**Dillon Cowan**

**CS 480 – Senior Project**

The project I am proposing will be a watered down version of Message Passing Interface (MPI)**.** I will make a program using the original MPI to test correctness of my version of MPI. Also I would want to make my own cluster from scratch. I would need at least 5 nodes including the head node, to somewhat start from somewhere that’s not to low and potentially have more added onto the cluster if there was a need.

**Project Goals**

**Build the Cluster**

* Gather the nodes, Ethernet switch, Ethernet cables, power cords, one head monitor, keyboard, mouse, power strip, etc.
* Connected everything
* Install Ubuntu on each machine and change necessary configure files, install the real MPI and get it up and running

**Calls to implement**

* MPI\_Init- starts up the environment
* MPI\_Comm\_size- returns the number processes that are available to the application
* MPI\_Comm\_rank- returns the specific communicator that is called on
* MPI\_Finalize-closes all connections and stops all processes in progress
* MPI\_Abort-closes the specific communicator
* MPI\_Get\_processor\_name- gets the name of the processor
* MPI\_Send- send some data to a specific communicator
* MPI\_Recv- receives data from some sender
* MPI\_Bcast- send to all communicators

I think I covered most of the calls that are the basis of MPI. <https://computing.llnl.gov/tutorials/mpi/> This is a link to all of the calls (all the bottom).

**Project in detail**

I am going to be writing the program in C++. For the first implementation I want to treat each connection as one processor even if there is more than one core on that connection. MPI\_Init has to be the first command called in a program before all others and it will only be called once. My MPI\_Init will not need to take any arguments. MPI\_Init will set up MPI\_Size and MPI\_Rank. When I’m first writing the program I will have a set amount of machines with known IP addresses. It would be nice to try to implement where my program can find each computer that are capable of working over the network automatically, instead of just knowing ahead of time. Something else that I want to implement is to allowing new machines to join the network while it’s working on a problem. MPI\_Init will also set up all the communications layers between all the processors. None of the other calls will have any “setting up” to do to talk to another process. For Example when you call MPI\_Send the “rank” is already set up to send immediately, MPI\_Comm\_size will just be getter methods. MPI\_Comm\_rank assigns each node a unique number so you know which node your talk to and which node your assigning work to. Each machine will have my version of MPI installed onto. Depending if my cluster has the same file system to read from, I may need to distribute, with a small program, the complied code on the head node to the other nodes. From what I’ve seen so far each node will need to have a server and that sever will be talking to the head node only. The head node will coordinate all the work. I’m going to assume that you’re going to want to use all the computers connected, so anything you need having to do with MPI\_COMM\_WORLD as an argument I’m going to exclude. For MPI\_Send and MPI\_Recv the arguments they will take are the variable to be used, the type of variable that it is (data type -- int, double, string) and where to send it to/who to receive from. I’m only allowing one variable to be passed through with MPI\_send and MPI\_recv. I will have block communication with sending and receiving. The program will wait to make sure that it has been successful before moving on. MPI\_Abort will take one argument and that is which single process to close and the call MPI\_Finalize will close all processes and you can’t make any more calls from MPI after calling finalize. MPI\_Bcast will take two arguments, the variable along with its data type. All calls will return a 0 if there’s no error. I need a deamon server on every node, Mpi\_init tells each daemon to start up the Mpi program with certain parameters (like where to connect to and what its rank, name and how many nodes are running) and each “slave” connects back to the “master”.

**Code example** (my version)

int main(int argc, char\*\* argv) {

MMPI\_Init();

// Get the number of processes

int world\_size;

MMPI\_Comm\_size(&world\_size);

// Get the rank of the process

int world\_rank;

MMPI\_Comm\_rank (&world\_rank);

// Print off a hello world message

printf("Hello world rank %d out of %d processors\n", world\_rank, world\_size);

int rc;

int buff;

rc = MMPI\_Send(&buff, MMPI\_INT, 1); // MMPI\_Send(&buf,datatype,dest)

if (rc != MMPI\_SUCCESS)

printf("failure on MPI\_SEND: fail on->> %d", rank);

rc = MMPI\_Recv(&buff, MMPI\_INT, 1); // MMPI\_Recv(&buf,datatype,source)

if (rc != MMPI\_SUCCESS)

printf("failure on MPI\_Recv: fail on->> %d", rank);

rc = MMPI\_Bcast(&buff, MMPI\_INT); // MMPI\_Bcast(&buf,datatype)

if (rc != MMPI\_SUCCESS)

printf("failure on MPI\_Bcast: fail on->> %d", rank);

MMPI\_Abort(rank); //MMPI\_Abort(dest)

// Finalize the MPI environment. No more MPI calls can be made after this

MMPI\_Finalize();

}

**Limitations**

I will need more machines to test my program out. Overall I will still be looking guidance as to what the final product of my senior project will be.

**Grading Scale**

|  |  |  |
| --- | --- | --- |
| Name | Out of | Complete? |
| Dsc\_mpi\_Init | 15 |  |
| Dsc\_mpi \_size | 5 |  |
| Dsc\_mpi \_rank | 5 |  |
| Dsc\_mpi\_processor\_name | 5 |  |
| Dsc\_mpi \_Finalize | 10 |  |
| Dsc\_mpi \_Abort | 10 |  |
| Dsc\_mpi \_Send | 20 | REQ |
| Dsc\_mpi \_Recv | 20 | REQ |
| Dsc\_mpi \_Bcast | 20 |  |
| Daemon server | 10 |  |
| Built cluster | 25 |  |
| TOTAL | 145 |  |

73 points - C, 84 points - B, 94 points - A

I might have understated building the cluster; I’m not sure how to put one together until I’m playing around with it. I was suggested about 1/3 of the project would be towards building the cluster, but that is up to the committees discretions.