



# MPI Basics

## Message Passing Interface

Tom Murphy

Director of Contra Costa College  
High Performance Computing Center



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# Preliminaries

answering: “What is a cluster”

- To set-up a cluster we must:
  - Configure the individual computers
  - Establish some form of communication between machines
  - Run the program(s) that exploit the above
- MPI is all about exploitation



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# So what does MPI do?

“What does the coder do?”

- Simply stated:
  - MPI allows moving data between processes
  - Data that is needed
    - for a computation
  - or
  - from a computation
- Now just wait a second!
  - Shouldn't that be processors!



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics

U Oklahoma, July 29 - Aug 4 2012





# Simple or Complex?

for now it's simple

- MPI has 100+ very complex library calls
  - 52 Point-to-Point Communication
  - 16 Collective Communication
  - 30 Groups, Contexts, and Communicators
  - 16 Process Topologies
  - 13 Environmental Inquiry
  - 1 Profiling
- MPI only needs 6 very simple complex library calls



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics

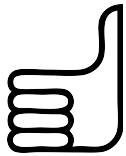
U Oklahoma, July 29 - Aug 4 2012





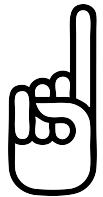
# Six Basic MPI commands

Via three key fingers



- How do I start and stop

- MPI\_Init
- MPI\_Finalize



- Know thy self (and others)

- MPI\_Comm\_rank
- MPI\_Comm\_size



- Middle Finger - The Message Passing

- MPI\_Send
- MPI\_Recv



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics

U Oklahoma, July 29 - Aug 4 2012





# Essential MPI Organization

that sometimes get in the way

- Data Representation is Standardized
  - MPI data types
- Harnessing Processes for a Task
  - MPI Communicators
- Specifying a kind of message
  - MPI Tags
- How many: Processes and Processors
  - -np
  - -machinefile



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# Data Representation

Exact -> Integer Types

- Signed
  - MPI\_CHAR
  - MPI\_SHORT
  - MPI\_INT
  - MPI\_LONG
- Unsigned
  - MPI\_UNSIGNED\_CHAR
  - MPI\_UNSIGNED\_SHORT
  - MPI\_UNSIGNED
  - MPI\_UNSIGNED\_LONG



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# Data Representation

Approximate -> Floating Point

- MPI\_FLOAT
- MPI\_DOUBLE
- MPI\_LONG\_DOUBLE



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012







# Data Representation

## Special Cases

- MPI\_BYTE
  - Device independent
  - Exactly 8 bits
- MPI\_PACKED
  - Allows non-contiguous data
    - MPI\_PACK
    - MPI\_UNPACK



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# Under the hood of the First Four

How do I start and stop

- `MPI_Init` (`int *argc, char ***argv`)
    - We gotta change (`int argc, char **argv`)
  - `MPI_Finalize` ()
- since
- MPI uses it to pass data to all machines



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# Under the hood of the First Four

Know thyself (and others)

- MPI\_Comm\_rank  
(MPI\_Comm comm, int \*rank)
- MPI\_Comm\_size  
(MPI\_Comm comm, int \*size)



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# MPI Hello World

Lets explore some code

- Henry previously regaled you with “Batch Computing on Boomer”
- Lets look at some code
  - MPI Hello World
  - MPI Greeting
- Exercises for Later
  - You run both above the above on Boomer
  - You modify extend these to further your knowledge



