15-440: Recitation 8

School of Computer Science
Carnegie Mellon University, Qatar
Fall 2011
Date: Oct 27, 2011

I- Intended Learning Outcome (ILO):

The ILO of this recitation is:

- Apply parallel programs according to the MPI standard.

II- Objectives:

- Provide students with a brief overview on the MPI programming model.
- Help students develop and run MPI parallel programs.

III- MPI Overview:

a- What is MPI?

- MPI = Message Passing Interface.
- MPI is a library of routines that can be used to create parallel programs.
- Major goal = Portability.

b- Reasons for using MPI

- Standardization.
- Portability.
- Performance Opportunities.
- Functionality.
- Availability.

c- Main Features:

- Can run simultaneously on different architectures.
- Independent and concurrent execution of programs.
- Each processor has private memory and address space.
- Deadlocks can occur.

d- Programming Model:

- With MPI all parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.

e- Fundamentals:

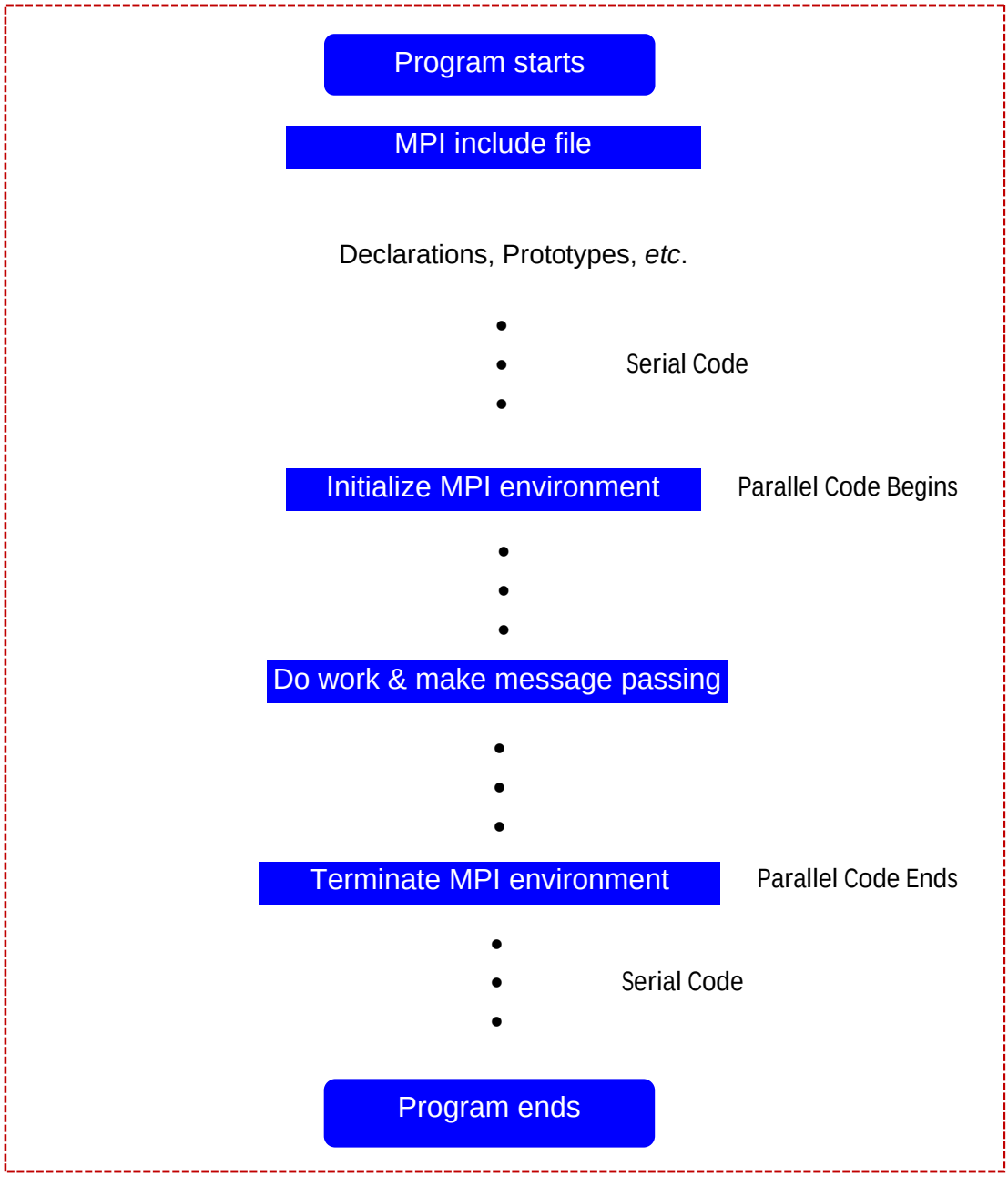
Communicators and groups:

- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other.
- Most MPI routines require you to specify a communicator as an argument.
- Communicators and groups will be covered in more detail in the class lectures. For now, simply use **MPI_COMM_WORLD** whenever a communicator is required - it is the predefined communicator that includes all of your MPI processes.

Rank:

- Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes. A rank is sometimes also called a "task ID". Ranks are contiguous and begin at zero.
- Used by the programmer to specify the source and destination of messages. Often used conditionally by the application to control program execution (if rank=0 do this / if rank=1 do that).

General MPI Program Structure:



f- Some of the Commonly MPI Routines (Will be used today in our lab):

1. ***MPI_Init(int *argc, char ***argv)***

This routine initializes the MPI execution environment. This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program.

2. ***MPI_Comm_size(MPI_Commcomm, int *size)***

This routine determines the number of processes in the group associated with a communicator. Generally used within the communicator MPI_COMM_WORLD to determine the number of processes being used by your application.

3. ***MPI_Comm_rank(MPI_Commcomm, int *rank)***

This routine determines the rank of the calling process within the communicator.

4. ***MPI_Wtime()***

This routine returns an elapsed wall clock time in seconds (double precision) on the calling processor.

5. ***MPI_Finalize()***

This routine terminates the MPI execution environment. This function should be the last MPI routine called in every MPI program - no other MPI routines may be called after it.

6. ***MPI_Send(void *buf, int count, MPI_Datatypedatatype, intdest, int tag, MPI_Commcomm)***

This routine is a basic blocking send operation. It returns only after the application buffer in the sending task becomes free for reuse.

7. ***MPI_Recv(void *buf, int count, MPI_Datatypedatatype, int source, int tag, MPI_Commcomm, MPI_Status *status)***

This routine receives a message and blocks until the requested data is available in the application buffer in the receiving task.

IV- **An example of MPI- *Hello World***

- Writing the HelloWorld MPI program:

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

int main(intargc, char **argv){

       intmyPID;
       intnum_procs;

       MPI_Init(&argc, &argv);

       MPI_Comm_rank(MPI_COMM_WORLD, &myPID);
       MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

       printf("Hello World. My ID = %d and the number of
processes created = %d\n", myPID, num_procs);

       MPI_Finalize();

       return 0;

}
```

- **Building and running an MPI program**(we assume the file name of your program is HelloWorld.c):

    - We already sat up MPI 2-1.4.1 for everyone of you on your 4VMs cluster.
    - Check that everything is in order at this point by doing:

    >$*which mpicc*
    >$*which mpiexec*

    All should refer to the commands in the bin subdirectory of your install directory.

    - Compile your program using ***mpicc***:

    >$mpiccHelloWorld.c -o HelloWorld

- Create a file named ***machinefile*** and include in it the list of the hosts that you want to run the executable on:

>$ cat machinefile

```
Host1        # Run 1 process on Host1
Host2:2      # Run 2 processes on Host2
```

- Now run your MPI program using ***mpiexec***:

>$mpiexec -f machinefile -n 2 ./HelloWorld

Hello World. My ID = 0 and the number of processes created = 2

Hello World. My ID = 1 and the number of processes created = 2

V- **Problem:**

Write an MPI program that will sum up elements of an array. The root process in your program acts as a master and the child processes act as slaves. The master allocates equal portions of the array to each slave. Each process then (the master as well as the slaves) calculates a partial sum of the portion of the array assigned to it. After done, each slave sends its partial sum to the master, who collect all partial sums from all the slaves and calculates a grand total.