CompSci131

Parallel and Distributed Systems

Prof. A. Veidenbaum

Today's topics

- **More MPI**

- **Reading assignment:**
  - **Today: 4.3 *and* lecture notes**
  - **Next lecture: Ch 5.**
    - » **Complete the assignment *before* next class**

Last Lecture Covered

- **Berkeley sockets**
- **MPI**

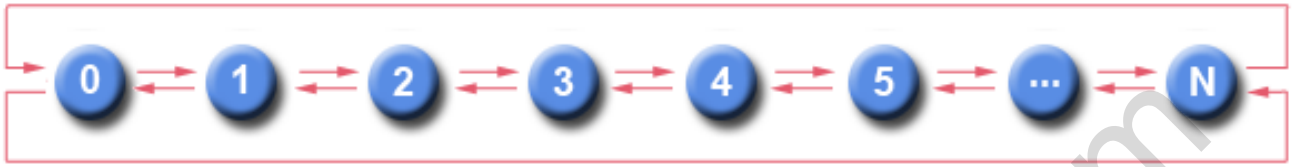
MPI Summary

- Communicator = set of tasks
- MPI_world = a communicator with all the tasks
- Rank = logical id within a communicator
  - 0 to N-1
- Blocking comm. Primitives : Send, Recv
  - Assumes communication is reliable and ordered!
    - » 2 message sent from task I to J are received by J in the same order
- Blocking means
  - Send blocks until its buffer is available for reuse
  - Recv blocks until its buffer has the message

MPI Summary

- Non-blocking comm. Primitives : Isend, Irecv
  - Async. Primitives, continue immediately
  - Return a request id for later tracking
    - » MPI_Wait takes the request id and will block until done
      - Same conditions as for Send, Recv to return
  - MPI_Test is a non-blocking test for a request
- Collective primitives: Barrier, Reduce

Example C program (LLNL tutorial)



- **Ring data exchange**

```
#include "mpi.h"
#include <stdio.h>
main(int argc, char *argv[]) {
int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;
MPI_Request reqs[4]; // required variable for non-blocking calls
MPI_Status stats[4];   // required variable for Waitall routine
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
prev = rank-1; next = rank+1;  // compute neighbours ranks
if (rank == 0) prev = numtasks - 1;
if (rank == (numtasks - 1)) next = 0;
```

Example program continued

- **After MPI_Init each process becomes part of**

```
// post non-blocking receives and sends for neighbors
MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD,
&reqs[0]);

MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD,
&reqs[1]);

MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD,
&reqs[2]);

MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD,
&reqs[3]);

// do some work while sends/receives progress in background

// wait for all non-blocking operations to complete
MPI_Waitall(4, reqs, stats);

// do more work

MPI_Finalize(); } own id (rank), total number of nodes
```

More useful MPI primitives

- MPI_Bcast (&buffer,count,datatype,root,comm)
  - Root task sends message to all others
- MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm)
  - Task with data distributes to all
- MPI_Gather = the opposite of MPI_Scatter
- MPI_Barrier
  - Wait for all
- MPI_Reduce