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# MPI Hello World

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {
    int my_rank, world_size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    printf("I am rank %d of %d\n",
           my_rank, (world_size-1));
    MPI_Finalize();
    return 0;
}
```

MPI Basics

U Oklahoma, July 29 - Aug 4 2012



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA





# Under the hood of the Last Two

The actual message passing

- MPI\_Send(  
void\* buf, int count, MPI\_Datatype datatype,  
int dest, int tag, MPI\_Comm comm)
- MPI\_Recv(  
void\* buf, int count, MPI\_Datatype datatype,  
int source, int tag, MPI\_Comm comm,  
MPI\_Status \*status)



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# MPI Greeting

A fugue in seven parts

1. Including the Right Stuff
2. General Declarations
3. MPI Setup
4. Client-side Code
5. Server-side Code
6. The Grand Finale
7. Error Handling



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA

MPI Basics

U Oklahoma, July 29 - Aug 4 2012





# MPI Greeting

## Part 1: Including the right stuff

- The first 54 lines can successfully be ignored for now, aside from:

Line 25

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

Line 50

```
void fatal(const int, const int);
```

Line 53

```
#define SERVER_RANK 0
```

- The ignored lines are for
  - Copyright info
  - Typical include files
  - Overview of program and compilation instructions



MPI Basics

U Oklahoma, July 29 - Aug 4 2012







# MPI Greeting

## Part 2: General Declarations

```
int main(int argc, char **argv) {  
  
#ifdef STAT_KIT  
    startTimer();  
#endif  
  
int my_rank, world_size, destination, tag, source;  
int length, mpiErr;  
char message[256], name[80];  
MPI_Status status;
```



# MPI Greeting

## Part 3: MPI Setup

```
MPI_Init(&argc, &argv);  
// note that argc and argv are passed by address
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);  
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# MPI Greeting

## Part 4: Client-side Code

```
/* client code */
if (my_rank != SERVER_RANK) {
    printf("I am the client, with rank %d of %d\n",
        my_rank, (world_size-1));
    MPI_Get_processor_name(name,&length);
    sprintf(message, "Greetings from process %d, %s!",
        my_rank,name);
    destination = SERVER_RANK;
    tag = 2;
    mpiErr = MPI_Send(message,strlen(message)+1, MPI_CHAR,
        destination,tag,MPI_COMM_WORLD);
    if(mpiErr != MPI_SUCCESS) {
        fprintf(stderr,"Rank %d - MPI_Send to %d failed!\n",
            my_rank,destination);
        fatal(mpiErr,my_rank);
    }
}
```

MPI Basics

U Oklahoma, July 29 - Aug 4 2012



INFORMATION  
TECHNOLOGY  
THE UNIVERSITY OF OKLAHOMA





# MPI Greeting

## Part 5: Server-side Code

```
/* Server code */
else {
    printf("I am the server, with rank %d of %d\n",
        my_rank, (world_size-1));
    tag = 2;
    for (source = 1; source < world_size ; source ++) {
        mpiErr = MPI_Recv(message,256,MPI_CHAR,
            source,tag,MPI_COMM_WORLD, &status);
        if(mpiErr != MPI_SUCCESS) {
            fprintf(stderr,"Rank %d - MPI_Recv from %d failed!\n",
                SERVER_RANK,source);
            fatal(mpiErr,my_rank);
        }
        fprintf(stderr, "%s\n", message);
    }
}
```

MPI Basics

U Oklahoma, July 29 - Aug 4 2012





# MPI Greeting

## Part 6: The Grand Finale

```
printf("Calling Finalize %d\n",my_rank);  
  
MPI_Finalize();  
  
#ifdef STAT_KIT  
    printStats("Hello world MPI",world_size,"mpi",  
              1, "1", 0, 0);  
#endif  
  
exit(EXIT_SUCCESS);  
}
```



MPI Basics  
U Oklahoma, July 29 - Aug 4 2012





# MPI Greeting

## Part 7: Error Handling

```
void fatal(const int mpiErr, const int rank) {  
    int resultLen;  
    char mpiErrStr[MPI_MAX_ERROR_STRING];  
    MPI_Error_string(mpiErr, mpiErrStr, &resultLen);  
    fprintf(stderr, "Rank %d - %s\n", rank, mpiErrStr);  
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);  
    exit(EXIT_FAILURE);  
}
```