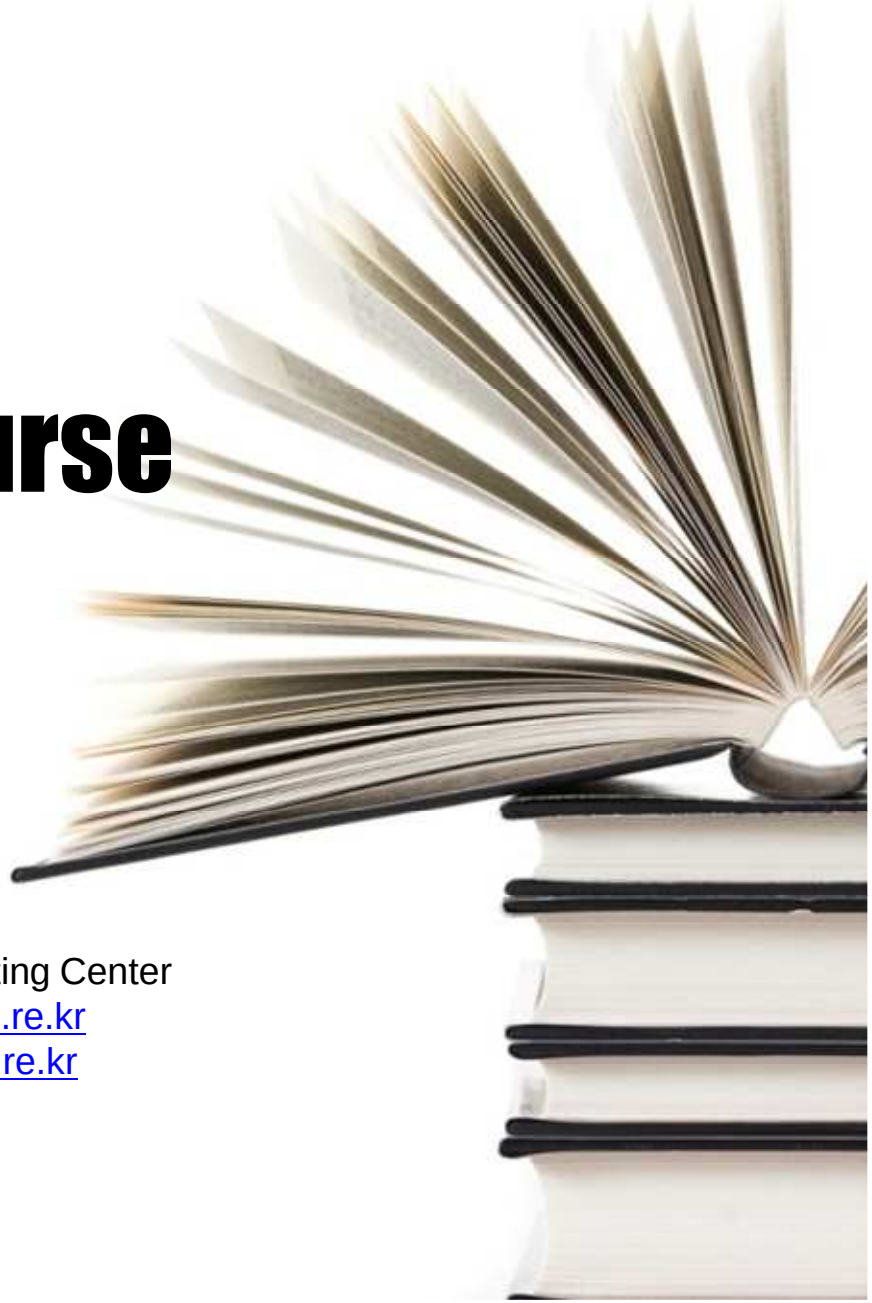
MPI Training Course

Part 1 Basic



2014. Aug

KISTI Supercomputing Center
http://www.ksc.re.kr
http://edu.ksc.re.kr

Objectives

- After successfully learning the tutorial in this module, you will be able to

  - ✓ Understand parallel program concept
  - ✓ Compile and run MPI programs using the MPI implementation
  - ✓ Write MPI codes using the core library

Syllabus (Basic Course)

09:30 - 10:30	▪ Motivation/Introduction to Education/ A first MPI program	
10:30 - 10:40	▪ **Break**	
10:40 - 12:00	▪ MPI Basic I – Six-function MPI	
12:00 - 13:30	▪ **Lunch**	
13:30 - 14:50	▪ MPI Basics II – P2P:Blocking Communication	
14:50 - 15:00	▪ **Break**	
15:00 - 16:30	▪ MPI Basics III – P2P:Non-Blocking Comm. & Hands on	
16:30 - 17:00	▪ **Summary**	

Reference & Useful Site

1. MPI Tutorial

<MPI 온라인 강좌>

- http://www.citutor.org/login.php

<MPI Tutorial>

- https://computing.llnl.gov/tutorials/mpi/
- http://mpitutorial.com/

<Domain Decomposition 강좌>

- http://www.nccs.nasa.gov/tutorials/mpi_tutorial2/mpi_II_tutorial.html

<MPI 한글 레퍼런스>

- http://incredible.egloos.com/3755171

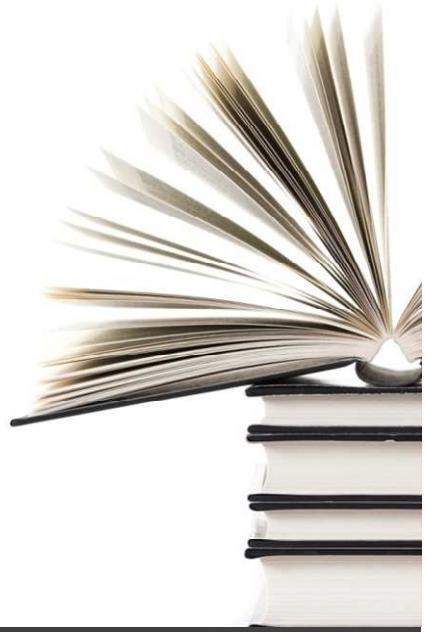
<SP Parallel Programming Workgroup>

- ~~http://www.itservices.hku.hk/sp2/workshop/html/samples/exercises.html~~

<PDC Center for HPC>

- http://www.pdc.kth.se/education/tutorials/summer-school/mpi-exercises/mpi-labs/mpi-lab-1-program-structure-and-point-to-point-communication-in-mpi/mpi-lab-1-program-structure-and-point-to-point-communication-in-mpi

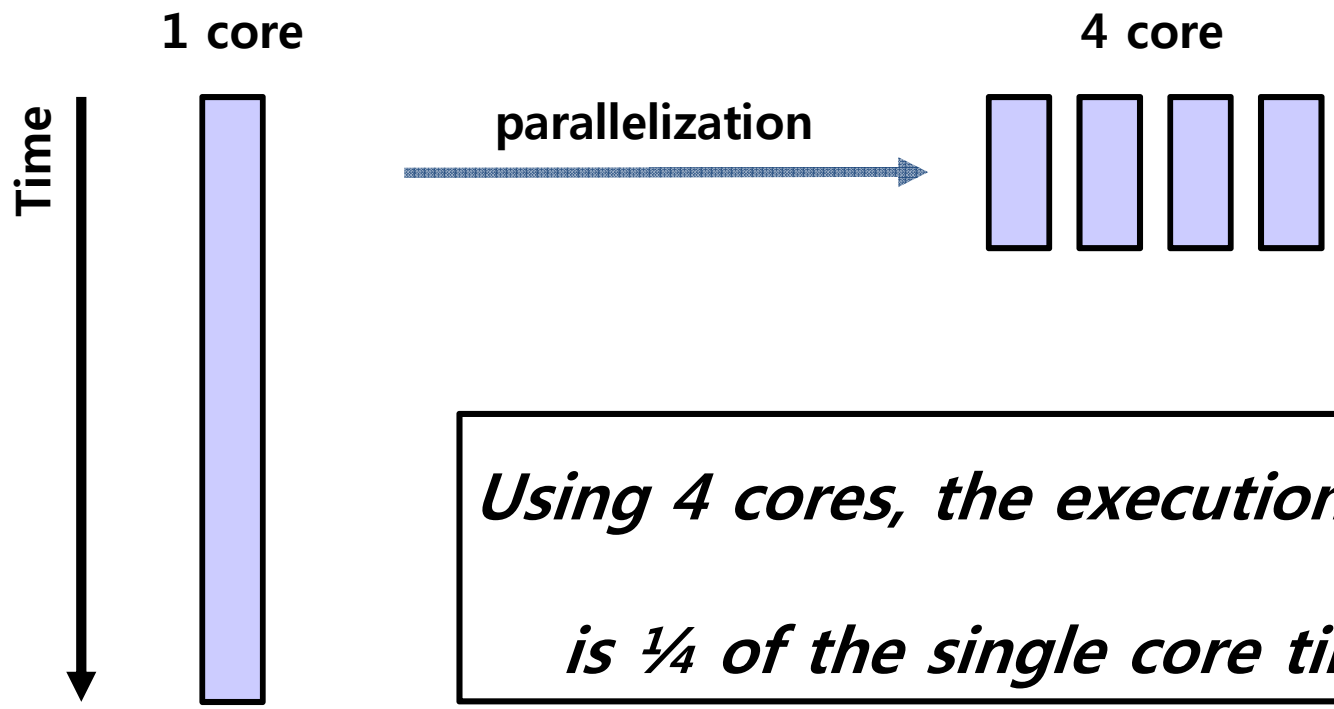
MPI Introduction and Concept



Why Parallelization?

Parallelization is another optimization technique
The goal is to reduce the execution time

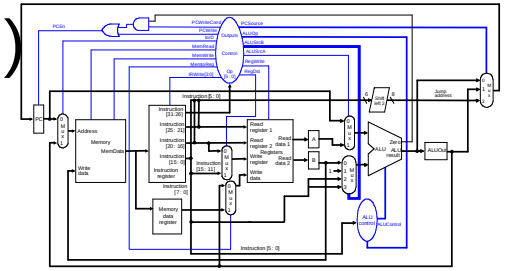
To this end, multiple processors, or cores, are used

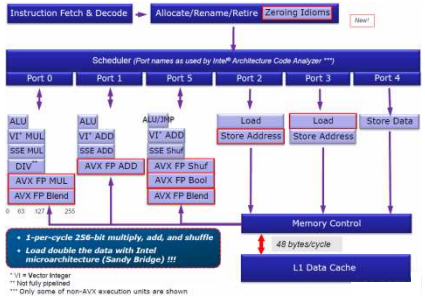


Using 4 cores, the execution time is ¼ of the single core time

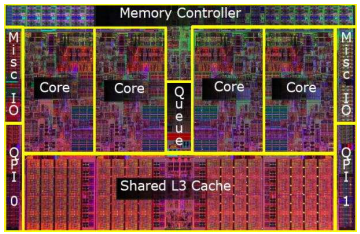
Parallelism

- ILP (Instruction Level)
  - Automatic
  - No User Control





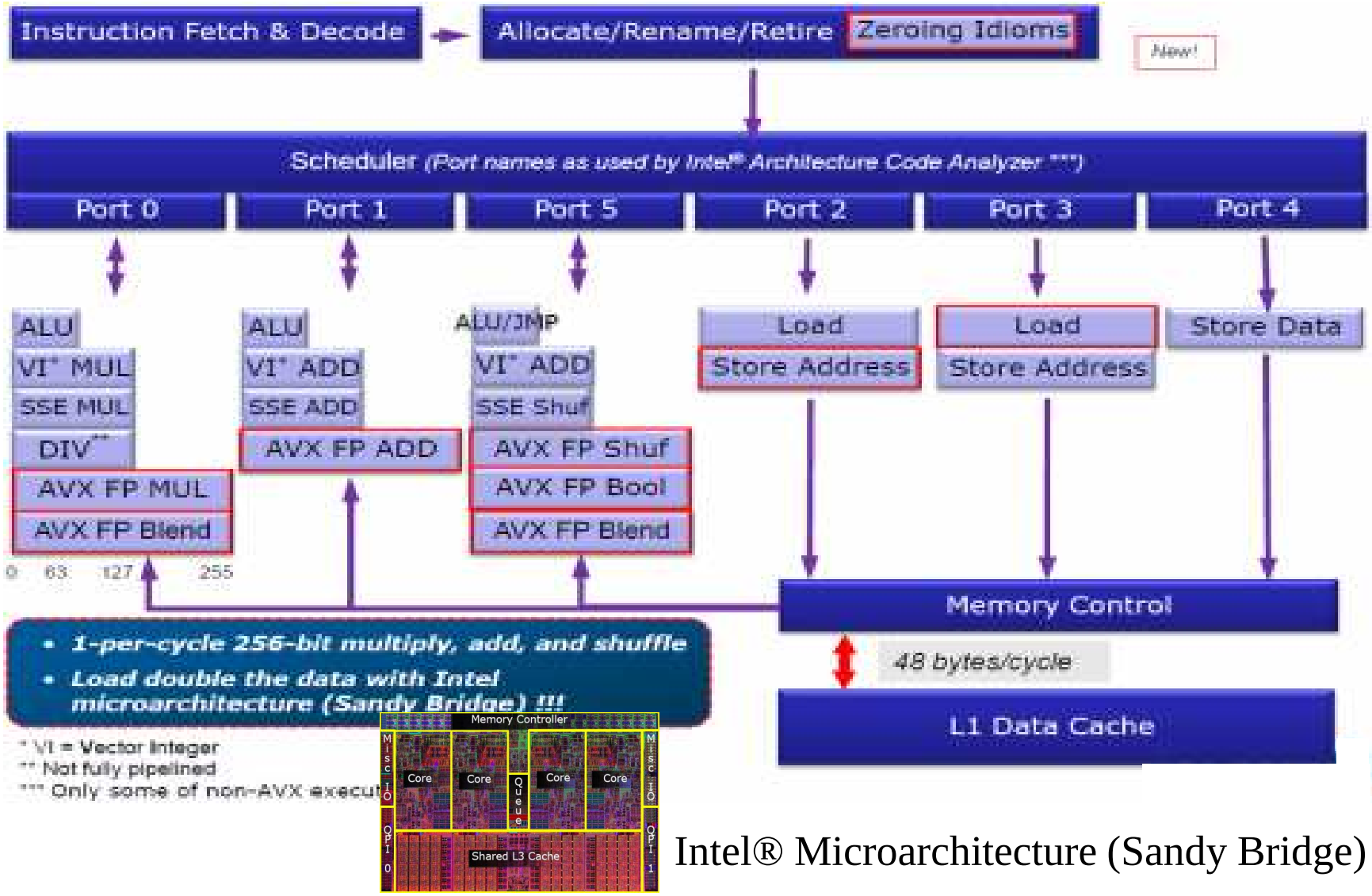
- Vectorization
  - Automatic
  - Compiler Option



