Abelardo Riojas
ISC 4933 2D Heat Equation MPI
Prof. Wang Spring 2022

2D Heat Equation MPI

Introduction:

In this project we use MPI to solve the classic 2D Heat Equation:

$$\frac{\partial H}{\partial t} - \left(\frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2}\right) = 0$$

With parameters:

Grid size = (100, 100)
delta_x = .03
delta_y = .04
delta_t = .0001
T = 20
a (diffusion coeff) = 1

Our initial and boundary conditions are:

Inside = 0.0
Top, Left, and Right sides = 1.0
Bottom side = 0.0

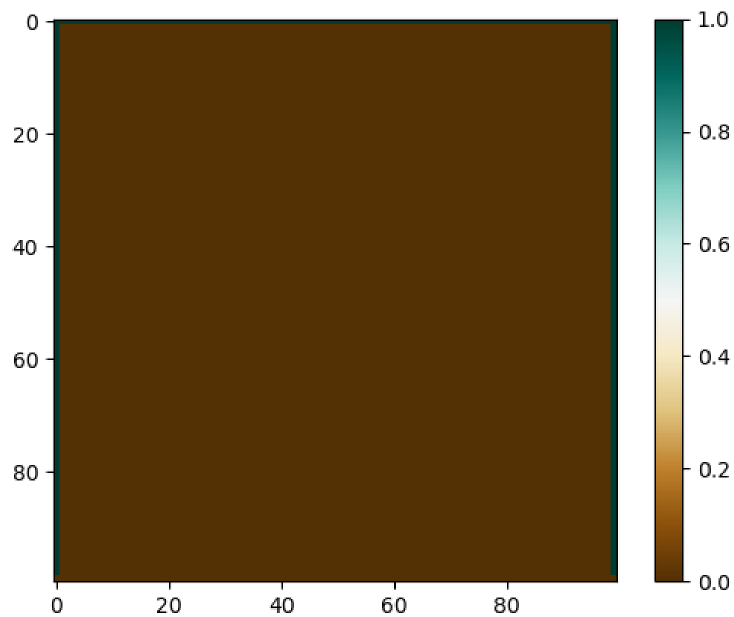
This gives us an approximation for the diffusion of heat on a thin rectangle of size 3x4 using a 100x100 grid. As for the parallelism, the exchange between sub-boundaries is done by **exchange.py** which sends and receives the boundaries in a clockwise manner (send to up, receive from down, send to down, receive from up, send to left, receive from right, send from right, receive from left). In **parallel.py** we create a class *Parallel* which initializes the parallel communication scheme. We use the helpful MPI.Create_cart method to set up a cartesian coordinate system and define operations like nup and ndown to shift communication between sub-boundaries. **evolve.py** does the typical finite difference method for 2D heat using numpy. In **heat_io.py** we define write_field which collects sub arrays and plots them as a full field using matplotlib. We initialize the problem using the initialize function from **heat_setup.py.** Everything happens in **heat.py.** Our field is initialized, we initialize exchanges, evolve the inner system, finalize exchanges, and then evolve the edges for each timestep**.** If iter % 1000 gives us 0 then we plot the current state of the system. We set up t0 =

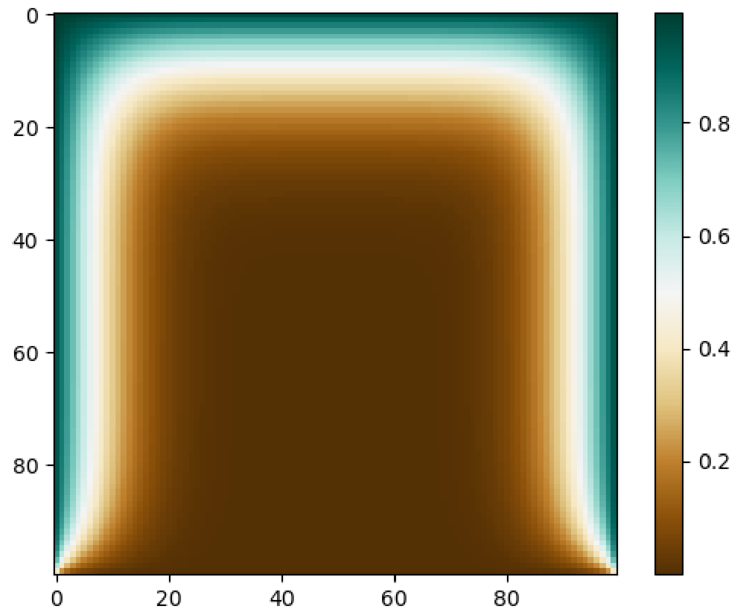
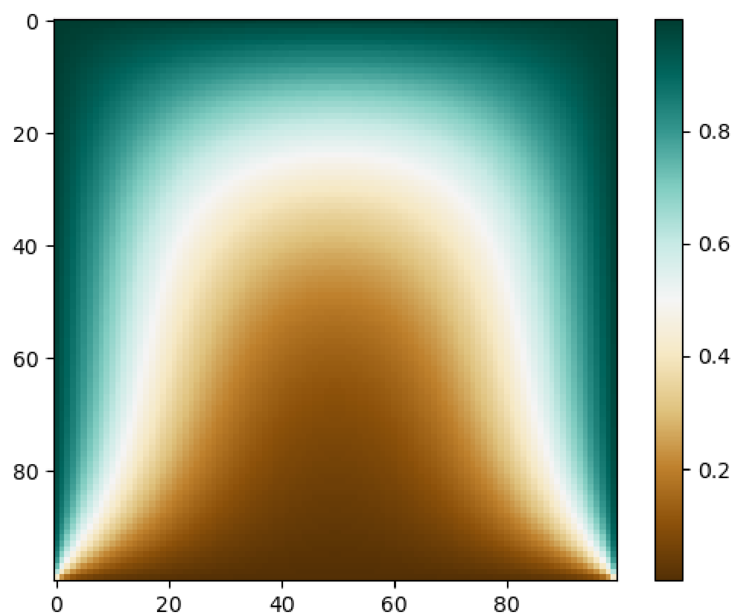
time.time() right after initialization, and stop after we've completed all of our time steps for the final running time.

