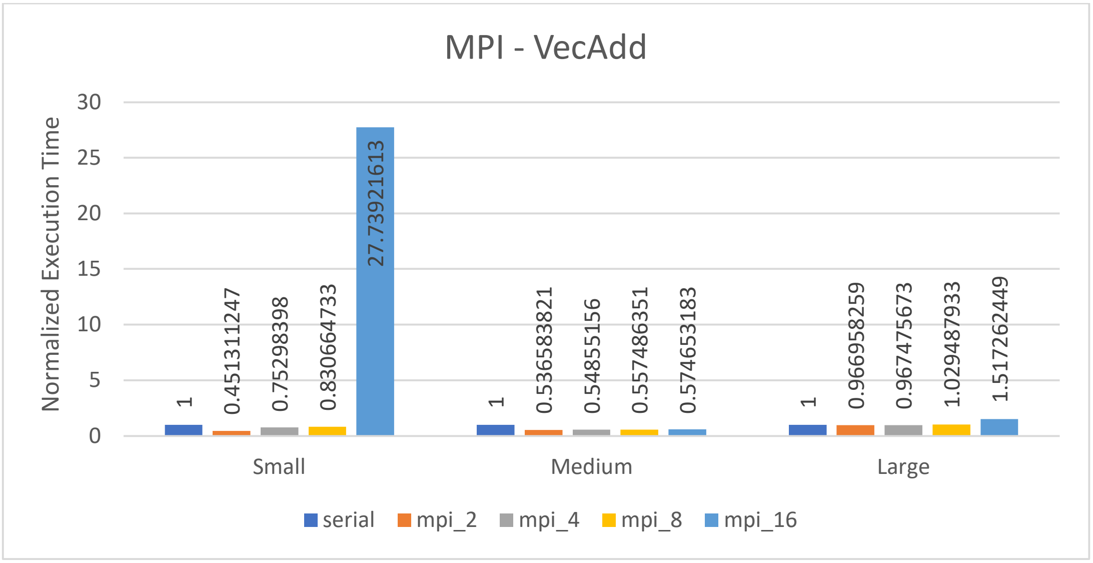

Figure 4: Normalized Execution Time for MPI based Vector Addition

For the small dataset, MPI provides highest speed up for 2 ranks and then slows down as we increase the ranks and worsens in case of 16 ranks as overheads significantly dominate any performance gained through parallelism as the dataset is small. For the medium and large datasets, MPI with 2 ranks provides the best performance. However, unlike small dataset, the 16 rank MPI performs about the same or slightly worse than the lower rank configurations because gain due to extracting parallelism offsets the high overheads as the dataset is larger.

