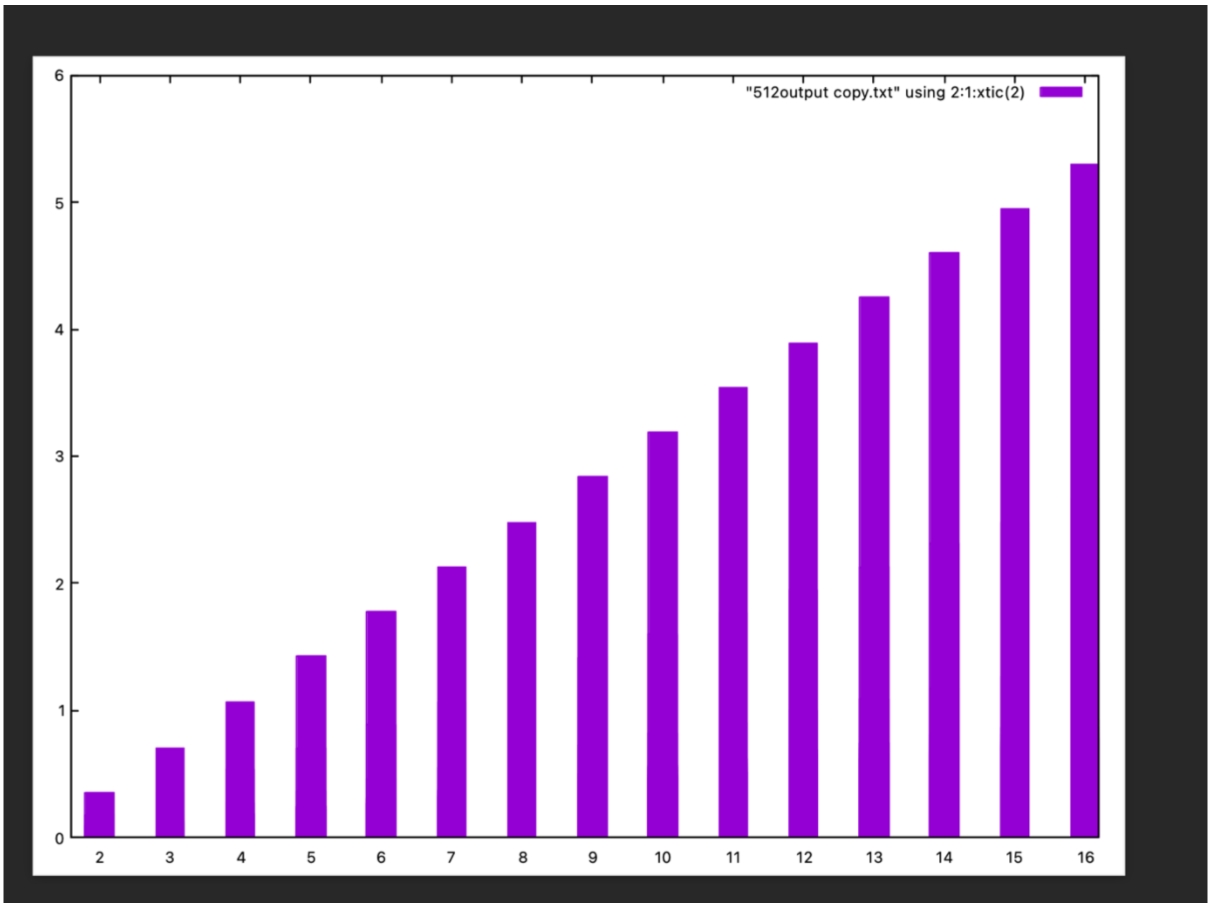
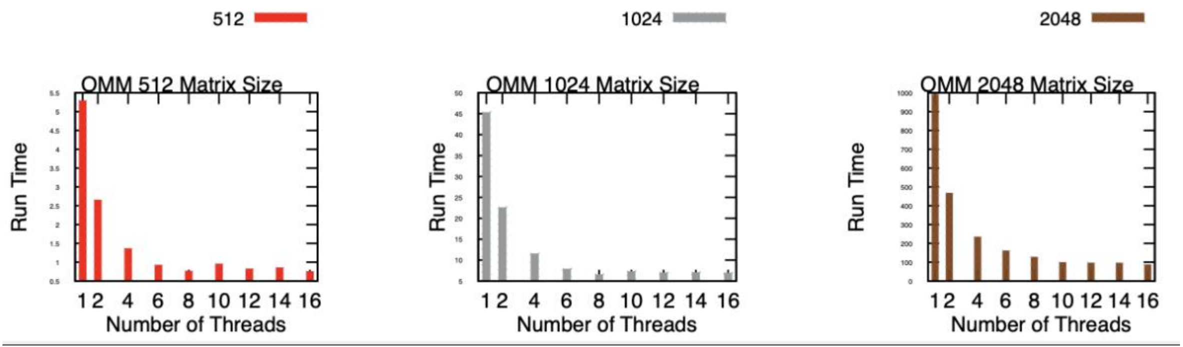
HPC LAB-1

