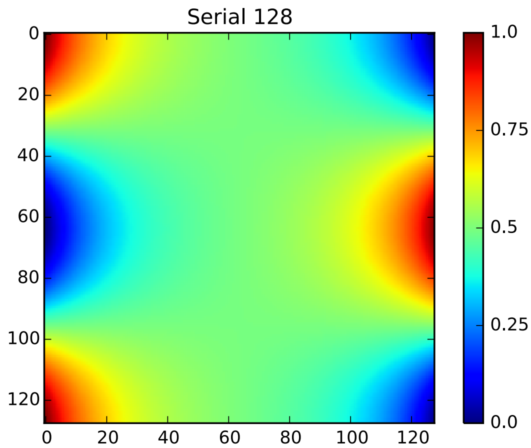
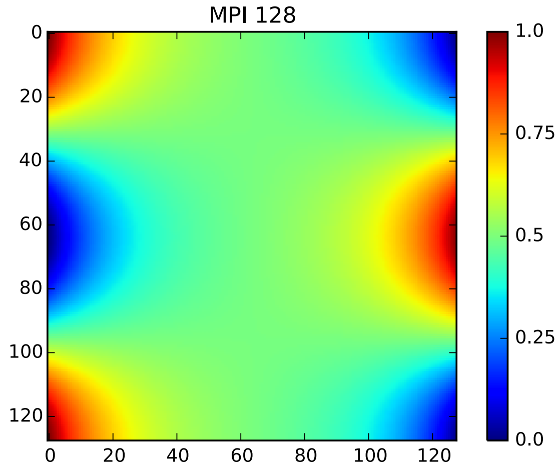
Time (Seconds)	Grid Size: 128	Grid Size: 256	Grid Size: 512
Serial 1 Thread	10.4304	163.806	2653.3
OpenMP 1 Thread	15.1437	243.05	4855.12
OpenMP 2 Threads	7.72278	122.377	2157.32
OpenMP 4 threads	4.0732	63.4796	982.849
OpenMP 8 threads	2.50784	33.5746	501.669
MPI 1	2.69359	42.5622	675.289
MPI 2	2.55984	23.3759	346.412
MPI 4	1.89915	12.1464	181.079
MPI 8	1.62562	6.88356	97.0981
MPI 16	1.52297	4.53383	56.9634

Volume Avg Temp	Grid Size: 128	Grid Size: 256	Grid Size: 512
Serial 1 Thread	0.497325	0.497207	0.497147
OpenMP 1 Thread	0.497325	0.497207	0.497147
OpenMP 2 Threads	0.497325	0.497207	0.497147
OpenMP 4 Threads	0.497325	0.497207	0.497147
OpenMP 8 Threads	0.497325	0.497207	0.497147
MPI 1	0.497325	0.497207	0.497147
MPI 2	0.497325	0.497207	0.497147
MPI 4	0.497325	0.497207	0.497147
MPI 8	0.497325	0.497207	0.497147
MPI 16	0.497325	0.497207	0.497147

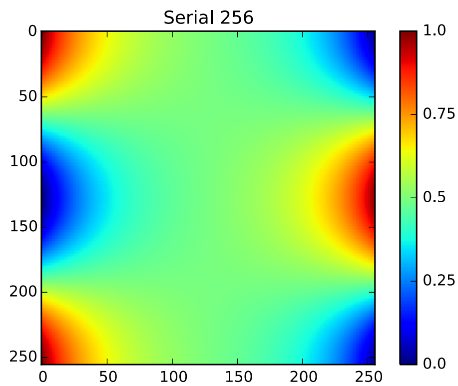
Heatmaps:

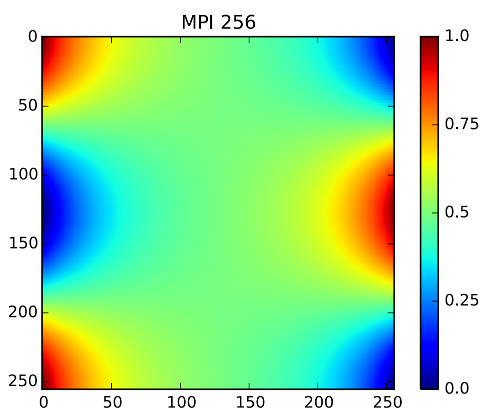
Grid Size of 128:



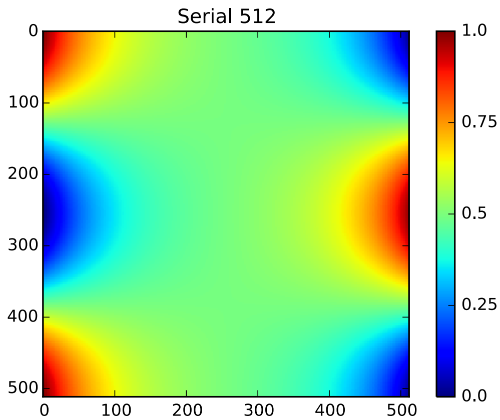


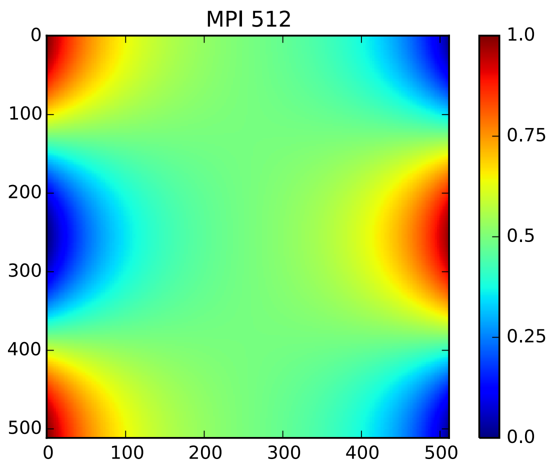
Grid Size of 256





Grid Size of 512





Comparison of OpenMP and MPI:

Starting from serial code, one obvious advantage of using OpenMP is that the code requires only very minor adjustment. This is clearly beneficial in terms of efficiency on the coder's working time. However, looking at the table of times, using OpenMP does add significant overhead in terms of computational time, which is a negative. Using 1 thread adds a significant increase in time, which I don't understand (is it just checking whether the parallel flag is being used that accounts for the difference? I'm not sure what else it could be). MPI doesn't suffer from such bad overhead, and actually even provides a speed-up only using one processor, which is also striking. MPI was particularly useful for this problem because the programmer has precise control over each process and can evenly distribute workload. It was also nice because the parallelization wasn't just over the for loops. It's also worthy of noting that although not discussed in this assignment, I would think that MPI uses much more memory than OpenMP, because each process gets its own section of memory, so it is as if we are running that program (# processors) time, and using n times as much memory, whereas in OpenMP each thread shares memory. Because we're updating values between two arrays in this assignment, I thought the ghost columns in MPI made the most conceptual sense to me, and was also easier to think about because I didn’t have to worry about writing over memory that another thread was going to then read while thinking it hadn’t been updated, as would be possible in OpenMP. However, it was of course very hard to re-write the code to incorporate MPI, which was a disadvantage (but a huge learning opportunity) of this approach.