- OpenMP
  - Manual
  - Compiler Directives

- MPI
  - Manual
  - Special Libraries

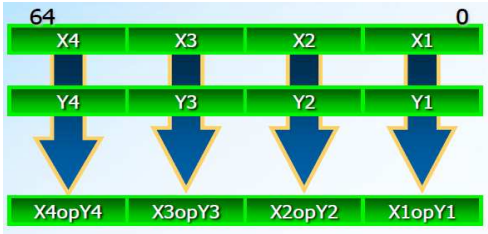
ILP (Instruction Level Parallelism)



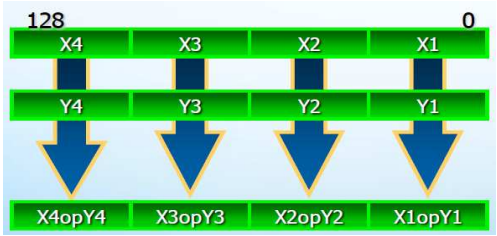
Intel® Microarchitecture (Sandy Bridge)

Evolution of Intel Vector Instructions

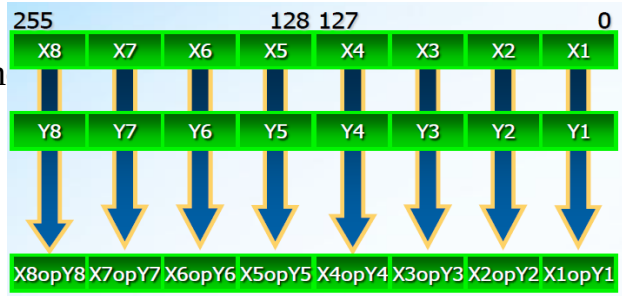
- MMX (1996, Pentium)
  - CPU-based MPEG decoding, Integers only, 64-bit divided into 2x32 to 8x8
  - Phased out with SSE4



- SSE (1999, Pentium III)
  - CPU-based 3D graphics, 4-way float operations, single precision
  - 8 new 128 bit register, 100+ instructions



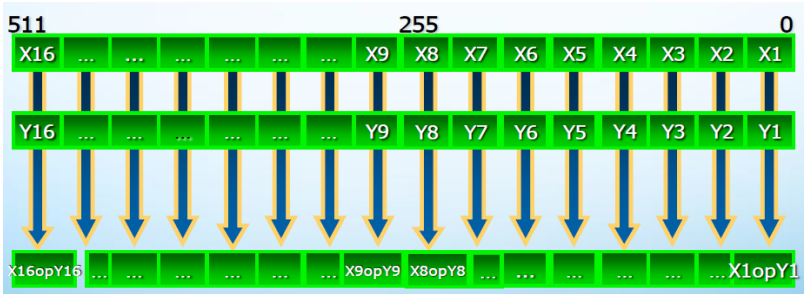
- SSE2 (2001, Pentium 4)
  - High performance computing
  - Adds 2-way float ops, double-precision; same registers as 4-way single-precision
  - Integer SSE instructions make MMX obsolete



- SSE3 (2004, Pentium 4E Prescott)
  - Scientific computing
  - New 2-way and 4-way vector instructions for complex arithm

- SSSE3 (2006, Core Duo)
  - Minor advancement over SSE3

- SSE4 (2007, Core2 Duo Penryn)
  - Modern codecs, cryptography, New integer instructions
  - Better support for unaligned data, super shuffle engine



- AVX (2011, Sandy Bridge, Bulldozer), AVX2 (2013, Haswell)
  - 256 bit register (YMM0 – YMM15)
  - Three-operand SIMD instruction format

- Intel MIC

CPU Performance

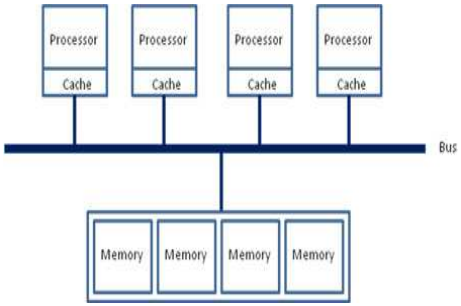
CPU performance =

Cores	✖	High frequency clock	✖	ISA extension
32 ~ 64		2 ~ 3 GHz		AVX, FMA

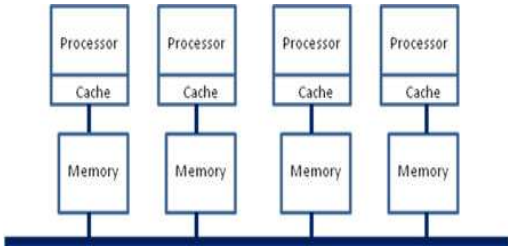
ex) Processor Number : E5-2690

Base clock : 2.9 GHz, Max Turbo : 3.8 GHz
1 socket(processor) : 8 (cores) * 3.8 (GHz) * 4 (vector) * 2 (FMA) = 243 (GFLOPS)
4 sockets : 4 (sockets) * 243 (GFLOPS) = **972 GFLOPS**

