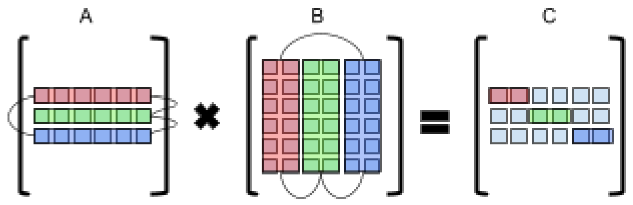
Joel Vastbinder

Parallel Matrix Multiplication

Partitioning Method

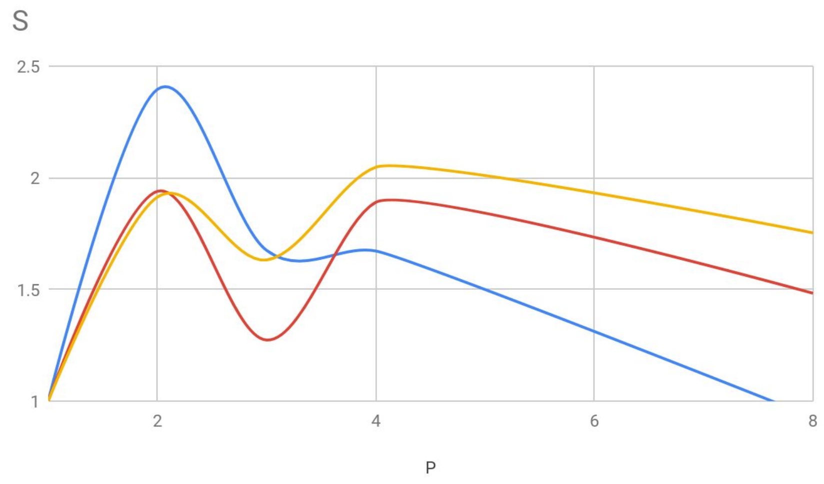
For partitioning for *x* CPUs, each CPU is assigned to m/x of the rows in an m x n matrix A and x/p columns of an n x p matrix B. The parts of the matrices are distributed by a single CPU. These sections map onto their assigned sections of the result matrix C. After all the spaces that can be calculated with the given pieces have been filled, the CPUs pass their data from matrices A and B to the next CPU to calculate the next part of their assigned sectors of C. Once the whole C matrix has been calculated, it is passed to one CPU to be written to disk.

