Message Passing Interface

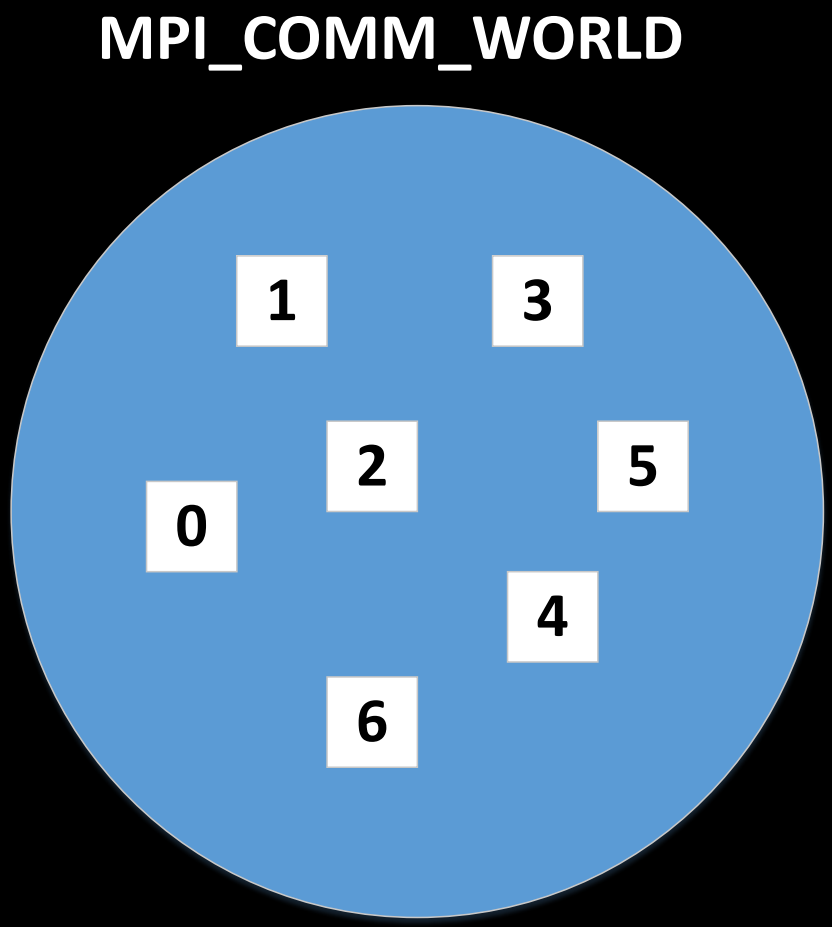
Professor Seokkoo Kang
Department of Civil & Environmental Engineering
kangsk78@hanyang.ac.kr

Compiling and Running

- Compile
  - mpicc: C code
  - mpicxx or mpiCC: C++ code
- Run
  - mpirun or mpiexec
  - Usage: mpirun –np 4 ./test

MPI Communicators

- A group of processes that communicate with each other



Rank

- An integer identifier assigned to each process
- Each process is assigned a unique number (0 … N-1)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

- If (rank==…) …

hello.c

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

main(int argc, char **argv)
{
  int node;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &node);

  printf("Hello World from Node %d\n",node);