- Shared Memory
  - Single address space for all processors

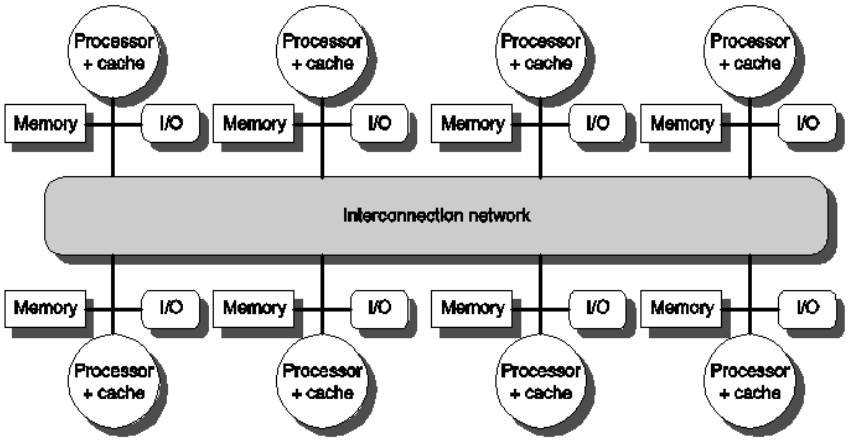


<UMA>



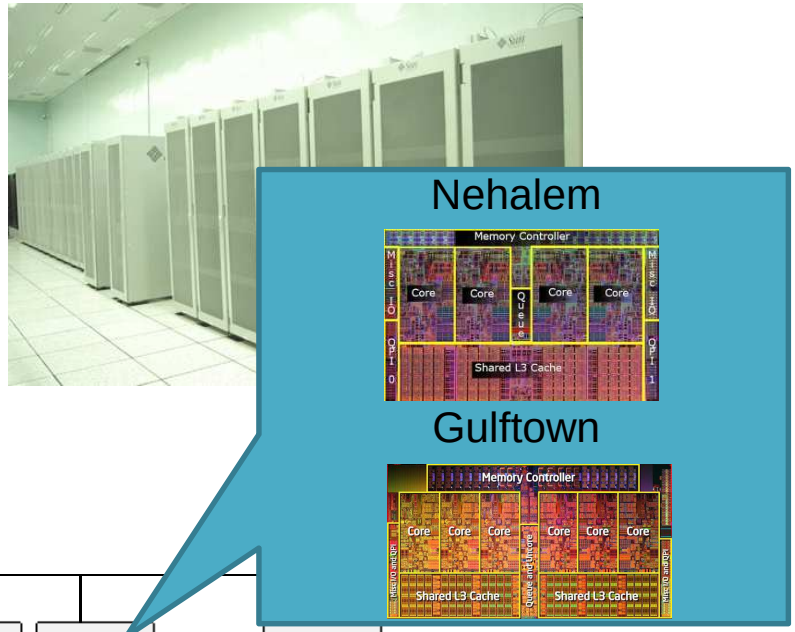
<NUMA>

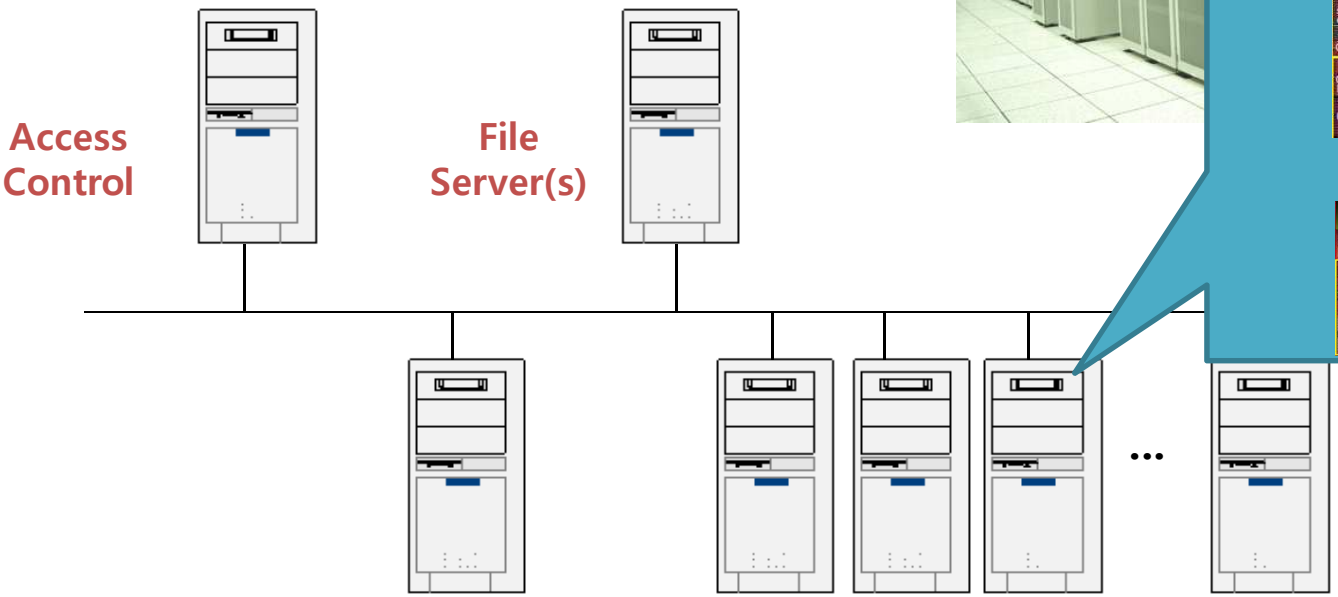
- Distributed Memory



Current HPC Platforms: COTS-Based Clusters

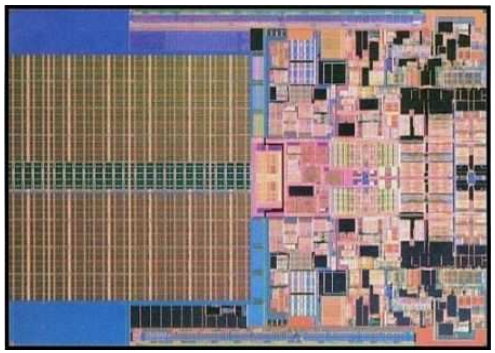
COTS = Commercial off-the-shelf



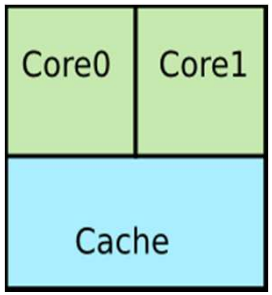


Evolution of Intel Multi-core CPU

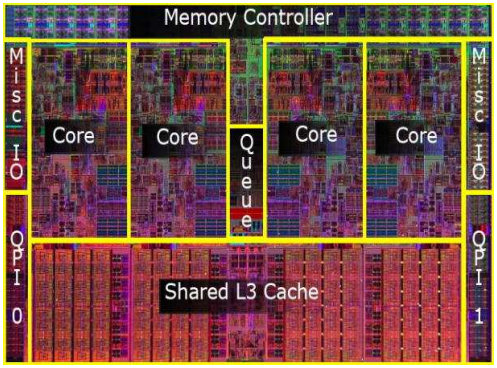
Penryn

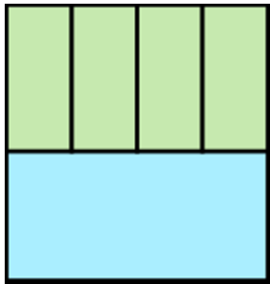


Multi Core

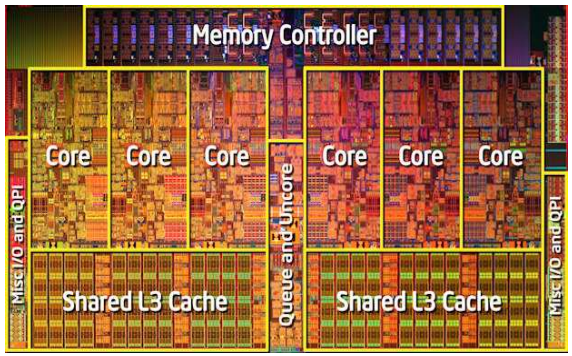


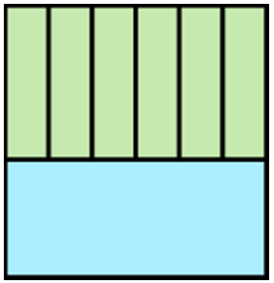
Bloomfield



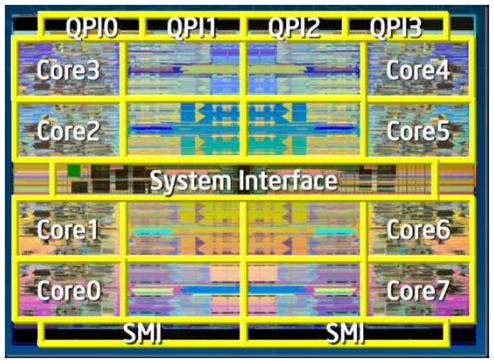


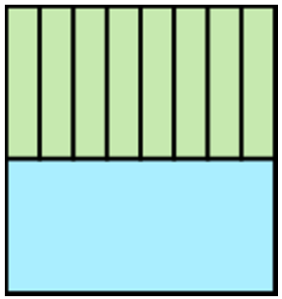
Gulftown



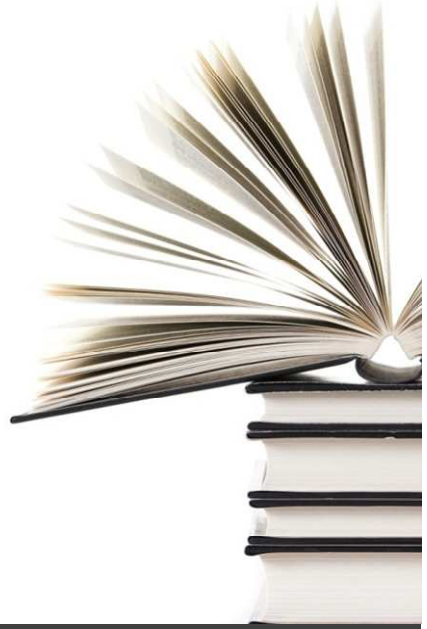


Beckton



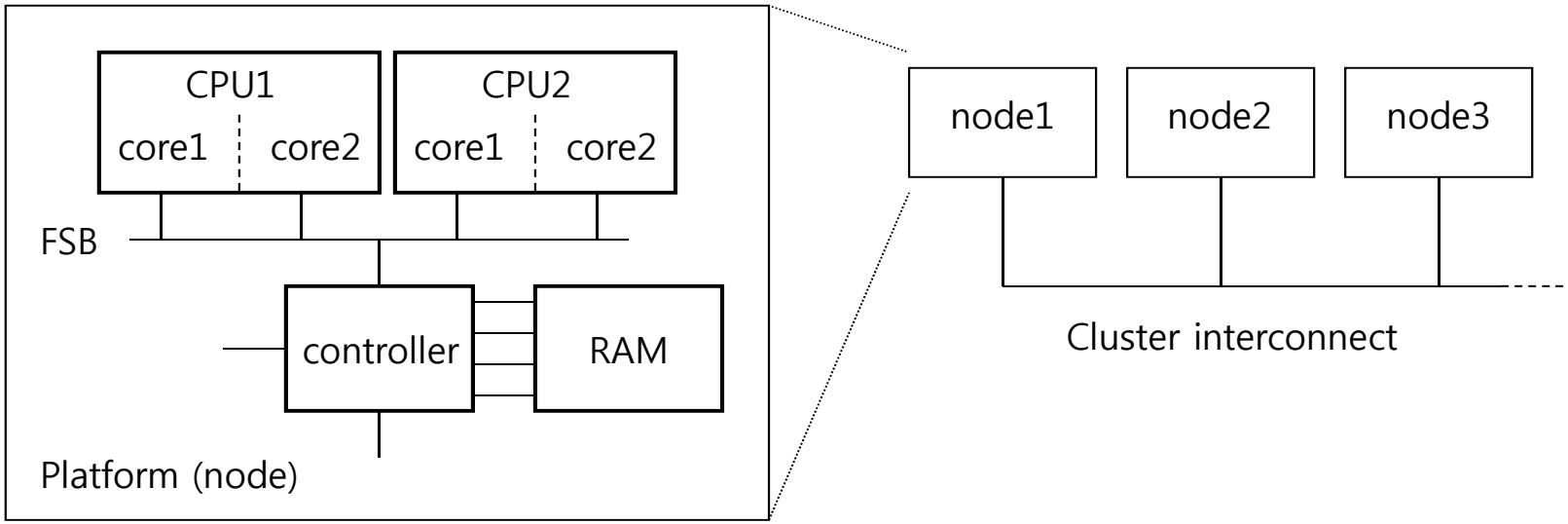


How To Program A Parallel Program?



Shared vs Distributed Memory

Multi-core CPUs in clusters – two types of parallelism to consider

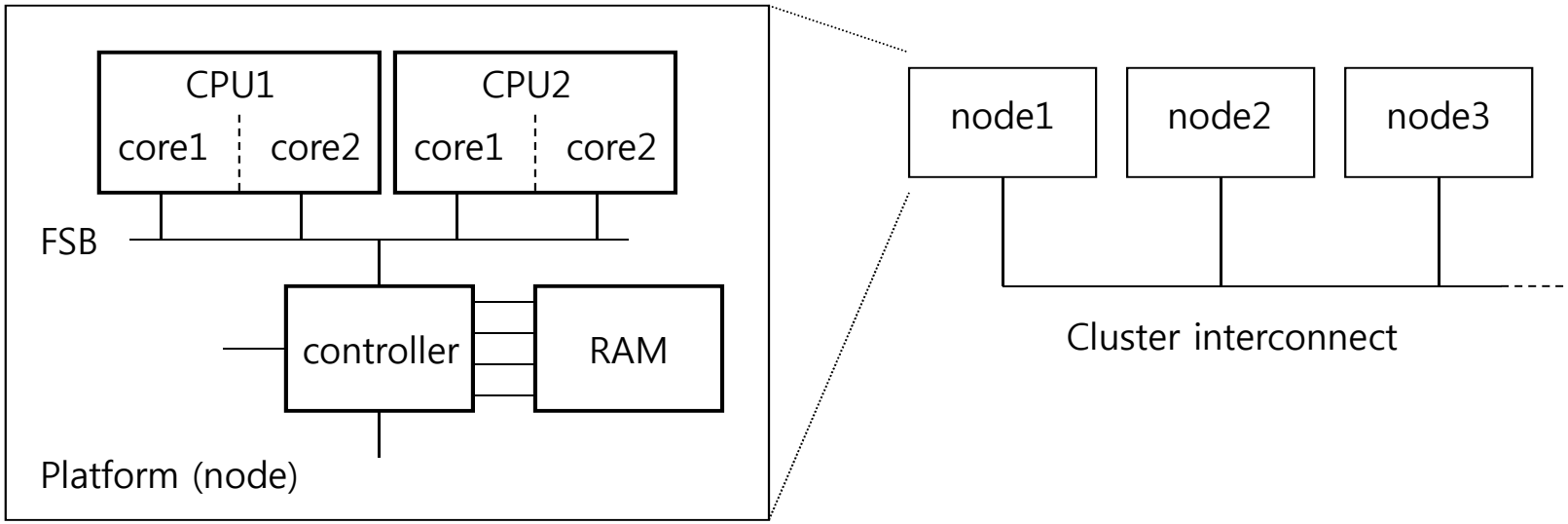


Shared memory on each node

Distributed memory across cluster

Shared vs Distributed Memory

Multi-core CPUs in clusters – two types of parallelism to consider



Shared memory on each node…

Pthreads
OpenMP
Automatic Parallelization

Distributed memory across cluster

Sockets
PVM
MPI

How To Program A Parallel Program?

- A Single System("Shared Memory")
  - Native Threading Model(standardized, low level)
  - Open MP(de-facto standard)
  - Automatic Parallelization(compiler does if for you)

- A Cluster of Systems("Distributed Memory")
  - Sockets(standardized, low level)
  - PVM – Parallel Virtual Machine (obsolete)
  - ⇨ MPI – Message Passing Interface (de-facto standard)

Parallel Models Compared

	MPI	Threads	OpenMP*
Portable	✓		✓
Scalable	✓	✓	✓
Performance Oriented	✓		✓
Supports Data Parallel	✓	✓	✓
Incremental Parallelism			✓
High Level			✓
Serial Code Intact			✓
Verifiable Correctness			✓
★ Distributed Memory	✓		

What is MPI?

- Message Passing Interface
- MPI is a library, not a language
- It is a library for inter-process communication and data exchange
- Use for Distributed Memory

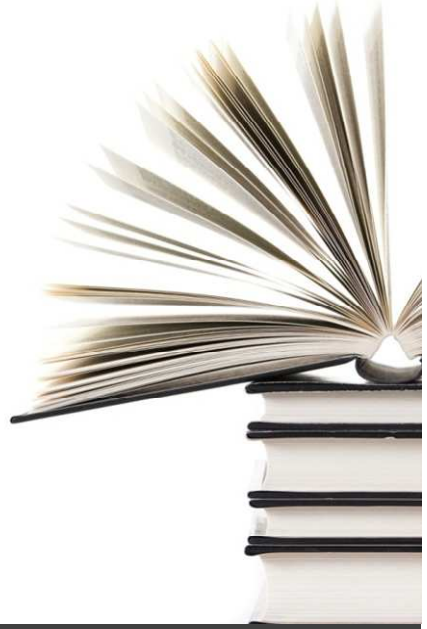
- History
  - MPI-1 Standard (MPI Forum) : 1994
    - http://www.mcs.anl.gov/mpi/index.html
    - MPI-1.1(1995), MPI-1.2(1997)
  - MPI-2 Announce : 1997
    - http://www.mpi-forum.org/docs/docs.html
    - MPI-2.1(2008), MPI-2.2(2009)
  - MPI-3 Announce : 2012
    - http://www.mpi-forum.org/docs/docs.html

Common MPI Implementations

- MPICH(Argonne National Laboratory)
  - Most common MPI implementation
  - Derivatives
    - MPICH GM – Myrinet support (available from Myricom)
    - MVAPICH – infiniband support (available from Ohio State University)
    - Intel MPI – Version tuned to Intel Architecture systems
- Open MPI(Indiana University/LANL)
  - Contains many MPI 2.0 features
  - FT-MPI: University of Tennessee (Data types, process fault tolerance, high performance)
  - LA-MPI: Los Alamos (Pt-2-Pt, data fault-tolerance, high performance, thread safety)
  - LAM/MPI: Indiana University (Component architecture, dynamic processes)
  - PACX-MPI: HLRS - Stuttgart (dynamic processes, distributed environments, collectives)
- Scali MPI Connect
  - Provides native support for most high-end interconnects
- MPI/Pro (MPI Software Technology)

교육시스템 접속방법

Environment Setup (New)



Basic User Environment

Login

- 사용자의 초기 접속은 총 4대의 로그인 노드로 한정됨
- 컴퓨팅 노드로의 액세스 불가
- 액세스 인터페이스는 ssh, sftp, ftp, X11 만 허용됨

