Parallel and Distributed Computing

CS3006

Lecture 14

Message Passing and MPI

11th May 2022

Dr. Rana Asif Rehman

Message Passing and MPI

Massage Passing Paradigm

Programming Using the Message Passing Paradigm

- Oldest and most widely used approach for distributed programming.
- The logical view of a machine supporting the message-passing paradigm consists of p processes, each with its own exclusive address space.
- Most of the communication is done using simple send/receive message passing.

Characteristics

- Provides high scalability
- Complex to program
- High communication costs
- No support for incremental parallelism

Message Passing Interface (MPI)

- MPI defines a standard library for message-passing that can be used to develop portable message-passing programs using either C or Fortran.

- The MPI standard defines both the syntax as well as the semantics of a core set of library routines.

- It is possible to write fully-functional message-passing programs by using only the six routines.

Message Passing Interface (MPI)

The minimal set of MPI routines.

`MPI_Init`	Initializes MPI.
`MPI_Finalize`	Terminates MPI.
`MPI_Comm_size`	Determines the number of processes.
`MPI_Comm_rank`	Determines the label of calling process.
`MPI_Send`	Sends a message.
`MPI_Recv`	Receives a message.

Starting and Terminating the MPI Library

- `MPI_Init` is called prior to any calls to other MPI routines. Its purpose is to initialize the MPI environment.
- `MPI_Finalize` is called at the end of the computation, and it performs various clean-up tasks to terminate the MPI environment.
- The prototypes of these two functions are:

```
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize()
```

- `MPI_Init` also strips off any MPI related command-line arguments.
- All MPI routines, data-types, and constants are prefixed by "`MPI_`". The return code for successful completion is `MPI_SUCCESS`.

Communicators

- A communicator defines a *communication domain*
  - a set of processes that can communicate with each other.
- Information about communication domains is stored in variables of type `MPI_Comm`.
- Communicators are used as arguments to all message transfer MPI routines.
- A process can belong to many different (possibly overlapping) communication domains.
- MPI defines a default communicator called `MPI_COMM_WORLD` which includes all the processes.

Querying Information

- The `MPI_Comm_size` and `MPI_Comm_rank` functions are used to determine the number of processes and the label of the calling process, respectively.
- The calling sequences of these routines are as follows:

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

- The rank of a process is an integer that ranges from zero up to the size of the communicator minus one.

Hello World Program


```
#include <mpi.h>
main(int argc, char *argv[])
{
    int np, myrank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    printf("From process %d out of %d, HelloWorld!\n",myrank, np);
    MPI_Finalize();
}
```

Sending and Receiving Messages

- The basic functions for sending and receiving messages in MPI are the `MPI_Send` and `MPI_Recv`, respectively.

- The calling sequences of these routines are as follows:

```
int MPI_Send(void *buf, int count, MPI_Datatype
datatype, int dest, int tag, MPI_Comm comm)
int MPI_Recv(void *buf, int count, MPI_Datatype
datatype, int source, int tag,MPI_Comm comm,
MPI_Status *status)
```

- MPI provides equivalent datatypes for all C datatypes. This is done for portability reasons.

- The message-tag can take values ranging from zero up to the MPI defined constant `MPI_TAG_UB`.

MPI Datatypes

MPI Datatype	C Datatype
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	

Sending and Receiving Messages

- MPI allows specification of **wildcard** arguments for both source and tag.
- If source is set to `MPI_ANY_SOURCE`, then any process of the communication domain can be the source of the message.
- If tag is set to `MPI_ANY_TAG`, then messages with any tag are accepted.

- On the receive side, the message must be of length equal to or less than the length field specified.

Sending and Receiving Messages

- On the receiving end, the status variable can be used to get information about the `MPI_Recv` operation.
- The corresponding data structure contains:

```
typedef struct MPI_Status {
  int MPI_SOURCE;
  int MPI_TAG;
  int MPI_ERROR; };
```

- `MPI_Status is usually used to take source and tag information in a 'receive' with wildcard entries on the corresponding positions.`

Sending and Receiving Messages

Example Program

```
if(my_rank==0){
	int sendBuff=10,tag=1,dest=1;
	printf("Process:%d is sending \'%d\' to process:%d \n",my_rank, sendBuff,dest);

	MPI_Send(&sendBuff, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);

}else if(my_rank==1){
	int recvBuff;int source=0,tag=1;

	MPI_Recv(&recvBuff, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
	printf("Process:%d is has received \'%d\' from process:%d\n",my_rank,
	recvBuff,source);

	}else{
	}
```

Questions

References

1. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (2017). *Introduction to parallel computing*. Redwood City, CA: Benjamin/Cummings.