Message Passing Interface

COMP750 Distributed Systems

MPI

- The Message Passing Interface (MPI) is a standard for programming parallel processors.
- MPI is a library of functions for Fortran, C and C++.
- There are several free implementations of MPI and several vendor implementations.
- Documentation and links to implementations are available on the class web site.

MPI Environment

- MPI uses message passing for inter-process communication.
- MPI can run on shared memory multi-processors, separate memory multi-processor or on a network of workstations.
- MPI assumes the Single Program Multiple Data (SPMD) model.
- MPI runs the same program on multiple CPUs.

Basic MPI program

```
#include "mpi.h"
void main(int argc, char *argv[]){
  // No MPI functions called before this
  MPI_Init(&argc, &argv);
  // MPI program here
  MPI_Finalize();
  // No MPI functions called after this
}
```

Communication

- MPI uses the concept of a ***communicator***, a collection of processes that can send and receive messages.
- For basic programs the only communicator needed is MPI_COMM_WORLD. It is predefined in MPI and consists of all the processes running when program execution begins.
- Communicators are of type **`MPI_Comm`**

Group Size

- An MPI program can learn how many processes are executing the program by

```
int MPI_Com_msize(MPI_Comm comm,
        int *size)
```

where:

- comm [in] is the communicator
- size [out] is the number of processes

Process ID

- An MPI program can learn its rank or process ID by

```
int MPI_Comm_rank(MPI_Comm comm,
                  int *rank);
```

where:

- comm [in] is the communicator
- rank [out] is this processes number from 0 to size-1.

Processor Name

- For documentation purposes, a process can learn its processor's name with:

```
MPI_Get_processor_name(
        char name[], int *namelen);
```

where

- name [out] is a char array to get the name
- namelen [out] receives the number of characters in the name.

Send Message

- One process can send a message to another process with MPI_Send

```
int MPI_Send( void *msg,
    int count, MPI_Datatype type,
    int dest, int tag,
    MPI_Comm comm);
```

where the all input parameters are:

- msg pointer to the data to send
- count number of data items to send

Send Message *(cont.)*

- dest The rank number of the destination
- tag An int < 64K used to identify different messages
- type The MPI_Datatype of the data item being sent. This is the MPI type, not the C data type.

MPI Data Types

MPI Data Type	C Data Type
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	unconverted

Send Semantics

- MPI_Send is a non-blocking buffered send.
- MPI_Send can send a single data item or an array of data items of the same type.

Receive Message

- Messages sent by MPI_Send can be received by MPI_Recv

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

where most parameters are identical to MPI_Send.

MPI_Recv

- source [in] The source of the message. Only message from this source will be accepted. If this parameter is MPI_ANY_SOURCE than messages from any source will be accepted.
- status [out] The status, source and tag of the message received.

MPI_Status Information

```
typedef struct {
    int MPI_SOURCE;      // source rank
    int MPI_TAG;         // send tag
    int MPI_ERROR;       // error status
} MPI_Status;
```

Broadcast Messages

- A process can send a message to all other processes with

```
int MPI_Bcast(void *msg,
    int cout, MPI_Datatype type,
    int root, MPI_Comm comm)
```

where all parameters are [in]

- msg is a pointer to the data to send.
- type is one of the MPI data types
- count the number of items to be sent.

MPI_Bcast *(cont)*

- root is the node that is sending the broadcast.
- Both sender and receivers call the **`MPI_Bcast`** function to effect a broadcast.
- If the process ID equals the root parameter then this node sends the message. All other nodes receive the message.

Combining Multiple Inputs

- A process can receive and combine messages from many other processes

int MPI_Reduce(void *operand, void *result, int count, MPI_Datatype, type, MPI_OP op, int root, MPI_Comm comm);

The specified operation is performed on all the input and operand to produce result.

Combining Operations

Operation Name	Meaning
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical And
MPI_BAND	Bitwise And
MPI_LOR	Logical Or
MPI_BOR	Bitwise Or
MPI_LXOR	Logical Exclusive Or
MPI_BXOR	Bitwise Exclusive Or
MPI_MAXLOC	Maximum and Location of Maximum
MPI_MINLOC	Minimum and Location of Minimum

Compiling MPI Programs

- MPICH is available for a variety of systems.
- The Microsoft Windows implementation provides libraries for gcc and Visual Studio
- Instructions for using Visual Studio are on the web.

Running MPI Programs

- The program executable must be located in the same directory on every processor.
- MPI_Config will establish the processors that will be used to run the program.
- The MPIrun program will start all the programs on every processor.