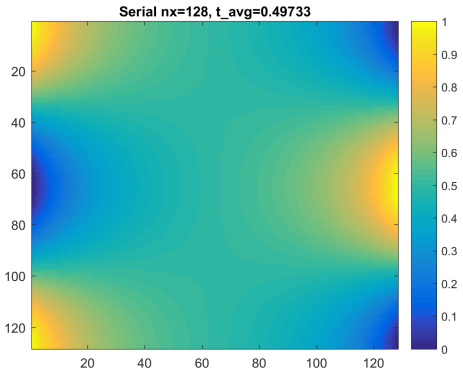
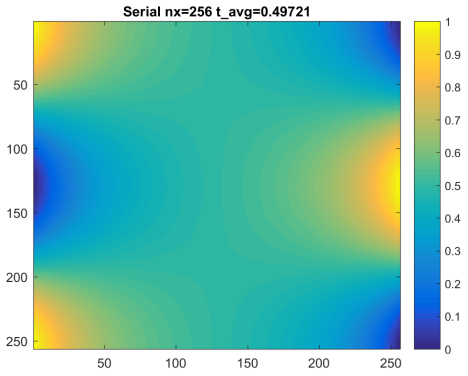
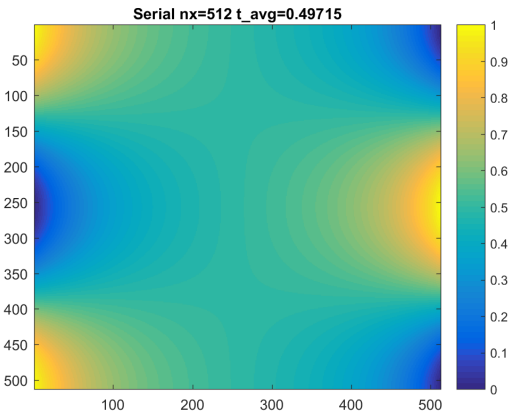
1 HW4 Summary

The figures below are the final arrays produced by running `heat_serial`, `heat_omp` and `heat_mpi`. Note that I encountered segfault problems running `heat_mpi` on Adroit even though I was able to run and test the program on my local system where it was working. The figures I managed to produce locally for MPI are included below as well as their timings, even though they does not serve for comparison purposes since it was run on a different hardware system.

