Homework 4

Christian Winwood
A0207402

February 15, 2021

1 Implementation

My implementation was fairly similar. For each of the different methods I decided to create a new function. In the main function, each process generates a random number to use throughout the entire program. They then go through and call all of the functions. Before each call, the root function prints out the status and then each process blocks to make sure each process is in sync before the function call.

1.1 Send and Receive

To implement this, I had each process send the root (process 0) their individual number. They then wait until they get the total back and report what they received. The root function receives all their numbers, adds them together and then sends the total to all the other processes individually.

1.2 Gather and BCast

Each process calls gather and their individual number is passed in. I have the root process receiving all of them into an array. The root than iterates through the replies, adds them together and sends out the answer with a BCast.

1.3 All Reduce

This was very easy. Each process calls the all reduce function with their number and an int to put the answer in. They then print what the answer is.

1.4 Ring

For the ring, the root process starts by sending their unique number. Each process then receives the current total, adds their number, and passes to the next process. When the root receives the final total, it sends out the total to all the other process to be printed out

1.5 Cube

The first thing I do is calculate how many iteration I need to do by using log base 2. I then do a for loop n times moving the flipping bit from the 0 bit to the nth bit. After each exchange, the totals are added together. Once all of them are done, each process already has the total so they each just print it out.

2 Compile and Run

The compile and run commands are the same they have always been. First you use

mpic++ allreduce.cpp

in order to compile the program. This creates an a.out files that you need to run. To run it you use the command

mpirun –oversubscribe -np ¡number-of-processes¿ a.out

and the program will run.

3 Output

Before each function is ran, which function is running gets printed. This is followed by the grand total that each process has at the end of the function.