MPI – Introduction

Ingyu Lee and Padma Raghavan

CSE 457 & CSE 557

Contents

- Message Passing Model
  - What is MPI ?
- How to run MPI program (in IST220)
  - Writing, Compiling and linking
  - Running MPI programs
  - Setup the environment
- Example codes
  - Point-to-point communication
    - Hello world !!
  - Collective communication
    - Compute Pi

Data Parallel vs. Message Passing Model

- Data Parallel
  - Single thread of control consisting of parallel operations
    - E.g. A processor per array element
      - Pointwise, Broadcast, nearest neighbor comm. Reductions.
    - Wrong level of granularity for current machines, Mapping is hard
- Message passing is
  - More general than data parallelism: not tightly synchronized
  - Potentially faster: programmer does mapping
  - What people use on any machine larger than 100 processors (including SMP)

What is MPI ?

- A Message Passing Library specificiation
  - Message-passing model
  - Not a language or compiler specification
  - Not a specific implementation or product
- For Parallel computers, clusters
  - Designed to permit the development of parallel software
  - Designed to provide access to advanced parallel hardware
- Not designed for fault tolerance

History of MPI

- MPI Forum: government, industry and academia.
  - Formal process began 11, 1992
  - Standard (1.0) 5, 1994
  - Standard(1.1) 6,1995
  - MPI-1.2, 7, 1997
  - MPI-2, 7, 1997
- Current product (MPI-1)
  - Public domain versions ANL/MSU (MPICH), OSC(LAM)
  - Proprietary versions available from all vendors

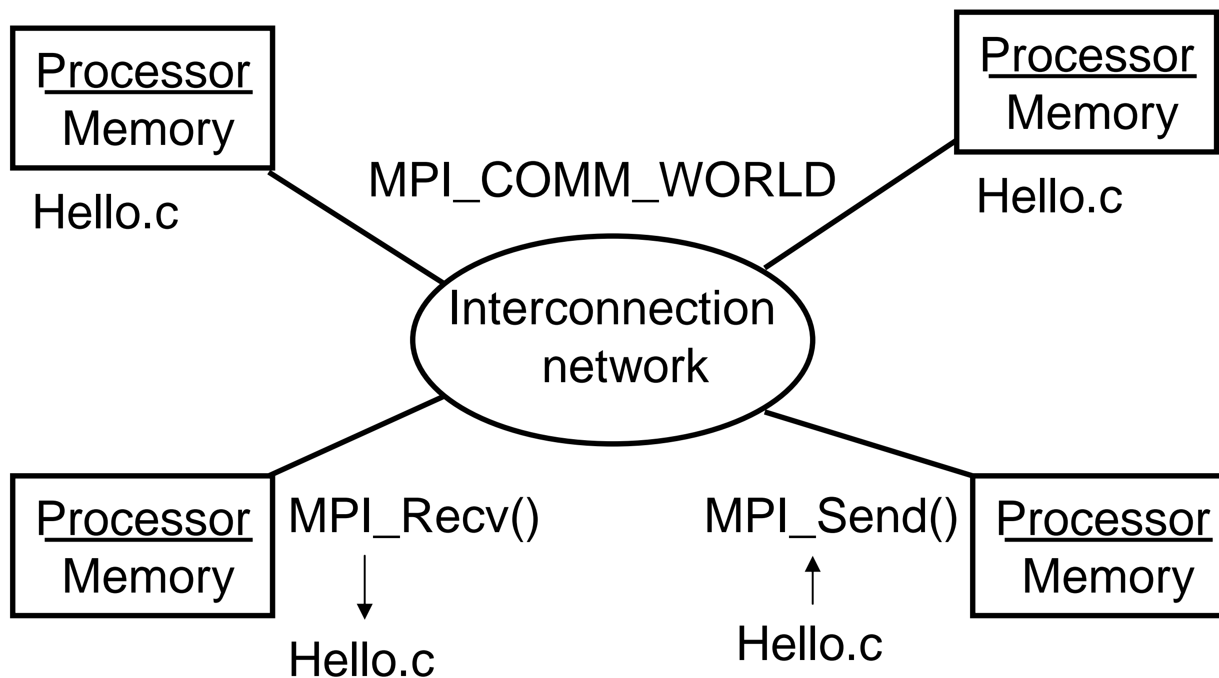
Parallel Programming Overview

- Creating Parallelism
  - SPMD model
- Communication between processors
  - Basic
  - Collective
  - Non-blocking
- Synchronization
  - Point-to-point synchronization is done by message passing
  - Global synchronization done by collective communication