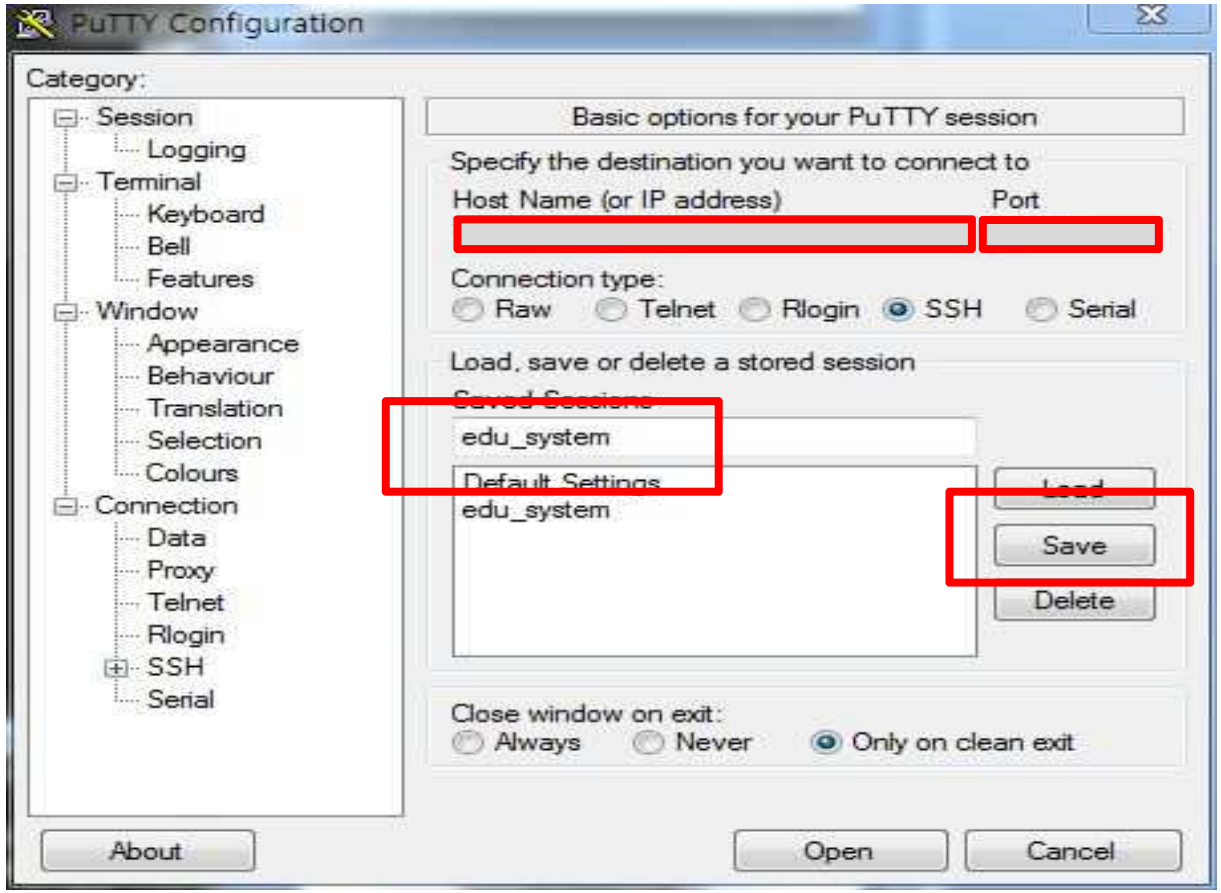
Unix or Linux

- $ ssh -l *사용자ID* xxx.xxx.xxx.xxx 또는
- $ ssh -l *사용자ID* IP address

Windows

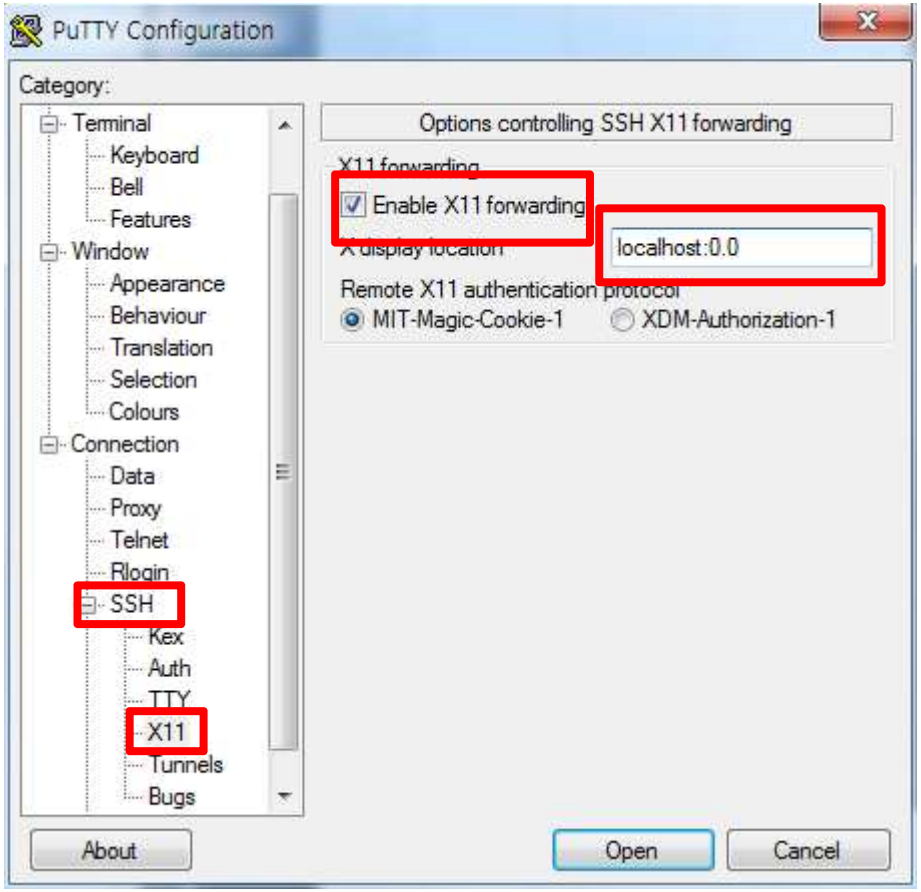
- putty 나 SSH Secure Shell Client 등의 ssh 접속 유틸리티를 이용함
  - 무료 Download 웹싸이트 : *http://www.putty.org/*

- Host Name : xxx.xxx.xxx.xxx (PuTTY Usage)



Introduction to Education System

- SSH -> X11 Tab - “Enable X11 forwarding” Check
- X display location : localhost:0.0
- Save



<Be careful>
Xming must be started in XP or Win7

➢ PuTTY Security Alert “Yes(Y)” Click

Introduction to Education System

- login as : edun## (## is your table number at the corner)
- edun##@ xxx.xxx.xxx.xxx's password :

```
edun20@master01:~
login as:
Access denied
edun20@150.183.147.250's password:
Last login: Mon Apr 16 17:02:49 2012 from 210.119.45.104
1070% [edun20@master01 ~]#
```

Introduction to Education System

- Login to Practice Node (s0001, s0002, s0003, s0004)
  - $ ssh s0001

```
edun20@s0001:~
1071% [edun20@master01 ~]# ssh s0001
Last login: Mon Apr 16 16:57:03 2012 from master01
1070% [edun20@s0001 ~]# hostname
s0001
1071% [edun20@s0001 ~]#
```

- Choose GNU Compiler : module command (add, rm, av, li, purge)
  - $ module

```
edun20@s0001:~
1076% [edun20@s0001 ~]# module
cmdModule.c(166):ERROR:11: Usage is 'module command  [arguments ...] '

  Modules Release 3.2.6 2007-02-14 (Copyright GNU GPL v2 1991):

  Usage: module [ switches ] [ subcommand ] [subcommand-args ]
```

```
edun20@s0001:~
  Available SubCommands and Args:
        + add|load              modulefile [modulefile ...]
        + rm|unload             modulefile [modulefile ...]
        + switch|swap           [modulefile1] modulefile2
        + display|show          modulefile [modulefile ...]
        + avail                 [modulefile [modulefile ...]]
        + use [-a|--append]     dir [dir ...]
        + unuse                 dir [dir ...]
        + update
        + refresh
        + purge
        + list
        + clear
```

- Choose GNU Compiler : module command
  - $ module av
  - $ module add compiler/gcc-4.1.2 mpi/openmpi-1.4.2

```
edun20@s0001:~
1083% [edun20@s0001 ~]# module av

-------------------------- /applic/Modules/modulefiles --------------------------
applic/abaqus-6.10-1      applic/mathlib-1.0        compiler/intel-10.1
applic/abaqus-6.9-1       applic/meep-1.1.1         compiler/intel-11.1
applic/abaqus-6.9-EF2     applic/mpip-3.1.2         compiler/pgi-8.0
applic/amber-10           applic/namd-2.8           compiler/pgi-8.0-6
applic/deform-10.2        applic/ncarg-4.4.2        compiler/pgi-9.0
applic/fds-5.5.1          applic/ncarg-5.2.1        compiler/pgi-9.0-4
applic/fftw-2.1.5         applic/ncarg-6.0.0        mpi/mvapich-1.1.0
applic/fftw-3.2.1         applic/ncview-2.1.1       mpi/mvapich-1.2.0
applic/g03                applic/netcdf4            mpi/mvapich-1.2.7
applic/g09                applic/netcdf4-4.1.3      mpi/mvapich2-1.2p1
applic/grads-2.0.a8       applic/numpy-1.6.0        mpi/mvapich2-1.4
applic/grads-2.0.a9       applic/papi-4.1.3         mpi/mvapich2-1.5
applic/gromacs-4.5.3      applic/pdt-3.16           mpi/openmpi-1.2.6
applic/hdf4               applic/rip4-4.3           mpi/openmpi-1.3.2
applic/hdf4-4.2.6         applic/tau-2.19.2         mpi/openmpi-1.3.3
applic/hdf5               applic/tau-2.20.2         mpi/openmpi-1.4
applic/hdf5-1.8.7         applic/totalview-8.7.0-7  mpi/openmpi-1.4.1
applic/java-1.6.0_16      applic/totalview-8.8.0-2  mpi/openmpi-1.4.2
applic/java-1.6.0_17      applic/xv-3.10            mpi/openmpi-1.4.3
applic/java-1.6.0_21      compiler/gcc-4.1.2        switcher/1.0.13(default)
applic/java-1.6.0_24      compiler/gcc-4.4.6
1084% [edun20@s0001 ~]# module add  compiler/gcc-4.1.2  mpi/openmpi-1.4.2
```

Introduction to Education System

- Choose GNU Compiler : module command
  - $ module li
  - $ module purge

```
edun20@s0001:~
1091% [edun20@s0001 ~]# module li
No Modulefiles Currently Loaded.
1092% [edun20@s0001 ~]#
1092% [edun20@s0001 ~]# module add compiler/gcc-4.1.2  mpi/openmpi-1.4.2
1093% [edun20@s0001 ~]#
1093% [edun20@s0001 ~]# module li
Currently Loaded Modulefiles:
  1) compiler/gcc-4.1.2   2) mpi/openmpi-1.4.2
1094% [edun20@s0001 ~]#
1094% [edun20@s0001 ~]# module purge
1095% [edun20@s0001 ~]#
1095% [edun20@s0001 ~]# module add compiler/gcc-4.1.2  mpi/openmpi-1.4.2
1096% [edun20@s0001 ~]#
1096% [edun20@s0001 ~]# module li
Currently Loaded Modulefiles:
  1) compiler/gcc-4.1.2   2) mpi/openmpi-1.4.2
1097% [edun20@s0001 ~]#
```

- GNU Compiler check
  - $ gcc –v
  - $ mpicc –v

```
edun20@s0001:~
1091% [edun20@s0001 ~]# gcc -v
Using built-in specs.
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/u
sr/share/info --enable-shared --enable-threads=posix --enable-checking=release -
-with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-
libgcj-multifile --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable
-java-awt=gtk --disable-dssi --enable-plugin --with-java-home=/usr/lib/jvm/java-
1.4.2-gcj-1.4.2.0/jre --with-cpu=generic --host=x86_64-redhat-linux
Thread model: posix
gcc version 4.1.2 20080704 (Red Hat 4.1.2-44)
1092% [edun20@s0001 ~]# mpicc -v
Using built-in specs.
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/u
sr/share/info --enable-shared --enable-threads=posix --enable-checking=release -
-with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-
libgcj-multifile --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable
-java-awt=gtk --disable-dssi --enable-plugin --with-java-home=/usr/lib/jvm/java-
1.4.2-gcj-1.4.2.0/jre --with-cpu=generic --host=x86_64-redhat-linux
Thread model: posix
gcc version 4.1.2 20080704 (Red Hat 4.1.2-44)
1093% [edun20@s0001 ~]#
```

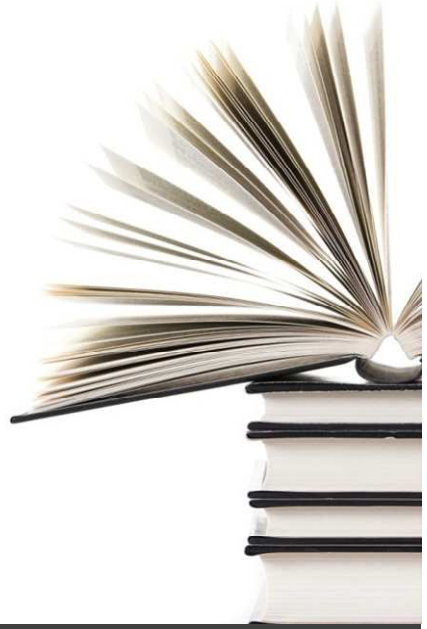
Checking Environment

- Making MPI hosts file
  - Use familiar editor : vi, emacs, gedit, etc…

```
$ cat  hosts
s0001
s0002
s0003
s0004
```

Break!!

How To Run MPI



MPI Basic Steps

- Writing a program
  - using “mpi.h” and some essential function calls
- Compiling your program
  - using a compilation script
- Specify the machine file

"Hello, World" in MPI - C

- Demonstrates how to create, compile and run a simple MPI program on the lab cluster using the Intel MPI implementation

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

int main (int argc, char* argv[])
{
   /* Initialize the library */
   MPI_Init(&argc, &argv);

   printf("Hello world\n");

   /* Wrap it up. */
   MPI_Finalize();
}
```

Initialize MPI Library

Do some work!

Return the resources

$ mpicc -o hello.x hello.c

$ mpirun -np 4 -hostfile hosts ./hello.x

"Hello, World" in MPI - Fortran

- Demonstrates how to create, compile and run a simple MPI program on the lab cluster using the Intel MPI implementation

```
PROGRAM hello
INCLUDE 'mpif.h'

        INTEGER iErr

        CALL MPI_Init(iErr)

        WRITE (*, *) 'Hello, World'

        CALL MPI_Finalize(iErr)
END
```

Initialize MPI Library

Do some work!

Return the resources

```
$ mpif90  -o  hello.x  hello.f90
$ mpirun  -np  4  -hostfile  hosts  ./hello.x
```

Compiling an MPI Program

- Most MPI implementations supply compilation scripts, eg.

Language	Command Used to Compile
Fortran 77	mpif77 mpi_prog.f
Fortran 90	mpif90 mpi_prog.f90
C	mpicc mpi_prog.c
C++	mpiCC or mpicxx mpi_prog.C

- Manual compilation/linking also possible
  - *Extremely complex*

```
$ gcc –o hello.x -L/applic/compilers/gcc/4.1.2/mpi/openmpi/1.4.2/lib64
  -I/applic/compilers/gcc/4.1.2/mpi/openmpi/1.4.2/include hello.c -lmpi
```

MPI Machine File

- A text file telling MPI where to launch processes
- Put separate host name on each line
- Example

교육용
시스템

```
node01
node02
node03
node04
```

```
s0001
s0002
s0003
s0004
```

- Check implementation for multi-processor node formats
- Default file found in MPI installation

Parallel Program Execution

- Launch scenario for MPIRUN
  - Find machine file(to know where to launch)
  - Use SSH or RSH to execute a copy of the program on each node in machine file
  - Once launched each copy establishes communications with local MPI lib (MPI_Init)
  - Each copy ends MPI interaction with MPI_Finalize

Starting an MPI Program

1

Execute on node1:

$ mpirun -np 4 –hostfile hosts ./hello.x

2

Check the MPI hostfile:

s0001
s0002
s0003
s0004

3

s0001

./hello.x (rank 0)

s0002

./hello.x (rank 1)

s0003

./hello.x (rank 2)

s0004

./hello.x (rank 3)

Lab #1

Activity 1: A First MPI Program - C

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

int main(int argc, char* argv[])
{
        MPI_Init(&argc, &argv);                    // MPI  Initialization

        printf ("Hello World \n");                 // Parallel  Code

        MPI_Finalize();                            // MPI  Close

        return 0;
}
```

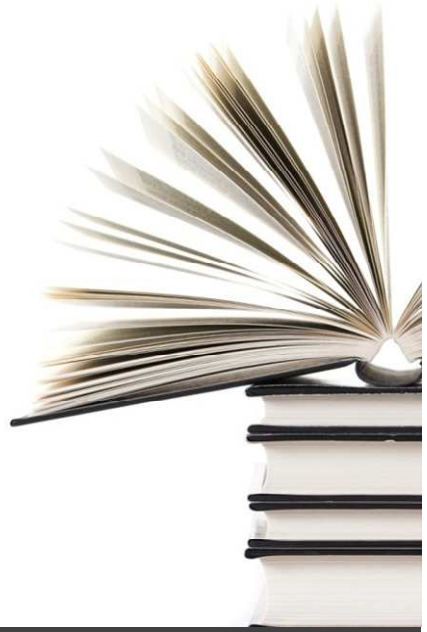
```
$ mpicc  -o  hello  –Wall  hello.c
$ mpirun  -np  4  –hostfile  hosts  ./hello
```

Activity 1: A First MPI Program - Fortran

```
PROGRAM hello
INCLUDE  'mpif.h'

      INTEGER  ::  iErr

      CALL MPI_Init(iErr)                    ! MPI  Initialization

      WRITE (*, *) 'Hello, World‘            ! Parallel  Code

      CALL MPI_Finalize(iErr)                ! MPI  Close
END
```