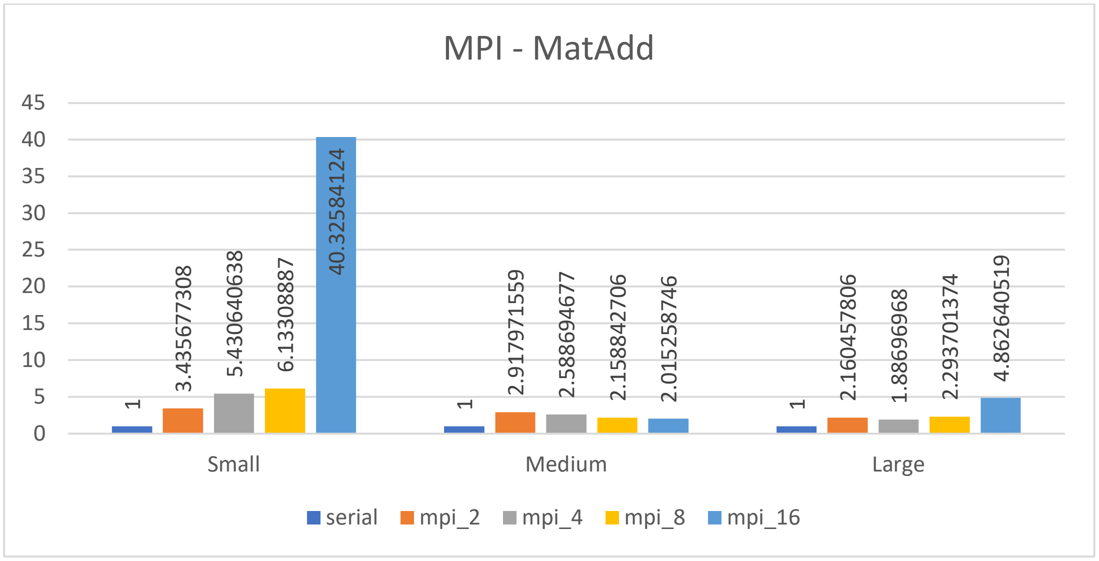

Figure 5: Normalized Execution Time for MPI based Matrix Addition

MPI seems to be slower than the than serial because, for a matrix a size x, number of additions is x. Work done per rank would be x/#ranks and scatter, gather cost is O(2x). So, for smaller dataset, the total amount of work is less, when this is distributes across ranks, the cost of scattering and gathering would be higher than reduction in execution time. This seems to be better in case of medium and large datasets as they require more computations to be done but synchronization cost is even higher.

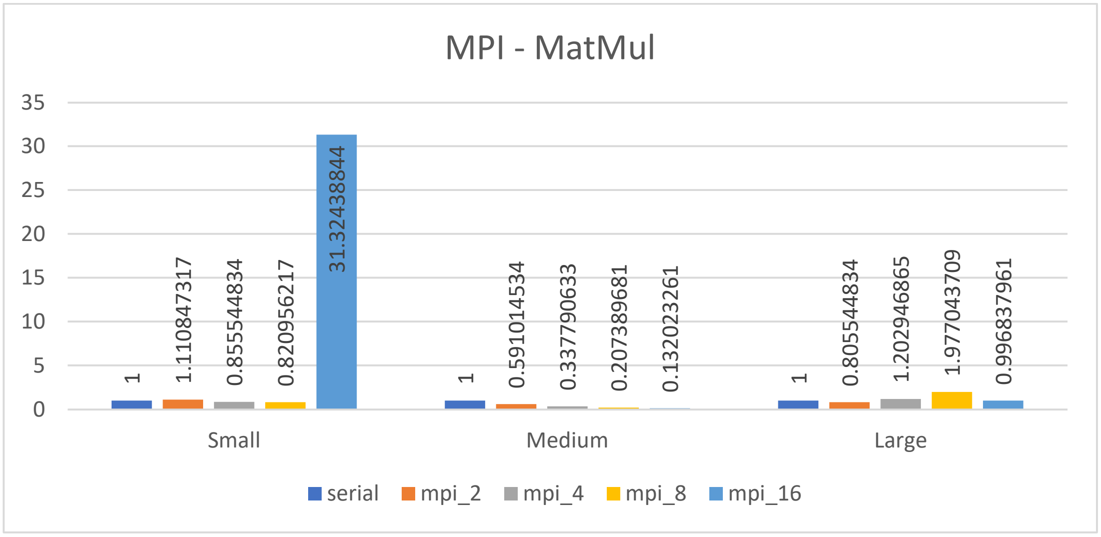

Figure 6: Normalized Execution Time for MPI based Matrix Multiplication

If the inputs to multiplied are square matrices of size N x N. The total amount of computation required would be N – multiplications, N-1 – additions per element. So total computation cost would be N x (N-1) x (N x N) -> O(N^4) and the cost of scattering and gathering would be O(ranks*N^2) (matrix multiplication algo requires broadcasting one input and scatter the other input). So, for small datasets best performance is achieved for rank 8 but worsens in case of rank 16 as overheads dominate. For medium datasets performance seems to be improving as we increase ranks until 16.

c. Serial vs OpenMP vs MPI

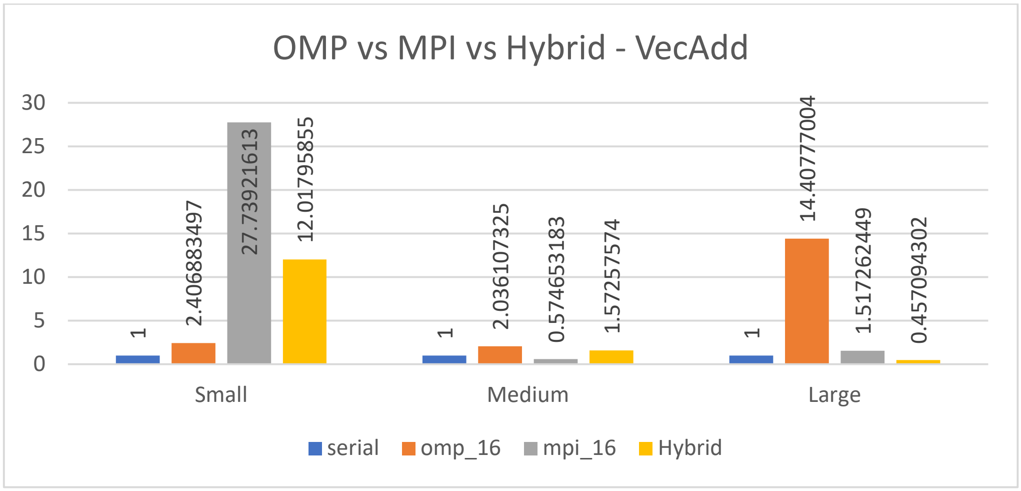

Figure 7: Normalized Execution Time comparison of Serial, OMP, MPI and Hybrid approaches for Vector Addition