  MPI_Finalize();
}
```

Compile and run the code

Run Terminal

> mpicc hello.c -o ./hello

> mpirun -np 4 ./hello

Output

Hello world from process 2 of 4

Hello world from process 0 of 4

Hello world from process 1 of 4

Hello world from process 3 of 4

Basic MPI functions

- MPI_Init: Initialize MPI
- MPI_Comm_rank: Get MPI rank information
- MPI_Comm_size: Get the number of the total rank
- MPI_Finalize: Finalize MPI
- MPI_Send: Send messages
- MPI_Recv: Receive messages

Send/Receive Functions

- Send and receive data between processes

  MPI_Send (buffer, count, datatype, dest, tag, comm)

  MPI_Recv (buffer, count, datatype, source, tag, comm, status)

buffer: data to send or receive

count: number of data

datatype: MPI_DOUBLE = double, MPI_INT = int, MPI_CHAR = char

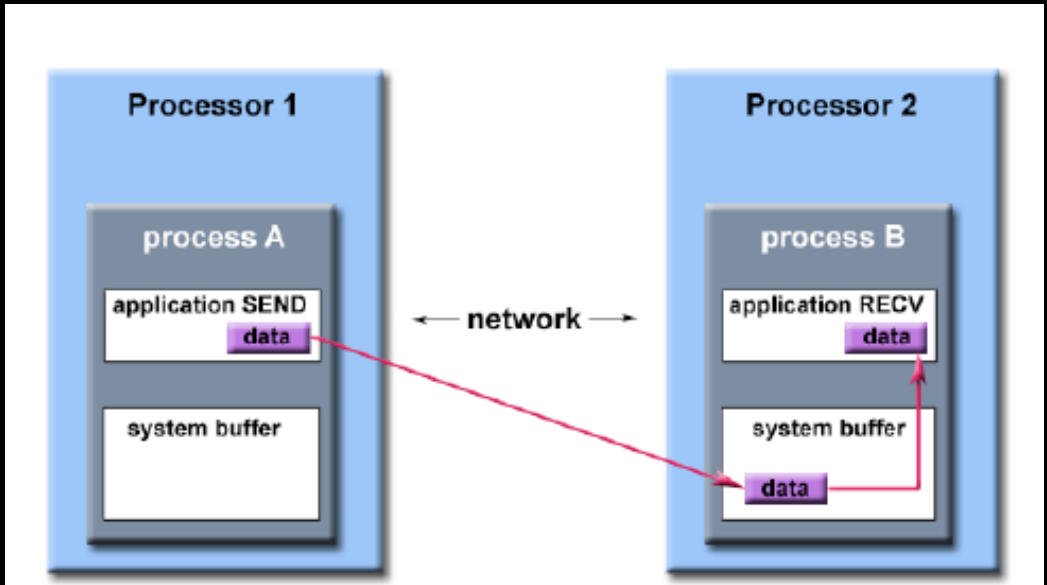
dest: rank # to which the data is sent.

tag: unique number assigned to each send/recv process; tags allow programmers to deal with the arrival of message in an orderly manner.

source: rank # from which the data is received.

Buffering

- A system buffer is reserved to hold data in transit.
- The buffer area is opaque to programmer and is managed by MPI library.
- Processor 2 will wait until RECV is called.



Path of a message buffered at the receiving process

Deadlock

- Two or more processes are in contention for the same set of resources
- Cause
  - All tasks are waiting for events that haven't been initiated yet
- Avoiding
  - Different ordering of calls between tasks
  - Non-blocking calls
  - Use of MPI_SendRecv

- **Below is an example that may lead to a deadlock**

Process 0	Process 1
Send(1)	Send(0)
Recv(1)	Recv(0)

- **An example that definitely will deadlock**

Process 0	Process 1
Recv(1)	Recv(0)
Send(1)	Send(0)

Note: the RECV call is blocking. The send call never execute, and both processes are blocked at the RECV resulting in deadlock.

- **The following scenario is always safe**

Process 0	Process 1
Send(1)	Recv(0)
Recv(1)	Send(0)

Case 1

- Process 0

  MPI_Send (buf, count, type, 1, tag, comm)

  MPI_Recv (buf, count, type, 1, tag, comm, status)

- Process 1

  MPI_Send (buf, count, type, 0, tag, comm)

  MPI_Recv (buf, count, type, 0, tag, comm, status)

- Possibly deadlock (sometimes not)
- If “count” is smaller than the system buffer size, it will not deadlock.

Case 2

- Process 0

  MPI_Recv (buf, count, type, 1, tag, comm, status)

  MPI_Send (buf, count, type, 1, tag, comm)

- Process 1

  MPI_Recv (buf, count, type, 0, tag, comm, status)

  MPI_Send (buf, count, type, 0, tag, comm)

- Always deadlock

Case 3

- Process 0

  MPI_Send (buf, count, type, 1, tag, comm)

  MPI_Recv (buf, count, type, 1, tag, comm, status)

- Process 1

  MPI_Recv (buf, count, type, 0, tag, comm, status)

  MPI_Send (buf, count, type, 0, tag, comm)

- Always safe

Example Send/Recv Code

```
// Find out rank, size
int world_rank, world_size;

MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int number;

if (world_rank == 0) {
   number = -1;
   MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1) {
   MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   printf("Process 1 received number %d from process 0\n",
         number);
}
```

Assignment

- Write a C/C++ code for calculating the sum from 1 to 10,000 and printing the result in two different ways.
  1. Calculate the partial sum in each rank (i) and send it to rank (0) so the rank (0) calculates and prints out the total sum.
  2. Send the accumulated sum from the rank (0) to (i) to the next rank (i+1). The final rank calculates and prints out the total accumulated sum.
- Use only MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize, MPI_Send, MPI_Recv functions.
- Run the code for np=1,2,3,4,..,10 and compare the final results.
- Upload all PPT material after each presentation.