Lecture 7

MPI Programming (III)

1

Topics

- Collective communication (cont'd)
- Point-to-point communication
  - Basic point-to-point communication
  - Non-blocking point-to-point communication
  - Four modes of blocking communication
- Process Topology
- Process Group
- Manager-Worker Programming Model

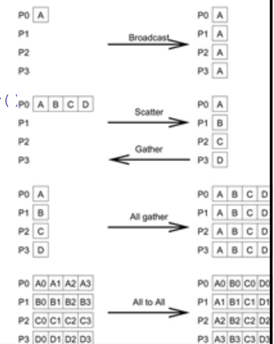
2

Collective Communication (cont'd)

3

Review

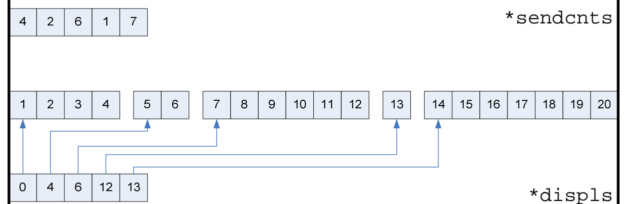
- Collective communication
  - MPI_Bcast(), MPI_Reduce()
  - MPI_Scatter(), MPI_Gather()
  - MPI_Allgather(), MPI_Allreduce()
  - MPI_Alltoall()
  - MPI_Barrier(), MPI_Scan()



Other MPI collective functions

```
int MPI_Alltoall(
   void* sendbuf, int scnts, MPI_Datatype sendtype,
   void* recvbuf, int rcnts, MPI_Datatype recvtype,
   MPI_Comm comm)

int MPI_Alltoallv(
   void* sendbuf, int *scnts, int *sdispls, MPI_Datatype sendtype,
   void* recvbuf, int *rcnts, int *rdispls, MPI_Datatype recvtype,
   MPI_Comm comm)

int MPI_Allgather(
   void* sendbuf, int scnts, MPI_Datatype sendtype,
   void* recvbuf, int rcnts, MPI_Datatype recvtype,
   MPI_Comm comm)

int MPI_Allgatherv(
   void* sendbuf, int  scnts, MPI_Datatype sendtype,
   void* recvbuf, int *rcnts, int *rdispls, MPI_Datatype recvtype,
   MPI_Comm comm)

int MPI_Reduce_scatter(
   void* sendbuf, void* recvbuf, int *rcnts,
   MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

5



6

08.cpp

- This is a program performing vector normalization (make the length of the vector to be unity).
  - Use MPI_Scatterv to distribute a vector onto all processors
  - Each processor compute its own squares of vector elements
  - Use MPI_Allreduce to sum sqaures of vector elements onto all processors
  - Each processor computes its normalized partial vector
  - Use MPI_Gatherv to put normalized vector back in one place.

7

Other MPI collective functions

```
void MPI_Barrier(MPI_Comm comm)

int MPI_Scan(
   void* sendbuf, void* recvbuf, int count,
   MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

int MPI_Op_create(
   MPI_User_function *function, int commute, MPI_Op *op)
int MPI_Op_free(MPI_Op *op)
```

commutative :
1: a#b = b#a
0: a#b != b#a

8

Synchronization

- MPI_Barrier() → used to synchronize all processes have called this subroutine.
  `int MPI_Barrier(MPI_Comm comm)`

- Processes started up on different machines run independently from each other. Therefore, different machines may be running different portion of a code in an instance, and running at different speed.
- It is sometimes necessary to ensure all processes are at the same point or at the same pace.
- For example, when friends go out for a long trip in different cars or motorcycles, it is necessary to set up some "synchronization point" so that everyone will reach the destination. (especially when there are drivers who doesn't know how to get there).
- Blocking communication usually results in synchronization.

Examples: 09a_noBarrier.c vs. 09b_barrier.c (compare the output)

9

MPI_Scan()

- Perform a scan ("partial reduction") of data
  - Also called "all-prefix-sums";

```
int MPI_Scan(void *sendbuf, void *recvbuf, int count,
       MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

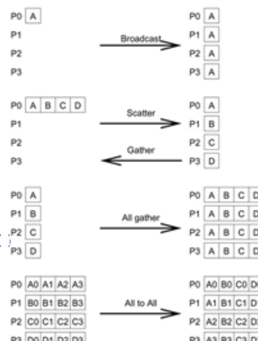
```
P0: [ 0  1  2]                    P0: [ 0  1  2]
P1: [ 3  4  5]     Count=3        P1: [ 3  5  7]
P2: [ 6  7  8]     MPI_SUM        P2: [ 9 12 15]
P3: [ 9 10 11]                    P3: [18 22 26]
```

