Beowulf Clustering & MPI Programming

Laurence Liew

Email: lliew@scs.com.sg

SCSES−LCC

---

Agenda

- Lesson 1: Introduction to Parallel Computing
- Lesson 2: How to Build a Beowulf Cluster
- Tea Break
- Lesson 3: Introduction to MPI
- Lunch
- Lesson 4: More MPI Programming
- Tea Break
- Lesson 5: Industry Beowulf

SCSES−LCC
Introduction to Parallel Computing

Compute–Intensive Applications

- Simulation and Modeling problems:
  - Based on successive approximations. More calculations, more precise the result
  - Eg: Fluid Dynamics
- Problems that dependent on computations and manipulations of large amounts of data
- Examples:
  - Genomics, Data–Mining
Will it rain today?

- Singapore Weather:
  - Assume 100 sq km area, 10m or 0.01 km accuracy
  - Grid pts=10,000km^2(area) x 20km(height) x 10000cubes per km^3 = 2x10^9
  - Assume 1000 calculations per pt per hour => 2x10^12
  - We want hourly predictions for 48 hours ~ 2x10^14
  - PIII 1GHz ~ 10^9 calculations per sec
    => 2x10^14/10^9 = 2x10^5 = 2.3 days!

- So how?
  - Use 128 PIII 1Ghz ~ 10^11 calculations per sec
  - => 2x10^14/10^11 = 2x10^3 = 33 mins!

Grand Challenge Problem

- **Drug design** To develop drug that cures cancer or AIDS by blocking the action of HIV protease
- **High-speed civil transport** To develop supersonic jet through computational fluid dynamics running on super computer
- **Fuel combustion** Designing better engine model via chemical kinetics calculations to reveal fluid mechanical effects.
- **Ocean modeling** Large scale simulation of ocean activities and heat exchange with atmospherically flow
- **Ozone depletion** To study chemical and dynamical mechanisms controlling the ozone depletion process.
- **Air pollution** Simulate air quality model
- **Protein structure design** 3-D structural study of protein formation using MPP
Limitation of Uni−processor Computer

• Single processors speed is limited by
  – Speed of Light
  – Physical properties of Semiconductor
    • Interference
  – Scalability
    • Memory expansion
    • I/O bandwidth

Parallel Processing

• Solving large problem by breaking it into a number of small problems, then solve them on multiple processors at the same time
• Real life example
  – building a house using many workers working on separate part
  – Assembly line in factory
Parallel Computer

Parallel computer is a special computer that is designed for parallel processing consists of
- Multiple processors
- High-speed Interconnection network that link these processors together
- Hardware and software that help coordinate computing tasks

Parallel Programming

- Parallel programming involves:
  - Decomposing an algorithm or data into parts
  - Distributing the parts as tasks for multiple processors
  - Coordination work and communications of those processor
- Parallel programming considerations:
  - Type of parallel architecture being used
  - Type of processor communications used
Classification of Computer

- Most widely used one is Flynn's Classification.
  - SISD (Single Instruction Stream, Single Data Stream)
  - SIMD (Single Instruction Stream, Multiple Data Stream)
  - MISD (Multiple Instruction Stream, Multiple Data Stream)
  - MIMD (Multiple Instruction Stream, Multiple Data Stream)

SISD

- Computer that consists of a single instruction stream and single data stream transfer from memory to processor
- Normal uni–processor system (Like Intel Pentium on PC)
SIMD

- Multiple processing element
- Single control unit, execute same instruction on large data set
- Exploit data parallelism in code
- Thinking Machines: CM1, CM–2

MISD

- Some researcher argue that this type of architecture does not exist!!!
- However, many argue that pipeline system can be classified into this class
- Vector machines? CRAY C90, NEC SX4
**MIMD**

- Multiple processor execute different program on different data
- Most commercial parallel supercomputer use this architecture (IBM SP, SGI Origin, HP CONVEX SPP)

**Massively Parallel Processor (MPP)**

- MPP is actually a computer system with multiple processors, large memory capacity, I/O and high speed interconnection
- There are 2 main types of MPP
  - Shared Memory MPP
  - Distributed Memory MPP
Shared Memory MPP
- Multiple Processors
- Global Shared Memory
- Processors communicate using shared memory

Distributed Memory MPP
- Multiple processors
- Each processor has local memory
- No global memory, use message passing to communicate
- Also called Multicomputer
Comparison of Shared and Distributed Memory Systems

**Shared Memory**

**Advantages**
- Easy to program
- Fast for small to moderate number of processors

**Disadvantages**
- Can not scale to large system

**Distributed Memory**

**Advantages**
- Fast for large system, large problem
- Scale to very large systems (Thousands processors)

**Disadvantages**
- Difficult to program

Distributed Shared Memory

- Hybrid between these 2 systems
- Physical hardware is distributed memory
- Use special hardware to help present the view of shared memory machine to application (CCNUMA – Cache–Coherency Non–uniform Memory Access use cache to help)
- Scalable and easy to program
- Sequent, SGI Origin!
Beowulf Cluster

• A parallel multicomputer built from COTS (Commodity–Off–The–Shelf) PCs and conventional high–speed network.
• Support parallel programming systems.

