Program 2

Riley Kopp

15 November 2020

Ping Ping

Program Description:

Ping is a standard tool used in the communication world to determine response time of a particular node on the network. In distributed computing, Ping is used for measuring response time (latency) and bandwidth between two given nodes.

For our implementation, Ping had to measure the response time between 2 nodes within the MPI cluster with packet size and repetitions passed in as arguments.

Using the Program:

To run PingPong, first enter into the directory and type make.

```
# Build the project
> cd ping_pong
> make
```

To run the program, type the following command:

```
> mpiexec -n 2 ping_pong <iterations> <packet_size>
```

Be sure to check that your host file's configuration only has one slot available on each host. (Attached in the directory is an example.)

The program will output 3 unlabeled message passing times, (Which made writing a script to do the data analysis really easy in python) below is a labeled example of the output:

```
> mpiexec -n 2 ping_pong 100 1000
90 # Avg Blocking Send/Recv time (micro-seconds)
43 # Avg Non Blocking Send/Recv time (micro-seconds)
43 # Avg Combined Send/Recv time (micro-seconds)
```

Data Analysis

Below is the graph of 450 consecutive runs of the program each increasing size by 1000 bytes.

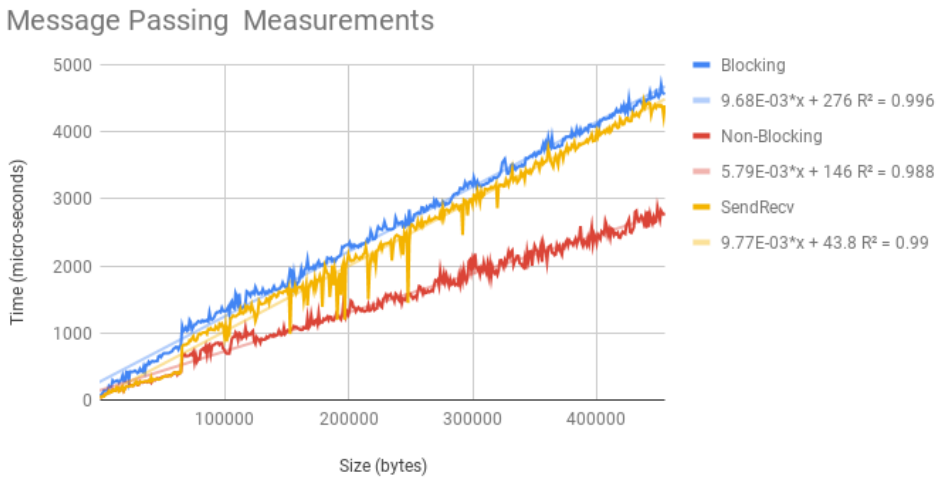

Figure 1: *Measured message passing times for Blocking, Non Blocking, and Combined Send/Receives*

As you can see in the graph, at approximately 70 kB the response times jumped significantly. Due to the fact that I gathered this data at 12pm on a Thursday it is fair to assume that the rise in time is correlated with the sudden influx of students in the lab for Assembly's lab. That or Shrader saw me working on MPI and decided to throttle my connection to mess with me, the world may never know.

Based on the linear regression lines, we can calculate our bandwidth and latency in the table below.

Ping Type	Latency (uS)	Bandwidth (bytes / uS)
Blocking	276	103.3
Non Blocking	146	172.7
Combined	43.8	102

Conway's Game of Life

Program Description:

"In 1970, Princeton mathematician John Conway invented the game of Life. Life is an example of a cellular automaton. It consists of a rectangular grid of cells. Each cell is in one of two states: alive or dead. The game consists of a number of iterations. During each iteration a dead cell with exactly three neighbours (out of eight) becomes a live cell. A cell with two or three neighbours (out of eight) stays alive. A cell with less than two or more than three neighbours becomes dead. All cells are updated simultaneously." – *Dr. Karlsson*

Using the Program:

To run the game of life, first enter into the directory and type make.

```
# Build the project
> cd game_of_life
> make
```

To run the program, type the following command:

```
> mpiexec -n <row_count> game_of_life <row_count> <col_count>
↪  <iterations> <initial live cell count> <print iterations>
```

Be sure to check that your host file's configuration has every slot available on each host. (Attached in the directory is an example.)

The program will output 3 unlabeled message passing times, (Which made writing a script to do the data analysis really easy in python) below is a labeled example of the output:

```
>mpiexec -n 6 game_of_life 6 9 4 20 1
Starting Board:
| O | - | - | O | - | - | - | - | - |
| - | O | - | - | - | - | - | O | - |
| - | - | O | - | O | - | - | O | O |
| O | O | O | - | - | - | O | O | - |
| - | O | - | O | - | - | - | - | - |
| O | - | - | O | O | - | - | O | O |
Board: 1
| - | - | - | - | - | - | - | - | - |
| - | O | O | O | - | - | - | O | O |
| O | - | O | O | - | - | - | - | O |
| O | - | - | - | - | - | O | O | O |
| - | - | - | O | O | - | O | - | O |
| - | - | O | O | O | - | - | - | - |
Board: 2
| - | - | O | - | - | - | - | - | - |
| - | O | - | O | - | - | - | O | O |
| O | - | - | O | - | - | O | - | - |
| - | O | O | - | O | O | O | - | O |
| - | - | O | - | O | - | O | - | O |
| - | - | O | - | O | O | - | - | - |
Board: 3
| - | - | O | - | - | - | - | - | - |
| - | O | - | O | - | - | - | O | - |
| O | - | - | O | - | - | O | - | O |
| - | O | O | - | O | - | O | - | - |
| - | - | O | - | - | - | O | - | - |
| - | - | - | - | O | O | - | - | - |
Board: 4
| - | - | O | - | - | - | - | - | - |
| - | O | - | O | - | - | - | O | - |
| O | - | - | O | O | O | O | - | - |
| - | O | O | - | - | - | O | - | - |
| - | O | O | - | O | - | O | - | - |
| - | - | - | - | - | O | - | - | - |
```

Testing and Verification

To test the game of life, I hand verified a few smallish (8x8 was the max size) games to ensure that every cell opererated as intended.

Note: Be warned that sometimes right at the end of the game an error might happen claiming a corrupted queue. I fought the issue for a while but eventually gave up since the program was technically completed.