Other types of MPI primitives

- **Has a number of communication modes, primitives**

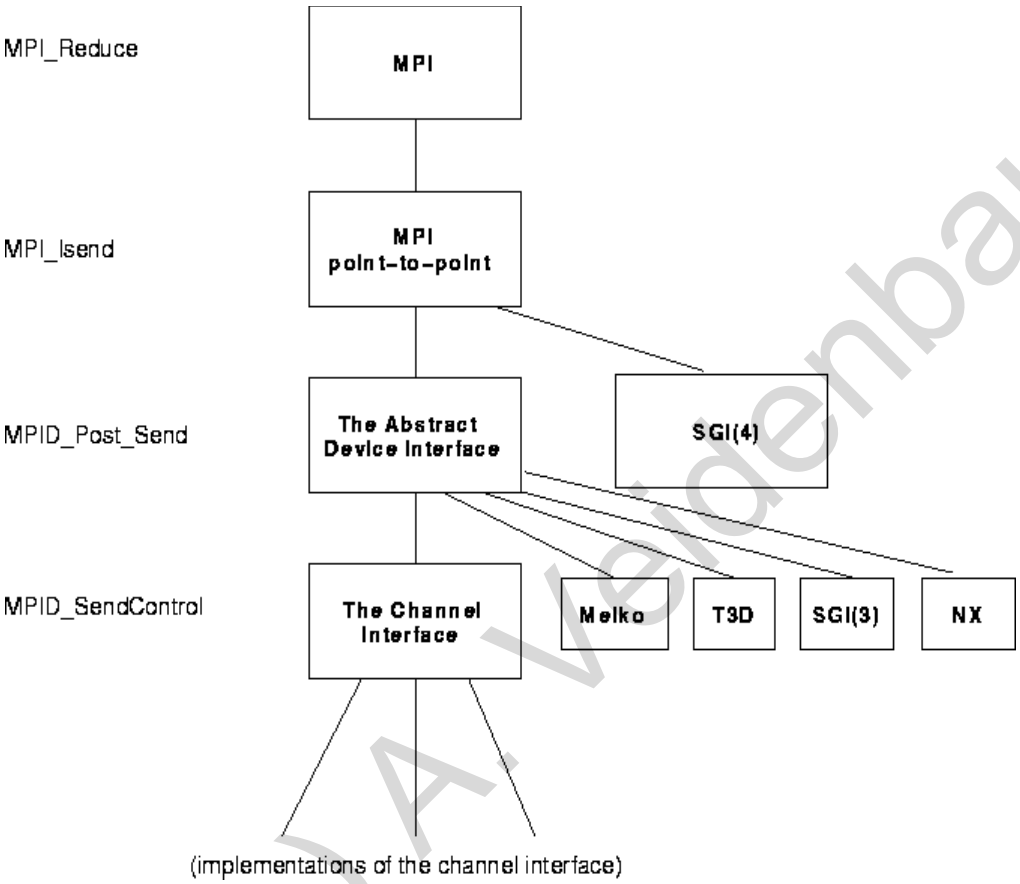
Primitive	Meaning
MPI_bsend	**Buffered send - append outgoing message to a local send buffer**
MPI_send	**Send a message and wait until copied to local or remote buffer**
MPI_ssend	**Synchronous send a message and wait until receipt starts**
MPI_sendrecv	**Send a message and wait for reply**
MPI_isend	**Pass reference to outgoing message, and continue**
MPI_issend	**Pass reference to outgoing message, and wait until receipt starts**
MPI_recv	**Receive a message; block if there are none**
MPI_irecv	**Check if there is an incoming message, but do not block**

- MPI_Irecv (&buf, count, datatype, dest, tag, comm, request)
- MPI_Wait(&request, &status)
  - Now block until finished
- MPI_Test(&request, &flag, &status)
  - Flag can be set when finished
- MPI_Get_count(&status, datatype, &count)

Portable MPI implementation (MPICH)

- From ANL, initially for MPI-1
- Goals: portability *and* good performance
- Defined ADI, a set of low-level primitives to
  - Implement all major MPI functions
  - Abstract away details of communication
    - » And possibly buffering
- One implementation defines an even lower interface, that requires only 5 functions

Abstract Device Interface (MPICH)



ADI functionality

- **Implements 4 sets of functions**
  1. specifying a message to be sent or received
  2. moving data between the ADI and the message-passing hardware
  3. managing lists of pending messages
     - » both sent and received
  4. providing basic information about the execution environment (e.g., how many tasks are there).
- **ADI layer contains the code for**
  - *packetizing messages* and attaching header information
  - managing multiple buffering policies
  - matching posted receives with incoming messages (or queuing them)
  - etc

Channel Interface

- **Five required functions**
  - **MPID_SendControl or MPID_SendControlBlock**
    - » **The latter is blocking**
  - **MPID_RecvAnyControl**
  - **MPID_ControlMsgAvail**
  - **MPID_SendChannel**
  - **MPID_RecvFromChannel**
- **Notice separation of control and data**
- **Can have additional functions**

How to implement these functions?

- **Can be done with standard Unix I/O operations**
  - **Select**
    - » **wait until the file descriptor becomes "ready" for some class of I/O operation (e.g., input possible)**
    - » **A file descriptor is ready if it is possible to perform the corresponding I/O operation without blocking**
      - **e.g., read(2)**
  - **Read**
    - » **from fd to the specified buffer, up to the specified # of bytes**
      - **“Up to” because of the EOF**
  - **Write**
    - » **to fd from the specified buffer, the specified # of bytes**

Buffering

- **Where is the message buffered?**
- **A critical issue for good performance**
- **The channel interface implements three data exchange mechanisms**
  - **Eager (MPICH default)**
    - » **data is sent to the destination immediately**
  - **Rendezvous**
    - » **When a receive is posted**
      - ***Control is always sent***
  - **Get**
    - » **Receiver reads data directly**
      - **Best on PGAS systems**

Sending Modes

- **Synchronous mode (MPI_Ssend):**
  - **the send does not complete until a matching receive has begun**
- **Buffered mode (MPI_Bsend)**
  - **the user supplies the buffer to system for its use**
- **Ready mode (MPI_Rsend)**
  - **user guarantees that matching receive has been posted.**
    - » ***undefined behavior if the matching receive is not posted***
- **Non-blocking versions**
  - **MPI_Issend, MPI_Irsend, MPI_Ibsend**

- ***Note that an MPI_Recv may receive messages sent with any send mode.***