This presentation was originally from Davies Muche & Jason Shields. It has been modified for CSC469/585 class
Outline

• A brief Background on Parallel Computers
• A Brief History of MPI
• MPI
What is a Parallel Computer?

- A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem.

This definition is broad enough to include parallel supercomputers that have hundreds or thousands of processors, networks of workstations, multiple-processor workstations, and embedded systems.
More on Parallel Computers

• At the heart of all parallel machines is a collection of processors, often in the hundreds or thousands

• Each processor has its own cache and local memory (perhaps)

• Parallel computers all have some communication facility
  – powerful switches are normally used to interconnect processors, while busses are used to connect processors in local clusters
Models of Parallel Machines

• There are three most important classes of parallel machines
Shared Memory

Diagram showing shared memory architecture with processors (P), memory (M), and disks connected through a switch or bus.
Shared Disk
Shared Nothing
Shared Nothing Continued

- All communication is via the communication network, from processor to processor
- Relatively inexpensive to build
- If one processor P wants to read data from the disk of another processor Q, Processor P sends a message to Q and sends the data over the network in another message
  - Both processors must execute a program that supports message transfer
• The concept of message transferring so that processes communicate with other processes by sending and receiving messages, is the core of the Message Passing Interface (MPI)
What is MPI?

• MPI is a message-passing library specification proposed as a standard by a committee of vendors, implementers, and users. It is designed to permit the development of parallel software libraries

• WHAT ITS NOT!
  - A compiler
  - A specific Product
A Brief History of MPI

• Related work before the MPI Standard:
  • PICL, PVM - Oak Ridge National Laboratory
  • PARMACS, P4, Chameleon - Argonne National Laboratory
  • Express - Caltech/ParaSoft
  • LAM - Ohio Supercomputer Center
  • TCGMSG - Specially designed for Quantum Chemistry
  • ISIS (Cornell University)
  • Linda (Yale),
Birth of MPI

• April 1992, during one-day workshop on Standards for Message Passing in Distributed-Memory Environment, they all realized that they were continuously reinventing the wheel duplicating each other's efforts.

• They got together (Supercomputing '92) and decided to thrash out a common standard in the same hotel in Dallas, where HPF Forum met.

• The first standard (MPI-1.0) was completed in May 1994.

• The Beta version of the second, enhanced standard (MPI-2.0) was released in July of 1997.

• Industrial participants included Convex, Cray, IBM, Intel, Meiko, nCUBE, NEC, Thinking Machines.
Where to use MPI

• MPI is primarily for SPMD and MIMD types of parallel computing environments

SPMD – Same Program, Different Data

MIND – Different Programs, Different Data
Features of MPI

• MPI is large. Its extensive functionality require many functions (126 functions)

• However, MPI can be small too!

Many parallel programs can be written with just six basic functions
The Six Basic Functions of MPI

MPI_INIT (Int *arg, char ***argv)

*Initiate an MPI computation.*

*argc, argv are required only in C language binding, where they are main program’s argument.*

MPI_FINALIZE ()

*Shutdown a computation.*

MPI_COMM_SIZE (comm, size)

*Determine the number of processes in a computation.*

*comm communicator (handle)*

*size number of processes in the group of comms (Integer)*
MPI_COMM_RANK (comm, rank)

Determine the identifier of the current process.

comm    communicator (handle)
rank or pid    process id in the group of comm (Integer)
Basic functions continued ...

