I. Algorithm Breakdown

To optimize the naïve sequential algorithm of matrix multiplication, I divide the matrixes into smaller blocks size of 64x64. Each block of A is multiplied with the corresponding block of B, and the partial results are accumulated into C. To achieve a multithreaded approach, I first fetched the hardware's concurrency level. Hardware concurrency is the ability of a computer's hardware to perform multiple operations at the same time. Your hardware can only run a limited number of threads at the same time. It leads to faster overall processing and more efficient multitasking. Each thread is responsible for a range of rows in the output matrix C, applying the block multiplication within that region.

II. Performance

The table below shows the performance of the standard implementation of the matrix multiplication using the naïve sequential algorithm and the optimized approach of matrix multiplication using the Blocking Algorithm and Hardware Concurrency. Each matrix sizes categories are tested 5 times and got their averages. The Speedup column is calculated using the ratio between the standard method and multithreaded method *(standard / multithreaded*).

Matrix Size	Standard (µs)	Multithreaded (µs)	Speedup
Very Small (10x5 * 5x25)	4.2	10337.2	<0.00
Small (60x50 * 50x70)	192.2	12950	0.01
Medium (150x100 * 100x175)	2578.8	12129.4	0.21
Large (300x200 * 200x400)	24912.6	20091	1.24
Big (650x500 * 500x750)	323559.6	144183.8	2.24
Very Big (800x850 * 850x800)	1362351.6	89617	15.20
Huge (1000x1000 * 1000x1000)	2054016.4	120171.4	17.09

Table 1: Comparison of Standard and Multithread Approach

Based on the table, the standard approach has the fastest time on the Very Small with a time of 4.2µs; the multithreaded approach has the fastest time on the Large with a time of 20091µs.

III. Standard Approach vs. Multithreaded Parallel Approach

Using the Speedup column, we can see the overall performance of the standard and multithreaded approach. We can see that starting from the matrix size Large (300x200 * 200x400), the multithreaded approach is having a performance advantage with 1.24x faster than the standard. As the matrix sizes increase, we can see the increase of performance advantage of multithreaded approach with a maximum value of 17.09x faster than the standard at the size Huge. At lower matrix sizes, we can observe that the multithreaded approach performs worse. It performs worse at Very Small with a value of <0.00 faster than the standard. Each thread takes time to be created, scheduled by the OS, and then later joined. For small matrices, the amount of actual work is tiny, often fewer operations than the cost of managing threads. In addition, context switch costs can be more expensive than just letting one thread finish the work