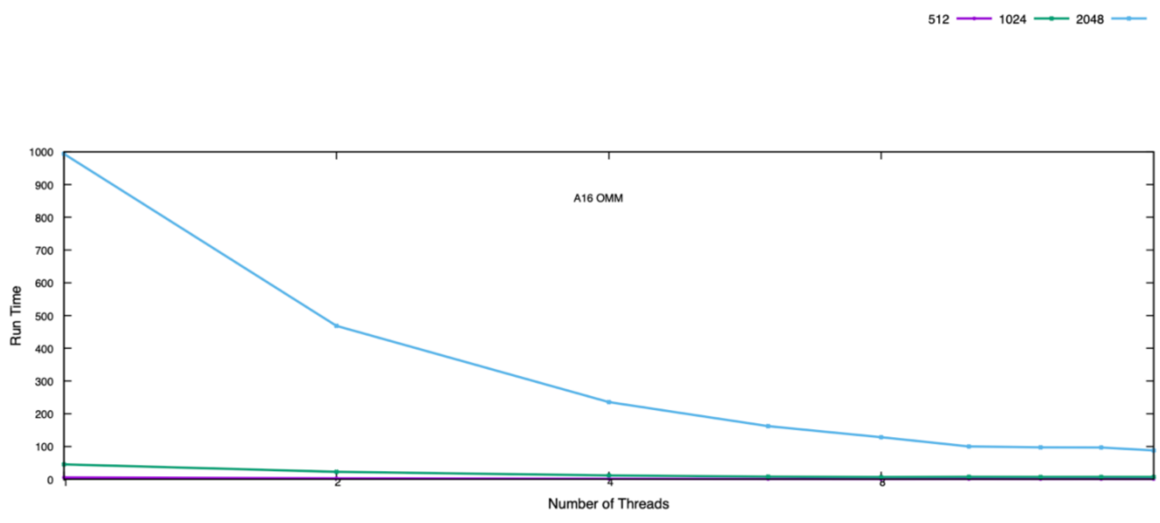
SE20UARI164 V Sai Srikar
SE20UARI179 B.Krishna Vamshi
SE20UARI002 Aashish Joshua James
SE20UARI006 Aditya Sake
SE20UARI005 ADITYA SAI



As you can see in the above graph the time required to run higher powers of matrix multiplication increases.