Cluster Computing

• Advantages
  – Low startup cost
  – Scalable in term of cost, performance
  – Easy to maintain
  – Better at tracking technology changes

• Disadvantages
  – Lower performance than commercial system at high–end tasks
  – Commercial systems offer better operating environment
Parallel Processing

- Speeding up the execution by splitting task into many independent subtask and run them on multiple processors
- Communication performance and task partitioning is a major concern

Parallelism in Application

- Parallelism in application is the potential of algorithm to be partitioned into a set of small tasks that run concurrently
- Important to the performance gained from parallel processing
- There are two types of parallelism
  - Functional Parallelism
  - Data Parallelism
Functional Parallelism

• An algorithm can be separated into multiple independent functions that operate concurrently
• For example
  \[ E = A \times B + C \times D \]
  \[ A \times B \] and \[ C \times D \] can execute simultaneously

Data Parallelism

• The same algorithm is executed on multiple node and process different data
• Example: Vector addition \[ A = B + C \]
• Each line of \[ A[I] = B[I] + C[I] \] execute concurrently using the same program on \( n \) processors
• \( N \) step process becomes a 1 step process on a \( N \) processor system!
Comparison of Data and Functional Parallelism

• Functional parallelism especially pipeline is commonly used but the parallel processing gain is limited
• Data parallelism depends on size and pattern of data but much more potential parallelism inside.

Currently functional parallelism is mostly used in instruction level such as superscalar technique
• Data parallelism is used mostly because of programming simplicity and potential parallelism that increase with the data size
Parallel Programming Tools

• Using Parallelizing Compiler
• Building applications yourself using parallel message passing library such as PVM and MPI
• Using parallel language such as High Performance Fortran and OpenMP
• Using parallel maths library!

Using Parallelizing Compiler

• Easiest to do since all development is done in sequential
• need to learn the trick of compiler directive
• less performance compared to hand tune code
Programming using Message Passing

• Partition task into multiple concurrent tasks that communicate by passing message
• MPI (message passing interface)
  – a de facto standard that is now supported by all platform.
  – Free implementations are also available (MPICH, LAM)

Lesson1: Introduction to Parallel Computing

Programming using Message Passing

• Advantages
  – Standard and portable
  – High performance
• Disadvantage
  – Very difficult to program
Programming using Parallel Language

• There are hundreds of parallel programming language. But very few are standard.
• Widely use parallel language
  – HPF (High Performance Fortran)
    • Issued by HPF Forum, SPMD data model, distributed memory
    • Based on inserting directives into code
  – OpenMP
    • New standard by OpenMP Consortium. Most commercial compiler started to support it
    • use shared memory model

Lesson 1: Introduction to Parallel Computing

Programming using Parallel Language

• Advantage
  – Easy to code
• Disadvantage
  – Lost some performance and scalability

Lesson 1: Introduction to Parallel Computing
HPF Program

```fortran
! Parallel Fortran Program using HPF
!
!HPF$ PROCESSEES p(number_of_processors())
real c(6000,6000)
real A(6000,6000), B(6000,6000)
!HPF$ distribute c(cyclic,*) onto p
!HPF$ align a(:, :) with c(:, :)
!HPF$ align b(:, :) with c(:, :)

a(1:6000,1:6000)=10.0
b(1:6000,1:6000)=5.0

program is done

end
```

Programming using Parallel Math Library

- Hide the complexity by providing easy to use math library
- Popular one are:
  - Scalapack by UTK
  - PetSc by Argonne National Lab
- Advantage
  - Easy to code, no knowledge of parallel systems required
Developing Parallel Apps

- Develop sequential application first
- Identify
  - Most time consuming part using profiling tool
  - Parallelism inherent in that part
- Choose strategy for data partitioning and task partitioning
- Choose the development tools on target parallel machine
- Add code for task control and communication
  - Compile, test, debug, measure performance, improve

