CSC 441 Program 2

Daniel Albl

Ping Pong:

I tested the performance of the school lab machines running MPI, using the "ping-pong" method, where a message is sent from machine A to machine B, and another from B to A. This was done with messages of 20, 40, 60, 80, and 100 megabytes, and using MPI's blocking, non-blocking, and "sendrecv" methods. The time elapsed was measured with calls to MPI's MPI_Wtime(). Each value below was obtained by running the program 5 times and taking the median. The time elapsed had very high variance, so I chose the median over the mean to prevent outliers from skewing the data.

megabytes	Sendrecv	nonblocking	blocking
20	402.488	381.531	348.413
40	628.447	657.597	627.493
60	1010.1	893.126	927.098
80	1368.12	1503.27	1389.32
100	1738.3	1810.79	1835.63

Linear Regressions:

Non-blocking: **y = 18.521x - 61.995**

SendRecv: **y = 17.056x + 6.1019**

Blocking: **y = 18.681x - 95.287**

Based on this data, the bandwidth for all methods appeared to be around 1/18 mb/ms or 56mb/s. The "latency" appears to be negative for blocking and non-blocking, which of course is non-sensical. I would attribute this to the noisiness of the data.

This program did not require any real testing as nothing was actually being computed.

```
$ make
$ mpiexec -np 2 ./ping_pong <num bytes>
```

Conway's Game of Life:

I implemented a parallel, distributed memory version of Conway's game of life using MPI. The grid of cells was divided horizontally into sections that were each handled by separate machines. Each machine has a "subGrid" class that runs the game for its given section of the grid. Because the updating of a given cell requires the 8 neighbors around it, each subGrid also stores the cells that border the section. The bordering cells must be calculated by the neighboring sections, and then sent between machines using MPI's non-blocking send and receive.

The program initializes the grid to be of a given size, and have a given number of alive cells. These alive cells are randomly shuffled throughout the grid using the Fisher-Yates shuffle algorithm. The game is run for a given number of iterations and prints the state of the game after a given number of runs.

Testing for this program was basically done visually by examining the output, and verifying that it does in fact follow the rules of the game.

```
$ make

$ mpiexec -np <num processes> ./life <i> <j> <k> <m> <n>

        i: live cells, j: iterations, k: iterations between prints, m: rows, n: columns
```

The files life.cpp and subGrid.h are for the Conway program, and ping_pong.cpp is for the ping pong program. Both programs are compiled by the single Makefile included. More detailed information about the programs can be found in the comments within the source files.