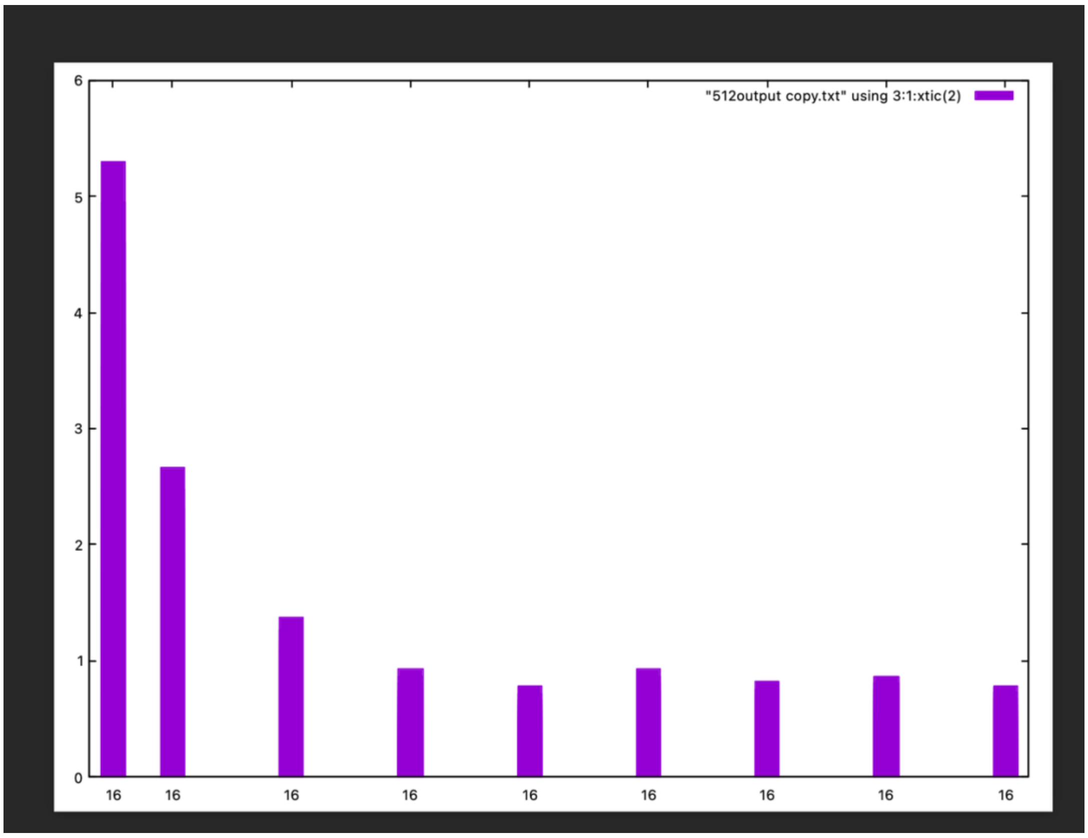
As you can see from the graphs below the amount of time increases drastically as the size of the matrix increases in the case of ordinary matrix multiplication and same is the case observed in Block Matrix multiplication keeping block size constant.





But we with the help of multiprogramming by parallelization solved the problem in lesser amount of time.

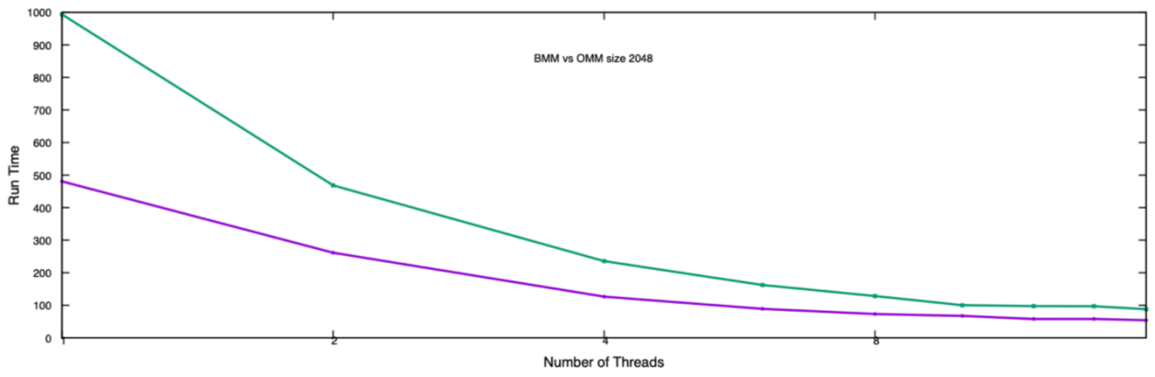
Multiprogramming in terms of threads refers to the ability of a computer system to run multiple threads of execution simultaneously within a single process. In this context, a thread is a lightweight, independent unit of execution within a process that shares the process's resources, including memory and file descriptors. With multithreaded programming, a process can have multiple threads executing different tasks concurrently, improving the system's overall performance and responsiveness.



As you can see in the above graph the same 512 matrix multiplication to the power 16 is taking place but on different threads and as the number of threads increased the amount of time elapsed to calculate the answer decreases.