Load Balancing

- Parallel computation stop when last task terminate execution
- If the computation time on each processors is not equal, faster processors have to wait for slow one. Performance loss
Causes of Work Load Imbalance

- Heterogeneity
  - Processors have different speed but received the same amount of work
- Synchronization
  - Processors trying to synchronize the operation but certain processors have a problem
    - Interfered by other running process
    - Wait for I/O
    - Network is busy

Task Granularity

- Granularity is a ratio between computation and communication
- Fine-grain parallelism
  - Short execution time between cycles of communication
  - Better potential for load balancing
  - High overhead, need special hardware support
- Coarse grain parallelism
  - Long computation between communication cycle
  - Lower overhead
  - Less parallelism, less potential for load balancing
  - Good for machine with low speed interconnection network
Data Dependency

- Data dependency exists when several computations shared the same data
- Frequently inhibit parallelism
- Example
  - $A = B + C, \ D = E + F$ ; no data dependency
  - $A = B + C, \ D = E + A$ ; first statement must finish before second one

Parallel Programming

- There are many alternative for parallel programming but the desire property are always
  - High Performance
  - Portable and standard
  - Easy to code
- In the past, most parallel computer comes with compiler or programming library that is proprietary
- Currently, MPI is the standard portable programming for MPP machines
How to build a Beowulf Cluster

What is Cluster Computing Technology?

- Cluster computing is a technology of linking multiple computers together with high speed network to build a high performance computer systems
Goal

- High-performance clustering
  - Link many computers together to team up and finish problem faster by having multiple computers working on the same problem independently

Beowulf Clustering

- Clustering technology originated from Beowulf Project at NASA by Thomas Sterling and Donald Becker, 1994
- Characteristics of Beowulf Cluster
  - Use PC based hardware and commodity components to gain high performance at low cost
  - Based on Linux OS, and open source software
  - Internal and dedicated high speed network
  - Support parallel programming (via. PVM, MPI)
Cluster System Structure

Cluster Software Structure

Lesson 2: How to build a Beowulf Cluster
Benefit of Clustering

- Scalability and Performance
- Rapid response, technology tracking
- More user–driven configuration
- Order of magnitude price–performance advantage
Is it Real?!

- Government/Academic
  - The Alliance
  - US DOE (ASCI Red/White, Fermi–Lab)
  - US Airforce
  - US Dept of Commerce (Weather Forecast)
  - IHPC, DSO, NUS
- Commercial
  - Shell (1024 nodes!)
  - Amerada Hess (saved US$2mil/year!)

Cluster System Structure

- There are hundreds of variation on cluster system structure since users are free to build their own system
- However, there are two most common structures
  - Scientific Computing Cluster
    - Linux supercomputer for high performance computing application
  - Information Server Cluster
    - Scalable web server, database server, mail, print server
    - Most cluster of this type are based on LVS technology
Lesson 2: How to build a Beowulf Cluster

Scientific Computing Cluster

- Master
- External Network
- Internal Network
- Slave (5 nodes)

Information Service Cluster

- Master
- Backup
- External Network
- Internal Network
- Slave (5 nodes)

SCSES-LCC
Let’s Build a Beowulf!

- Buy all equipment (PC, network switches, racks, etc.)
- Select cluster structure and connect all hardware together.
- Configure Master node
- Configure Slave node
- Install Software

System Configuration

- Configure Master node first
  - Install Linux from the distribution such as Redhat, Suse, Debian, Turbo Linux on the master node
  - Configure
    - RSH enable, /etc/hosts.equiv, /root/.rhosts, /etc/pam.d/rlogin, /etc/pam.d/rsh, /etc/pam.d/rexec
    - /etc/xinetd.d/rsh, /etc/xinetd.d/rlogin, /etc/xinetd.d/rexec
    - NFS Server (/home, /usr/local)
    - Hostname resolution /etc/hosts or /etc/reslove.conf
    - DHCP server use for remote boot (optional)
    - Install software (MPI, PVM, Scalapack etc)
Server node configuration

- **Disk full Server**
  - Each slave node must have local disk for storing OS and user data
  - Node boot from OS installed locally
- **Diskless Server**
  - Each slave node may or may not have local disk
  - Node boot and load OS remotely from network to ram disk
  - Local disk is used for caching and local data store

Comparison

**Diskfull**
- User can still work on slave node even when master goes down
- Reduce network traffic
- Scale to very large configuration
- Software, library, tools, and data may be out of sync

