Joshua Wallace

HW4

The plots I made for the analysis portion of this homework are 3-d ``wire'' plots. I think they make sense once you see them. I did not know how to put axes on them so I wish to explain here what the axes are on each of the plots. The vertical axis is the temperature T for each plot, while the bottom left axis in the foreground is x and the bottom right axis in the foreground is y.

Figure 1 shows the temperature as a function of position for the serial run with grid size of 128. Looks pretty good. Figure 2 shows the same for the serial 256 grid size run and Figure 3 is the same for the serial 512 grid size run. The average temperatures for each run are as follows: 128, 0.496830306313; 256, 0.49695963511; 512, 0.497022899482.

The OpenMP runs had the same exact output data as the serial runs no matter how many numbers of threads I set for the OpenMP run. Thus, I decided to not waste space or time by including the same looking figures again. Please refer to Figures 1,2, and 3 for the corresponding plots for grid size of 128, 256, and 512, respectively, irrespective of number of threads. Similarly the average temperatures for the various runs all correspond with the temperatures I gave for the serial cases.

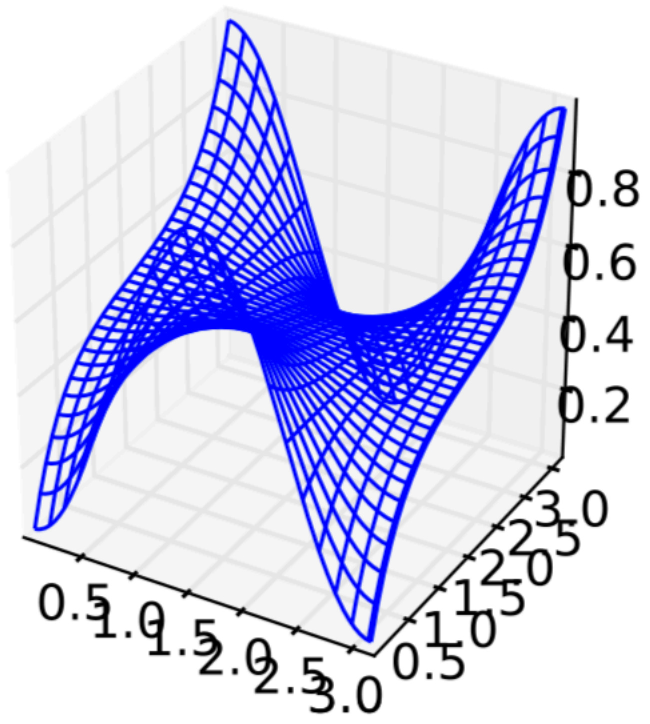

Figure 1: Temperature as a function of position for the serial run with grid size of 128. The vertical axis is the

temperature T, while the bottom left axis in the foreground is x and the bottom right axis in the foreground is y.

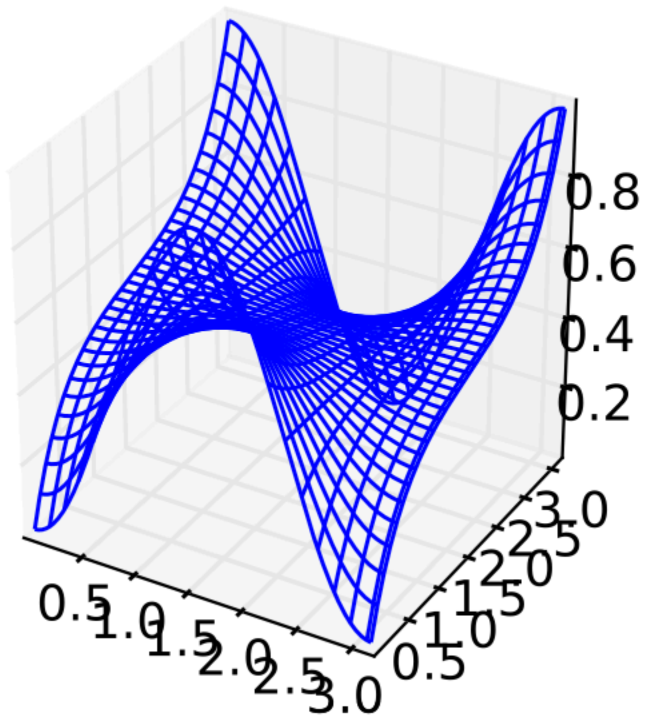

Figure 2: Temperature as a function of position for the serial run with grid size of 256. The vertical axis is the temperature T, while the bottom left axis in the foreground is x and the bottom right axis in the foreground is y.

