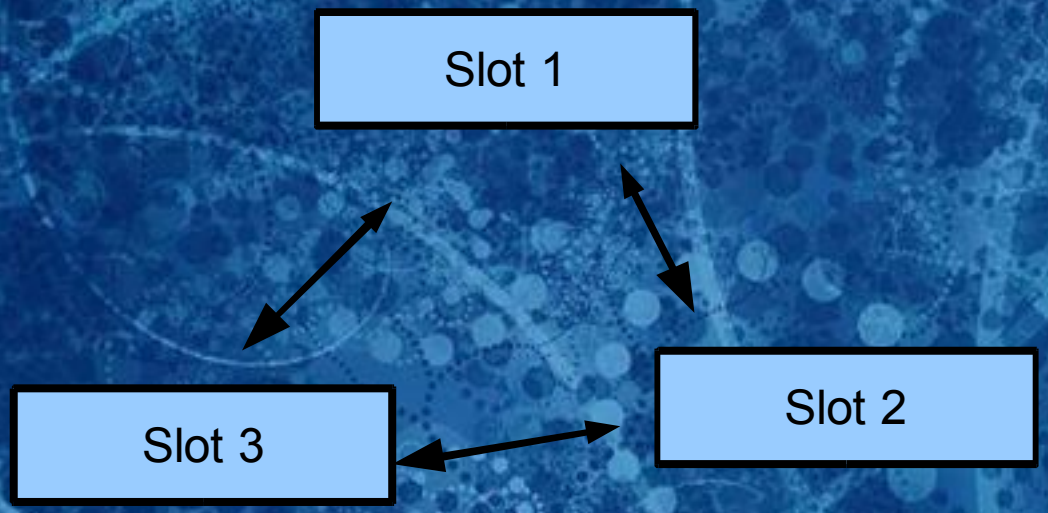
Introduction to MPI Programming

Joey Bernard, ACEnet CRC, UNB

What is MPI?

- MPI – Message Passing Interface
- A specification, not a library
- You use an implementation
  - MPICH
  - openMPI
  - LAM

What does it look like?



How to compile?

- MPI implementations include scripts
  - mpicc, mpiCC, mpif77, mpif90
- These scripts automatically set parameters **required** by MPI
- You can add compiler specific options
- *-showme* will print out the compiler command to be executed
- Can also set environment variables
  - CC, CFLAGS, F77, FC, FFLAGS, FCFLAGS

How to run

- MPI implementations have scripts for running
  - mpirun, mpiexec
- Common parameters:
  - *-np X* — *X* is the number of slots
  - *- -hostfile name* — *name* is the file with the list of hosts

What about Grid Engine?

- You need to tell grid engine how to run MPI code
  - *#$ -pe ompi X* *X* is the number of slots
  - *mpirun my_code* actually execute
- Grid Engine sets up all of the required parameters to run the MPI program successfully

Bare Essentials

- We need to initialize and finalize an MPI program

```
int MPI_Init(&argc, &argv);
MPI_INIT(IERROR)
```

- No other MPI routines can be called before this one except *MPI_Initialized*

Bare Essentials 2

- We need to initialize and finalize an MPI program

  ```
  int MPI_Finalize();
  MPI_FINALIZE(IERROR)
  ```

- After calling finalize, no other MPI routines maybe called except for *MPI_Get_version*, *MPI_Initialized*, and *MPI_Finalized*

Who am I?

- We need to know who we are, and how many siblings we have

```
int MPI_Comm_rank(MPI_Comm comm, int
    *rank);
MPI_COMM_RANK(COMM, RANK, IERROR)

int MPI_Comm_size(MPI_Comm comm, int *size);
MPI_COMM_SIZE(COMM, SIZE, IERROR)
```

How can we talk?

