# Introduction to MPI

March 20, 2015

## What is MPI

- MPI is a library for passing messages between processes.
- MPI is a standard, not an implementation. We happen to be using the `openmpi` implementation.
- Since MPI is a standard, any correctly implemented code relying on MPI should be able to run using any properly implemented MPI library.
- For us, TORQUE is the mechanism for reserving resources. MPI is the mechanism that is helping us run processes on the those resources and communicate between the processes.

## Basic concepts in MPI

- The fundamental tasks that MPI addresses are **sending** and **receiving** messages.
- MPI allows us to group processes
- MPI allows us to tag messages

- In this course we will use the **S**ingle **P**rogram **M**ultiple **D**ata model (SPMD). In this model all the processes are copies of the same executable.
- Behavior of each process is determined by the **rank**, which is managed by MPI.

## MPI functions

The following functions make up most of the functionality of MPI:

- `MPI_Init` – initialize MPI
- `MPI_Comm_size` – get the number of processes in the given communicator
- `MPI_Comm_rank` – get the rank of this process in the given communicator
- `MPI_Send` – send a message
- `MPI_Recv` – receive a message
- `MPI_Finalize` – clean up

## Hello, World

```
#include<mpi.h>
#include<stdio.h>

int main(int argc, char** argv){
  MPI_Init(&argc,&argv);

  printf("Hello, World\n");
  MPI_Finalize();

}
```

- To compile the previous code, use
  `mpicc helloworld.c -o helloworld`
- To run the code, use
  `mpirun helloworld` INSIDE AN INTERACTIVE QSUB SESSION!

- Note also that `printf` worked properly, even though it was being executed on a remote node. This is the MPI runtime at work for us!
- `MPI_COMM_WORLD` is the world communicator, every process is included in this communicator.

We will frequently make behavior depend upon rank. The process with rank 0 is usually designated the “master” or “head”, and all the other processes are called “workers”.

```
#include<mpi.h>
#include<stdio.h>

int main(int argc, char** argv){
  MPI_Init(&argc,&argv);
  int total_procs;
  int rank;
  MPI_Comm_size(MPI_COMM_WORLD,&total_procs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  if (rank==0){
    printf("Hello, this is process %d  and
I'm the master.\n",rank);
  }
  else{
    printf("Hello, this is process %d  and
I'm a lowly worker. :(\n",rank);
  }
  MPI_Finalize();

}
```

## Sending and Receiving

- `int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag,MPI_Comm comm)`
  - buf is a pointer to the beginning of the data to send. Note that it can be of any type.
  - count is the number of data pieces to send
  - datatype is the type of data being sent (tells us how big the data pieces are), and is `MPI_INT`, `MPI_DOUBLE`, `MPI_CHAR`, etc...
  - dest is the rank of the process we are sending the message to
  - tag we will usually set to 0
  - comm we will set to `MPI_COMM_WORLD`

- `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
  - `buf` is a pointer to the memory allocated to store the message
  - `count` is the size of the buffer, we can not receive more data than this.
  - `datatype` is the type of data we are receiving and is one of `MPI_INT`,`MPI_DOUBLE`,`MPI_CHAR`, etc
  - `source` is the rank of the process we want to receive from. Set to `MPI_ANY_SOURCE` to receive messages from anyone.
  - `tag` is the message tag we want to receive. Usually set to `MPI_ANY_TAG`
  - `comm` is always `MPI_COMM_WORLD`
  - `status` gets further information about the message
    - `status.MPI_SOURCE` tells us who sent the message
    - `MPI_Get_count(status)` tells us how long the message was

- Page 67 (Section 3.5 of the book) has a **quick** overview of MPI.
- http://www.mpitutorial.com/
- http://www.lam-mpi.org/tutorials/one-step/ezstart.php
- http://www.mcs.anl.gov/research/projects/mpi/tutorial/index.html

```
int main(int argc, char** argv){
  MPI_Init(&argc,&argv);
  int total_procs;
  int rank;
  MPI_Comm_size(MPI_COMM_WORLD,&total_procs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  if (rank==0){
    int i;
    char* message="I am your master!";
    char recv_buffer[10];
    MPI_Status status;
    for (i=1;i<total_procs;i++){
      MPI_Send(message,18,MPI_CHAR,i,0,MPI_COMM_WORLD);
    }
    for (i=1;i<total_procs;i++){
      MPI_Recv(recv_buffer,10,MPI_CHAR,MPI_ANY_SOURCE,MPI_A
      printf("Process %d: Message from %d: %s\n",rank,statu
    }
  }
```