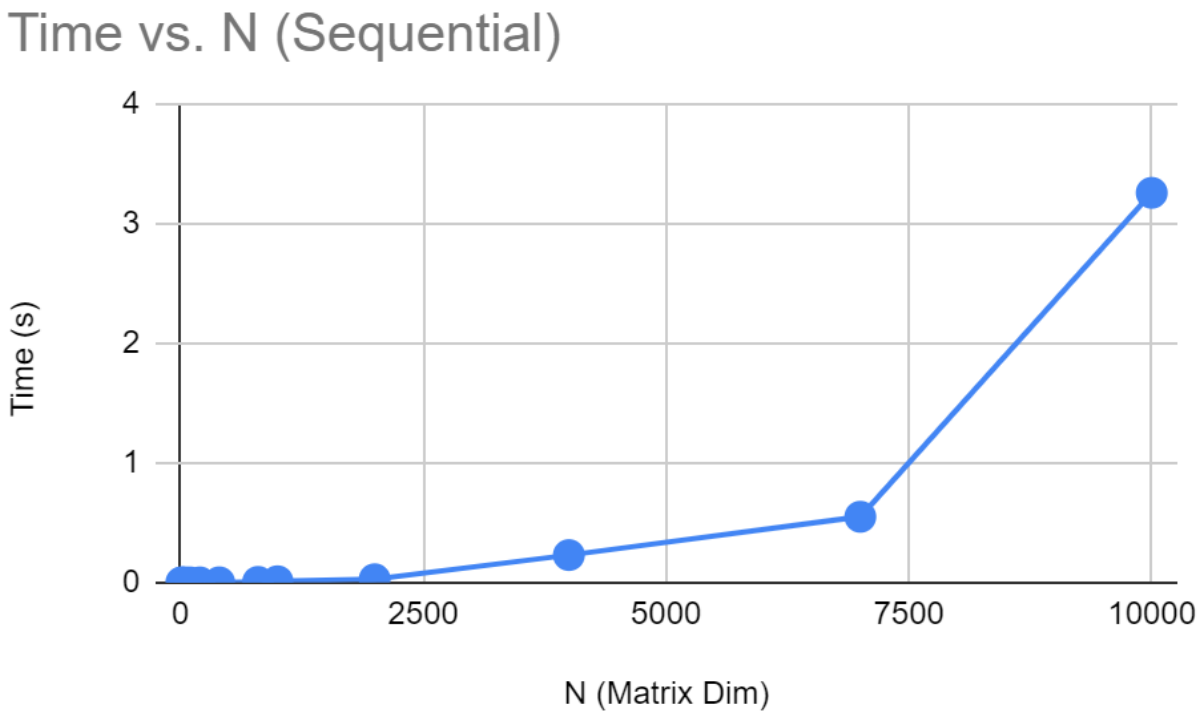
Grant Kinsley (gkinsley)
Ethan Duong (duonget)

231P: HW 5

1. A characterization of the sequential transposer execution time as the size of the matrix increases. Add plots and figures as necessary to explain your results.

The data shows that as the size of the matrix increases, the sequential transposer execution time also increases. The below chart shows the execution times given matrices of size N = { 10, 20, 50, 100, 200, 400, 800, 1000, 2000, 4000, 7000, 10000 } have the beginnings of an exponential curve.

The sequential transposer follows a triple for loop: swapping values of the matrix in place. It would make sense that as the size of the matrix increases, the number of swaps to be done increases and so execution time follows.