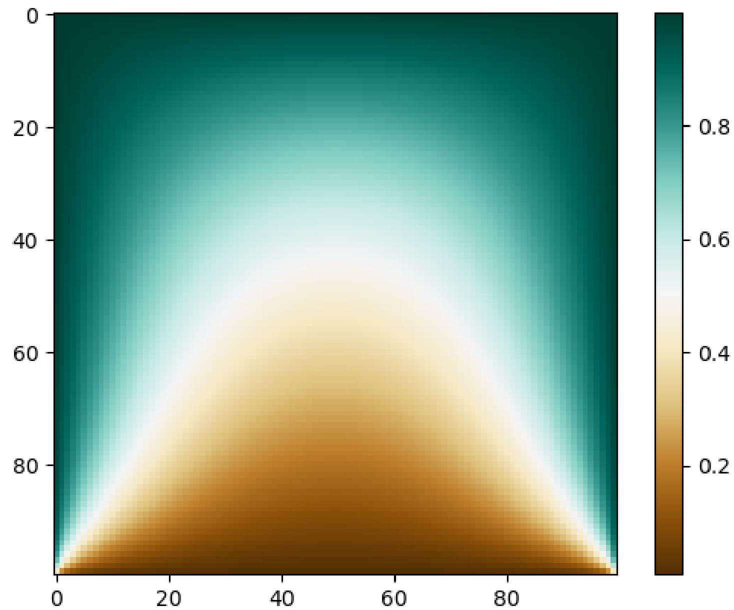
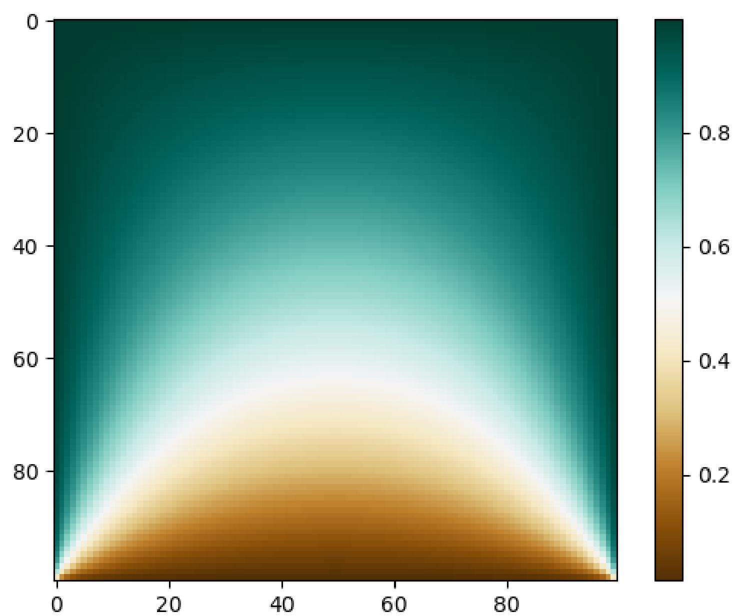
Run the program with

