CSC 410 Parallel Programming
Dr. Christer Karlsson
Benjamin Kaiser
Programming Assignment 2
Part 1 Ping Pong

Description of Program

This program is a test of the networks bandwidth by using a ping pong program to time when a message is sent to another machine and then when the other machine sends that message back. Dr. Karlsson wanted us to implement blocking, sendrecv, and nonblocking versions of the ping-pong algorithm.

Description of algorithms and libraries used

The only two things that were really used in this program were the OpenMPI library for C++ and then also the ping-pong algorithm. According to open-mpi.org, the OpenMPI library is "an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers."

In other words, it is a distributed processing library that runs the exact same code on all of the processes.

The ping-pong algorithm is literally just having process 0 send a message to process 1. Process 1 then turns around and sends it back to process 0 who then checks to make sure the same message was received.

Description of functions and program structure

This program consists of four functions including main. The other three are the different implementations of the ping-pong program. The first function called "normal" or "regular" is the normal blocking implementation of the ping-pong program. The second function called "sendrecv" is the SendRecv version of the program. The final function called "nonblocking" is what I believe to be the non-blocking version of the ping pong program.

Description of testing and verification process

In order to test this program I did a variety of things. The first thing I did was run the program with only a single time through each message passing. This is what I have submitted.

However I did run it several different ways including running a message through the size I could statically allocate without issues (~4,000,000 bytes), looping running through this up to 5,000 times. I also started with a byte size of 2 and incremented in powers of 2 both running just one time per data size and also repeating each one about 5,000 times. The data for these latter two were saved to a file as described in the "Data" section of this document. I also used the linux ping command to verify some of my data. I also used the ethtool for more verification. This is described in more detail in my analysis of data.

Data

The data for this program is contained in pingpongresults.txt and pingpongresultssingle.txt. Pingpongresults.txt contains the data for when the program was run looping 5,000 times on each power of 2. Pingpongresultssingle.txt contains the data for a single run incrementing by a power of 2. The reasons that this data was collected is explained further in the analysis section of this document.

Analysis of Data

Graphs and charts and stuffs are contained in PingPongAnalysis.pdf. The data used to create these graphs is the data collected in pingpongresultssingle.txt.

After looking at the graphs and calculating the trendlines, I concluded that Blocking is the slowest, followed by SendRecv followed by Nonblocking. My trendlines are actually interesting. I Calculated ~.73515898 Gbps for Blocking bandwidth, 1.00920903242084 Gbps for SendRecv bandwidth and 1.98511166253101 Gbps for Nonblocking. My latencies were .2554 ms, -.0313 ms, and -.0967 ms respectively. I obviously don't trust the last two trendlines because I had negative latency which is 100% impossible. I also wanted to confirm the bandwidth numbers so I used the ethtool command on eno1 which told me that the connection between machines has a maximum speed of 1000 Mbps which is equal to 1 Gbps. So the bandwidth I got for SendRecv is off. I believe that I know why this is and that is because I may have placed my timing statements in the wrong spot which may be throwing a not accurate fast speed.

In order to compare to something that is already known, I ran the ping command on the same two linux machines that I was using to test my data (linux10 and linux11). The latency that my trendline predicted for Blocking lined up almost exactly with the times that the ping returned. It was interesting to note however that when sending 64 bytes of data, the MPI ran almost twice as fast as the ping command. I did check consistency of this and this appeared about right.

To check consistency, I ran each data size about 5000 times and this data is contained in the pingpongresults.txt file. As you can see, the data is fairly consistent across data sizes.

Description of Submission

The contents of the submission of this program are as follows:

- Ping_pong.cpp - source code for part 1
- Pingpongresults.txt - data for part 1
- Pingpongresultssingle.txt - data for part 1
- Life.cpp - source code for part 2
- Life.h - source code for part 2
- PingPongAnalysis.pdf - data analysis for part 1
- Homework3.pdf