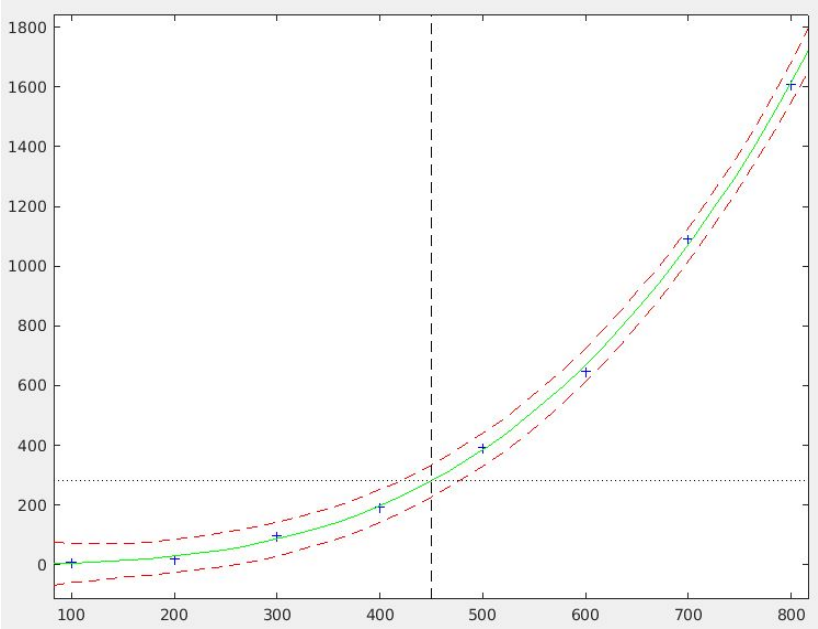
Gentry Atkinson
CS7331, Spring 2019
Homework: Constant Time Matrix Multiply

Matrix multiplication is a standard benchmark for high performance computing systems and is also an excellent benchmark for standard performance computing systems. As the first step of this investigation I developed a simple C++ program to create a 2 matrices of user-defined size, filled them with random integers, and then filled a third matrix with their multiple. Test runs generated the following runtimes:

Matrix Size: n	**Runtime in milliseconds**
100	7
200	21
300	98
400	192
500	392
600	648
700	1092
800	1608

By providing these values to the polyfit function in Matlab, we find that our values can be neatly fit with a 2nd order polynomial curve:



I then altered the original matrix multiplying to instead build 3 matrices using 3 user-input values in ascending order. The multiplier would allocate a number of threads equal to the square of the ratio between the matrix sizes. If parallelism was accomplishing a perfect speed up, then the workload and the performance boost would both be increasing relative to n^2 and we would see all three runs complete in roughly the same amount of time. A test run of the altered program produces the values:

Matrix Size	**Runtime in Milliseconds**	**Threads Allocated**
200	37	1
300	39	2
400	44	4

These runs show us that the speedup is absolutely accounting for the increasing workload as the matrix sizes grow but also illustrate the burden created by thread management. The smallest matrix multiplication takes roughly 50% longer than the original, sequential code took to run while the largest matrix is roughly 4 times faster than its original sequential run.

This small program, while essential just a toy that does not produce any useful product, can still serve to illustrate the conditions under which parallelism can be a useful addition to a program and the conditions under which it is merely a burden to both the system and the developer. Parallelism is a powerful and useful tool