$ mpif90 -o hello.x –Wall hello.f90
$ mpirun -np 4 –hostfile hosts hello.x

Six-function MPI



The Whole Library

- MPI is large and complex
  - MPI 1.0 have 125 functions
  - MPI 2.0 is even larger

- But, many MPI features are rarely used
  - Inter-communicators
  - Topologies
  - Persistent communication
  - Functions designed for library developers

The Absolute Minimum

- Six MPI functions
  - Many parallel algorithms can be implemented efficiently with only these functions

Fortran	C
`MPI_INIT()`	`MPI_Init()`
`MPI_COMM_SIZE()`	`MPI_Comm_size()`
`MPI_COMM_RANK()`	`MPI_Comm_rank()`
`MPI_SEND()`	`MPI_Send()`
`MPI_RECV()`	`MPI_Recv()`
`MPI_FINALIZE()`	`MPI_Finalize()`

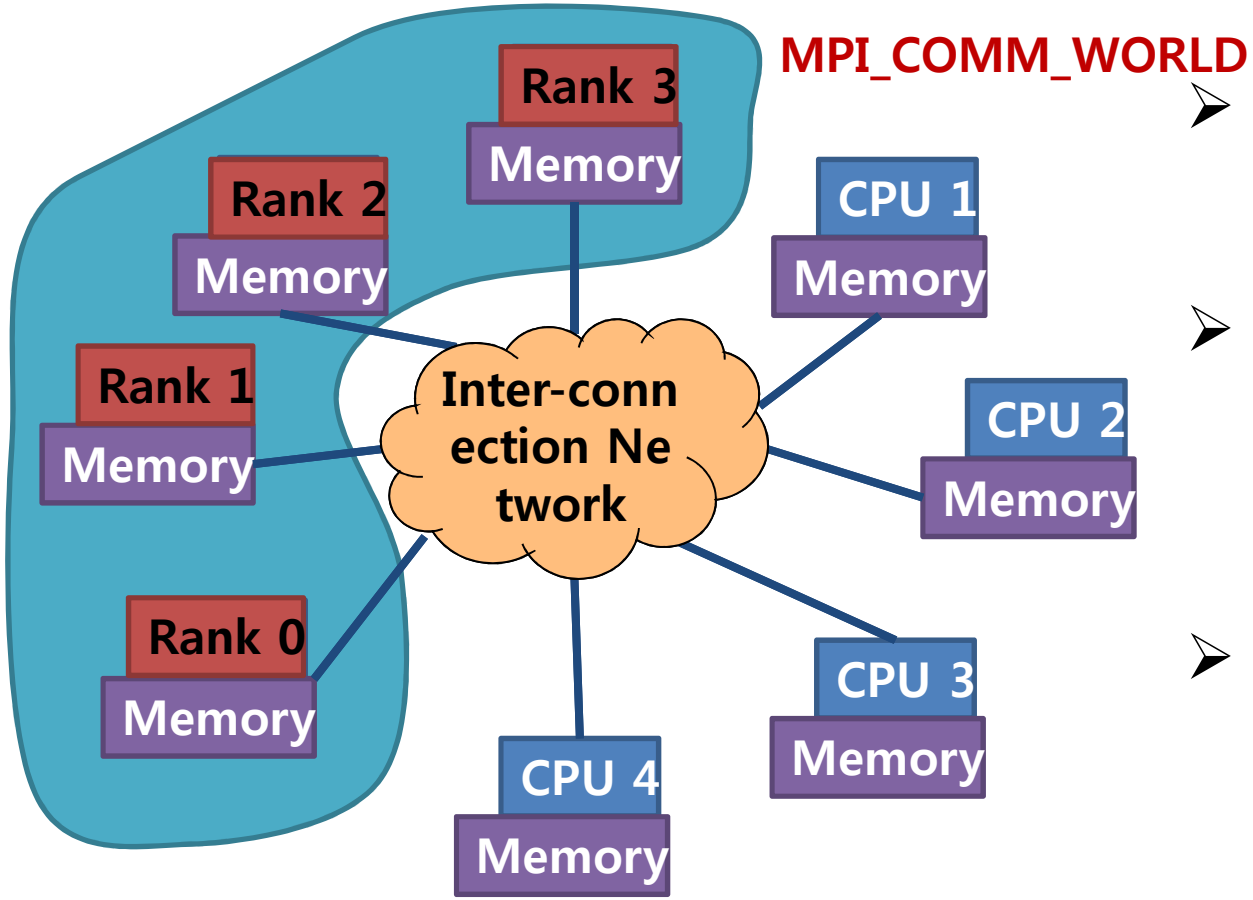
The Absolute Minimum

- Six MPI functions
  - Many parallel algorithms can be implemented efficiently with only these functions

Fortran	C
MPI_INIT()	MPI_Init()
MPI_COMM_SIZE()	MPI_Comm_size()
MPI_COMM_RANK()	MPI_Comm_rank()
MPI_SEND()	MPI_Send()
MPI_RECV()	MPI_Recv()
MPI_FINALIZE()	MPI_Finalize()

Starting MPI

Fortran	C
`CALL MPI_INIT(ierr)`	`int MPI_Init(&argc, &argv)`



- `MPI_Init` prepar es the system for MPI execution
- Call to `MPI_Init` may update argu ments in C
  - Implementation de pendent
- No MPI functions may be called bef ore `MPI_Init`

Exiting MPI

Fortran	C
`CALL MPI_FINALIZE(ierr)`	`int MPI_Finalize();`

- MPI_Finalize frees any memory allocated by the MPI library
- No MPI function may be called after calling MPI_Finalize
  - `Exception:MPI_Init`

If any one process does not reach the finalization statement, the program will appear to hang

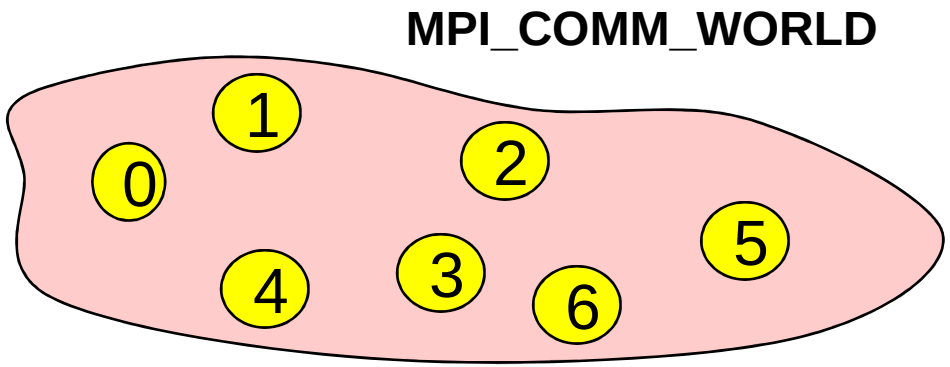
The Absolute Minimum

- Six MPI functions

Fortran	C
MPI_INIT()	MPI_Init()
MPI_COMM_SIZE()	MPI_Comm_size()
MPI_COMM_RANK()	MPI_Comm_rank()
MPI_SEND()	MPI_Send()
MPI_RECV()	MPI_Recv()
MPI_FINALIZE()	MPI_Finalize()

MPI Communicator

- A handle representing a group of processes that can communicate with each other(more about communicators later)
- All MPI communication calls have a communicator argument
- Most often you will use MPI_COMM_WORLD
  - Defined when you call MPI_Init
  - It is all of your processors.



Communicator Size

- How many processes are contained within a communicator?

Fortran	`CALL MPI_COMM_SIZE(comm, size, ierr)`
C	`int MPI_Comm_size(MPI_Comm comm, int *size)`

- MPI_Comm_size returns the number of processes in the specified communicator
- The communicator structure, MPI_Comm, is defined in mpi.h

Processor Rank

- Process ID number within communicator

Fortran	`CALL MPI_COMM_RANK(comm, rank)`
C	`int MPI_Comm_rank(MPI_Comm com, int *rank)`

- MPI_Comm_rank returns the rank of calling process within the specified communicator
- Processes are numbered from *0 to N-1*

Lab #2

Lab 2: "Hello, World" with hostname - C

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

int main(int argc, char *argv[])
{
    int       nRank, nProcs;
    char      procName[MPI_MAX_PROCESSOR_NAME];
    int       nNameLen;

    MPI_Init(&argc, &argv);                          // MPI Start
    MPI_Comm_rank(MPI_COMM_WORLD, &nRank);           // Get current processor rank id
    MPI_Comm_size(MPI_COMM_WORLD, &nProcs);          // Get number of processors

    MPI_Get_processor_name(procName, &nNameLen);

    printf("Hello World. (Process name = %s, nRank = %d, nProcs = %d)\n", procName, nRank, nProcs);

    MPI_Finalize();                                  // MPI End
    return 0;
}
```

$ mpicc -o hello_host –Wall hello_host.c

$ mpirun -np 4 –hostfile hosts ./hello_host

Lab 2: “Hello, World” with hostname - C

```
$ cat hosts
s0001
s0002
s0003
s0004

$ mpicc  -o  hello_host.x  hello_host.c

$ mpirun  -np  4  -hostfile  hosts  ./hello_host.x
Hello World. (Process name = s0003, nRank = 2, nProcs = 4)
Hello World. (Process name = s0002, nRank = 1, nProcs = 4)
Hello World. (Process name = s0004, nRank = 3, nProcs = 4)
Hello World. (Process name = s0001, nRank = 0, nProcs = 4)
```

Lab 2: "Hello, World" with hostname - Fortran

```
PROGRAM  hello
IMPLICIT  NONE
INCLUDE  'mpif.h'

   INTEGER  nRank, nProcs, nNameLen, iErr
   CHARACTER(10)  procName
   INTEGER  myar(5, 5)

   CALL  MPI_Init(iErr)
   CALL  MPI_Comm_size(MPI_COMM_WORLD, nProcs, iErr)
   CALL  MPI_Comm_rank(MPI_COMM_WORLD, nRank, iErr)

   CALL  MPI_Get_processor_name(procName, nNameLen, iErr)

   WRITE (*, *) 'Hello World. (Process name = ', procName, ', nRank = ', nRank, ', nProcs = ',
                  nProcs, ')'

   CALL  MPI_FINALIZE(iErr)

END
```

$ mpif90 -o hello_host.x –Wall hello_host.f90

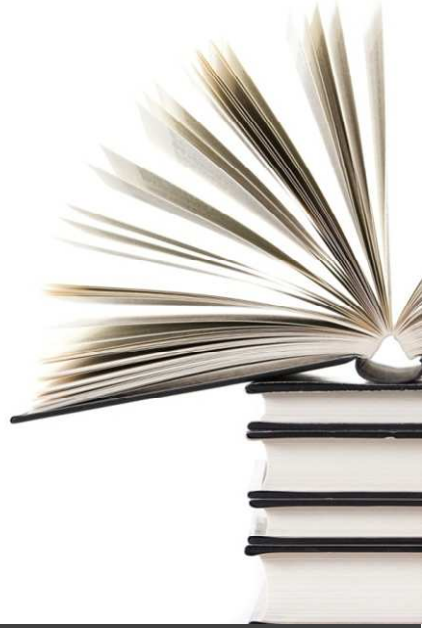
$ mpirun -np 4 –hostfile hosts ./hello_host.x

Lab 2: "Hello, World" with hostname - Fortran

