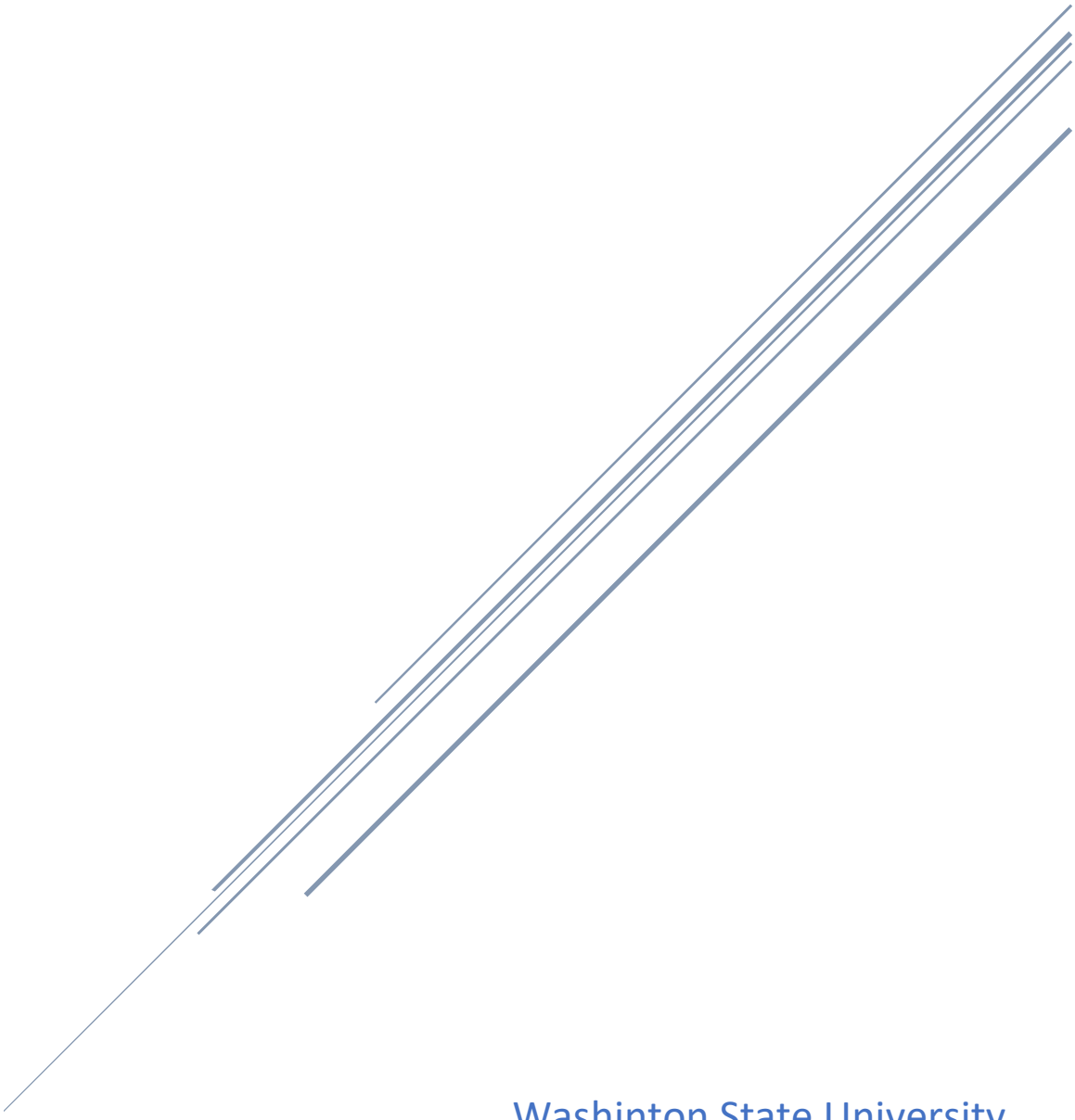
EFFECT OF MULTITHREADING ON MATRIX MULTIPLICATION

By Timothy J Cain



Washinton State University
CPTS360 Systems Programming

Multithreading is a technique in programming that allows multiple processors to execute a single set of code simultaneously. When used correctly, multithreading can greatly decrease the runtime of your program by allowing you to utilize more of the computer's hardware.