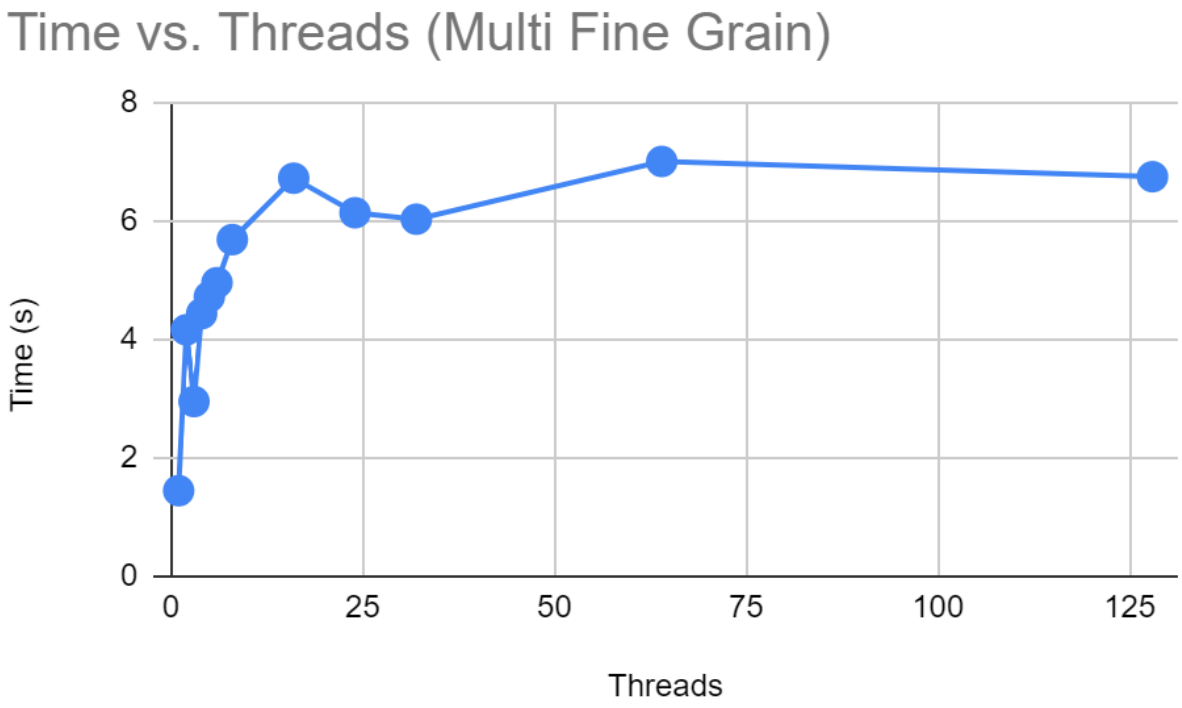


2. A characterization of the multi threaded transposer execution time for a fine-grain configuration. Keep the number of exchanges per thread constant, grain=1. Keep the size of the matrix constant, n=7,000. Vary the number of threads in {1, 2, 3, 4, 5, 6, 8, 16, 24, 32, 64, 128}. Add plots and figures as necessary to explain your results.

The below chart was constructed using the multi threaded transposer and the matrix size and grain were kept constant at N = 7,000 and grain = 1. Varying the threads between 1 and 128m we see that there are rapid increases in execution time between threads = 1 to 16. We then see that increasing the number of threads doesn't affect execution time as greatly.

Since the grain is 1, each thread only does 1 unit of work on the matrix before getting its next unit of work. For each exchange in the matrix, a thread has to read and increment the correct counters tracking matrix exchanges. As the number of threads increases, execution time slows down since threads are waiting on the mutex protected variables.