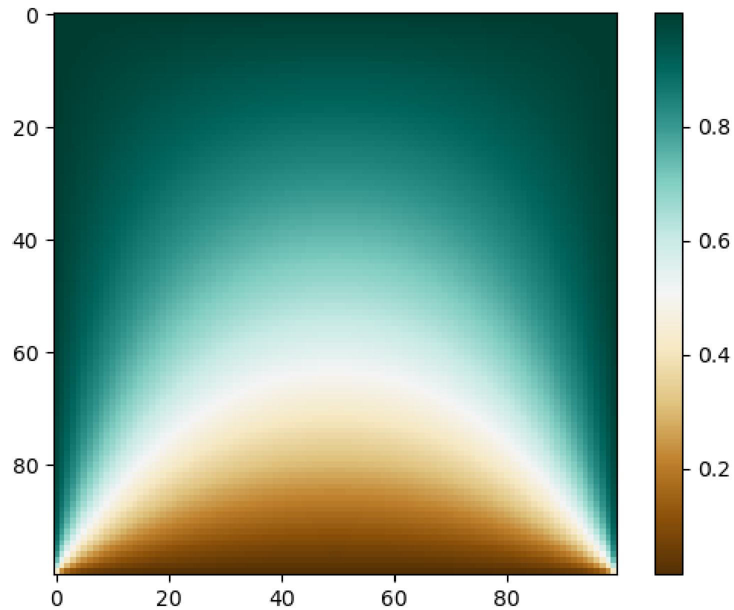
mpirun -np <processes> python heat.py <grid_x> <grid_y> <num_timesteps>

Below are a few of the evolved states at different time intervals using -np 4 (the system evolves very quickly so changes start diminishing after
t ~= 50000)

Timestep	System
0 (initial)	

1000	0 20 40 60 80 0 20 40 60 80 0.8 0.6 0.4 0.2
5000	0 20 40 60 80 0 20 40 60 80 0.8 0.6 0.4 0.2

10000	
50000 (pretty much at equilibrium)	

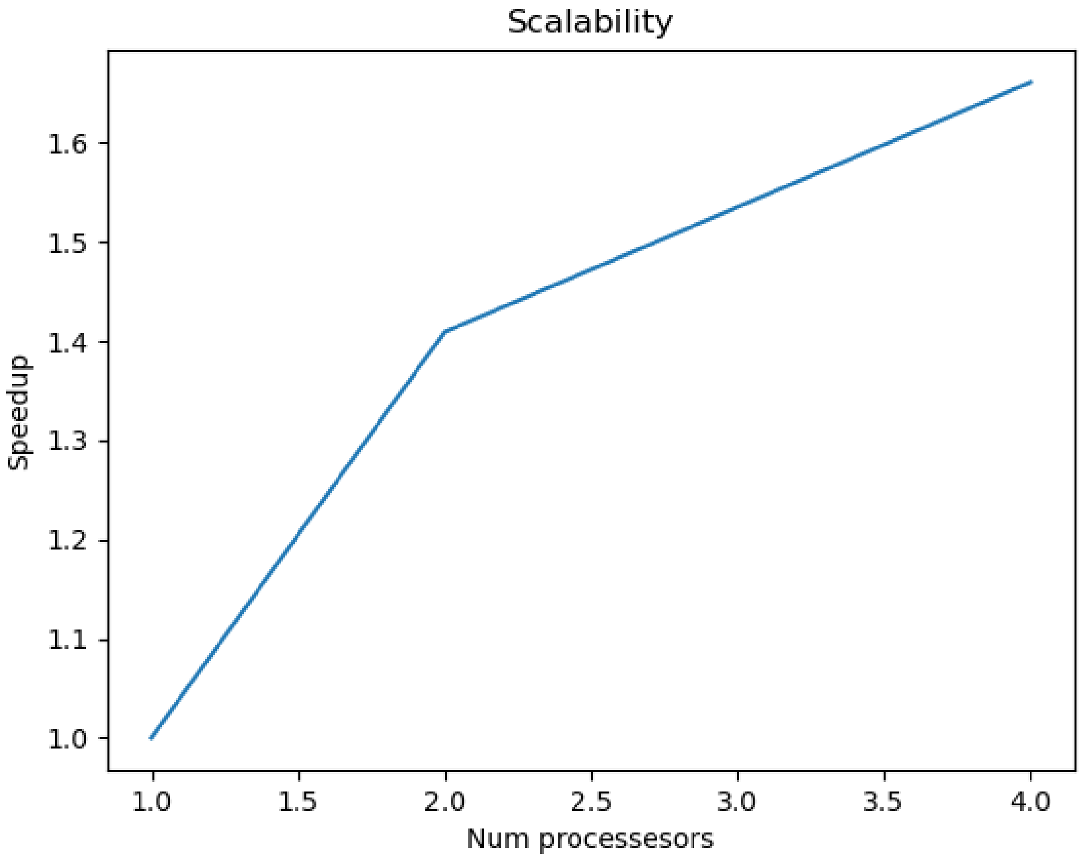
At time T = 20, (t=20000)	

And the running time vs the number of processes (only plotted last timestep):

My machine has 6 cores. And the code only works with processes that are factors of 100 to cleanly cut up the grid. Therefore, the code was only tested with 1, 2 and 4 processes.

Num processors	Elapsed time
1 (serial)	66.51 seconds
2	47.20 seconds
4	40.04 seconds

Not much of a scalability plot but here you go.


Scalability
Speedup
Num processesors
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0