The benefit of multithreading is it allows multiple cores to work together on the same program to simultaneously execute the same piece of code to decrease the total runtime. This does not affect CPU runtime because each instruction still takes the same amount of time for the processor, but it *does* decrease the Real runtime because multiple instructions are being executed by multiple cores at the same time. In this experiment, I tested runtime using a program that multiplies two matrixes together and returns the result. All tests were conducted on the ELEC System at Washinton State University Tri Cites.

The first part of this experiment did not utilize any multithreading. I tested runtime by starting with two matrixes of equal size, taking the average of 5 time values for each size and plotting the results on a log-log graph. I started with a four-by-four matrix and doubled the size until 1024x1024. As expected, the runtime increased exponentially as the size of the matrixes increased. I then took the size of the matrixes and plotted them against the average runtime of each size to create the Figure 1-1.

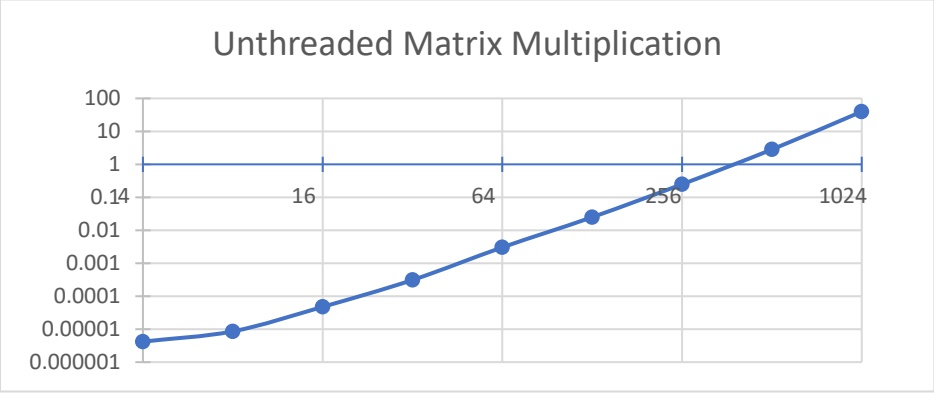


Figure 1-1: Unthreaded Matrix Multiplication

The runtime of the unthreaded program grew linearly as the matrix size increased. This was expected because the time it takes to do each operation does not change, it is the total number of operations that increases between sizes. The runtime ranged from below a second to nearly a minute.

The second part of the assignment utilized multithreading to test how it affected the total runtime. Like the first part, I used matrixes of the same size and compared the effect of using different number of threads and graphed the results. Figure 2-1 show the results of using 1-6 threads and how they affect the runtime of the program. I plotted these using a log-log graph.