Matrix Multiplication can take place by-:

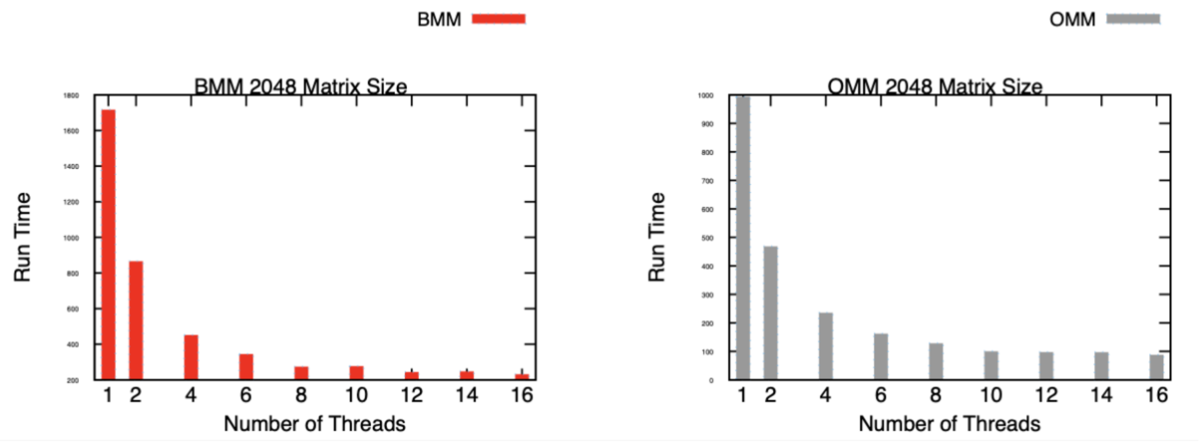
1. Ordinary Matrix Multiplication
2. Block Matrix Multiplication



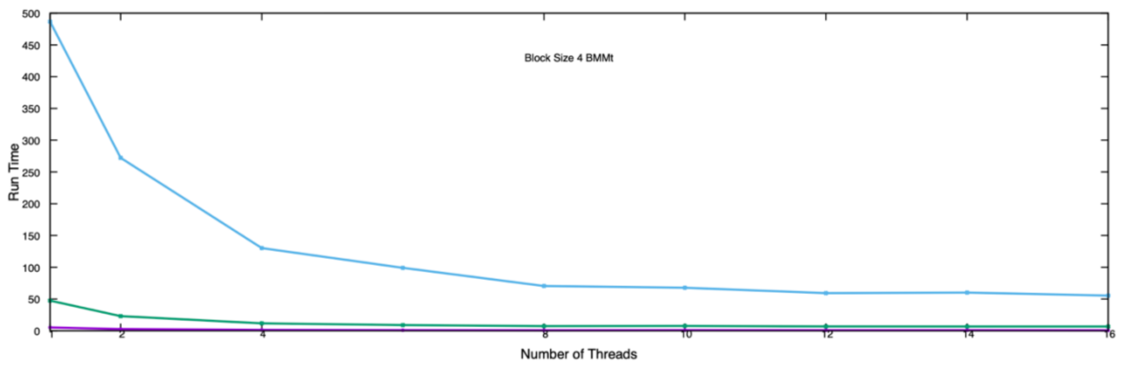
- In an ordinary matrix multiplication, the resulting matrix is found by taking the dot product of each row of the first matrix and each column of the second matrix. This operation is commonly used in linear algebra for solving systems of linear equations, and in various mathematical and computational applications.