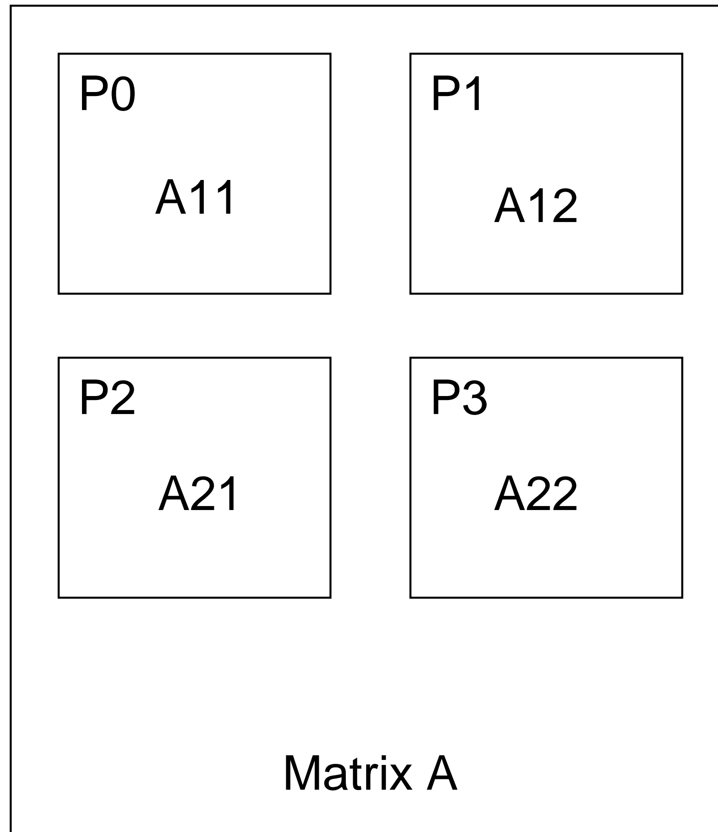
SPMD Model

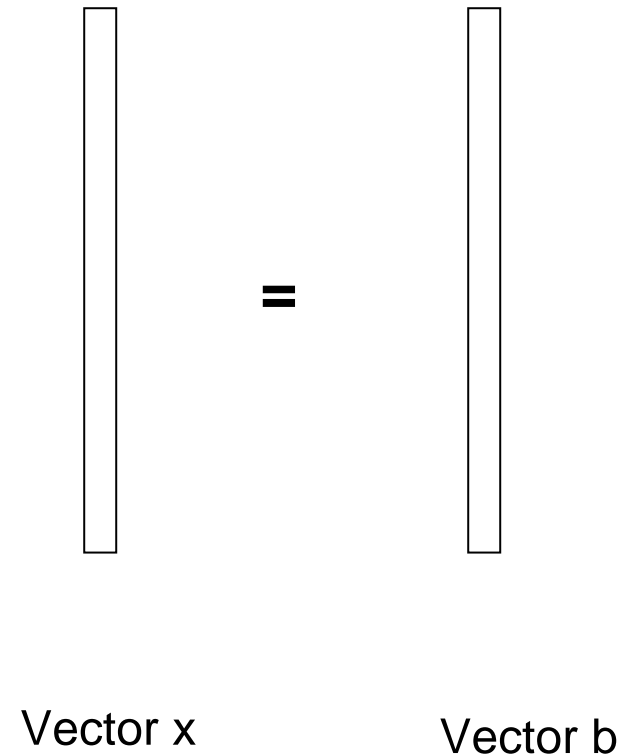
- Single Program Multiple Data model
  - Each processor has a copy of the same program
  - All run them at their own rate
  - May take different paths through the code
- Process-specific control through variables like:
  - My process number
  - Total number of processors
- Processors may synchronize, but nothing is implicit---everything is explicit

Message Passing Model



Matrix Vector Multiplication





Features of MPI

- General
  - Communicators combine context and group
  - MPI_Init(), MPI_Comm_rank(), MPI_Comm_size(), MPI_Finalize()
- Point-to-point communication
  - Structured buffers and derived datatypes
  - Modes: blocking and non-blocking
  - MPI_Send(), MPI_Recv()
- Collective
  - Both built-in and user-defined collective operations
  - Large number of data movement routines
  - Subgroups defined directly or by topology
  - MPI_Bcast(), MPI_Reduce()