**Diskless**
- Easier to build and manage the system since the configuration is centralized
- Software installed are more synchronize
- Node is transparent so it can be shutdown and replace any time
- More network traffic due to NFS root file system
How the diskless system work

- On master node
  - Create slave node configuration
  - Create common NFS mount area
  - Create boot ROM or boot floppy for slave node
- There are many tools that can help create this
  - LUI by IBM
  - Diskless Cluster Tools, Beowulf Builder
    [http://smile.cpe.ku.ac.th/software](http://smile.cpe.ku.ac.th/software)
  - SCYLD Distribution do this automatically

System Imager

- VA System Imager
  [http://systemimager.org](http://systemimager.org)
  - Help install multiple machine across network
  - Image is created on one client called “Golden client”
  - Then image will be download to “Image server”
  - Other can be install by replicating these image from image server
LUI

- LUI (Linux Utilities for Cluster Install)

- Resource base installation
  - Tool to create cluster resource description on Master node first
  - Help install it on number of slave node
  - Using RPM based distribution installation

Beowulf Builder

- Beowulf Builder Tools
  - Create diskless slave configuration and filesystem by extracting master node configuration
  - Tool that manages the collection of resources and configuration of slave nodes
  - Web base interface for remote operation

[http://amata.cpe.ku.ac.th/~b40eksr/project](http://amata.cpe.ku.ac.th/~b40eksr/project)
Diskless Boot Process using Ethernet Boot Rom

- Slave node power on
- Slave send bootp request to obtain IP
- Slave node use tftp to download kernel
- Kernel boot itself and use NFS to mount root file system
- System started the usual way
- Note that kernel can be stored on Master Node

Diskless Boot Process from Media (CD, Floppy)

- Slave node power on
- Boot from CD or floppy disk. Kernel is loaded
- Slave obtains IP using DHCP
- Kernel mount NFS root from master or use CD rom File system
- System boot in the usual way
PXE: **Preboot Execution Environment**

- Industry standard client/server interface that allows networked computer to be configure and boot remotely by an administrator
- Steps:
  - Client notify server that it uses PXE
  - Server send list of boot server which contains OS image
  - Client contact server and obtain the name of file to be download
  - Client down load file using TFTP protocol
  - That file down load the rest of OS and start the execution

Some notes about Diskless System

- Diskless slave uses broadcast of DHCP, BOOTP. If too many system are started simultaneously, the boot process may fail due to packet loss.
- Common practice is to start only a set of nodes at one time due to power requirement
- Diskless boot protocol assume that master and slave are on the same sub-network
Installing Cluster Software: Distributed

- Distributed Software Installation
  - Install software on every node
  - Possible now because of lower disk storage cost and large capacity
  - May have the problem with unsynchronized version of software

Installing Cluster Software: Global

- Global Software Installation
  - Use NFS mount volume to shared software installation
  - Single point software install, easy to manage
  - More network overhead, slower
Single System Image

- Cluster will be much easier to manage if we can control the whole collection of nodes as a single machine!
  - Single account and protection
  - Single File System Structure
  - Single Software Installation Point
  - Single Process Space

Single User Account

- Using NIS (Network Information Services)
  - One server, usually master, act as a NIS Server
  - Easy to setup and good for small cluster
  - Can not scale and very slow for large cluster

- Using File Replication
  - Account control done on Master node only
  - Replicate important control file to every node using tool such as rsync
  - More efficient and scale much better
Single File System Structure

- **Shared NFS**
  - Use single server to serve common directory such as /home
  - Perfect for small cluster but will not scale for very large cluster

- **Replication**
  - Use replication utility to push file down to every node once in a while
  - Scale much better but need some effort by users

- **Global File system**
  - GFS, PVFS can provides single image over multiple system
  - Proven technology?

Cluster Security

- Always use secure shell to connect to cluster from any location outside the system
- Put cluster behind some kind of firewall or use master node as a firewall
- Always alert about the attack
- Within cluster: RSH or SSH. Note SSH overheads!
Linux Cluster Distribution

- Recent effort to streamline the installation of Linux cluster and software
- There are many distributions
  - OSCAR
  - SCYLD
  - SCE

OSCAR
(Open Source Cluster Application Resources)

- Work of Open cluster group
  http://www.openclustergroup.org
- Effort to integrate a set of open source tools that are "tweaked" to work together out of the box
- Still at the very early stage
- Focus on the building of Scientific Computing Cluster
- Oscar information http://www.csm.ornl.gov/oscar/
What is in OSCAR?