Example: 10_scan.cpp

10

Summary

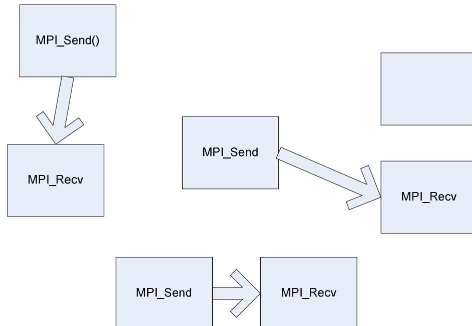
- Information Enquiry
  - MPI_Initialize()
  - MPI_Get_processor_name()
  - MPI_Get_version()
  - MPI_Comm_size()
  - MPI_Comm_rank()
  - MPI_Wtime()
  - MPI_Finalize()
- Collective communication
  - MPI_Bcast(), MPI_Reduce()
  - MPI_Scatter(), MPI_Gather()
  - MPI_Allgather(), MPI_Allreduce()
  - MPI_Barrier(), MPI_Scan()
  - MPI_Alltoall()



Basic
Point-to-point communication

12

Point-to-point Communication



Point-to-point Communication

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

// MPI_Bcast(&data, 1, MPI_INT, bRoot, MPI_COMM_WORLD);

if(rank==bRoot) {
    for(i=0;i<size;i++) {
       if(i != rank) {
          MPI_Send(&data, 1, MPI_INT,
                 i, aTag, MPI_COMM_WORLD);
       }
    }
}
else {
    MPI_Recv(&data, 1, MPI_INT,
           bRoot, aTag, MPI_COMM_WORLD, &mpistatus);
}
```


Broadcast

- The sequential algorithm in the previous slide is a naïve algorithm that works well for small-scale problems.
- Larger-scale broadcast usually uses tree with parallel p2p communications..

Round 1	Round 2	Round 3	Round 4
0 → 1	0 → 2	0 → 4	0 → 8
	1 → 3	1 → 5	1 → 9
		2 → 6	2 → 10
		3 → 7	3 → 11
			4 → 12
			5 → 13
			6 → 14
			7 → 15

Broadcast to n nodes requires $\log_2 n$ rounds

MPI_Send()

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

Input Parameters

buf	initial address of send buffer (choice)
count	number of elements in send buffer (nonnegative integer)
datatype	datatype of each send buffer element (handle)
dest	rank of destination (integer)
tag	message tag (integer)
comm	communicator (handle)

Notes
This routine may block until the message is received.

MPI_Recv()

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

Output Parameters

buf	initial address of receive buffer (choice)
status	status object (Status)

Input Parameters

count	maximum number of elements in receive buffer (integer)
datatype	datatype of each receive buffer element (handle)
source	rank of source (integer) → [MPI_ANY_SOURCE]
tag	message tag (integer) → [MPI_ANY_TAG]
comm	communicator (handle)

Notes
The count argument indicates the maximum length of a message; the actual number can be determined with MPI_Get_count.

MPI_Status

- The structure MPI_Status enables recipient to know various information regarding the incoming message.
- MPI_Status

The `MPI_Status` datatype is a structure. The three elements for use by programmers are

`MPI_SOURCE`
Who sent the message

`MPI_TAG`
What tag the message was sent with

`MPI_ERROR`
Any error return

Non-blocking point-to-point communication

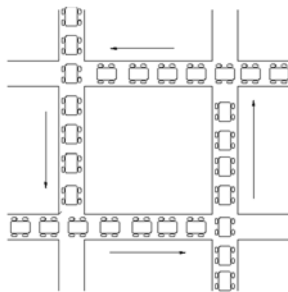
19

Non-blocking point-to-point communication

- Can be used to eliminate deadlocks
  - A situation wherein two or more processed are unable to proceed because each is waiting for one of the others to do something.
- Can be used to "overlap communication with computation"
  - To reduce T_s, the non-parallel portion of codes
  - Also known as latency hiding
  - This can be important for high-performance computing.

20

Deadlock



http://www.cs.rpi.edu/academics/courses/fall04/os/c10/gridlock.gif

21

Non-Blocking Communication Functions

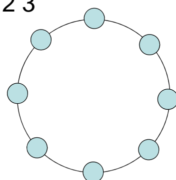
```
int MPI_Isend(void *buf, int count, MPI_Datatype datatype,
       int dest, int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype,
       int source, int tag, MPI_Comm comm, MPI_Request *request);


int MPI_Iprobe(int source, int tag, MPI_Comm comm, int *flag,
               MPI_Status *status )

int MPI_Probe(int source, int tag, MPI_Comm comm,
               MPI_Status *status )

int MPI_Wait(MPI_Request *request, MPI_Status *status);