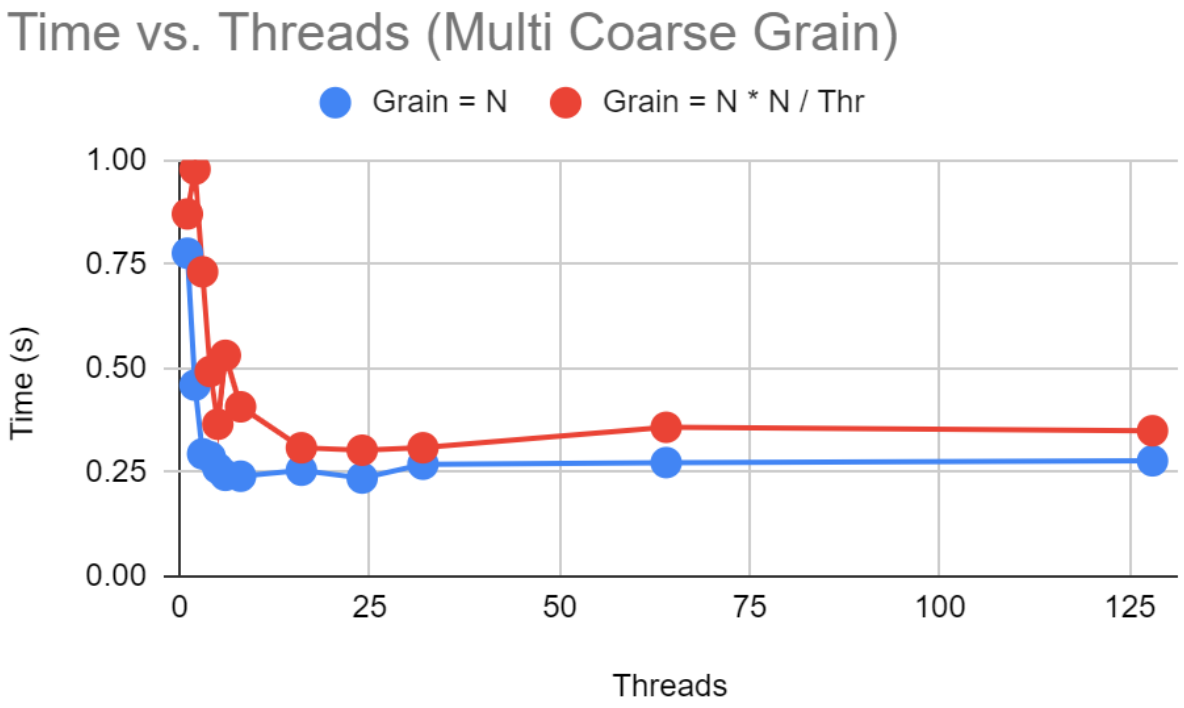
Therefore, a fine grain configuration is not as efficient in this use case since each work unit is a simple exchange of numbers in the matrix. So threads get stuck waiting for the mutex to open so that they can get their next unit of work.



3. A characterization of the multi threaded transposer execution time for a coarse-grain configuration. Experiment with different values for the parameter grain, particularly grain=n and grain=(n×n)/threads.Keep the size of the matrix constant, n=7,000. Vary the number of threads in {1, 2, 3, 4, 5, 6, 8, 16, 24, 32, 64, 128}. Add plots and figures as necessary to explain your results.

The below chart shows execution times with N = 7,000 and 2 different variations of coarse grain: grain = N and grain = N*N/Thr. In both configurations, the execution time decreases rapidly from Threads = 1 to 16. Then there are not as many performance gains when the number of threads > 16. We see that throughout all trials, the configuration where grain = N performs better. When grain = N, we see the most performance gain and least diminishing returns when Threads = 8.

This configuration is more efficient than the fine grain configuration, since each work unit consists of many exchanges. So there is no bottleneck at the mutex, as threads are busy doing the matrix exchanges instead of waiting for the mutex.