- LUI installation tools from IBM
- C3 Management Tool from Oak Ridge National Lab
- OpenPBS
- OpenSSH
- MPI and PVM

SCYLD Beowulf Operating System

- Integrated Distribution, automated installation from CD
- Bproc kernel patch for unify process space
- Large file system support
- Special version of MPICH that support Bproc
- Library and tools
- Commercially supported by SCYLD Computing Corp. Very well developed
Cluster Management Tools

- There are many open source software tools available.
  - C3 from Oak ridge National Laboratory
  - SCMS from Kasetsart University
  - Cluster Probe from Hong Kong University
  - VACM from VA Linux
  - Scripts: prsh, bpsh

Common feature for Cluster Management Tool

- Parallel Unix command
- Management script
- Some GUI interface to management tools
- Real time Monitoring
- Web interface (scms, cluster probe)
Remote management

- Most of the remote management is still done through X and ssh session
- Most of the web base to is
  - Manage only a single machine (webmin)
  - Monitoring the system (SCMS/KCAP, Cluster Probe)
- Problem is how to maintain the security

SCMS: Cluster Management Tool for Beowulf Cluster

- A collection of system management tools for Beowulf cluster
- Package includes
  - Portable real-time monitoring
  - Parallel Unix command
  - Large collection of graphical user interface tools for users and system administrator
- Download
  http://smile.cpe.ku.ac.th/software
Selector works up to 10000 nodes

Cluster configuration browser

Measure Communication Delay

Real-time Monitoring

Lesson 2: How to build a Beowulf Cluster

Commerical support?

- Scyld Cluster OS
- Portland Group's: HPF, OpenMP
- Scientific/Academic:
  - BLAST, FASTA, NAMD
- Commercial HPC Software
  - FLUENT, MSC, NumericalObjects' DiffPack
  - AMI, Hypermesh
- IBM, Compaq, SGI
Conclusion and Trends

- Building and Managing Cluster is now easier with many tools
- Cluster distributions started to appear.
  - Easier to install and manage cluster system
  - Integrate all tools and some application reduce the time to install application and tools

Introduction to MPI
The Message−Passing Model

• Parallel program consists of a set of processes. These processes are mapped to one or more processors
• Large computation are divided among these process. The computation are performed concurrently
• Process communicate with each others by passing network message
  – Exchange data
  – Synchronization

What is MPI?

• A message−passing library specification by MPI Forum
  – MPI 1.1 Standard developed from 1992–1994
• MPI is implementation and machine independent
  – Works for parallel computers, clusters, and heterogeneous networks
• Many implementation exists
  – MPICH from Argonne National Laboratory
  – LAM from Notre−Dam University
  – MPI−PRO from MPITech
Why Use MPI?

- MPI is portable
- MPI is powerful
- MPI is standard
- MPI is supported
- MPI is being improved with every version!

Accessing the Cluster

- Telnet session: telnet X.X.X.X
- Login: beowulferX
- Password: beowulf
- Home directory: /home/beowulferX
A Minimal MPI Program (C)

/* hello.c */

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

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}

Running MPI program (with MPICH)

• % mpicc –o hello hello.c
• % mpirun –np 4 hello
Header Files and Function

- All MPI constants are defined in a Header File
  C: #include <mpi.h>
- Mpicc compiling script helps locate all these declaration
- Format of MPI function
  Error = MPI_Fxxxxxx(parameter, ...);

MPI is Easy or Hard?

- MPI is complex (129 Functions) but for a beginner, only 6 functions are enough to implement simple algorithms.
- **MPI_INIT**(int *argc, char **argv);
  - Init MPI environments
- **MPI_FINALIZE**();
  - Exit MPI environments
- **MPI_COMM_SIZE**(comm,size);
  - How many processes participates?
- **MPI_COMM_RANK**(comm,pid);
- **MPI_SEND**(buf,count,datatype,dest,tag,comm);
- **MPI_RECV**(buf,count,datatype,source,tag,comm,status);
MPI Basic Concepts

• Rank
  – Every process has a unique rank or id assigned by the system at start up. Rank begin at 0.
  – Rank is used as source and destination of messages.

• Group and Communicator
  – MPI process started in group. Each process in a group is associated with a unique rank start from 0 to N-1.
  – A group is always associated with a communicator which is used to perform group operations.
  – Initially, all process are members of predefined communicator MPI_COMM_WORLD.
Initialization

• MPI program must initialize MPI by calling

```
int MPI_Init(int argc, char **argv)
```

Finishing the Task

• All MPI program must end with function

```
MPI_Finalize();
```

**Example:**

```
int main(int argc, char **argv)
{
    MPI_Init(&argc,&argv);
    MPI_Finalize();
}
```
How many processes?