```
$ cat hosts
s0001
s0002
s0003
s0004

$ mpif90  -o  hello_host.x  hello_host.f90

$ mpirun  -np  4  -hostfile  hosts  ./hello_host.x
Hello World. (Process name = s0003, nRank = 2, nProcs = 4)
Hello World. (Process name = s0002, nRank = 1, nProcs = 4)
Hello World. (Process name = s0004, nRank = 3, nProcs = 4)
Hello World. (Process name = s0001, nRank = 0, nProcs = 4)
```

Break!!

Basic Communication



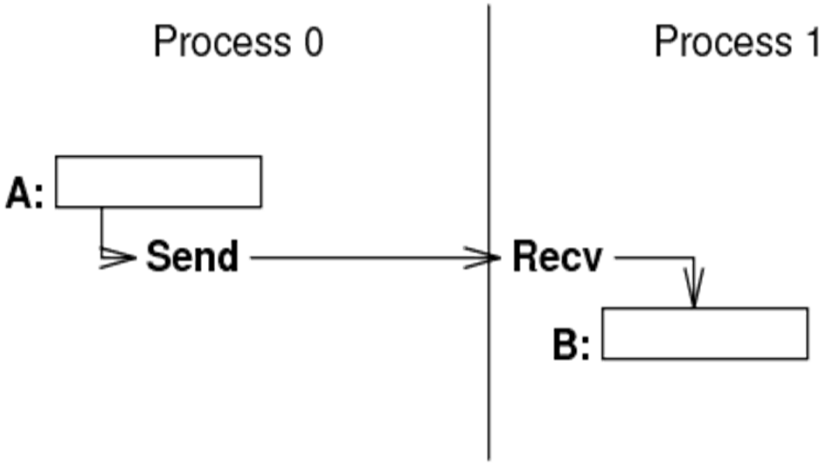
The Absolute Minimum

- Six MPI functions

Fortran	C
MPI_INIT()	MPI_Init()
MPI_COMM_SIZE()	MPI_Comm_size()
MPI_COMM_RANK()	MPI_Comm_rank()
MPI_SEND()	MPI_Send()
MPI_RECV()	MPI_Recv()
MPI_FINALIZE()	MPI_Finalize()

Sending and Receiving Message

➢ Basic Message Passing Process



- Where to send
- What to send
- How many to send

- Where to receive
- What to receive
- How many to receive

Message Organization in MPI

- Message is divided into data and envelope
  - Data
    - buffer
    - count
    - data type
  - Envelope
    - process identifier (source/destination rank)
    - message tag
    - communicator

MPI Data Types (1/2)

MPI Data Type	C Data Type
MPI_CHAR – 1 Byte character	signed char
MPI_SHORT – 2 Byte integer	signed short int
MPI_INT – 4 Byte integer	signed int
MPI_LONG – 4 Byte integer	signed long int
MPI_UNSIGNED_CHAR – 1 Byte u char	unsigned char
MPI_UNSIGNED_SHORT – 2 Byte u int	unsigned short int
MPI_UNSIGNED – 4 Byte u int	unsigned int
MPI_UNSIGNED_LONG– 4 Byte u int	unsigned long int
MPI_FLOAT – 4 Byte float point	float
MPI_DOUBLE – 8 Byte float point	double
MPI_LONG_DOUBLE- – 8 Byte float point	long double

MPI Data Types (2/2)

MPI Data Type	Fortran Data Type
MPI_INTEGER – 4 Byte Integer	INTEGER
MPI_REAL – 4 Byte floating point	REAL
MPI_DOUBLE_PRECISION – 8 Byte	DOUBLE PRECISION
MPI_COMPLEX – 4 Byte float real	COMPLEX
MPI_LOGICAL – 4 Byte logical	LOGICAL
MPI_CHARACTER – 1 Byte character	CHARACTER(1)

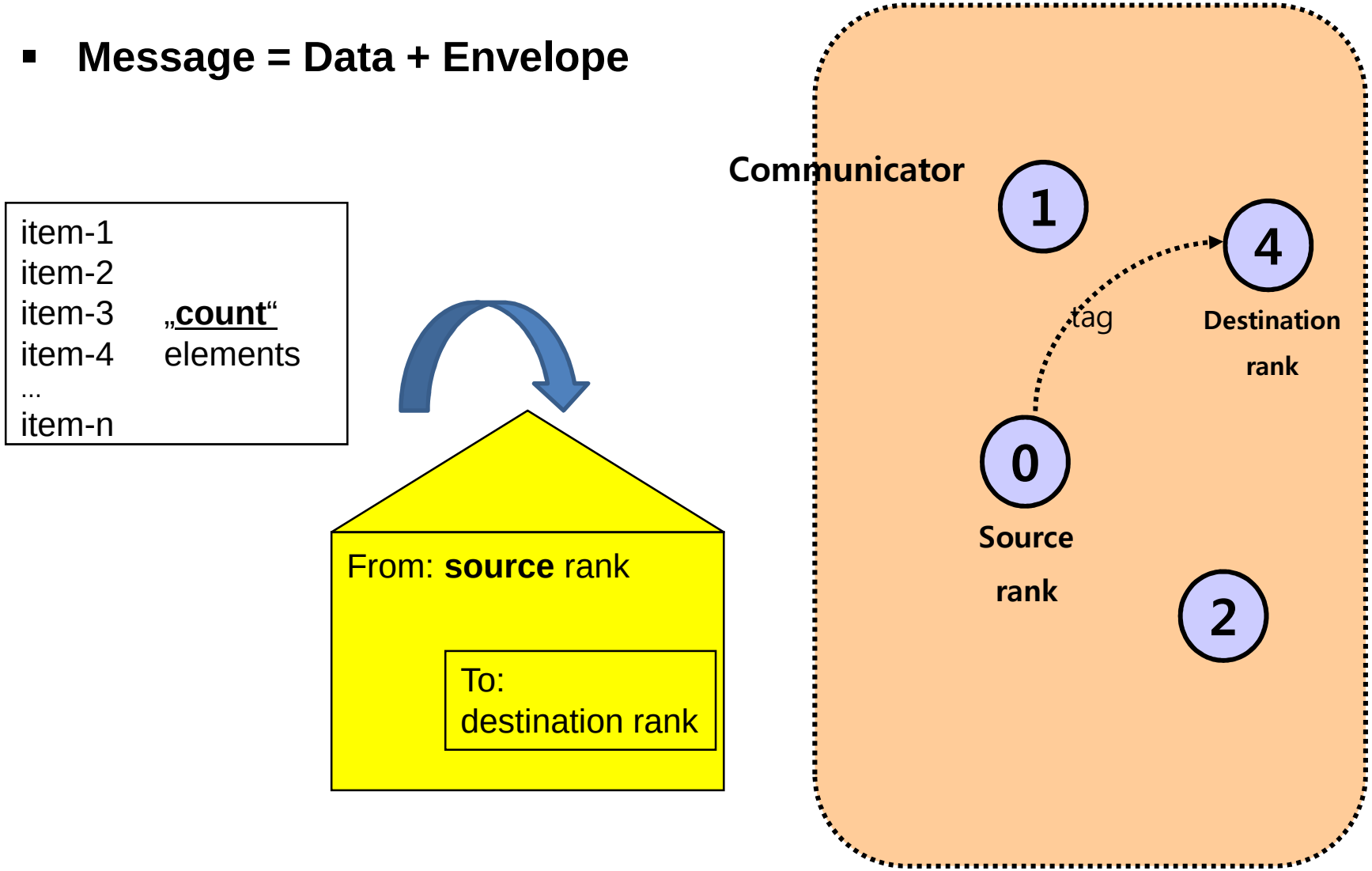
2345	654	96574	-12	7676

count=5
datatype=MPI_INTEGER

INTEGER arr(5)

Communication Envelope

- **Message = Data + Envelope**



MPI Blocking Send & Receive

- MPI_Send() vs MPI_Recv()

```
MPI_Send(buf, count, datatype, dest, tag, comm)
```

- buf: Address of send buffer
- count: Number of items to send
- datatype: Datatype of each item
- dest: Rank of destination process
- tag: Message tag
- comm: Communicator

```
MPI_Recv(buf, count, datatype, src, tag, comm, status)
```

- buf: Address of receive buffer
- count: Maximum number of items to receive
- datatype: Datatype of each item
- src: Rank of source process
- tag: Message tag
- comm: Communicator
- status: Status after operation

Blocking Recv - Status

- Status Information
  - Send Process (Rank)
  - Tag
  - Data size : MPI_GET_COUNT

Information	Fortran	C
source	status(MPI_SOURCE)	status.MPI_SOURCE
tag	status(MPI_TAG)	status.MPI_TAG
Error	status(MPI_ERROR)	status.MPI_ERROR
count	MPI_GET_COUNT()	MPI_Get_count()

MPI Send & Receive Example

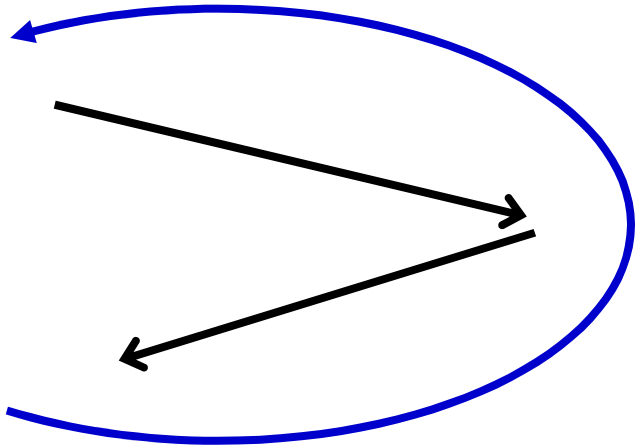
- To send an integer x from process 0 to process 1

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank); /* find rank */

if (myrank == 0) {
   int x;
   MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x, 1, MPI_INT,
   0,msgtag,MPI_COMM_WORLD,status);
}
```

Lab #3

Ping Pong Example

rank=0		rank=1
Send **(dest=1)**		
	(tag=55)	
		Recv **(source=0)**
		Send **(dest=0)**
	(tag=88)	
Recv **(source=1)**		



```
if (my_rank==0)                    /* i.e., emulated multiple program */
        MPI_Send( ... dest=1 ...)
        MPI_Recv( ... source=1 ...)
else
        MPI_Recv( ... source=0 ...)
        MPI_Send( ... dest=0 ...)
fi
```

Ping Pong Example

Fortran

```
PROGRAM Pingpong
INCLUDE 'mpif.h '
   INTEGER nrank, nprocs, ierr, status(MPI_STATUS_SIZE)
   INTEGER :: tag = 55, ROOT = 0
   INTEGER :: send = -1, recv = -1

   CALL MPI_INIT(ierr)
   CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

   IF (nrank == ROOT) THEN
      PRINT *, 'Before : nrank=', nrank, 'send=', send, 'recv=',
           recv
      send = 7
      CALL MPI_SEND(send, 1, MPI_INTEGER, 1, tag, MPI_COMM_WORLD,
                    ierr)
   ELSE
      CALL MPI_RECV(recv, 1, MPI_INTEGER, ROOT, MPI_ANY_TAG,
MPI_COMM_WORLD,
                    status, ierr)
      PRINT *, 'After  : nrank=', nrank, 'send=', send, 'recv=',
          recv
   END IF

   CALL MPI_FINALIZE(ierr)
END
```

$ mpif90 –o pingpong.x pingpong.f90
$ mpirun –np 2 –hostfile hosts ./pingpong.x

C

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

int main(int argc, char *argv[])
{
   int nrank, nprocs, tag = 55, ROOT = 0;
   int send = -1, recv = -1;
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

   if (nrank == ROOT) {
      printf("Before : nrank(%d) send = %d, recv = %d₩n",
             nrank, send, recv);
      send = 7;
      MPI_Send(&send, 1, MPI_INTEGER, 1, tag,
             MPI_COMM_WORLD);
     }
   else {
      MPI_Recv(&recv, 1, MPI_INTEGER, ROOT, MPI_ANY_TAG,
              MPI_COMM_WORLD, &status);
      printf("After  : nrank(%d) send = %d, recv = %d₩n",
               nrank, send, recv);
   }

   MPI_Finalize();
   return 0;
}
```

$ mpicc –o pingpong pingpong.c
$ mpirun –np 2 –hostfile hosts ./pingpong

The Absolute Minimum

- Six MPI functions

Fortran	C
MPI_INIT()	MPI_Init()
MPI_COMM_SIZE()	MPI_Comm_size()
MPI_COMM_RANK()	MPI_Comm_rank()
MPI_SEND()	MPI_Send()
MPI_RECV()	MPI_Recv()
MPI_FINALIZE()	MPI_Finalize()

Break!!

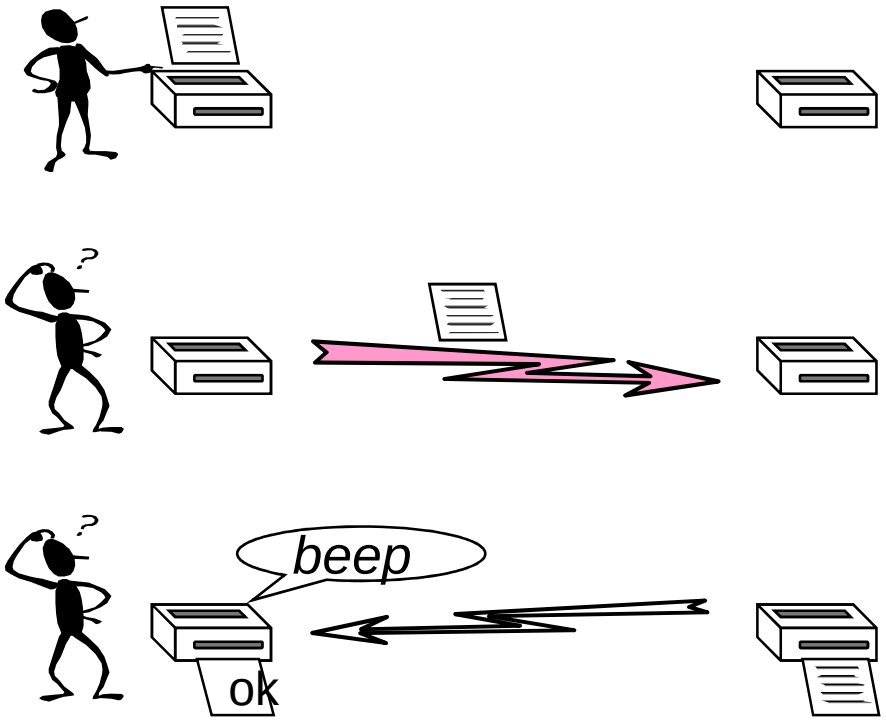
P2P: Blocking Communications

Point-to-Point Communication

- Simplest form of message passing.
- One process sends a message to another.
- Different types of point-to-point communication:
  - synchronous send
  - buffered = asynchronous send

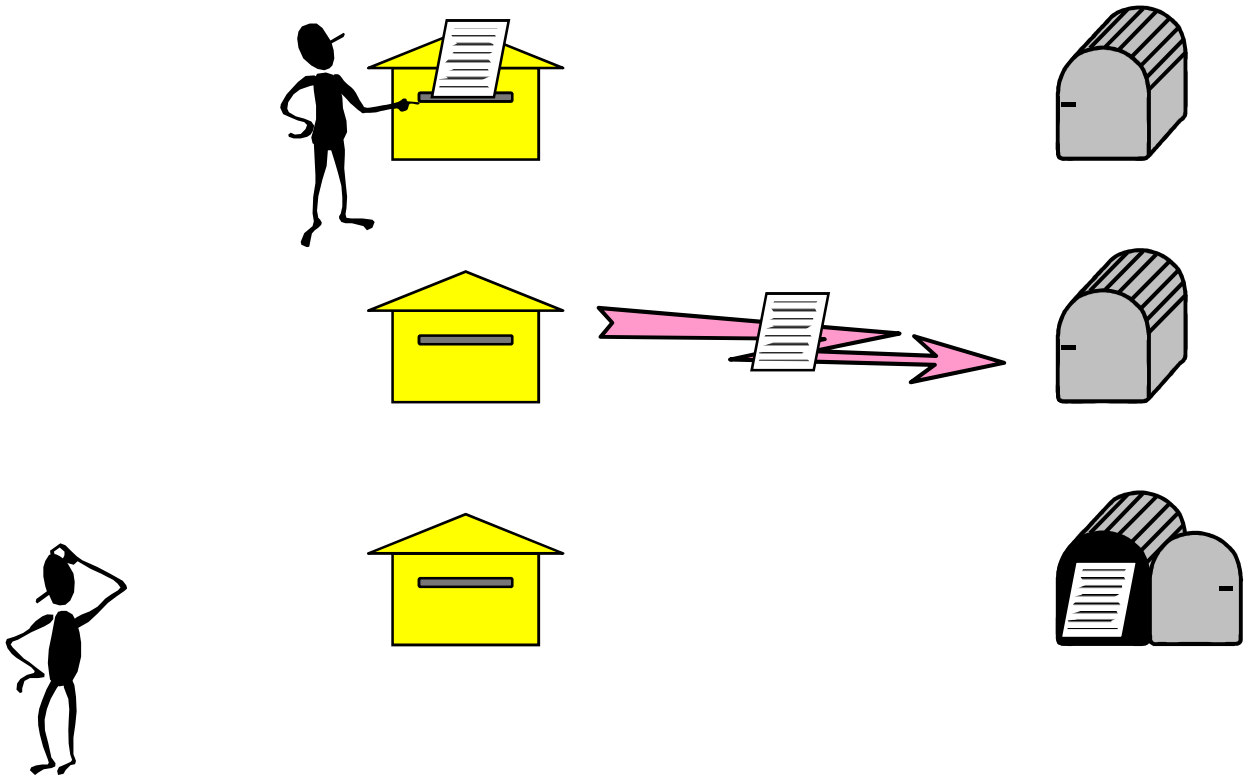
Synchronous Sends

- A synchronous communication does not complete until the message has been received
- Analogue to the beep or okay-sheet of a fax



Buffered = Asynchronous Sends

- An asynchronous communication completes as soon as the message is on the way.
- A post card or email



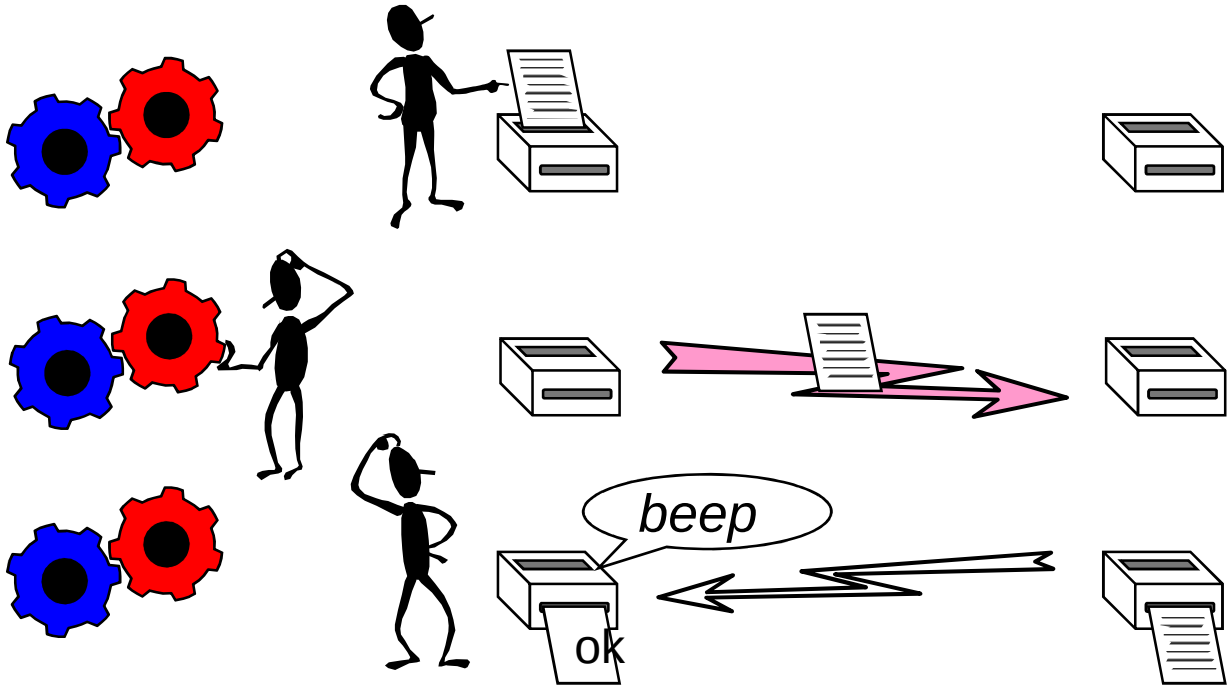
Blocking Operations

- Some sends/receives may block until another process acts:
  - synchronous send operation blocks until receive is issued;
  - receive operation blocks until message is sent.

- Blocking subroutine returns only when the operation has completed.

Non-Blocking Operations

- Non-blocking operations return immediately and allow the sub-program to perform other work



Communication Mode

Mode	MPI Call Routine	
	Blocking	Non Blocking
Synchronous	MPI_SSEND	MPI_ISSEND
Ready	MPI_RSEND	MPI_IRSEND
Buffer	MPI_BSEND	MPI_IBSEND
Standard	MPI_SEND	MPI_ISEND
Recv	MPI_RECV	MPI_IRECV

Blocking Send (standard)

C	**int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)**
Fortran	**MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)**

(CHOICE) buf : initial address of send buffer (IN)

INTEGER count : number of elements in send buffer (IN)

INTEGER data type : data type of each send buffer element (IN)

INTEGER dest : rank of destination (IN)

If communication is not needed, MPI_PROC_NULL

INTEGER tag : message tag (IN)

INTEGER comm : MPI communicator (IN)