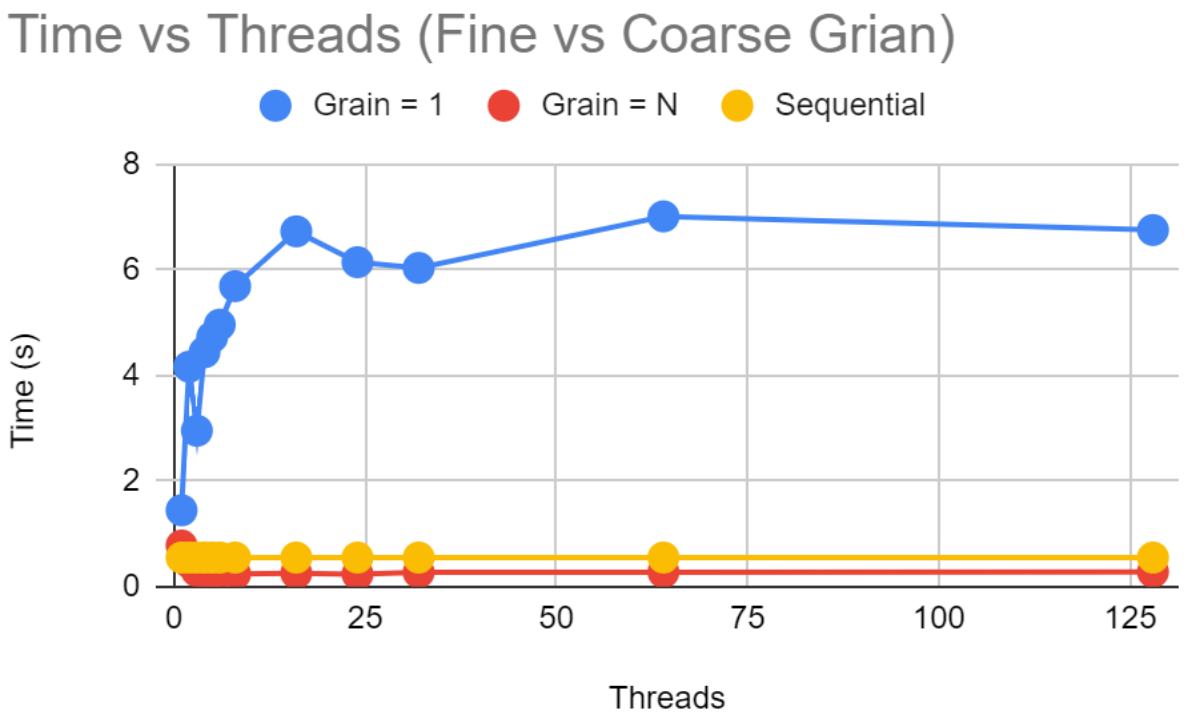


4. Fixing n=7,000, grain=1 for fine grain, and grain=n for coarse-grain; graph the speedup obtained by using the fine and coarse grain multi-thread version with respect to the sequential one, as the number of threads used in the multi-threaded version varies in {1, 2, 3, 4, 6, 8, 16, 32, 64, 128}. Explain your results.

The below chart was constructed from the fine grain data from Question 2, the coarse data from Question 3, and the sequential execution time for N = 7000 from Question 1.

We see that between fine grain, coarse grain, and sequential transposer all perform similarly when the multi threaded versions only have 1 thread (seq < coarse < fine). As the number of threads increases, we see the execution time increase logarithmically. These results were discussed and explained in our answer to Question 2.

Between the coarse grain and sequential performance, it just took the use of 2 threads to see the coarse grain transposer perform better than the sequential transposer. There is another speedup from 2 to 3 threads, then the execution times stay mostly the same as threads increase.



5. Consider matrices of size n from 10 to 10,000. Graph the speedup obtained by the fastest coarse-grain transposer with respect to the sequential version. Explain your results.

From the data shown in Question 3, the better performing transposer was when grain = N. We plotted the data from Thr = 8, 24, and 128 to compare to the sequential transposer. We'll focus on Thr = 8 since that one performed the best of the 3 overall. As we vary N, the sequential transposer performed better from N = 10 up until N = 800. Then as N get larger after N = 1,000 all the way to N = 10,000 the coarse grain performed better.

This makes sense since the use of parallel threads are most efficient when we have more and more data to work with. When the amount of exchanges is kept low, the sequential transposer works well, even better. Once the matrix gets large enough, then splitting the work up between threads will yield faster execution times.