- How many processes are participating in this computation?

  MPI_Comm_size( MPI_COMM_WORLD, &mysize);

  Example:
  int mysize;

  MPI_Comm_size( MPI_COMM_WORLD, &mysize);

Get Rank of Current Process

- To get rank of current process, call

  MPI_Comm_rank(MPI_Comm comm, int *rank);

  Example:
  int rank;

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
Timing MPI Program

MPI_Wtime();

• Return time in second from an arbitrary point in the past

• Usage

```c
double t1, t2, elapsetime;
t1 = MPI_Wtime();
t2 = MPI_Wtime();
elapsetime = t2 - t1;
```

Better Hello (C)

```c
/* hello2.c */
#include "mpi.h"
#include <stdio.h>

int main(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("I am rank %d of size %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```
MPI Send

**MPI_SEND** (start, count, datatype, dest, tag, comm)

- The message buffer is described by (start, count, datatype).
- The target process is specified by dest, which is the rank of the target process in the communicator specified by comm.
- When this function returns, the data has been delivered to the system and the buffer can be reused. The message may not have been received by the target process.

MPI Receive

**MPI_RECV** (start, count, datatype, source, tag, comm, status)

- Waits until a matching (on source and tag) message is received from the system, and the buffer can be used.
- source is rank in communicator specified by comm, or MPI_ANY_SOURCE.
- status contains further information
- Receiving fewer than count occurrences of datatype is OK, but receiving more is an error.
MPI Datatypes

- The data in a message to sent or received is described by a triple (address, count, datatype)
- Data type help MPI calculate exact message size to be send on network
- MPI_DATATYPE
  - Basic Types
  - Derived Types

<table>
<thead>
<tr>
<th>MPI_Datatype</th>
<th>C data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_CHAR</td>
<td>char</td>
</tr>
<tr>
<td>MPI_SHORT</td>
<td>short</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>int</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>long</td>
</tr>
<tr>
<td>MPI_UNSIGNED_CHAR</td>
<td>unsigned char</td>
</tr>
<tr>
<td>MPI_UNSIGNED_SHORT</td>
<td>unsigned short</td>
</tr>
<tr>
<td>MPI_UNSIGNED</td>
<td>unsigned int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_LONG</td>
<td>unsigned long</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>MPI_BYTE</td>
<td></td>
</tr>
<tr>
<td>MPI_PACKED</td>
<td></td>
</tr>
</tbody>
</table>
MPI Tags

- Messages are sent with an accompanying user-defined integer tag, to assist the receiving process in identifying the message.
- Messages can be screened at the receiving end by specifying a specific tag, or not screened by specifying MPI_ANY_TAG as the tag in a receive.

Lesson 3: Introduction to MPI

Final Hello! –1

/* hello3.c */
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main(int argc, char* argv[]){
    int my_rank; /* rank of process*/
    int p; /* number of processes */
    int source; /* rank of sender */
    int dest; /* rank of receiver */
    int tag = 0;
    char message[100];
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Status status;
}
Final Hello! – 2

/* hello3.c */

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

/* find out process rank */
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

/* find out number of processes */
MPI_Comm_size(MPI_COMM_WORLD, &p);

/* get processor name */
MPI_Get_processor_name(processor_name,&namelen);

if (my_rank != 0) {
    sprintf(message, "Greetings from process %d[proc %s]!",
            my_rank, processor_name);
    dest = 0;
    MPI_Send(message, strlen(message)+1, MPI_CHAR, dest, tag,
              MPI_COMM_WORLD);   }
else { /* my_rank == 0 */
    printf("Greetings from master process %d[proc %s]\n", my_rank,
            processor_name);
    for (source = 1; source < p; source++) {
        MPI_Recv(message, 100, MPI_CHAR, MPI_ANY_SOURCE, tag,
                  MPI_COMM_WORLD, &status);
        printf("%s\n", message);
    }
    MPI_Finalize();
} /* main */

Final Hello! – 3
mpirun −np 4 hello3

If (my_rank == 0)
... for each P,
call MPI_Recv(..)

If (my_rank != 0)
... MPI_Send(..)

[laurence@law src] mpirun −np 4 hello3
Greetings from master process 0[proc law]
Greetings from process1[proc law]!
Greetings from process2[proc law]!
Greetings from process3[proc law]!

What’s going on?

• one process on one processor (cpu)
• mpirun directive => a copy of executable program on each processor
• each processor runs its copy of the executable
• Different processes can execute different statements by branching (if conditions!) with the program based on process ranks (id)
Two style of Parallel Programming

- **SPMD (Single Program Multiple Data)**
  - Multiple instance of the same code compute on different data
  - Synchronizing from time to time
- **Master/Slave**
  - Two kind of processes
  - Master: control the execution and task allocation
  - Slave: perform the execution
Lesson 3: Introduction to MPI

More MPI Programming

Lesson 4: More MPI Programming
Trapezoidal Rule

\[ \int_{a}^{b} f(x) \, dx \]

\[ \text{area of trap} = \frac{1}{2} h [f(x_{i-1}) + f(x_i)] \]

\[ \text{total} = \frac{f(x_0)}{2} + \frac{f(x_n)}{2} + f(x_1) + f(x_2) + \ldots + f(x_{n-1}) \cdot h \]

See trap_s.c

MPI communication

- **Point-to-Point Communication**
  - Mechanism to send/receive message between 2 processes
- **Collective Communication**
  - Well structure communication or operations among group members. eg. Multicast, Gather/Scatter, Reduced
- **Asynchronous/Synchronous communication**
  - For asynchronous communication, communication and be performed at the same time as computations.
  - MPI support asynchronous operations
MPI Point to Point Communication

- Communication between two processes
- Source process send message to destination process
- Rank is used to identify source and destination
- MPI_Send
- MPI_Recv

Parallel Trapezoidal Rule

See trap_p.c
Collective Communications

- Parallel algorithms often call for coordinate communication operations involving multiple processes. eg. Finding a summation or maximum, minimum number of distributed data.

- MPI support the following operations
  - Barrier synchronization: Synchronizes all processes.
  - Broadcast: Send data from one process to all processes
  - Gather: Gather data from all processes to one process
  - Scatter: Scatters data from one process to all processes.
  - Reduction: sum, multiply etc. distributed data
Collective Communication

- Broadcast
  - `MPI_Bcast(inbuf, incnt, intype, root, comm)`
  - Broadcast data in buffer `inbuf` from one process (root) to all processes