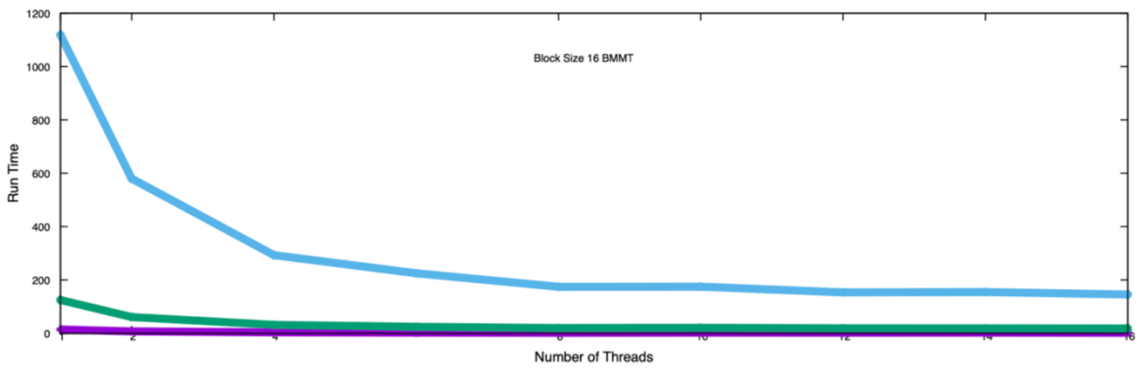
- Block matrix multiplication is a way of partitioning large matrices into smaller blocks or submatrices, and then performing matrix multiplication on these smaller blocks. The goal of block matrix multiplication is to optimize the performance of matrix multiplication by reducing the number of memory accesses, reducing the amount of computation required, or both.

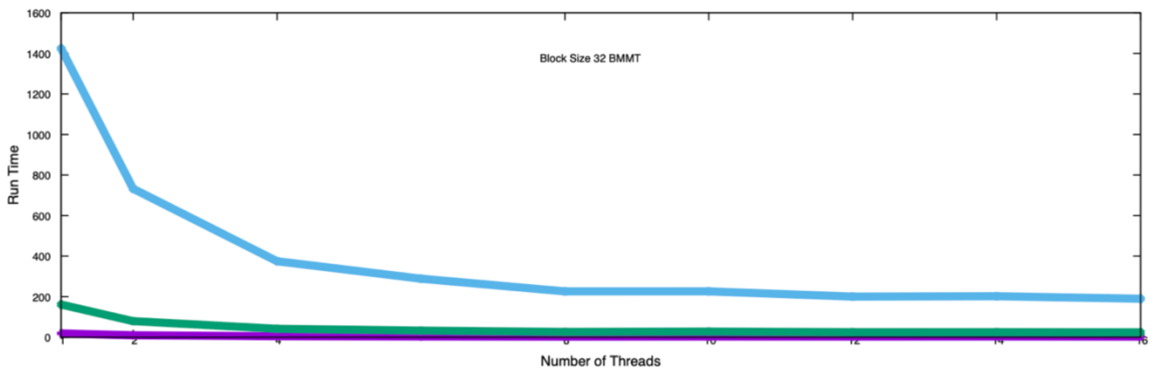
As you can see in the above line graph BMM runs faster when compared to OMM.

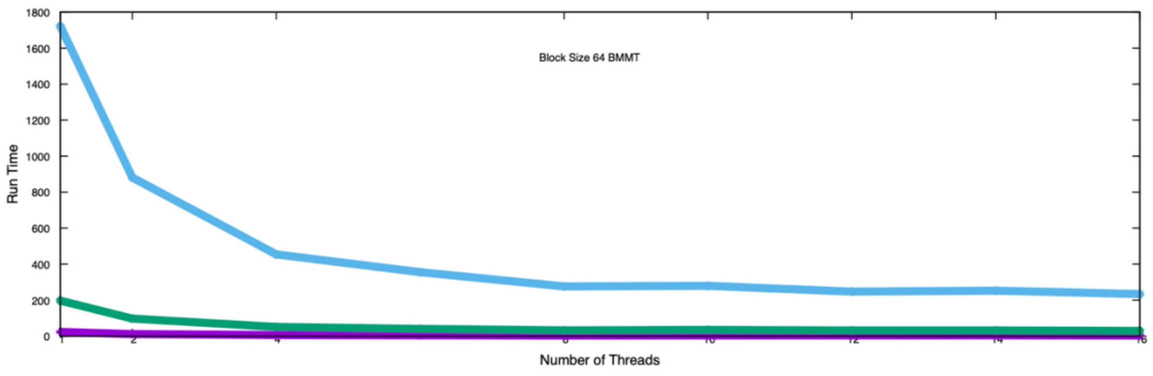


For BMM the amount of time elapsed also depends on the block size that is the smallest blocks we can make before parallel multiplication.