MPI_SEND (buf, count, datatype, dest, tag comm)

Send a message.

buf Address of send buffer (choice)
count number of elements to send (Integer >=0)
datatype datatype of send buffer elements (handle)
dest process id of destination process (Integer)
tag message tag (Integer)
comm communicator (handle)
Basic functions continued …

MPI_RECV (buf, count, datatype, source, tag, comm, status)

Receive a message.

buf Address of receive buffer (choice)
count size of receive buffer, in elements (Integer >=0)
datatype datatype of receive buffer elements (handle)
source process id of source process, or MPI_ANY_SOURCE (Integer)
tag message tag, or MPI_ANY_TAG (Integer)
comm communicator (handle)
status status object (status)
Sample C Program

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

main(int *argc, char** argv) {
    int size, rank;
    MPI_Init(argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Hello world! I'm %d of %d\n", rank+1, size);
    MPI_Finalize();
}
Sample Output run over four processors

Hello world! I’m 1 of 4
Hello world! I’m 3 of 4
Hello world! I’m 0 of 4
Hello world! I’m 2 of 4

Since the Printf is a local statement every processor execute it
Compiling and Running MPI Programs

Compiling:

Mpicc can be used to compile small programs
Example: `mpicc –o execute myprog.c`
For larger programs, it is ideal to make use of a makefile

Running:

The MPI standard does not specify how a parallel computation is started
Example: a typical mechanism could be a command line argument indicating the number of processes that are to be created: for example, `myprog -n 4`, where myprog is the name of the executable
Commentary

• All MPI programs must include a header file. mpi.h

• All MPI programs must call MPI_INIT as the first MPI call.  
  \textit{This establishes the MPI environment.}

• All MPI programs must call MPI_FINALIZE as the last call,  
  \textit{this terminates the MPI environment.}

• Only one invocation of MPI_INIT can occur in each program

• NO calls to MPI can be made after MPI_FINALIZE is called

• All non MPI routine are local; i.e. Print*, ‘Hello world! ...’  
  runs on each processor
Parallel Programs in MPI

- Can be done by data partitioning among processes.
- Local calculations can be used to form the final result.
- In MPI one can use the rank of the process to divide data.

```plaintext
// Suppose we have an interval from "a" to "b".
// We want to calculate the integral over it.
// Then each process takes its share of data in this fashion.
width = (b-a)/n
local_a = a + my_rank*width;
local_b = local_a + width;
```
Parallel Programs

```cpp
if (my_proc_rank == 0)
    MPI_Send(..);
else if (my_proc_rank == 1)
    MPI_Recv(..);
```

- This approach is called "Single program multiple data"
More MPI functions

- MPI has more than 126 functions.
- Most used 6 functions.
  - See previous slides
- Others
  - MPI_Bcast
  - MPI_reduce
MPI_Bcast

- Broadcasts a message from the process with rank "root" to all other processes of the communicator
- `int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm )`
- **Buffer**…starting address of buffer (choice)
- **Input Parameters**
  - **count**
  - number of entries in buffer (integer)
  - **datatype**
  - data type of buffer (handle)
  - **root**
  - rank of broadcast root (integer)
  - **comm**
  - communicator (handle)
MPI_Reduce

- int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

- **Input Parameters**
  - **sendbuf**
    - address of send buffer (choice)
  - **count**
    - number of elements in send buffer (integer)
  - **datatype**
    - data type of elements of send buffer (handle)
  - **op**
    - reduce operation (handle)
  - **root**
    - rank of root process (integer)
  - **comm**
    - communicator (handle)

- **Output Parameter**
  - **recvbuf**
    - address of receive buffer (choice, significant only at root)
PI calculation in MPI

- The method evaluates the integral of $4/(1+x^2)$ between $-1/2$ and $1/2$.
- The integral is approximated by a sum of $n$ intervals; the approximation to the integral in each interval is $(1/n)*4/(1+x^2)$.
PI calculation in MPI

- The master process (rank 0) asks the user for the number of intervals; the master should then broadcast this number to all of the other processes.

- Each process then adds up every n'th interval \((x = -1/2+rank/n, -1/2+rank/n +size/n,\ldots)\). Finally, the sums computed by each process are added together using a reduction in MPI.
References:

- http://www.mgnet.org/~douglas/ccd-old-classes.html
- http://beige.ucs.indiana.edu/B673/node115.html

- ns/sc99/sc99_tutorial_files/frame.htm