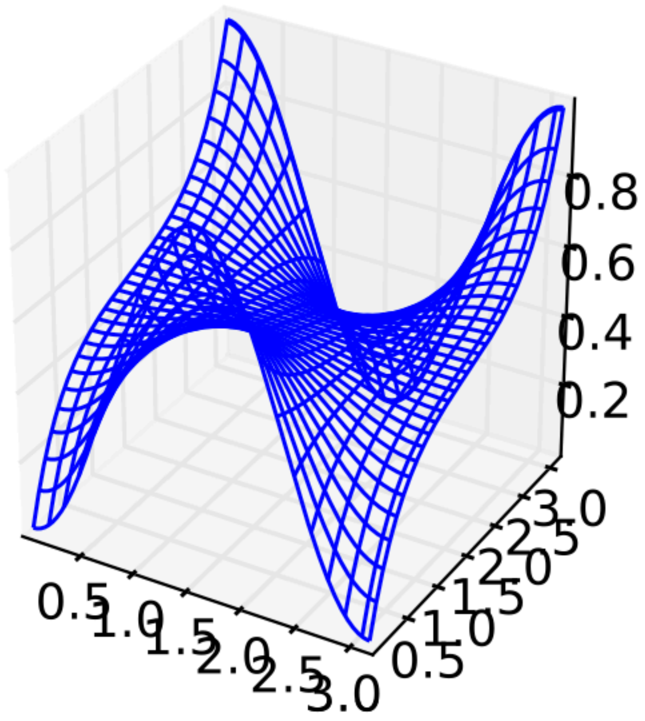

Figure 3: Temperature as a function of position for the serial run with grid size of 512. The vertical axis is the temperature T, while the bottom left axis in the foreground is x and the bottom right axis in the foreground is y.

The MPI runs also had the same plots, the same data, and the same exact average temperatures as the serial and OpenMP runs, independent of the number of processors the run uses .

Generally increasing the number of processors increased the speed of each run. I did not time my serial runs but my single thread OpenMP and single processor number MPI run can serve as a proxy for the serial runs. I will quote serial run values as I remember them. Specifically, for nx=128 I had the following run lengths (in seconds):

N processors:	1	2	4	8	16
Open MP:	28	22	15	6	
MPI:	60	31	16	9	5

For nx=256

N processors:	1	2	4	8	16
Open MP:	442	349	174	89	
MPI:	954	478	234	122	64

For nx=512

N proccesors:	1	2	4	8	16
Open MP:	6689	5569	2788	1405	
MPI:	15125	7676	3733	1928	1150

So it is clear that increasing the number of processors, for a given code for a given nx, increases the run speed, but in each case for a given number of threads/processors OpenMP is faster than MPI. I believe this is because the MPI threads all access the same memory and don't have to spend time talking to each other and sharing messages, while MPI has to deal with this. If I remember right, the serial run is faster than the 1 thread/processor runs in each case. This makes sense because the serial run doesn't have to deal with the overhead of setting up a parallel program only to not take advantage of the parallelism.

Another thing to note is that the MPI run times decrease almost entirely in half every time we double the number of processors, while the OpenMP runs don't show such favorable scalability. This is because, as I wrote it, the OpenMP code sets up the parallel processes at every timestep, so there is overhead associated with every single timestep. MPI, since it runs parallel basically the whole time, doesn't have to deal with this. In this way MPI is advantageous to OpenMP as far as scalability goes, although it still lags OpenMP for a small number of processors. However, collecting the final data from MPI is more difficult than OpenMP since the latter accesses and prints a memory space in the same manner than the serial code does while the domain decomposition of the MPI implementation requires a more complicated way of printing the data. **To handle MPI's output, I had each processor print its data separately then combined the data post-run with a shell script.** In this way OpenMP is easier to use. OpenMP was also MUCH MUCH MUCH easier to write than MPI.

PLEASE ALSO NOTE that I used a Python script post-run to calculate the average temperature in each case. That is why such code is not included in my C source code.