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);

int MPI_Waitall(int count, MPI_Request array_of_requests[],
        MPI_Status array_of_statuses[]);
```

22

Examples: Passing value in a ring

- Each process passes a value (initially its own rank) to the right until it receives its rank back again. (Complete one cycle)
  - Three processes: 0 1 2 3

Rank	0	1	2	3
0	0	1	2	3
1	3	0	1	2
2	2	3	0	1
3	1	2	3	0
4	0	1	2	3



23

Example: Passing value in a ring

- Version 0: 11_ring_1.cpp
- Version 1: 12_ring_2.cpp
  - MPI_Send, MPI_Recv
  - Blocking point-to-point communication
- Version 2: 13_ring_3.cpp
  - MPI_Sendrecv() / MPI_Sendrecv_replace()
- Version 3: 14_ring_4.cpp
  - MPI_Isend, MPI_Irecv
  - Non-blocking point-to-point communication

24

Process Topology

Process Topology (1)

- It is sometimes convenient to arrange processes in certain topology to match the underlying algorithm that communicates with its neighbors.
- Example: Passing value in a ring
  - 15_ring_5.c
- More examples to come in future lectures
  - Matrix algorithms → 2-D Cartesian
  - 2-D Laplace equation solver → 2-D Cartesian
  - 3-D Laplace equation solver → 3-D Cartesian


```
int MPI_Cart_create (MPI_Comm comm_old, int ndims,
       int *dims, int *periods, int reorder, MPI_Comm *comm_cart);

int MPI_Cart_shift ( MPI_Comm comm, int direction, int displ,
                    int *source, int *dest )

int MPI_Cart_get (MPI_Comm comm, int maxdims, int *dims,
        int *periods, int *coords )

int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank );

int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims,
       int *coords )
```


Note: Italicized & underlined parameters are outputs!

16_cart.c

0 (0,0)	1 (0,1)	2 (0,2)
3 (1,0)	4 (1,1)	5 (1,2)
6 (2,0)	7 (2,1)	8 (2,2)

Process Group

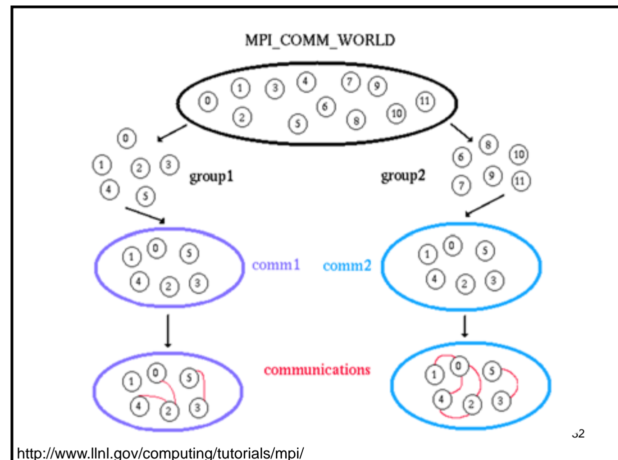
Communicators

- Communicator: a “handle” to a group of processes to perform communications
  - Collective communication: communicator
  - Point-to-point communication: communicator + rank
- With communicators
  - Many groups can be created
  - A process may belong to many different groups
- System defined the following two communicators for us after MPI_Init();
  - MPI_COMM_WORLD: all processes the local process can communicate with after initialization (MPI_INIT)
  - MPI_COMM_SELF: the process itself

Communicators

- MPI-1: In a static-process-model implementation of MPI, all processes that participate in the computation are available after MPI is initialized. (mpich, lam-mpi)
- In MPI-2 with dynamic process mode, processes can dynamically join an MPI execution.
  - In such situations, MPI_COMM_WORLD is a communicator incorporating all processes with which the joining process can immediately communicate.
  - Therefore, MPI_COMM_WORLD may simultaneously have different values in different processes.

31



http://www.llnl.gov/computing/tutorials/mpi/

32

Group communication

- It is sometimes convenient to sub-group MPI_COMM_WORLD to do group communications
- Example: 17_group.c
  - The entire "universe" is divided into two groups, even rank group & odd rank group
  - MPI_COMM_WORLD → group → sub-group → Create communicator from sub-group

33

Six categories in MPI APIs

- Point to point communication
  - MPI_Send, MPI_Recv, …
- Collective communication
  - MPI_Bcast, MPI_Reduce, MPI_Scatter, …
- Process topology
  - MPI_Cart_create, MPI_Cart_shift
- Groups, Contexts, and Communicators
  - MPI_Comm_Group, MPI_Group_incl, MPI_Comm_create, ..
- Environment Inquiry
  - MPI_Get_processor_name, MPI_Get_version, …
- Profiling
  - Not discussed!

34

Manager-Worker Programming Model

Manager-Worker Programming Model

- Originally known as Master-Slave Programming Model
- Manager-worker model can be used in a heterogeneous cluster, and automatically load-balance the available nodes
  - Example: finding n-narcissistic number
  - n-narcissistic number: An n-digit number which is the sum of the nth powers of its digits is called an n-narcissistic number, or sometimes an Armstrong number or perfect digital invariant (Madachy 1979). For example:

    153 = 1^3 + 5^3 + 3^3

    548834 = 5^6 + 4^6 + 8^6 + 8^6 + 3^6 + 4^6
  - 18_manager.cpp