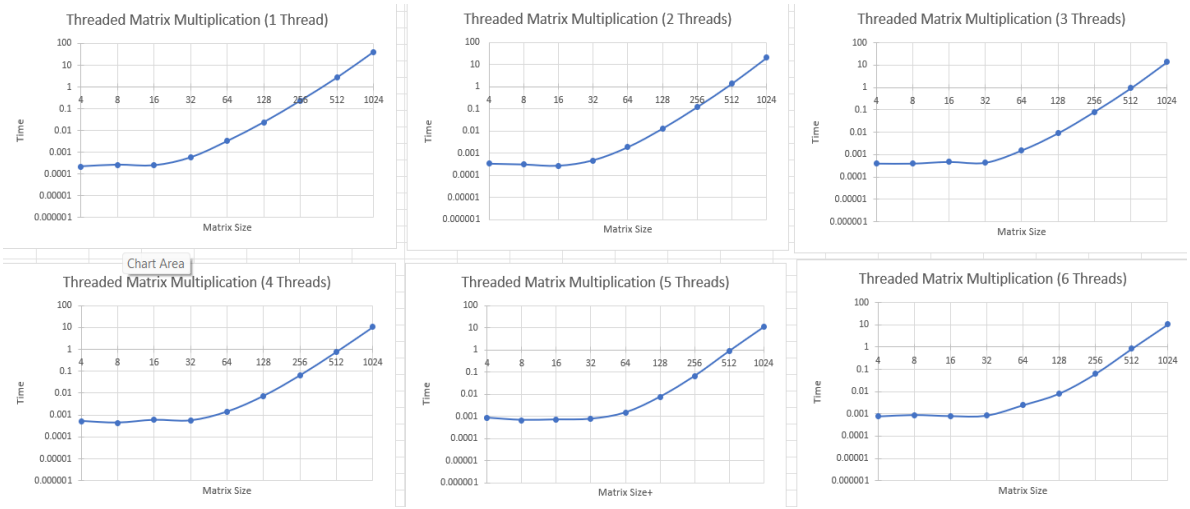


Figure 2-1: Thread 1-6 Run Times

As expected, the addition of more threads caused an overall decrease in runtime for large amounts of data. However, using more threads on smaller matrixes show a slight increase in runtime. This is probably caused by the time it takes a thread to be created because the increase in time was greater with more threads. In addition to this, the number of rows multiplied is not the same between all threads. With lower number of threads, it was common for multiple threads to have multiplied 0 threads, which would account for the increase in runtime because the

program needs to initialize all threads. With small datasets only one or two threads actually multiplied rows, and the other initialization of the other threads was wasted time. Figure 2-2 shows the comparison between the runtimes of different threads.

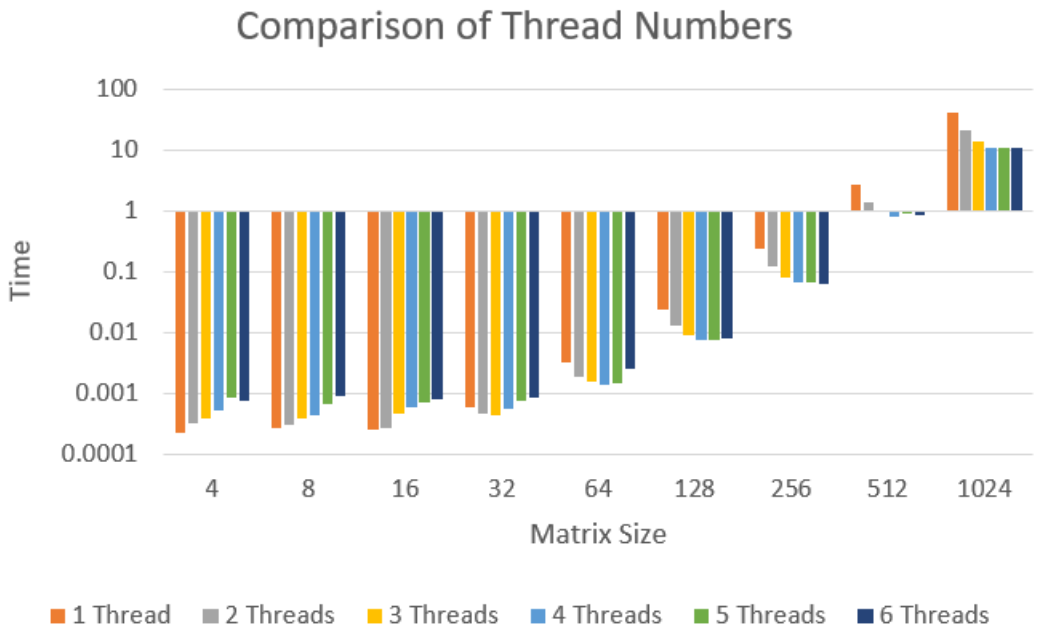


Figure 2-2 Comparison of Thread Numbers

In conclusion, threads are more effective when used on large datasets or programs with many operations. Across all thread numbers, there was a slight increase in time in matrix sizes 4-64, and a negligible decrease in sizes 128-512 with a large decrease in runtime for the 1024 row matrix. There is a drop off in the effectiveness of threads at around 3 threads. This probably because most modern processors can only run up to 2 threads per core and cannot devote all their time to a single process. It is possible to create more threads than your processor can manage at one time, however the cores will need to split their time between the threads and as a result there will be a slight increase in runtime.