Hybrid implementation would have lesser overheads as compared to MPI but more than that of OpenMP because OpenMP uses shared memory whereas MPI uses messages to share data. So, from the execution times for small dataset we can see that hybrid is slower than OpenMP but faster than MPI whereas for medium dataset, MPI has least execution time and large dataset hybrid has the least execution time.

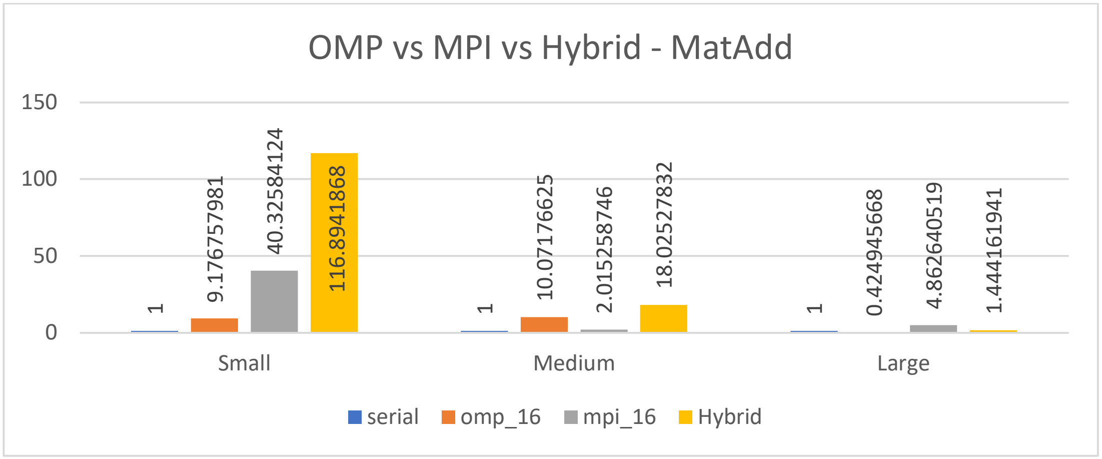

Figure 8: Normalized Execution Time comparison of Serial, OMP, MPI and Hybrid approaches for Matrix Addition

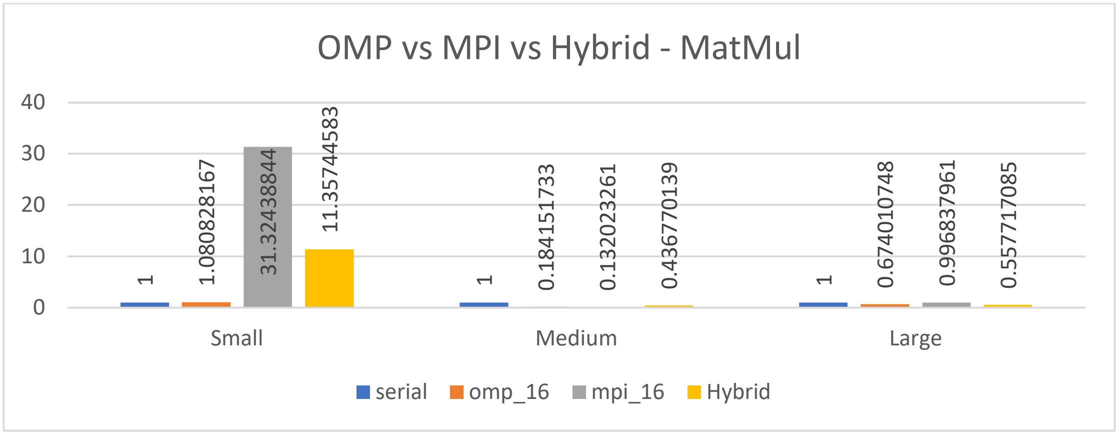

Figure 9: Normalized Execution Time comparison of Serial, OMP, MPI and Hybrid approaches for Matrix Multiplication