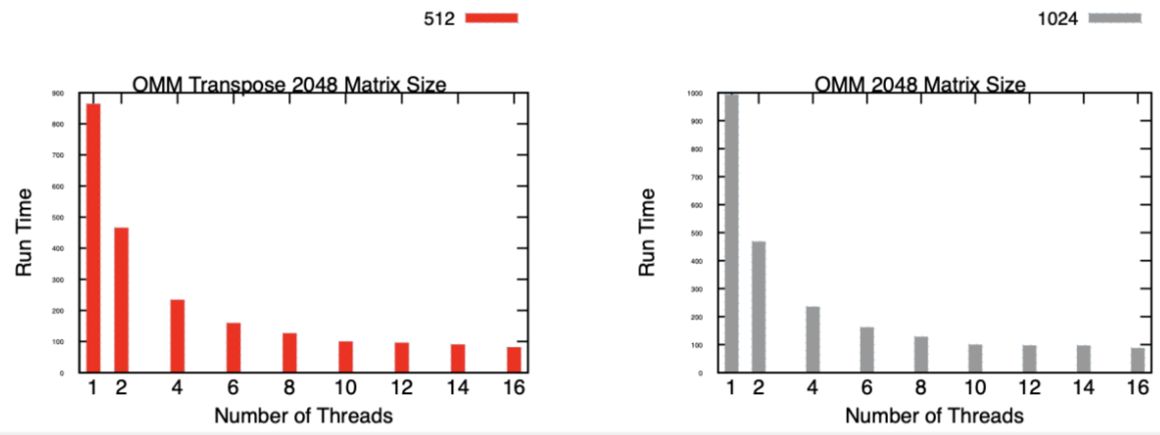




In the above graphs we can see how for varied block sizes the time taken varies.
The performance of block matrix multiplication depends on the size of the blocks or submatrices used. The choice of block size affects both the computational cost and the memory access patterns of the algorithm.

If the block size is too small, the overhead of partitioning and recombining the blocks may dominate the computation, reducing the overall performance of the algorithm. On the other hand, if the block size is too large, the memory access patterns may become inefficient, leading to cache misses and decreased performance.

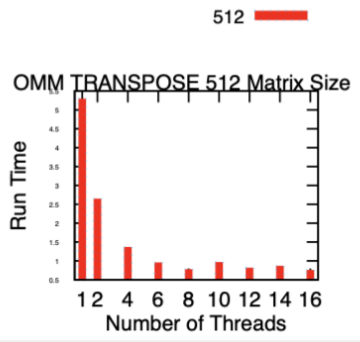
Choosing an appropriate block size requires a trade-off between these factors. In general, the block size should be chosen based on the characteristics of the specific application and the underlying hardware. Empirical studies and experience can help determine the best block size for a given problem.

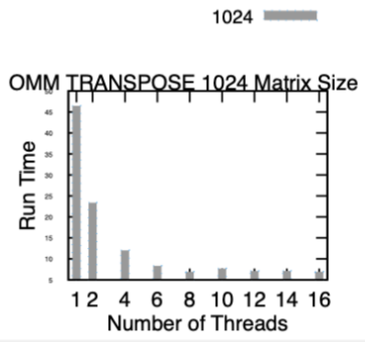


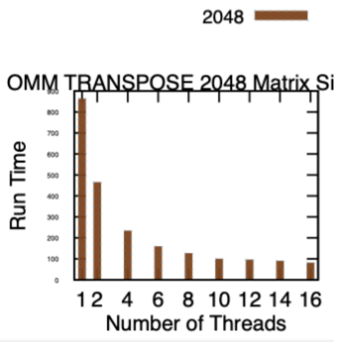
As you can see in the above graph the amount of time taken to compute 2048 matrix multiplication is less in the case of transpose than without transpose.

When performing matrix multiplication, the number of memory accesses and cache misses is proportional to the size of the matrices and the stride of the memory access patterns. By transposing one or both matrices involved in the multiplication, the stride of the memory access patterns can be reduced, leading to fewer cache misses and improved performance.

