Logistics

• Any Questions?

• Gone over the weekend to State Bowling Tournament.
Message Passing History

• Loosely based on Message Queues of System V Interprocess Communication Protocols (IPC).

• Common practice on early parallel systems.
  ➢ Usually Hypercube, ring, or Mesh Networks.
  ➢ Home grown message passing libraries.
  ➢ Vendor Specific libraries.

• Intel Hypercube systems
  ➢ Some of the first general purpose systems available to scientific community
  ➢ Did much to further development of Message Passing programming systems.
Message Passing History [2]

- In the late 1980s and early 1990s
  - Researchers and the vendor community started discussing the notion of an industry standard.
- 1994 MPI 1.0 standard was published
  - Don Heller formerly from the CS department and the Ames Laboratory’s SCL participated in this effort.
    - Now at PSU 😞
- 1995 MPI 1.1 published (maintenance release)
- 1997 MPI 1.2 and MPI 2.0
  - 1.2 updates/fixes of 1.1 standard
  - 2.0 introduced new features
    - One-sided communication and Process spawning
Lots of tools available

- Older tools mostly no longer used much
  - TCGMSG (research)
  - Express (ISV)
  - Nx (Intel)
  - P4 (Parallel Programs for Parallel Processors)

- Tools mostly used today
  - MPI Standard
    - Vendor Specific implementations and MPICH
      - MPICH derivatives
  - PVM (Parallel Virtual Machine)
  - New OpenMPI release
    - Has fault tolerance mechanisms

- Many more in both categories.
  - MP_Lite (developed at Ames Laboratory).
General Message Passing Model

- **Multiple instances of execution (SPMD).**
  - Separate processes with their own thread of control.
  - May execute different code
  - Both control and data parallelism are supported.

- **Asynchronous Parallelism**
  - Special constructs used to synchronize processes
    - Barriers and collective communication mechanisms

- **Separate Address Space**
  - One process cannot “see” memory of another process.

- **Explicit Interactions**
  - Programmer must code all interactions

- **Explicit Allocation**
  - Workload and data are explicitly assigned to specific processes.
Hypercube Network
Ring Network
Mesh Network
Ring on a Mesh Network
Communication Modes

• Important issues understanding of context
  ➢ How many processes (and possibly threads) are involved.
  ➢ How the processes are synchronized
  ➢ How are communication buffers are managed

• Three modes of operation
  ➢ Synchronous
  ➢ Blocking
  ➢ Non-blocking
Synchronous Mode

Time

send

receive

continue
Blocking Mode

- send
- continue
- buffer
- receive

Time
Blocking Mode [2]
Non-Blocking Mode

- send
- continue
- receive
- buffer
Non-Blocking Mode [2]
## Mode Comparison

<table>
<thead>
<tr>
<th>Communication Operation</th>
<th>Synchronous</th>
<th>Blocking</th>
<th>Non-Blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Start</td>
<td>Both Send and Received reached</td>
<td>Send reached</td>
<td>Send reached</td>
</tr>
<tr>
<td>Return of send</td>
<td>Message received</td>
<td>Message sent</td>
<td>Send initiated</td>
</tr>
<tr>
<td>Semantics</td>
<td>clean</td>
<td>Mostly clean</td>
<td>complex</td>
</tr>
<tr>
<td>Buffering</td>
<td>Not needed</td>
<td>needed</td>
<td>required</td>
</tr>
<tr>
<td>Status</td>
<td>Not needed</td>
<td>Not needed</td>
<td>required</td>
</tr>
<tr>
<td>Wait Overhead</td>
<td>High</td>
<td>modest</td>
<td>Some Minimized</td>
</tr>
<tr>
<td>Overlap of Communication and Computation</td>
<td>No</td>
<td>Some</td>
<td>Yes Maximized</td>
</tr>
</tbody>
</table>

3/6/2005  
ComS 425  
Spring 2005  
17 of 29: Lecture 20
Message Passing Libraries

• Most Message Passing libraries are either
  ➢ Blocking or Non-blocking
  ➢ Usually allow both.
  ➢ Few are synchronous.