- MPI is Message Passing Interface
- There are many methods of sending and receiving messages
- You can create channels
- You can send to one or to all

Basic Sending

- The most basic send moves data from one slot to another (blocking mode)

```
int MPI_Send(void *buf, int count,
    MPI_Datatype type, int dest, int
    tag, MPI_Comm comm);
MPI_SEND(BUF, COUNT, DATATYPE,
    DEST, TAG, COMM, IERROR)
```

Basic Receiving

- Now we need to receive that data (blocking mode)

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

MPI_RECV(BUF, COUNT, DATATYPE,
    SOURCE, TAG, COMM, STATUS,
    IERROR)
```

Receive Status

- What kind of status do we get on the receive?
  - status->MPI_source
  - status->MPI_tag
  - status->MPI_ERROR
- Why?
  - Source can be set to MPI_ANY_SOURCE
  - Tag can be set to MPI_ANY_TAG

What kind of datatypes?

MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Can we talk to more than one?

- Sometimes we need to talk to everybody at once

```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype type, int root,
    MPI_Comm comm);

MPI_BCAST(BUF, COUNT, TYPE, ROOT,
    COMM, IERROR)
```

- This broadcasts from one slot (root) to everybody, including root

Reductions

- What if we want to simply consolidate results across the slots?

```
int MPI_Reduce(void *sendbuf, void
    *recvbuf, int count, MPI_Datatype
    type, MPI_Op op, int root,
    MPI_Comm comm);

MPI_REDUCE(SENDBUF, RECVBUF,
    COUNT, TYPE, OP, ROOT, COMM,
    IERROR)
```

Built-in Reduction Operations

MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bit-wise and
MPI_LOR	logical or
MPI_BOR	bit-wise or
MPI_LXOR	logical exclusive or
MPI_BXOR	bit-wise exclusive or
MPI_MAXLOC	max. value and location
MPI_MINLOC	min. value and location

Kinds of Reductions

- *MPI_Reduce* – the result of the reduce is delivered to the root slot
- *MPI_Allreduce* – the result of the reduce is delivered to all of the slots
- *MPI_Reduce_scatter* – the array result of the reduction is scattered to all slots
- *MPI_Scan* – inclusive prefix reduction on slots 0 to i

How do you distribute data?

- One of the first tasks is to distribute initial data to all of the slots

```
int MPI_Scatter(void *send, int
    sendcnt, MPI_Datatype type, void
    *recv, int recvcnt, MPI_Datatype
    type, int root, MPI_Comm comm);
```

- Takes the initial data in send and divides it into equal chunks and sends it out to everyone, including root

Then what?

- Once we've finished our calculations, we need to gather up the results

```
int MPI_Gather(void *send, int
    sendcnt, MPI_Datatype type, void
    *recv, int recvcnt, MPI_Datatype
    type, int root, MPI_Comm comm);
```

What if we don't want to wait?

- All of the communication calls till now have been blocking
- What if we don't want to wait for these communications to finish before we get back to work?
- We can use the provided non-blocking routines
- Includes a final parameter to track the request

```
MPI_Request *handle
```

How do we know we're done?

- With non-blocking sends and receives, we don't necessarily know that the data has finished being transferred

```
int MPI_Wait(MPI_Request *request,
    MPI_Status *status);

int MPI_Waitall(int cnt, MPI_Request
    *reqarray, MPI_Status *status);
```

Profiling

- How do we track time in an MPI program?

```
double MPI_Wtime();
```

- This gives us the number of seconds since some start time in the past.

```
double MPI_Wtick();
```

- This gives us the resolution of the time

What are COMMs?

- What are all the references to MPI_Comm objects?
- Messages are passed over ommunication channels
- When MPI initialization is done we get MPI_COMM_WORLD for free
- Everyone belongs to this communicator

Communicators

- First create a new group of slots

```
int MPI_Group_incl(MPI_Group old, int
    new_size, int *old_ranks,
    MPI_Group *new);
```

- Then create a new communicator for this group

```
int MPI_Comm_create(MPI_Comm
    old_comm, MPI_Group group,
    MPI_Comm *new_comm);
```

Putting it all together

```
#include <mpi.h>
int main(int argc, char **argv) {
  int i, id;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &id);
  MPI_Comm_size(MPI_COMM_WORLD, &i);
  …
  MPI_Finalize();
  return 0;
}
```