CS475 PA1

In the project I parallelized 3 algorithms. The stencil algorithms seemed to be smoothing out some data in either a 1 dimensional list or a 2 dimensional list. The mat_vec program is calculating one row of the product of two matrices.

All experiments were run on the steamboat.cs.colostate.edu machine. It has 6 hyperthreaded cores on an Intel Xeon E5-1650 processer clocked at 3.6Ghz. The cache sizes are :

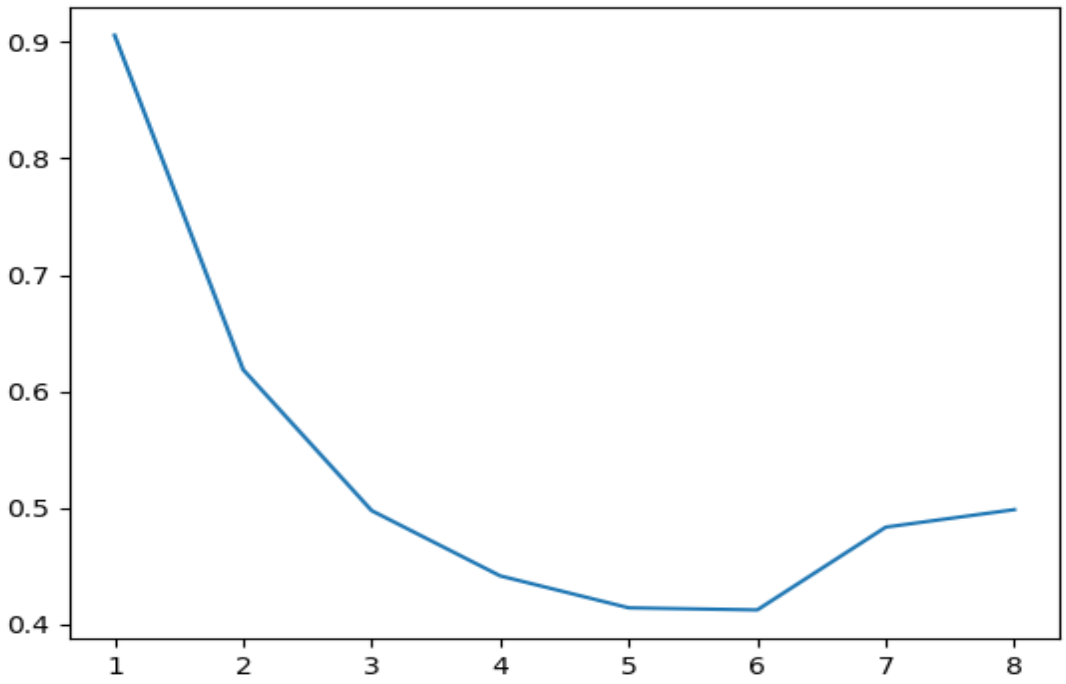
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	15360K

For all the experiments I ran my parallelized versions of the programs with some constant arguments specifying the number of threads from 1 to 8, and taking the average of 7 times to run the program of each. I then graphed the results.

Stencil 1D

In the stencil 1D algorithm, my approach to parallelization was to parallelize the for loop that calculates the smoothing for one iteration of the list. The scheduling I use is static, because the workload should not differ significantly from iteration to iteration. The while loop is dependent on the last iteration and cannot be parallelized.