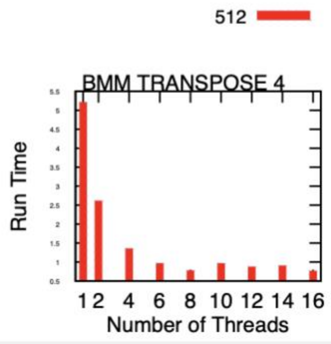
For example, if two matrices are being multiplied and one of them is cache-friendly (i.e., its elements are stored in contiguous memory), transposing the other matrix can make it cache-friendly as well, improving the performance of the multiplication. It's important to note that transposing a matrix can also increase the storage required for a matrix and may not always improve performance, so the decision to transpose a matrix should be based on a careful analysis of the specific case.

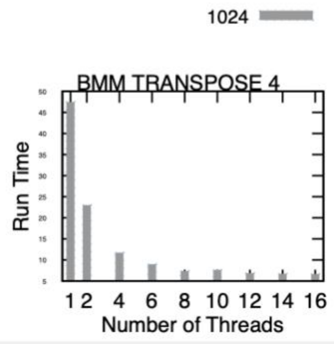


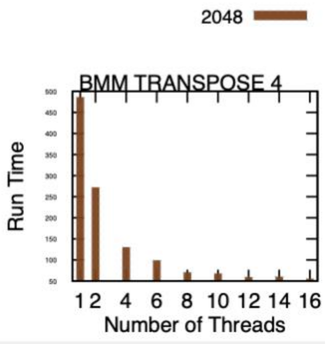


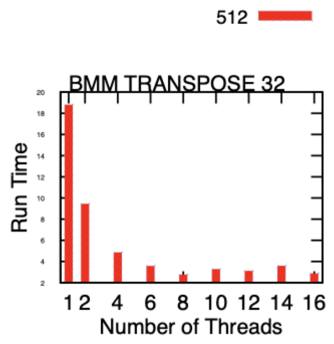


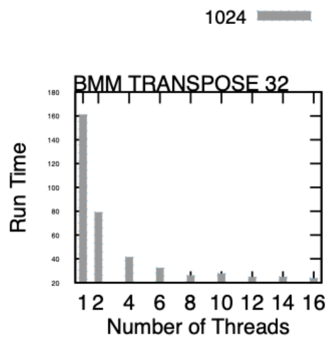
In the case of BMM we have graphs based on the block size for each of the matrices. A few of them can be seen below.

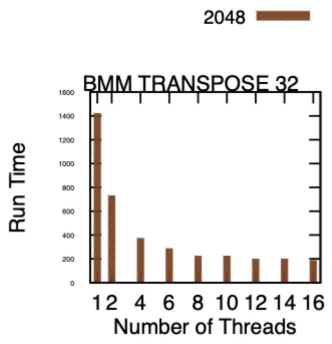


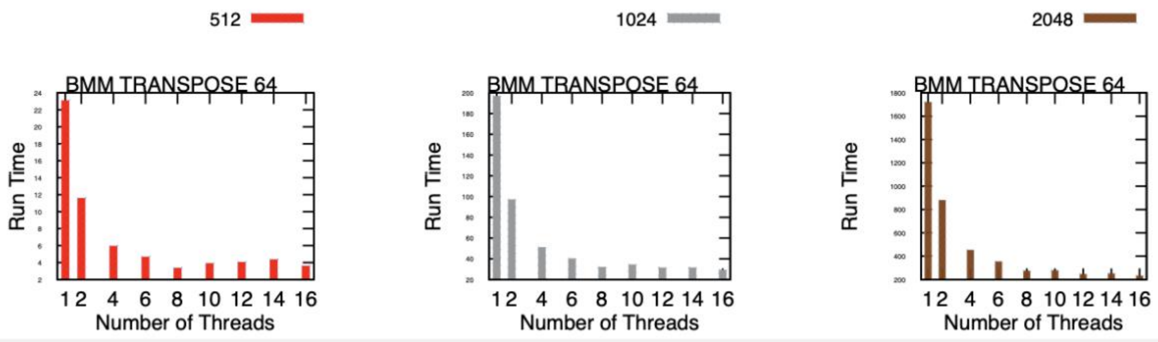












In conclusion we can see how depending on the matrix size and power, number of threads, block size and method of multiplication(OMM,BMM transpose or not) has an effect on matrix multiplication.