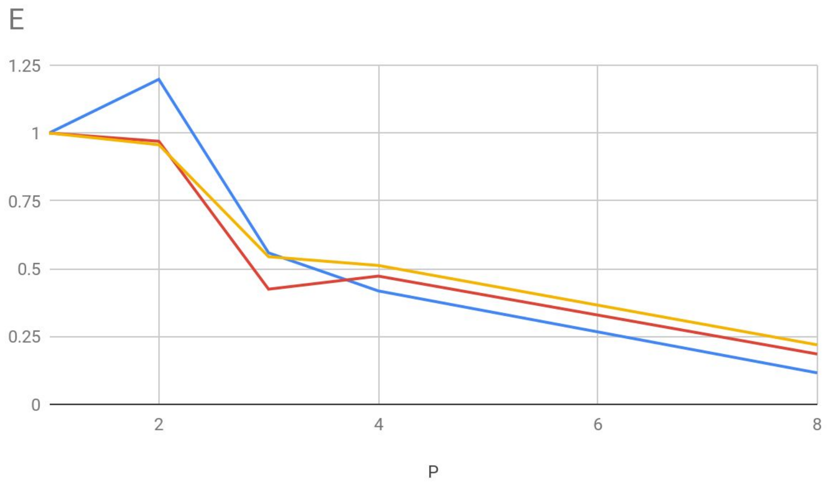


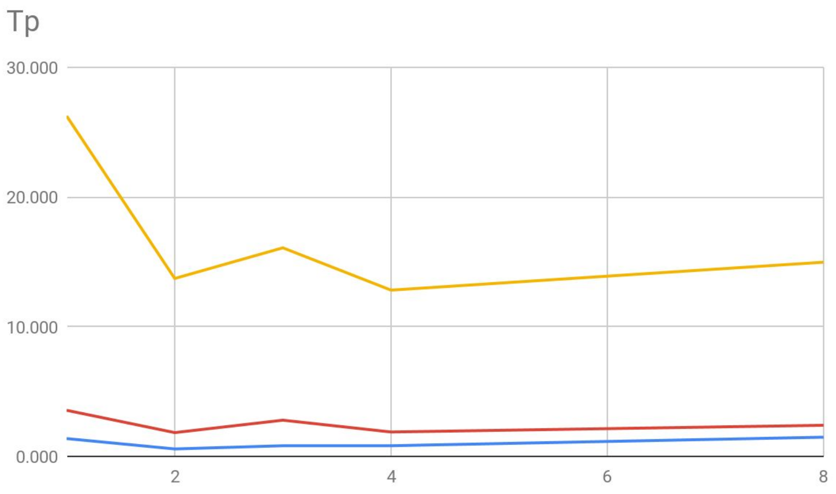
Verification

To verify correctness, sample matrices were tested against a web utility for performing matrix operations found at matrix.reshish.com. The parallel implementation does not work 100% correctly, it outputs a lot of correct numbers, but they are not in the right sequence nor are all the correct numbers output.

Size	P	T_P	S	E
144 x 120, 120 x 288	1	1.371	1	1
	2	0.572	2.396	1.198
	3	0.819	1.675	0.558
	4	0.820	1.672	0.418
	8	1.479	0.927	0.116
1024 x 1024, 1024 x 1024	1	3.552	1	1
	2	1.831	1.939	0.970
	3	2.789	1.273	0.424
	4	1.879	1.891	0.473
	8	2.395	1.483	0.185
2048 x 2048, 2048 x 2048	1	26.246	1	1
	2	13.716	1.914	0.957
	3	16.080	1.632	0.544
	4	12.818	2.048	0.512
	8	14.970	1.753	0.219


S
2.5
2
1.5
1
2
4
6
8
P


E
1.25
1
0.75
0.5
0.25
0
2
4
6
8
P


Tp
30.000
20.000
10.000
0.000
2
4
6
8

Discussion

Performance

My code would not compile or run on the lab machines, so I had to run the tests on my laptop. I attempted both to compile the code using mpicc and transferring the executable file to the lab machine, but neither strategy worked. As a result the times are most likely slower overall, and although it has 4 physical cores, my personal machine seems to dislike to utilizing more than 2 at a time which would account for the decreased speedup as the number of processes increased. There could be additional overhead associated as the number of processes spun up by MPI increased, but most of the decrease in speedup was probably due to the hardware being underutilized.

I was pleased to see that the speedup at least from 1 to 2 processes on each matrix was almost linear. Hopefully if this was run on better hardware that trend would mostly continue at least up through the number of physical cores on the machine.

Surprises

I was surprised by how much the speedup affected even the small matrix. I assumed that the parallel overhead would negate the benefits gained from having multiple processes working on a matrix of that small.

Assignment Difficulty

I think what made this assignment most difficult was how much the amount of work increased from the last two projects. Instead of taking a serial implementation and modifying it to run on multiple threads, we had to add file i/o to the serial implementation, and do a complex operation with memory constraints we had never had before. I worked probably 20 hours on this assignment and did not completely finish.

Part of that difficulty came from thinking the next part would be easy once I overcame the current hurdle. For example I figured once I got the matrices properly distributed to the different processes, the multiplication would be relatively straightforward. Keeping track of array indices and figuring out where in the C matrix that process's A and B contributed to was very difficult. I ended up in quite a few situations where I realized the next piece was just as if not more complex than previous pieces.