MPI Basics

Message Passing Interface

Andrew Fitz Gibbon

OU SuperComputing Symposium

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

So what does MPI do?

Actually it's "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!

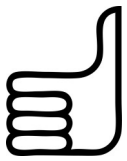
How do you want it Simple or Complex?

The correct answer for now is 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 needs 7 very simple library calls

Seven Basic MPI commands

Via the three “right hand rule “ fingers



- How do I start and stop
  - MPI_Init
  - MPI_Finalize

- Know thy self (and others)
  - MPI_Comm_rank
  - MPI_Comm_size
  - MPI_Get_processor_name

- Middle Finger - The Message Passing
  - MPI_Send
  - MPI_Recv

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

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

Data Representation

Appoximate -> Floating Point

- MPI_FLOAT
- MPI_DOUBLE
- MPI_LONG_DOUBLE

Data Representation

Special Cases

- MPI_BYTE
  - Device independent
  - Exactly 8 bits
- MPI_PACKED
  - Allows non-contiguous data
    - MPI_PACK
    - MPI_UNPACK

Under the hood of the Seven

How do I start and stop

- MPI_Init (int *argc, char ***argv)
  - We gotta change (int argc, char **argv)

    since
  - MPI uses it to pass data to all machines
- MPI_Finalize ()

Under the hood of the Seven

Know thyself (and others)

- MPI_Comm_rank
  (MPI_Comm comm, int *rank)
- MPI_Comm_size
  (MPI_Comm comm, int *size)
- MPI_Get_processor_name
  (char *name, int *resultlen)

Under the hood of the Seven

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 Hello World

A fugue in six parts

1. Using the Right Stuff
2. General Initialization
3. MPI Setup
4. Client-side Code
5. Server-side Code
6. The Grand Finale

MPI Hello World

Part 1: Using the right stuff

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

#define SERVER_NODE 0
```

MPI Hello World

Part 2: General Initialization

```
int main(int argc, char **argv) {
    int my_rank, world_size;
    int destination, source;
    int tag, length;

    char message[256], name[80];

    MPI_Status status;
```

MPI Hello World

Part 3: MPI Setup

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

MPI Hello World

Part 4: Client-side Code

```
if (my_rank != SERVER_NODE) {
    printf("I am the client, with rank %d of %d\n",
        my_rank, world_size);
    MPI_Get_processor_name(name, &length);
    sprintf(message,
        "Greetings from process %d%s!",
        my_rank, name);
    destination = 0; tag = 2;
    MPI_Send(message, strlen(message)+1, MPI_CHAR,
            destination, tag, MPI_COMM_WORLD);
```

MPI Hello World

Part 5: Server-side Code

```
} else {
    printf("I am the server, with rank %d of %d\n",
        my_rank, world_size);
    tag = 2;
    for (source = 1; source < world_size ; source ++) {
    MPI_Recv(message, 256, MPI_CHAR,
            source, tag, MPI_COMM_WORLD,
            &status);
    fprintf(stderr, "%s\n", message);
    }
}
```

MPI Hello World

Part 6: The Grand Finale

```
    printf("Calling Finalize %d\n",my_rank);

    MPI_Finalize();
}
```