```
MPI_SEND(a, 50, MPI_REAL, 5, 1, MPI_COMM_WORLD, ierr)
```

Blocking Recv (standard)

C	**int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)**
Fortran	**MPI_RECV(buf, count, datatype, source, tag, comm, status, ierr)**

(CHOICE) buf : initial address of receive buffer (OUT)

INTEGER count : number of elements in receive buffer (IN)

INTEGER datatype : datatype of each receive buffer element (IN)

INTEGER source : rand of source (IN)

If communication is not needed, MPI_PROC_NULL

INTEGER tag : message tag (IN)

INTEGER comm : MPI communicator (IN)

INTEGER status(MPI_STATUS_SIZE) : Have information of received message (OUT)

```
MPI_RECV(a,50,MPI_REAL,0,1,MPI_COMM_WORLD,status,ierr)
```

Precautions for Successful Communication

- The receiver has exact rank of sender
- The sender has exact rank of receiver
- Same communicator
- Same message tag
- Enough buffer size of receiver

```
[s0001:20135] *** An error occurred in MPI_Recv
[s0001:20135] *** on communicator MPI_COMM_WORLD
[s0001:20135] *** MPI_ERR_TRUNCATE: message truncated
[s0001:20135] *** MPI_ERRORS_ARE_FATAL (your MPI job will now abort)
```

Lab #4

Lab #4 Parallel Sum

- <Problem>
  - Parallel sum, ex) 1 ~ 100

- <Hint & Requirement>
  - Decide the sum range of each rank
    - Use processor size (rank), sum number (100)
  - ROOT (0 Rank) involve in the own calculation and get total sum

Lab #4 Parallel Sum – C (1/2)

```
/*
     Example Name    : parallel_sum.c
     Compile         : $ mpicc -g -o parallel_sum -Wall parallel_sum.c
     Run             : $ time mpirun -np 6 -hostfile hosts parallel_sum
*/
#include <stdio.h>
#include <mpi.h>

#define   NUMBER    100

int main(int argc, char *argv[])
{
     int i, nRank, nProcs, ROOT = 0;
     int nQuotient, nRemainder, nMyStart, nMyEnd, nMySum = 0, nTotal = 0;
     MPI_Status status;

     MPI_Init(&argc, &argv);
     MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
     MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

     nQuotient = NUMBER / nProcs;
     nRemainder = NUMBER % nProcs;

     nMyStart = nRank * nQuotient + 1;
     nMyEnd = nMyStart + nQuotient – 1;
```

Lab #4 Parallel Sum – C (2/2)

```
    if (nRank == nProcs-1)   nMyEnd += nRemainder;

    printf("nRank (%d) : nMyStart = %3d ~ nMyEnd = %3d\n", nRank, nMyStart, nMyEnd);

    // Sum each scope
    for (i = nMyStart; i <= nMyEnd; i++)         nMySum += i;

    if (nRank == ROOT) {
        nTotal = nMySum;

        for (i = 1; i < nProcs ; i++ ) {
            MPI_Recv(&nMySum, 1, MPI_INT, i, 10, MPI_COMM_WORLD, &status);
            nTotal += nMySum;
        }
        printf("\nTotal sum (1 ~ %d) = %d.\n", NUMBER, nTotal);
    }
    else
        MPI_Send(&nMySum, 1, MPI_INT, ROOT, 10, MPI_COMM_WORLD);

    MPI_Finalize();

    return 0;
}
```

Lab #4 Parallel Sum - Compile & Run

```
$ mpicc -g -o parallel_sum -Wall parallel_sum.c
$ time mpirun -np 6 -hostfile hosts parallel_sum
nRank (3) : nMyStart =  49 ~ nMyEnd =  64
nRank (4) : nMyStart =  65 ~ nMyEnd =  80
nRank (0) : nMyStart =   1 ~ nMyEnd =  16
nRank (1) : nMyStart =  17 ~ nMyEnd =  32
nRank (5) : nMyStart =  81 ~ nMyEnd = 100
nRank (2) : nMyStart =  33 ~ nMyEnd =  48

Total sum (1 ~ 100) = 5050.
```

Lab #4 Parallel Sum – Fortran (1/2)

```
! Example Name      : parallel_sum.f90
! Compile           : $ mpif90 -g -o parallel_sum.x -Wall parallel_sum.f90
! Run               : $ time mpirun –np 6 -hostfile hosts parallel_sum.x

PROGRAM parallel_sum
IMPLICIT NONE
INCLUDE 'mpif.h'
     INTEGER i, nRank, nProcs, iErr, status(MPI_STATUS_SIZE)
     INTEGER :: ROOT = 0, nMySum = 0, nTotal = 0
     INTEGER nQuotient, nRemainder, nMyStart, nMyEnd
     INTEGER        NUMBER
     PARAMETER (NUMBER = 100)

     CALL MPI_INIT(iErr)
     CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)
     CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)

     nQuotient = NUMBER/ nProcs
     nRemainder = mod(NUMBER, nProcs)

     nMyStart = nRank * nQuotient + 1
     nMyEnd = nMyStart + nQuotient - 1

     IF (nRank == nProcs-1) nMyEnd = nMyEnd + nRemainder
```

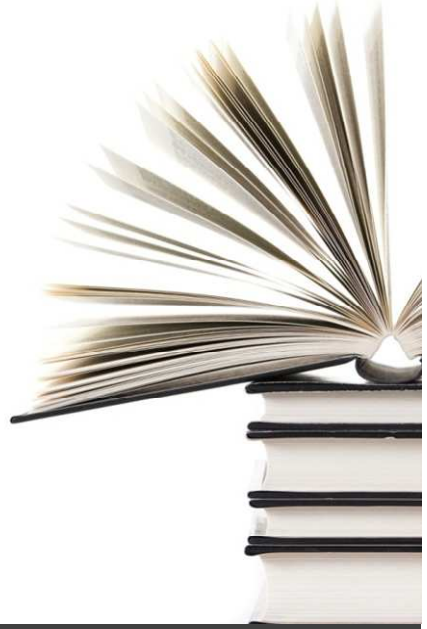
Lab #4 Parallel Sum – Fortran (2/2)

```
    PRINT *, 'nRank=', nRank, 'nMyStart=', nMyStart, 'nMyEnd=', nMyEnd

    ! sum each scope
    DO i=nMyStart, nMyEnd
        nMySum = nMySum + i
    END DO

    IF (nRank == ROOT) THEN
        nTotal = nMySum

        DO i=1, nProcs-1
            CALL MPI_RECV(nMySum, 1, MPI_INTEGER, i, 55, MPI_COMM_WORLD, status,
                          iErr)
            nTotal = nTotal + nMySum
        END DO

        PRINT *
        PRINT *, 'Total sum (1 ~ ', NUMBER, ') = ', nTotal
    ELSE
        CALL MPI_SEND(nMySum, 1, MPI_INTEGER, ROOT, 55, MPI_COMM_WORLD, iErr)
    END IF

    CALL MPI_FINALIZE(iErr)
END
```

Lab #4 Parallel Sum – Compile & Run

```
$ mpif90 -g -o parallel_sum.x -Wall parallel_sum.f90

$ time mpirun -np 6 -hostfile hosts parallel_sum.x
 nRank=           3 nMyStart=          49 nMyEnd=          64
 nRank=           4 nMyStart=          65 nMyEnd=          80
 nRank=           1 nMyStart=          17 nMyEnd=          32
 nRank=           2 nMyStart=          33 nMyEnd=          48
 nRank=           5 nMyStart=          81 nMyEnd=         100
 nRank=           0 nMyStart=           1 nMyEnd=          16

 Total sum (1 ~          100 ) =         5050
```

P2P : Non-Blocking Communication



Non-Blocking Communication

- Communication has three steps
  1. **Initialization : Posting send or recv**
  2. **Perform other job**
     - Do communication and calculation at the same time
  3. **Completion : Waiting or Testing**

- Easier to write dead-lock free code
- Reduce communication overhead

Non-Blocking's Initialization

C	`int MPI_ISend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)`
Fortran	`MPI_ISEND(buf, count, datatype, dest, tag, comm, request, ierr)`

C	`int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)`
Fortran	`MPI_IRECV(buf, count, datatype, source, tag, comm, request, ierr)`

INTEGER request : communication request handle (OUT)

Non-blocking Irecv has no status argument

Non-blocking's Completion

- Waiting or Testing
  - Waiting
    - Process blocking until the communication is complete
    - **Non-blocking communication + Waiting = Blocking Comm**
  - Testing
    - Return true or false, depending on communication completion

MPI_WAIT

C	`int MPI_Wait(MPI_Request *request, MPI_Status *status)`
Fortran	`MPI_WAIT(request, status, ierr)`

INTEGER request : request handle (IN)

INTEGER status(MPI_STATUS_SIZE) : status object (OUT)

MPI_TEST

C	`int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)`
Fortran	`MPI_TEST(request, flag, status, ierr)`

INTEGER request : request handle (IN)

LOGICAL flag : true if operation completed, or false (OUT)

INTEGER status(MPI_STATUS_SIZE) : status object (OUT)

Lab #5

One Sided Communication (1 / 2)

Blocking Send, Blocking Recv

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
   ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_RECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
   istatus, ierr)
ENDIF
```

Non-Blocking Send, Blocking Recv

```
IF (myrank==0) THEN
   CALL MPI_ISEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
   ireq, ierr)
   CALL MPI_WAIT(ireq, istatus, ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_RECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
   istatus, ierr)
ENDIF
```

One Sided Communication (2 / 2)

- Blocking Send, Non-Blocking Recv

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
   ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_IRECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
   ireq, ierr)
   CALL MPI_WAIT(ireq, istatus, ierr)
ENDIF
```

- Non-Blocking Send, Non-Blocking Recv