Features of MPI(cont')

- Process groups
  - Built-in support for grids and graphs.
- Profiling
  - Hooks allow users to intercept MPI calls to install their own tools- lookup, jumpshot
- System Environmental
  - Inquiry
  - Error control
- Non-message-passing concepts not included
  - Process management
  - Remote memory transfers
  - Active messages
  - Threads

Compile and Run

- MPI home directory
  - General, /usr/local/mpich
  - In IST220, /home/software/mpich
  - Add path in your .cshrc.cat
    - # Add MPI directories to the program search path.
    - set path = (/usr/software/mpich/bin $path)
- Compile Commands
  - mpicc -c hello.c
  - mpif77 -c hello.f
- Link
  - mpicc -o hello hello.o -lmpich

Makefile

```
CC = /home/software/mpich/bin/mpicc
CP = /home/software/mpich/bin/mpiCC

INC = -I/home/software/mpich/include
MPILIB = -L/home/software/mpich/lib -lmpich -lmpe -lsocket -lnsl

all: hello_world_cc hello_world cpi1 cpi2 cpilog
hello_world_cc: hello_world_cc.o
	$(CP) -o hello_world_cc hello_world_cc.o $(MPILIB)
hello_world: hello_world.o
	$(CC) -o hello_world hello_world.o $(MPILIB)
cpi1: cpi1.o
	$(CC) -o cpi1 cpi1.o -lm $(MPILIB)
cpi2: cpi2.o
	$(CC) -o cpi2 cpi2.o -lm $(MPILIB)
cpilog: cpilog.o
	$(CC) -o cpilog cpilog.o -lm $(MPILIB)
.c.o:
	${CC} $(INC) $@ -c  $<
.cc.o:
	${CP} $(INC)  -c  $<
clean:
	rm *.o; rm hello_world_cc hello_world cpi1 cpi2
```

Compile and Run(cont.)

- Run program
  - mpirun –np 2 –machinefile sample.list hello
- Sample.list
  - narn
  - zerg
  - protoss
  - minbari
- Setting .rhosts (vi ~user-id/.rhosts)
  - narn.cse.psu.edu user-id
  - zerg.cse.psu.edu user-id
  - protoss.cse.psu.edu user-id
  - minbari.cse.psu.edu user-id

Six Function MPI

- These six functions allow you to write many programs
  - MPI_Init
    - Initialize MPI
  - MPI_Comm_size
    - To find the number of processes
  - MPI_Comm_rank
    - To determine a process's ID number
  - MPI_Send
    - A process to send a message to another process
  - MPI_Recv
    - A process to receive a message sent by another process
  - MPI_Finalize
    - To shut down MPI

Simple Program – Hello World (1)

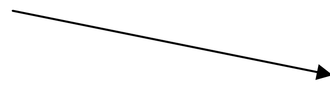
```
#include "mpi.h"
#include <stdio.h>

int main(argc,argv)
int argc;
char **argv;
{
        Int rank, size;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&size);
        Printf("Hello World ! I am %d of %d\n",rank,size);
        MPI_Finalize();
        Return 0;
}
```

MPI Basic Send/Receive

- We need to fill in the details in

Process 0   Process 1

Send(data)



Receive(data)

- Things that need specifying
  - How will processes be identified
  - How will data be described
  - How will the receive recognize
  - What will it mean for these operations to complete

Identifying Processes: MPI Communicators

- In general, processes can be subdivided into groups:
  - Group for each component model
  - Group to work on subdomain
- Supported using a “communicator” a message context and a group of processes
- In MPI program all processes do the same thing
  - The set of all processes make up the “world”
    - MPI_COMM_WORLD
  - Name processes by number (called “rank”)

MPI Datatype

- MPI datatype is predefined, corresponding to a data type form the language
  - MPI_CHAR
  - MPI_DOUBLE
  - MPI_FLOAT
  - MPI_INT
  - MPI_LONG
  - MPI_LONG_DOUBLE
  - MPI_SHORT
  - MPI_UNSIGNED_CHAR
  - MPI_UNSIGNED
  - MPI_UNSIGNED_LONG
  - MPI_UNSIGNED_SHORT

