Kevin Cho
A20393339

Pthreads

Correctness

For Pthreads, I know that my solution is correct because I divided N evenly into different workloads for each thread based on the number of threads. Each thread then receives a “range” structure which contains a “start” and an “end” value for the normalization row. The “start” and “end” value determines which parts of the matrix each thread works on, so that they do not interfere with calculations that other threads are doing. Therefore, I had no race conditions in my solution since the normalization row in the matrix multiplication never intersected which meant row and column also never intersected.

Alternative versions

In an alternative version, I tried to take the whole matrix and assign it to all threads. Then I would have each thread run calculations on the matrix while skipping every “numthread” row. Doing this ensured that none of the threads ever touched the same row and column, but performance for this version suffered on higher threads. For example, if I wanted to calculate N = 1000, the time compared to serial would reasonably decrease for every increase in threads up to 8 threads. Once I went past 8 threads performance started getting worse than the serial implementation and I believe this occurred because of the high overhead of assigning the matrix N to every thread. With my correct version, I have no such problem, and it properly has speedup up to 32 threads.

OpenMP

Correctness

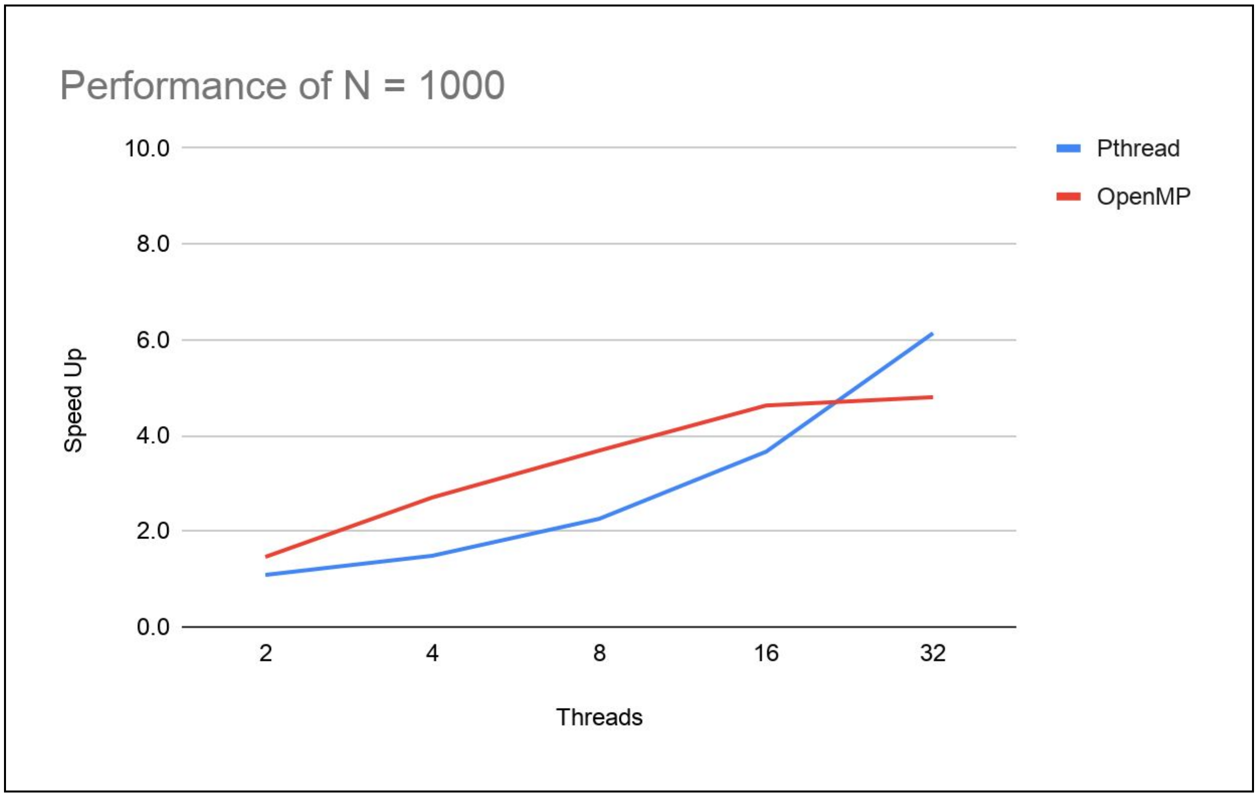
For OpenMP, I know that my solution is correct because I made the multiplier, row, and column private variables. This ensures that none of the threads would be overwriting any of the numbers in matrix A or B.

Alternative versions

Also I had my pragma within the norm loop because it was much more efficient to have a single thread run through the normalization rows once and then have other threads do the calculations. If the pragma enveloped the entire gaussian elimination loop then I would have gotten the same problem as my alternative pthreads version where I would have gotten degrading performance on higher thread counts. I further set the default number of threads to 2 since the default for the pragma would have been 32 which does not work well for small N sizes.