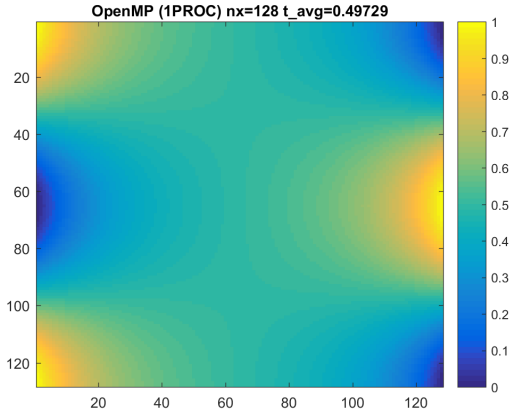
Serial

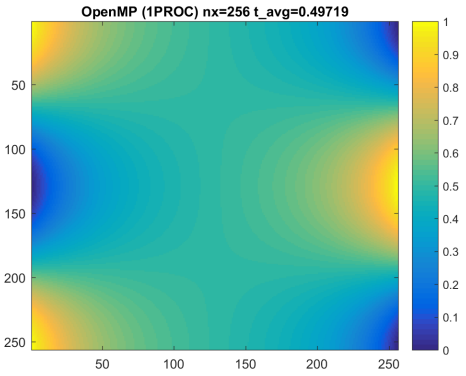


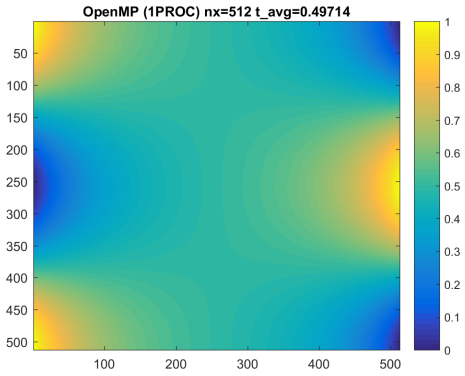


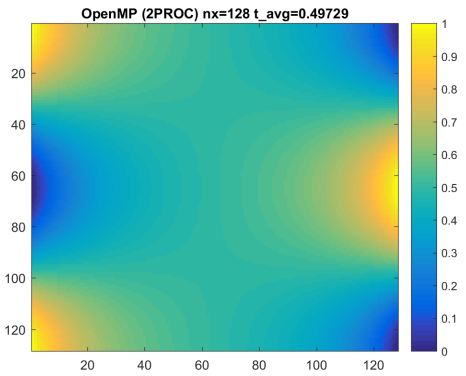


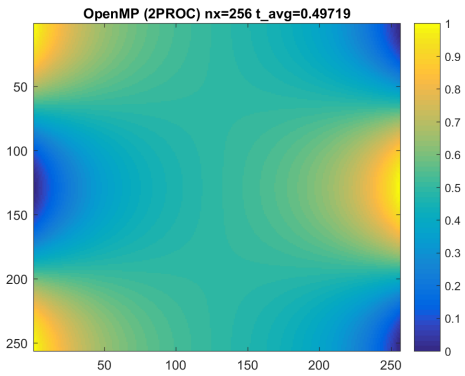
OpenMP

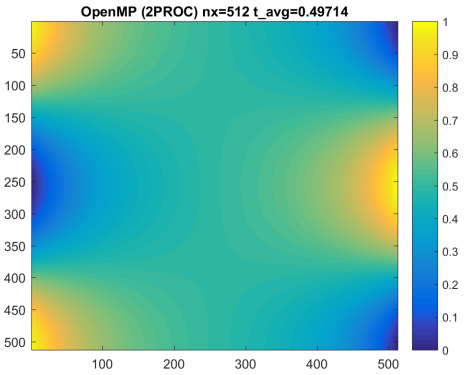

OpenMP (1PROC) nx=128 t_avg=0.49729
20
40
60
80
100
120
20
40
60
80
100
120
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

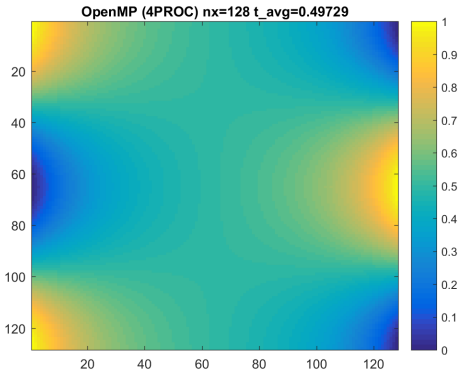

OpenMP (1PROC) nx=256 t_avg=0.49719
50
100
150
200
250
50
100
150
200
250
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

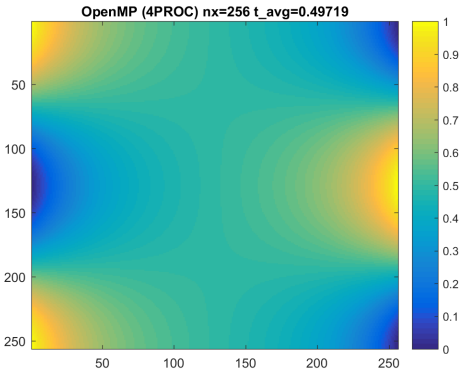

OpenMP (1PROC) nx=512 t_avg=0.49714
50
100
150
200
250
300
350
400
450
500
100
200
300
400
500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

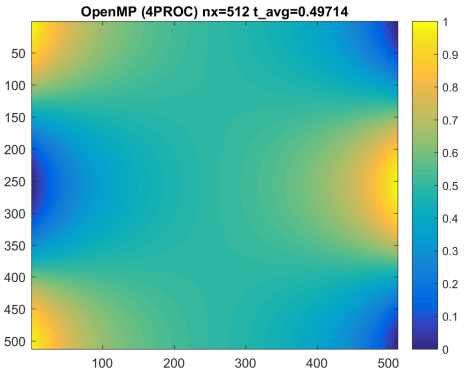

OpenMP (2PROC) nx=128 t_avg=0.49729
20
40
60
80
100
120
20
40
60
80
100
120
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

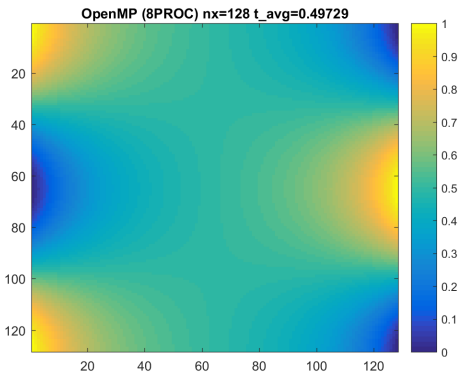

OpenMP (2PROC) nx=256 t_avg=0.49719
50
100
150
200
250
50
100
150
200
250
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

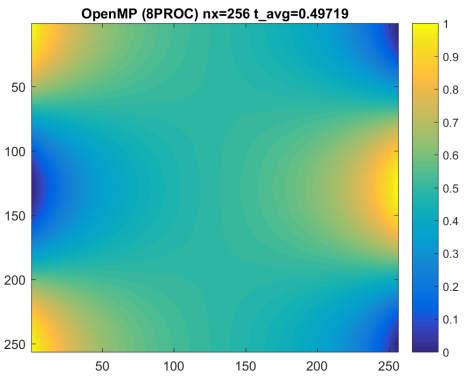

OpenMP (2PROC) nx=512 t_avg=0.49714
50
100
150
200
250
300
350
400
450
500
100
200
300
400
500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