MPI Tags & Status

- Messages are sent with a user-defined integer tag
  - Allow receiving process in identifying the message
  - Receiver may also screen messages by specifying a tag
  - Use MPI_ANY_TAG to avoid screening
- Status is a data structure allocated in the user's program

MPI_SEND & MPI_RECV

- MPI_SEND(start,count,datatype,dest,tag,comm)
  - Completion: When this function returns, the data has been delivered to the system and the data structure can be reused. The message may not have been received by the target process
- MPI_RECV(start,count,datatype,source,tag,comm,status)
  - Waits until a matching message is received from the system, and the buffer can be used
  - Receiving fewer than count occurrences of datatype is OK but receiving more is an error

Simple Program – Hello World (2)

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

int main(argc,argv)
int argc;
char *argv[];
{
        Int rank, n, message;
        Char buff[1000];
        MPI_Status status;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&n);
        If (rank==0) { // Process 0 will output data
            printf("\n Enter anything to get started\n"); scanf("%s",buff);
            printf("\n You entered: %s\n",buff);
            printf("\n Hello from process %3d",rank);
            for (i=1;i<n;i++) {
                MPI_Recv(&message,1,MPI_INT,MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,&status);
                printf("\n Hello from process %3d\n", message);
            }
        } else
            MPI_Send(&rank,1,MPI_INIT,0,111,MPI_COMM_WORLD);

        MPI_Finalize();
        Return 0;
}
```

Simple Program – Pi

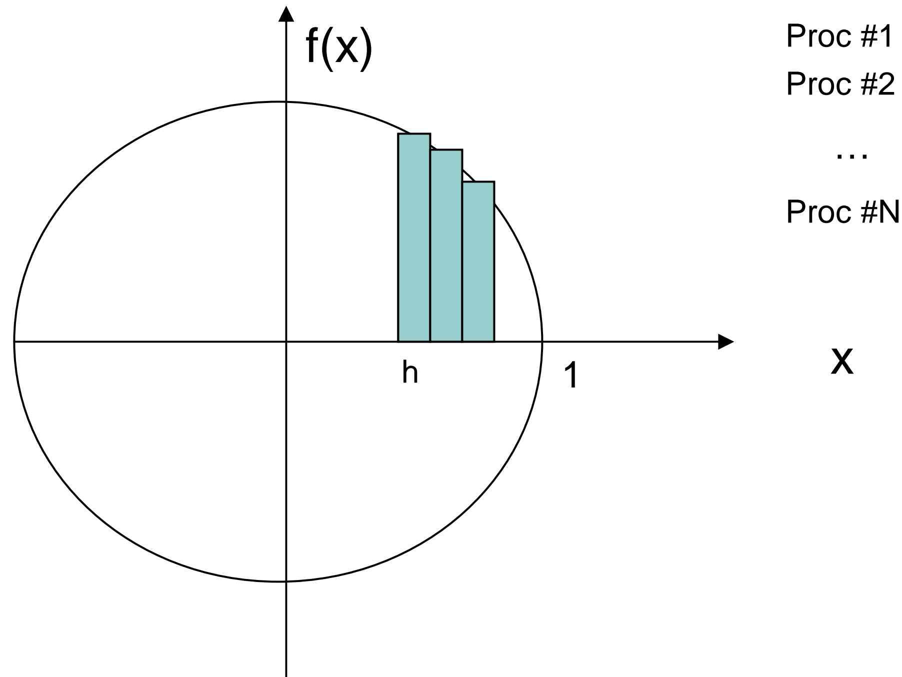
- Taylor series

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} \quad for\ |x|<1$$

$$\frac{d}{dx}\arctan x = 1 - x^2 + x^4 - x^6 + \ldots$$

$$\arctan 1 = \int_0^1 (1 - x^2 + x^4 - x^6 + \ldots)dx = \int_0^1 \frac{1}{1+x^2}dx$$

Simple Program – Pi



$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx = h\sum_{i=1}^{n} \frac{4}{1+x_i^2} = h\sum_{i=1}^{n} \frac{4}{1+((i-1/2)h)^2}$$

Simple Program – Pi

- Collective operations to send data to and from all of the running processes.
  - MPI_Init
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Bcast
  - MPI_Reduce
  - MPI_Finalize

Collective communication

- Collective operations are called by all processes in a communicator
  - MPI_Bcast
    - Sends data from one process(root) to all others.
    - MPI_Bcast(*buffer,count,MPI_Datatype,root,MPI_Comm)
- MPI_Reduce
  - Combines data from all processes (by adding them in this case), and returning the results to a single process.
  - MPI_Reduce(*operand,*result,count,MPI_Datatype,MPI_Op,root,MPI_Comm)

MPI built-in reduction operators

- MPI_BAND
- MPI_BOR
- MPI_BXOR
- MPI_LAND
- MPI_LOR
- MPI_LXOR
- MPI_MAX
- MPI_MAXLOC
- MPI_MIN
- MPI_MINLOC
- MPI_PROD
- MPI_SUM

Simple Program – Pi (cont')