```c
MPI_Bcast(buff, 100, MPI_Float, 0, MPI_COMM_WORLD);
```
Collective Communications

- **Gather**
  - `MPI_Gather(inbuf, incnt, inctype, outbuf, outcnt, outtype, root, comm)`
  - Collect data from every processes to one process (root)

- **Scatter**
  - `MPI_Scatter(inbuf, incnt, inctype, outbuf, outcnt, outtype, root, comm)`
  - Reverse of gather, quickly spread the data to every process involved
Collective Communications

- Reduction operation
- `MPI_reduce(inbuf, outbuf, count, type, operation, root, comm)`
  - Apply the operation to each element in inbuf of each process in group and return result to one process
- `MPI_Allreduce(inbuf, outbuf, count, type, operation, root, comm)`
  - Same as `MPI_Reduce` but return result to all processes
  - Support `MPI_MAX`, `MPI_MIN`, `MPI_SUM`, `MPI_PROD`, `MPI_LAND`, `MPI_LOR`, `MPI_LXOR`, `MPI_BAND`, `MPI_BOR`, `MPI_BXOR`

Lesson 4: More MPI Programming

```c
MPI_Reduce( inbuf, outbuf, 2, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
```

Return minimum value of two set of distributed data.

See `trap_col.c`
When to use MPI

- Portability and Performance
- Irregular Data Structures
- Building Tools for Others
  - Libraries
- Need to Manage memory on a per processor basis

When not to use MPI

- Regular computation matches HPF
- Solution (e.g., library) already exists
- Require Fault Tolerance
- Distributed Computing
  - CORBA, DCOM, etc.
MPI–2: Extending the Message Passing Interface

- Dynamic Process Management
  - Dynamic process startup
  - Dynamic establishment of connections

- One–sided communication
  - Put/get
  - Other operations

- Parallel I/O

- Other MPI–2 features
  - Generalized requests
  - Bindings for C++/ Fortran–90; interlanguage issues

---

Industry Beowulf
ScaLAPACK and BLAS

- Stand on the shoulders of greatness → use libraries!
- ScaLAPACK → Scalable LAPACK (Linear Algebra PACKage)
- Subroutines for
  - Linear system solutions
  - Eigenvalue problems
- Itself also based on another library → BLAS (Basic Linear Algebra Subprogram)
- BLAS → vector/matrix operations

See linsolve.c
More Libraries

- PetSc (Portable Extensible Toolkit for Scientific Computation)
  - Solves linear/non-linear equations
  - Basic graphics
- DiffPack by Numerical Objects (commercial)
  - object-oriented problem-solving environment for the numerical solution of partial differential equations (PDEs)

Industry Tools Supporting Linux Beowulf Clusters

- MSC (CAE type tools)
  - Nastran, Visual Nastran
- Fluent (CFD)
- Platform Computing
  - Load Sharing Facility (LSF)
- MatLab (requires parallel extensions)
- Etc etc...
Trends..

- Cray -> SuperCluster (Alpha CS20 cluster with Myrinent)
- IBM
  - SP technology being ported to Linux clusters
  - You can now buy pre-configured cluster over the web!
- Major HPTC software vendors are in!
- Local R&D facilities are building large clusters!

More Information
• The TFCC is an international forum promoting cluster computing research and education. It participates in helping to set up and promote technical standards in this area. The Task Force is concerned with issues related to the design, analysis, development and implementation of cluster–based systems. Of particular interest are: cluster hardware technologies, distributed environments, application tools and utilities, as well as the development and optimisation of cluster–based applications.