• The mechanisms (yellow arrows) that move data lead to
  ➢ Overhead
    ▪ Each buffer implies a copy which is more overhead.
  ➢ Race conditions and Dead/Live Lock
    ▪ Must be prevented by library mechanisms
    ▪ Users are allowed to mess up 😊
Buffers in Message Passing

• Used in most libraries
  ➢ Allocated by the Message Passing Library
    ▪ Little or no control by the application
    ▪ Are limited in size by system resources
      ♦ The user code needs some memory too!
    ▪ Similar to disk I/O buffers
      ♦ E.g., read ahead and write behind mechanisms in UNIX
  ➢ Allocated by the user application
    ▪ Explicitly controlled by the developer
    ▪ Also limited by what the application resources that are required

• If you understand the C library \textit{aio\_} interface the semantics for message passing are very similar.
  ➢ Also System V IPC Messages are very similar
PVM & MPI Resources on the WWW

• **Official Forum Homepage**
  ➢ [www.mpi-forum.org](http://www.mpi-forum.org)

• **Argonne National Laboratory, MCS Links**

• **Official MPICH home page**

• **MP_Lite**

• **PVM Home Page**
Notes on PVM

- One of the first robust libraries available for message passing.
- Open Source from day one.
- Provided Significant input to MPI standardization efforts.
  - Helped lead the way
- Still under development today.
  - HARNESS is the new extension beyond Message Passing
    - Fault tolerance and WAN savvy
  - Components are in the new OpenMPI distribution
- Should not be used for normal message passing.
  - There is a standard use it.
MPI is a standard

- Okay it is a standard
  - Unix is a standard too right??
  - Implementations differ because interpretations of the standard differ.

- MPICH 1.2
  - A base implementation that has the proper functional semantics.
  - Open Source so that vendors can have something to work from.
    - API stays the same the details behind change over time.
  - Provides 100% of the MPI-1 standards
  - Small part of the MPI-2 standards.
  - MPICH 2.0 beta releases available
MPI-2

• Is relatively new
• Only Fujitsu, NEC, Hitachi, CRAY have a fully compliant MPI-2 standard (as of Jan 2005).
  ➢ Note the standard was published in 1997.
  ➢ Beta releases of MPICH-2 are available.
• The one-sided communications mechanisms in the standard are
  ➢ Not truly one-sided, passive and active modes
  ➢ Hard to implement
• Process spawning is hard to do
  ➢ Two modes fork or exec
  ➢ Setting up connections easy over UNIX socket mechanisms but not over a proprietary network system.
    ▪ E.g., myrinet, Quadrics, SCI
MPI

• We will use the MPI-1 Standard Programming Model for the course.
• There are many aspects to MPI programming
  ➢ Everything is Private
    ▪ Absolutely No Memory is shared among processes
  ➢ All interactions must be coordinated
    ▪ For every send there is a receive!
    ▪ Every task is specifically allocated to a process
  ➢ Types of interactions
    ▪ Collective
      ♦ Init, broadcast, reductions, barriers
    ▪ Non-collective or point-to-point
      ♦ Send, receive
How to compile and run

• **Compile**
  - C        mpicc file.c –o file
  - C++     mpiCC file.C –o file
  - FORTRAN77  mpif77 file.f –o file
  - FORTRAN90  mpif90 file.f90 –o file

• **Running the application**
  - mpirun –np 3 file
  - Give me three processes from the default “machine_file”
    ▪ More about this later.
MPI Hello World Example

```c
#include <stdio.h>
#include <string.h>
#include "mpi.h"
#define BUF_LEN 255

int main(int argc, char* argv[]) {
    int rank_id;    /* rank id of process */
    int nproc;      /* number of processes */
    int sorc;       /* rank of sending proc */
    int dest;       /* rank of receive proc */
    int tag = 0;    /* message tag */
    char mydata[BUF_LEN]; /* message buffer */
    MPI_Status status;  /* return status */
}```
Hello World Example [2]

dest = 0; /* process to print messages */

/* Start up MPI */
MPI_Init(&argc, &argv);

/* Find out process rank id */
MPI_Comm_rank(MPI_COMM_WORLD, &rank_id);

/* Find out number of processes */
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
Hello World Example [3]

if (rank_id != 0) {
    /* Create message and send message to 'dest' */
    sprintf(mydata, "Hello there from process %d!", rank_id);
    MPI_Send(mydata, strlen(mydata)+1, MPI_CHAR, 
              dest, tag, MPI_COMM_WORLD);
} else { /* rank_id == 0 */
    for (sorc = 1; sorc < nproc; sorc++) {
        MPI_Recv(mydata, BUF_LEN, MPI_CHAR, sorc, tag, 
                  MPI_COMM_WORLD, &status);
        printf("%s\n", mydata);
    }
}

/* Shut down MPI */  MPI_Finalize();
}
Processes and processors

• The MPI standard deals with processes
  ➢ Single address space software construct
  ➢ Can be multi-threaded
    ▶ Yes this is scary 😊

• Processors are a piece of hardware

• Some implementations:
  ➢ Limit one MPI process per processor
  ➢ Allow multiple MPI processes per processor
  ➢ Limit one MPI process per SMP node

• The MPI standard does not dictate this at all.
  ➢ Implementation dependant
  ➢ Users/developers must know how their implementation works!