Message Passing Interface

MPI Basics

Tom Murphy

Director of Contra Costa College
High Performance Computing Center

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?

"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!

Simple or Complex?

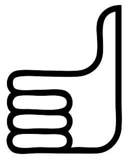
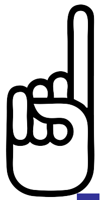
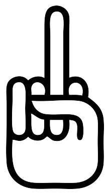
for now it's simple

- 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

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

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 Six

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 Six

Know thyself (and others)

- MPI_Comm_rank
  (MPI_Comm comm, int *rank)
- MPI_Comm_size
  (MPI_Comm comm, int *size)

Under the hood of the Six

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

Lets explore some code

- Fire up a qsub interactive shell on AC
  - ssh <account>@ac.ncsa.uiuc.edu
  - cp ~tra5/mpihello.c .
  - qsub –I
  - mpdstartup
  - mpicc –o mpihello mpihello.c
  - mpirun -np 4 ./mpihello

MPI Hello World

Lets explore some code

- Fire up a qsub interactive shell on sooner
  - ssh **yourusername**@sooner.oscer.ou.edu
  - cp -r ~hneeman/BWUPEP2011_exercises/MPIHello/ ~/BWUPEP2011_exercises/
  - cd ~/BWUPEP2011_exercises/MPIHello/C/
  - **Edit mpihello.bsub : replace e-mail address and username**
  - make_cmd
  - bsub < mpihello.bsub

MPI Hello World

A fugue in six parts

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

MPI Hello World

Part 1: Including the right stuff

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

#define SERVER 0
#define TAG 2
```

MPI Hello World

Part 2: General Declarations

```
int main(int argc, char **argv) {

  int my_rank, world_size;
  int destination=SERVER, source;
  int tag=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_Get_processor_name(name, &length);

sprintf(message, "Process %d (%d of %d) on %s!",
        my_rank, my_rank+1, world_size, name);
```

MPI Hello World

Part 4: Client-side Code

```
if (my_rank != SERVER) {          // Client Section
  fprintf(stderr, "Client: process %d\n", my_rank);
  MPI_Send(message, strlen(message)+1, MPI_CHAR,
           destination, tag, MPI_COMM_WORLD);
```

MPI Hello World

Part 5: Server-side Code

```
} else { // Server Section
   fprintf(stderr, "Server: process %d\n", my_rank);

   fprintf(stderr, "%s\n", message);
   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

```
    fprintf(stderr, "Calling Finalize %d\n",my_rank);
    MPI_Finalize();
}
```