• http://www.computer.org/tab/tclist/tfcc.html

General Information

• BEOWULF
  – http://www.beowulf.org/

• EEE TFCC
  – http://www.computer.org/tab/tclist/tfcc.htm

• High Performance Computing Page
Companion Material

- Online examples available at http://www.mcs.anl.gov/mpi/tutorials/per
- mpiexmpl.tar.gz contains source code and run scripts that allows you to evaluate your own MPI implementation

Extra Notes
Little Fact About Performance

- Performance depends on several factors
  - Parallelism inherent in algorithms/application
  - I/O Requirement
  - Communication requirement of parallel application
  - Matching between parallel application and parallel computer architecture

Performance Metric for Parallel System

- Runtime
  - Sequential runtime ($T_s$) is an elapse time between the beginning and the end of execution of a program on sequential computer
  - Parallel runtime ($T_p$) is the elapse time from the moment that first processors start the parallel execution until the last processor finish its execution
Performance Metric for Parallel System

• SPEED UP
  – Ratio between sequential runtime and parallel runtime
  – Measure the gain obtained from parallelizing the code
  – For system with P processors, perfect speedup is $S = P$

Performance Metric for Parallel System

• Efficiency
  – A measure of the fraction of time for which a processor is used
  – For parallel system with P processors
  – Perfect efficiency is when $S = P$, $E = 1$
Limitation of Parallel Processing

- Sequential Portion of code
- Communication overhead
- Scalability in term of memory and number of processors

Amdahl’s law

- For Fixed−size problem, performance of parallel program is limited by sequential part left in code
  - $s =$ fraction of parallel code (e.g.. 0.01 or 1 %)
  - $P =$ number of processors
  - $Speedup = \frac{1}{s + \frac{(1 - s)}{P}}$

  - Ignore communication overhead, assume perfect case
Amdahl’s Law

- Maximum speedup are limited to $1/s$
- If program has sequential code = 1% ($s=0.01$)
  Maximum speedup is only 100
- Care must be taken to eliminate sequential part

![Graph showing speedup vs number of processors]

How to Overcome Amdahl’s Law Limitation

- Some problem such as finite element, if more computing power is available, we need more precision not faster speed.
- This kind of problem is Fixed-time problem
- In this case, Amdahl’s law can be overcome by increasing problem size
- This fact has been discovered by Gustafson and Barsis and leads to so called Gustafson–Barsis Law
Gustafson–Barsis Law

• Assume that problem is fixed–time
• if \( s = \) sequential
  Fraction of code, \( P = \) number of processors

\[
S = \frac{s T_p + (1 - s) P T_p}{T_p} \\
S = s + (1 - s) P
\]

Gustafson–Barsis Law

• In this case, speedup always increase as the number of processors increases
• The only condition is that the computing workload must be increased as well
Communication Overhead

- A major factor in parallel performance

\[ W = \text{Computational work} \]

\[ T_{o}(W,P) = \text{Communication overhead function} \]

\[ S = \frac{Ts}{Tp} \Rightarrow S = \frac{Ts}{\frac{Ts}{P} + T_{o}(W,P)} = \frac{1}{\frac{1}{P} + \frac{T_{o}(W,P)}{Ts}} \]
Discussion

- Optimum number of processors and speed depends on
- Algorithm dependent parameters
  - Problem Size (W)
  - Communication Function m(W,p)
- System Parameter
  - CPU Speed (Texe)
  - Interconnection Network Speed (Tc)
The End.