Performance Analysis

Data can be found in README

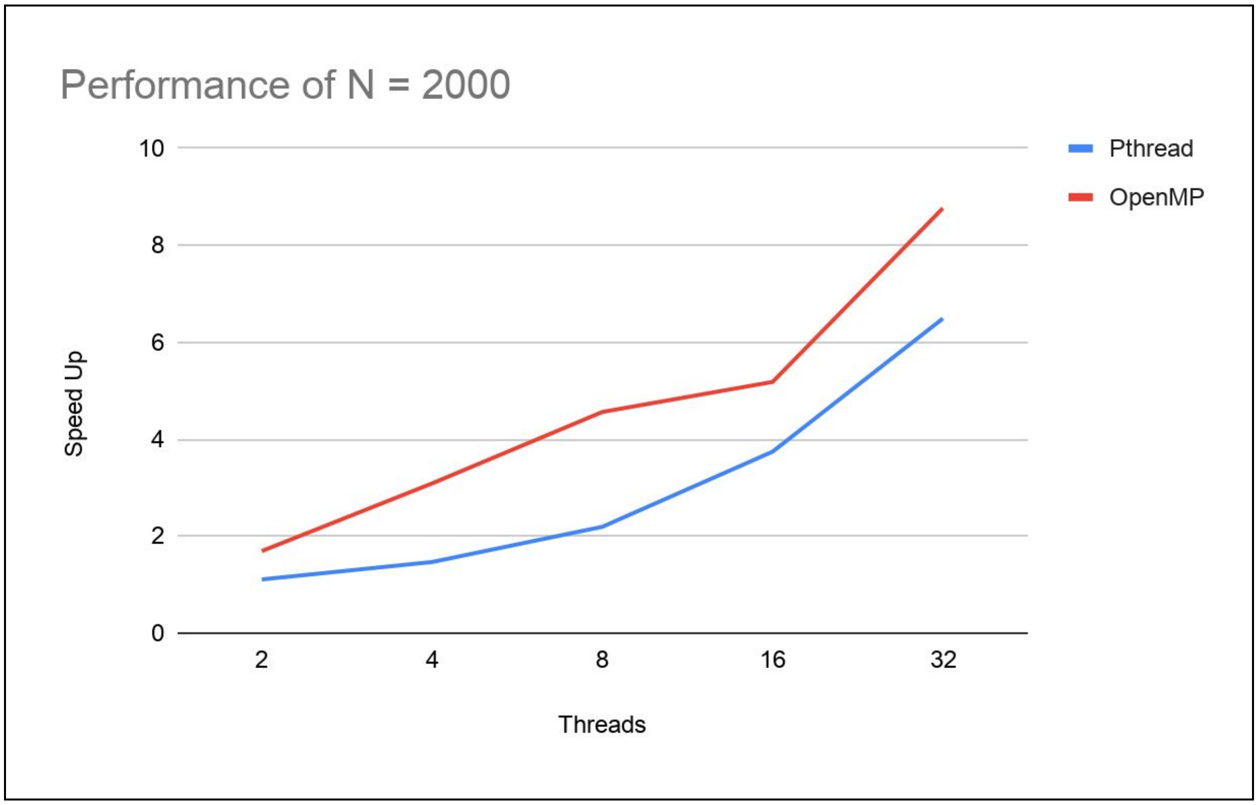


Serial speed = 1104.96 ms

Pthread	Threads	Speed	Speed up		OpenMP	Threads	Speed	Speed up
	2	1016.070	1.1			2	754.940	1.5
	4	742.736	1.5			4	408.370	2.7
	8	489.042	2.3			8	299.822	3.7
	16	301.706	3.7			16	238.789	4.6
	32	180.029	6.1			32	230.158	4.8

Pthread’s 2 thread performance is not much better than serial’s speed but there is a speedup when increasing the cores.

OpenMP’s baseline performance is higher than Pthread, but around 32 threads, Pthread begins to gain linear speedup while OpenMP’s speedup plateaus.



Serial Speed = 7432.93 ms

Pthread	Threads	Speed	Speed up		OpenMP	Threads	Speed	Speed up
	2	6703.00	1.1			2	4389.920	1.7
	4	5064.02	1.5			4	2406.510	3.1
	8	3387.43	2.2			8	1628.980	4.6
	16	1985.11	3.7			16	1434.080	5.2
	32	1145.00	6.5			32	847.869	8.8

Once again OpenMP’s baseline performance is better than my implementation of Pthread. Both programs already outspeed serial’s performance at 2 threads, and both programs begin achieving linear speedup at 32 threads.