MPI Lab 1

BME 590L Lab 6

Daniel Puleri

Duke University

October 11, 2018

Section 1

MPI Basics

What is MPI?

- Message passing *interface*
- It is not another programming language, but an interface to which different organizations can create their own implementations of MPI
- When you compile with `mpicc` you are using a wrapper around a regular compiler such as `gcc`, `clang`, or `icc` that includes all of the MPI libraries for you

Why would we want to use MPI over OpenMP?

- You might want to:
  - solve a problem that requires more memory than the biggest node you can get access to
  - solve a problem that would take forever on one node
- Example: your biggest node has 256 GB of RAM, but you need to load a detailed map of the US into memory and that map is 1 TB

How to compile and run with MPI?

```
# compile the same as you would with gcc
mpicc myprog.c -o myprog
# run the program with mpirun
mpirun -np X myprog
```

- If you run `mpicc -compile_info` you will see what is really going on during compilation
- In the example above "X" represents the number of MPI tasks you wish for your program to be run with

Hello world in MPI

- Make sure that you can run the following program and then we will dissect it together

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

int main(int argc, char **argv) {
    MPI_Init(&argc,&argv);
    int rank, size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Hello, World, from task %d of %d\n",
           rank, size);
    MPI_Finalize();
    return 0;
}
```

Hello world in MPI explanation:

- What does `MPI_Init()` do?
  - Initialized the MPI environment
  - Anything before `MPI_Init` is relying on undefined behavior
- What does `MPI_COMM_WORLD` mean?
  - This is your global communicator, it contains all of the tasks you can communicate to
- What does `MPI_COMM_size()` do?
  - This returns the size of the communicator you pass (in this case, the default global communicator)
  - What is this number equal to when you pass `MPI_COMM_WORLD`?
- What does `MPI_Comm_rank()`?
  - Tells you what your rank (think, task ID) is inside of a given communicator

Things to test out:

- Write a `for` loop to print out 1 to 10
- Have **one** rank print out a message

Section 2

Exercise 1

Ex1: Domain decompositon

1. Write code to have each rank square and print out three numbers in an array that goes from 1 to 3 times the size of your parallel domain
   - Scaffold code available on Piazza (if you need it)
2. Write code to have each rank square and print out an *as-equal-as-possible* portion of an array from 1 to 100
   - This problem will require domain decomposition!

Ex1: Go over solution to 1.1 together

Ex1: Go over solution to 1.2 together

Section 3

MPI Sending and Recieving

MPI_Send

- `MPI_Ssend` (Synchronous Send)
  - returns when message is **delivered**
- `MPI_Bsend` (Buffered Send)
  - routine returns before the message is delivered
  - system copies data into a buffer and sends it later on
- `MPI_Send` (standard Send)
  - Internally can be either a Bsend or an Ssend ...
  - Assume it's an Ssend and you won't have problems

MPI_Recv

- Recieve is always synchronous!!
- Meaning, the program will wait until it has recieved the message

Sending and recieving fucntion signature

```
int MPI_Send(const void *buf, int count
             MPI_Datatype datatype,
             int dest, int tag,
             MPI_Comm comm)

int MPI_Recv(void *buf, int count,
             MPI_Datatype datatype,
             int source, int tag,
             MPI_Comm comm, MPI_Status *status)
```

- Note that you need to specify who you are sending to
- On the recieving end, you need to specify who is sending you information

How do I find out more information about these functions?

Use the man pages!

```
man MPI_Send
man MPI_Recv
```

It will tell you what the function signatures are, what the functions expect, and how they will act.

Section 4

Exercise 2

Ex2: Messages in circles

- Send a message to the next and previous processor
  - let the next processor be rank+1
  - let the previous be rank-1
  - wrap around
  - let the message be 10 ints long and be the rank repeated
- Do this with **blocking communication only** (`MPI_Send` not `ISend`)

Code to print out recieved message:

```
printf("Rank %d recieved: ", myrank);
for (int i = 0; i < 10; i++) {
    printf("%d ", recvd_msg[i]);
}
printf("\n");
```

Section 5

Q&A on HW