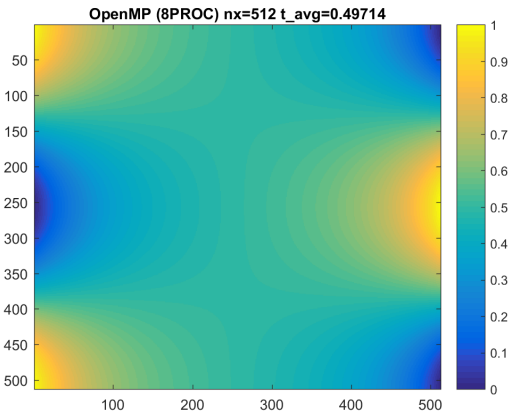

OpenMP (4PROC) nx=128 t_avg=0.49729
20
40
60
80
100
120
20
40
60
80
100
120
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


OpenMP (4PROC) nx=256 t_avg=0.49719
50
100
150
200
250
50
100
150
200
250
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

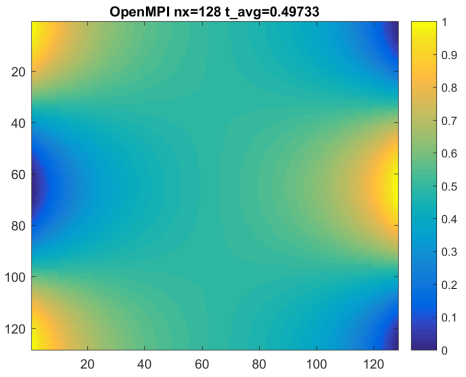

OpenMP (4PROC) nx=512 t_avg=0.49714
50
100
150
200
250
300
350
400
450
500
100
200
300
400
500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

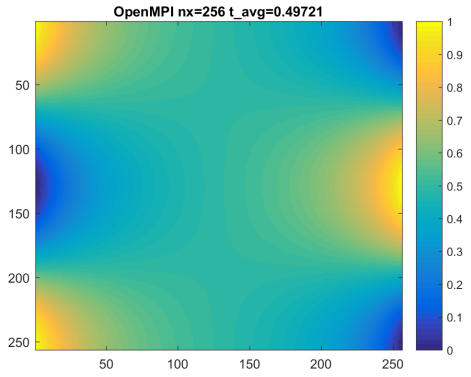

OpenMP (8PROC) nx=128 t_avg=0.49729
20
40
60
80
100
120
20
40
60
80
100
120
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

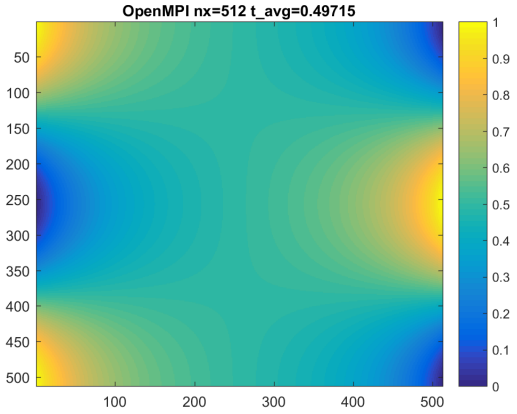

OpenMP (8PROC) nx=256 t_avg=0.49719
50
100
150
200
250
50
100
150
200
250
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


OpenMP (8PROC) nx=512 t_avg=0.49714
50
100
150
200
250
300
350
400
450
500
100
200
300
400
500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

OpenMPI







Timing and Temperature Average

Serial <nx>	**Time** [s]	**Temp. Avg.**
128	14.84	0.497325
256	238.9	0.497207
512	4198.22	0.497147

Table 1: Serial

Array Length	OpenMP 1PROC		OpenMP 2PROC		OpenMP 4PROC	
<nx>	**Time** [s]	**Temp. Avg.**	**Time** [s]	**Temp. Avg.**	**Time** [s]	**Temp. Avg.**
128	15.1401	0.49729	8.07095	0.49729	4.29149	0.49729
256	243.407	0.497189	127.235	0.497189	66.2138	0.497189
512	4266.07	0.497137	2314.11	0.497137	1191.14	0.497137

Table 2: OpenMP

Array Length	OpenMP 8PROC	
<nx>	**Time** [s]	**Temp. Avg.**
128	2.40089	0.49729
256	34.5503	0.497189
512	599.978	0.497137

Table 3: OpenMP

OpenMPI <nx>	**Time** [s]	**Temp. Avg.**
128	11.4013	0.497325
256	186.598	0.497207
512	3041.26	0.497147

Table 4: OpenMPI. Note that this has been run locally

OpenMP and OpenMPI

While OpenMP is very simple to use in C++ and provides a significant time advantage compared to the serial run, OpenMPI offers even faster runtime (which I unfortunately was not able to see due to Segfault errors when running the program on the Adroit Cluster). However, OpenMPI is significantly more laborious to implement since it requires passing data between threads, where OpenMP simply has access to all variables. This meant reformatting much of the code as well as changing code which was not able to run in MPI environment. With each thread having its own memory and not having any shared memory the division of the array required addition of halo columns in order to pass information. The timing of this process is also required to be controlled so that information is updated at the same time. For the future I found that it might be easier to implement OpenMPI while writing the code, since it was very laborious to adapt the serial code to OpenMPI after.