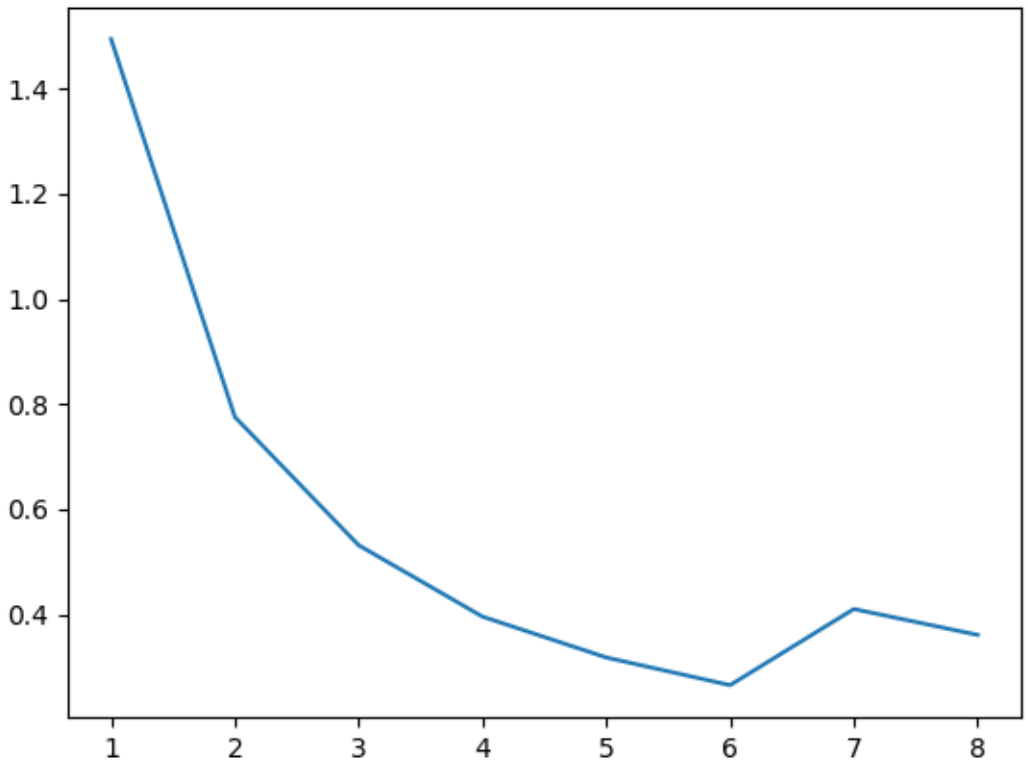
Thread number vs. Time



The speedup that the program gets with more threads decreases nicely and is always greater than 1 since the program is faster than sequential for any thread number greater than 1. At 7 threads the execution time gets longer, which is expected since it is running on a 6 core processor and multiples of 6 should be the most efficient values since less contest switching is needed at those values.

Stencil 2D

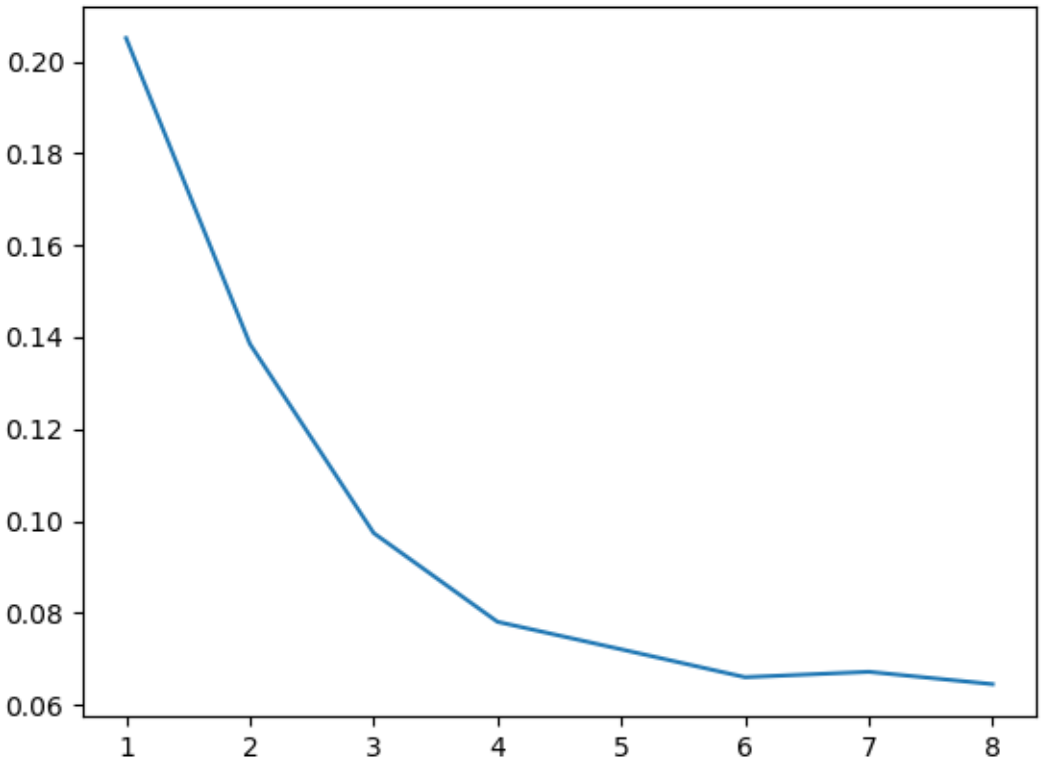
In the stencil 2D algorithm, I used the same idea, but with the doubly nested loop that does the same smoothing calculation for one iteration. Again, the iterations should not differ significantly in workload so static scheduling is best.



Just like stencil 1d, the speedup increases with threads smoothly until it moves from 6 threads to 7 threads because we working on a 6 core CPU. The speedup is always greater than 1, but drops a bit going from 6 to 7 threads.

MAT VEC

In the mat_vec I parallelized the nested for loops that compute the C vector. I chose to parallelize the outer loop for better efficiency. Again, the iterations should have relatively equal workloads so static scheduling is best.



The program runs faster than the sequential version when given more threads, and therefore has a speedup > 1. Going from 6 to 7 there is less of a decrease in speedup than in the stencil algorithms but one still exists.