```
#include "mpi.h“
 #include <math.h>

int main(argc,argv)
int argc;
char *argv[];
{
      int done = 0, n, myid, numprocs, i, rc;
      double PI25DT = 3.141592653589793238462643;
      double mypi, pi, h, sum, x, a;

      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
      MPI_Comm_rank(MPI_COMM_WORLD,&myid);
```

Simple Program – Pi (cont')

```
while (!done) {
   if (myid == 0) {
     printf("Enter the number of intervals: (0 quits) ");
     scanf("%d",&n);
     }
     MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
     if (n == 0) break;
     h = 1.0 / (double) n;
     sum = 0.0;
     for (i = myid + 1; i <= n; i += numprocs) {
          x = h * ((double)i - 0.5);
          sum += 4.0 / (1.0 + x*x);
     }
     mypi = h * sum;

   MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
   if (myid == 0)
             printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
 }

 MPI_Finalize();
}
```

Non-blocking communication

- The following is called an “unsafe” MPI program
  - Process0 Process1
    - Send(1) Send(0)
    - Recv(1) Recv(0)
- Split communication operations into two parts
  - First part initiates the operation. It does not block
  - Second part waits for the operation to complete
  - MPI_Recv
    - MPI_Irecv
    - MPI_Wait

Using Non-blocking Receive

- Two advantages
  - No deadlock (correctness)
  - Data may be transferred concurrently (performance)
- Obvious caveats
  - You may not modify the buffer between Isend() and corresponding wait()
  - You may not look at or modify the buffer between Irecv() and corresponding wait()

Communicators

- What is MPI_COMM_WORLD
  - A communicator consists of
    - A group of processes
      - Numbered 0,…,N-1
      - Never changes membership
    - A set of private communication
      - Message sent with one communicator cannot be received by another
      - Implemented using hidden message tags
    - Why
      - Restricting communication to subgroup is useful
  - A communicator is private virtual communication domain
    - All communication performed w.r.t a communicator
    - Source/destination ranks w.r.t communicator
    - Message sent on one cannot be received on another

Notes on C and Fortran

- MPI is language independent, and has "language bindings" for C and Fortran and many other languages
- In C:
  - Mpi.h must be #included
  - MPI functions return error codes or MPI_SUCESS
- In Fortran:
  - Mpif.h must be included, or use MPI module (MPI-2)
  - All MPI calls are to subroutines, with a place for the return code in the last argument.
- C++ bindings, and Fortran-90 issues, are part of MPI-2

MPI-2

- Dynamic process management
  - Spawn new processes
  - Client/server
  - Peer-to-peer
- One-sided communication
  - Remote Get/Put/Accumulate
  - Locking and synchronization mechanisms
- I/O
  - Allows MPI processes to write cooperatively to a single file
  - Makes extensive use of MPI datatype to express distribution of file data among processes
  - Allow optimizations such as collective buffering
- I/O has been implemented; 1-sided becoming available

Free MPI Implementations

- MPICH from Argonne National Lab. And Mississippi State Univ.
- Runs on
  - Networks of workstations
  - MPP
  - SMPs using shared memory
- Strengths
  - Free, with source
  - Easy to port to new machines and get good performance
  - Easy to configure, build
- Weaknesses
  - Large
  - No virtual machine model for networks of workstations

Free MPI Implementations(‘cont)

- LAM(Local Area Multicomputer)
- Developed at the Ohio Supercomputer Center
- Runs on
  - SGI, IBM, DEC, HP, SUN, LINUX
- Strengths
  - Free, with source
  - Virtual machine model for networks of workstations
  - Lots of debugging tools and features
  - Has early implementation of MPI-2 dynamic process management
- Weaknesses
  - Does not run on MPPs