```
IF (myrank==0) THEN
   CALL MPI_ISEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
   ireq, ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_IRECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
   ireq, ierr)
ENDIF
CALL MPI_WAIT(ireq, istatus, ierr)
```

Non-blocking Send/Recv - C

```
/* Example name : isend.c */
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
      int nrank, nprocs, i, ROOT = 0;
      float data[100], value[200];
      MPI_Request req;
      MPI_Status status;

      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
      MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

      if (nrank == 1) {
            for (i=0; i<100; i++)
                        data[i] = 3.0;
            MPI_Isend(data, 100, MPI_FLOAT, ROOT, 55, MPI_COMM_WORLD, &req);
            MPI_Wait(&req, &status);
      }
      else if (nrank == ROOT){
            for (i=0; i<200; i++)
                        value[i] = 0.0;
            MPI_Irecv(value, 200, MPI_FLOAT, 1, 55, MPI_COMM_WORLD, &req);
            MPI_Wait(&req, &status);

            printf("rank(%d) value[5] = %f\n", nrank, value[5]);
      }

      MPI_Finalize();
      return 0;
}
```

```
$ mpicc -o isend -Wall isend.c
$ mpirun -np 2 -hostfile hosts isend
rank(0) value[5] = 3.000000
```

Non-blocking Send/Recv - Fortran

```
! Example name : isend.f90
PROGRAM isend
INCLUDE 'mpif.h'
      INTEGER :: ierr, nrank, req, ROOT = 0
      REAL data(100), value(200)
      INTEGER status(MPI_STATUS_SIZE)

      CALL MPI_INIT(ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

      IF (nrank == 1) THEN
            data = 3.0
            CALL MPI_ISEND(data, 100, MPI_REAL, ROOT, 55, MPI_COMM_WORLD, req, ierr)
            CALL MPI_WAIT(req, status, ierr)
      ELSE IF (nrank == ROOT) THEN
            value = 0.0
            CALL MPI_IRECV(value, 200, MPI_REAL, 1, 55, MPI_COMM_WORLD, req, ierr)
            CALL MPI_WAIT(req, status, ierr)

            PRINT *, 'rank : ', nrank, ' value(5) = ', value(5)
      ENDIF

      CALL MPI_FINALIZE(ierr)

END

$ mpif90 -o isend.x -Wall isend.f90
$ mpirun -np 2 -hostfile hosts isend.x
rank :            0  value(5) =    3.000000
```

Nonblocking Example

Fortran

```
Example name : non_p2p.f90
PROGRAM non_p2p
INCLUDE 'mpif.h'
   INTEGER ierr, nrank, req
   INTEGER status(MPI_STATUS_SIZE)
   INTEGER :: send = -1, recv = -1, ROOT = 0

   CALL MPI_INIT(ierr)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

   IF (nrank == ROOT) THEN
      PRINT *, 'Before : nrank = ', nrank, 'send = ', send, 'recv =
', recv
      send = 7
      CALL MPI_ISEND(send, 1, MPI_INTEGER, 1, 55, MPI_COMM_WORLD,
                     req, ierr)
      PRINT *, 'Other job calculating'

      CALL MPI_WAIT(req, status, ierr)
   ELSE
      CALL MPI_RECV(recv, 1, MPI_INTEGER, ROOT, 55, MPI_COMM_WORLD,
                    status, ierr)
      PRINT *, 'After  : nrank = ', nrank, 'send = ', send, 'recv =
', recv
   ENDIF

   CALL MPI_FINALIZE(ierr)

END
```

$ mpif90 –o non_p2p.x non_p2p.f90
$ mpirun –np 2 –hostfile hosts ./non_p2p.x

C

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

int main(int argc, char *argv[])
{
   int nrank, nprocs, tag = 55, ROOT = 0;
   int send = -1, recv = -1;
   MPI_Request req;
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

   if (nrank == ROOT) {
      printf("Before : nrank(%d) send = %d, recv = %d₩n", nrank,
      send, recv);
      send = 7;
      MPI_Isend(&send, 1, MPI_INTEGER, 1, tag, MPI_COMM_WORLD,
                &req);

      printf("Other job calculating.₩n₩n");
      MPI_Wait(&req, &status);
   }
   else {
      MPI_Recv(&recv, 1, MPI_INTEGER, ROOT, tag, MPI_COMM_WORLD,
      &status);
      printf("After  : nrank(%d) send = %d, recv = %d₩n", nrank,
      send, recv);
   }

   MPI_Finalize();

   return 0;
}
```

$ mpicc –o non_p2p non_p2p.c
$ mpirun –np 2 –hostfile hosts ./non_p2p

Two Sided Communication (1 / 9)

➢ 선 송신, 후 수신 1. : 메시지 크기에 따라 교착 가능

– Tachyon : Threshold Byte = 8kB = 2048 real = 1024 double

```
IF (myrank == 0) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank == 1) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, ...)
ENDIF
```

Two Sided Communication (2 / 9)

➢ 선 송신, 후 수신 2. (1.의 경우와 동일)

```
 IF (myrank == 0) THEN
    CALL MPI_ISEND(sendbuf, …, ireq, …)
    CALL MPI_WAIT(ireq, …)
    CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank == 1) THEN
    CALL MPI_ISEND(sendbuf, …, ireq, …)
    CALL MPI_WAIT(ireq, …)
    CALL MPI_RECV(recvbuf, ...)
ENDIF
```

Two Sided Communication (3 / 9)

➢ 선 송신, 후 수신 3. : 메시지 크기와 무관 하게 교착 없음

```
 IF (myrank == 0) THEN
    CALL MPI_ISEND(sendbuf, …, ireq, …)
    CALL MPI_RECV(recvbuf, ...)
    CALL MPI_WAIT(ireq, …)
ELSEIF (myrank == 1) THEN
    CALL MPI_ISEND(sendbuf, …, ireq, …)
    CALL MPI_RECV(recvbuf, ...)
    CALL MPI_WAIT(ireq, …)
ENDIF
```

Two Sided Communication (4 / 9)

- 전송 데이터 크기에 따른 교착여부 확인
- 교착을 피하기 위해 논블록킹 통신 사용

```
INTEGER N
PARAMETER (N=1024)
REAL a(N), b(N)
…
IF( myrank .eq. 0 ) THEN
    CALL MPI_SEND( a, N, …;
    CALL MPI_RECV( b, N, …)
ELSE IF( myrank .eq. 1 ) THEN
    CALL MPI_SEND( a, N, …)
    CALL MPI_RECV( b, N, …)
ENDIF
…
```

Two Sided Communication (5 / 9)

➢ 선 수신, 후 송신 1. : 메시지 크기와 무관하게 교착

```
IF (myrank == 0) THEN
   CALL MPI_RECV(recvbuf, ...)
   CALL MPI_SEND(sendbuf, ...)
ELSEIF (myrank == 1) THEN
   CALL MPI_RECV(recvbuf, ...)
   CALL MPI_SEND(sendbuf, ...)
ENDIF
```

Two Sided Communication (6 / 9)

➢ 선 수신, 후 송신 2. : 메시지 크기와 무관하게 교착 없음

```
IF (myrank == 0) THEN
   CALL MPI_IRECV(recvbuf, …,ireq, …)
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_WAIT(ireq, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_IRECV(recvbuf, …,ireq, …)
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_WAIT(ireq, …)
ENDIF
```

Two Sided Communication (7 / 9)

- 전송 데이터 크기와 무관한 교착발생 확인
- 교착을 피하기 위해 논블록킹 통신 사용

```
…
REAL a(100), b(100)
…
IF (myrank == 0) THEN
  CALL MPI_RECV(b, 100,…)
  CALL MPI_SEND(a, 100, …)
ELSE IF (myrank == 1) THEN
  CALL MPI_RECV(b, 100, …)
  CALL MPI_SEND(a, 100, …)
ENDIF
…
```

Two Sided Communication (8 / 9)

- 한쪽은 송신부터, 다른 한쪽은 수신부터
  : 블록킹, 논블록킹 루틴의 사용과 무관하게 교착 없음

```
IF (myrank == 0) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_RECV(recvbuf, …)
   CALL MPI_SEND(sendbuf, ...)
ENDIF
```

Two Sided Communication (9 / 9)

➢ 권장 코드

```
IF (myrank == 0) THEN
   CALL MPI_ISEND(sendbuf, …, ireq1, …)
   CALL MPI_IRECV(recvbuf, …, ireq2, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_ISEND(sendbuf, …, ireq1, …)
   CALL MPI_IRECV(recvbuf, …, ireq2, …)
ENDIF

CALL MPI_WAIT(ireq1, …)
CALL MPI_WAIT(ireq2, …)
```

Example : deadlock_blocking.c

```
/* Example name : deadlock_blocking.c */
#include <stdio.h>
#include <mpi.h>

/* if BUF_SIZE > 1500, deadlock */
#define BUF_SIZE    (1500)

int main(int argc, char *argv[])
{
   int nprocs, nrank, i, ROOT = 0;
   MPI_Status status;
   double a[BUF_SIZE], b[BUF_SIZE];

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

   for (i=0; i<BUF_SIZE; i++)
      a[i] = i;

   if (nrank == ROOT) {
      for (i=0; i<10; i++)
         printf("before> a[%d] = %.0f, b[%d] = %.0f\n", i, a[i], i, b[i]);    }
   }
   if (nrank == 0) {
      MPI_Send(a, BUF_SIZE, MPI_DOUBLE, 1, 17,
   MPI_COMM_WORLD);
      MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 1, 19,
   MPI_COMM_WORLD, &status);
   }
   else if (nrank == 1) {
      //MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 0, 17,
   MPI_COMM_WORLD, &status);
      MPI_Send(a, BUF_SIZE, MPI_DOUBLE, 0, 19,
   MPI_COMM_WORLD);
      MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 0, 17,
   MPI_COMM_WORLD, &status);
   }
   printf("\n\n");
   if (nrank == ROOT) {
      for (i=0; i<10; i++)
         printf("after > a[%d] = %.0f, b[%d] = %.0f\n", i, a[i], i, b[i]);
   }

   MPI_Finalize();
   printf("\n");
   return 0;
```

```
$ mpicc  -o  deadlock_blocking  –Wall  deadlock_blocking.c
$ mpirun  -np  2  –hostfile  hosts  ./deadlock_blocking
```

Example : deadlock_blocking.f90

```
! Example name : deadlock_blocking.f90
! if buf_size > 1500, 1600 is deadlock
! ----------------------------------------------------------
PROGRAM  deadlock_blocking
INCLUDE  'mpif.h'
    INTEGER  nrank, nprocs, i, ierr
    INTEGER,  PARAMETER :: buf_size = 1500
    DOUBLE PRECISION , DIMENSION(buf_size) :: a, b
    INTEGER status(MPI_STATUS_SIZE)

    CALL  MPI_INIT(ierr)
    CALL  MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)
    CALL  MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)

    IF (nrank == 0) THEN
                CALL MPI_SEND(a, buf_size,
MPI_DOUBLE_PRECISION, 1, 17, MPI_COMM_WORLD, ierr)
                CALL MPI_RECV(b, buf_size,
MPI_DOUBLE_PRECISION, 1, 19, MPI_COMM_WORLD,
status, ierr)
    ELSE IF (nrank == 1) THEN
                CALL MPI_SEND(a, buf_size,
MPI_DOUBLE_PRECISION, 0, 19, MPI_COMM_WORLD, ierr)
                CALL MPI_RECV(b, buf_size,
MPI_DOUBLE_PRECISION, 0, 17, MPI_COMM_WORLD,
status, ierr)
    ENDIF

    CALL MPI_FINALIZE(ierr)
END
```

```
$ mpicc  -o  deadlock_blocking.x  –Wall  deadlock_blocking.f90
$ mpirun  -np  2  –hostfile  hosts  ./deadlock_blocking.x
```

Example : deadlock_nonblocking.c

```
/* Example name : deadlock_nonblocking.c */
#include <stdio.h>
#include <mpi.h>

#define		BUF_SIZE	(1600)

int main(int argc, char *argv[])
{
	int nprocs, nrank, i;
	//MPI_Request req[2];
	//MPI_Status status[2];
	MPI_Request req;
	MPI_Status status;
	double a[BUF_SIZE], b[BUF_SIZE];

	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
	MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

	for (i=0; i<BUF_SIZE; i++)
		a[i] = i;

	if (nrank == 0) {
		MPI_Isend(a, BUF_SIZE, MPI_DOUBLE, 1, 17,
	MPI_COMM_WORLD, &req);
		MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 1, 19,
	MPI_COMM_WORLD, &status);
		//MPI_Wait(&req[0], &status[0]);
	}
	else if (nrank == 1) {
		MPI_Isend(a, BUF_SIZE, MPI_DOUBLE, 0, 19,
	MPI_COMM_WORLD, &req);
		MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 0, 17,
	MPI_COMM_WORLD, &status);
		//MPI_Wait(&req[1], &status[0]);
	}

	MPI_Wait(&req, &status);

	if (nrank == 0) {
		for (i=0; i<20; i++) {
			printf("b[%d]=%.0f ", i, b[i]);
			if (i != 0 && i%9 == 0) printf("\n");
		}
	}

	MPI_Finalize();
	printf("\n");

	return 0;
}
```

```
$ mpicc  -o  deadlock_nonblocking  –Wall  deadlock_nonblocking.c
$ mpirun  -np  2  –hostfile  hosts  ./deadlock_nonblocking
```

Example : deadlock_nonblocking.f90

```
! Example name : deadlock_nonblocking.f90
! if buf_size > 1500, 1600 is not deadlock
! ----------------------------------------------------------
PROGRAM  deadlock_nonblocking
INCLUDE  'mpif.h'
    INTEGER  nrank, nprocs, i, ierr
    INTEGER, PARAMETER :: buf_size = 1600
    DOUBLE  PRECISION , DIMENSION(buf_size) :: a, b
    INTEGER  status(MPI_STATUS_SIZE), req

    CALL MPI_INIT(ierr)
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)
    CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)

    IF (nrank == 0) THEN
        CALL MPI_ISEND(a, buf_size, MPI_DOUBLE_PRECISION,
    1, 17, MPI_COMM_WORLD, req, ierr)
        CALL MPI_RECV(b, buf_size, MPI_DOUBLE_PRECISION,
    1, 19, MPI_COMM_WORLD, status, ierr)
    ELSE IF (nrank == 1) THEN
        CALL MPI_ISEND(a, buf_size, MPI_DOUBLE_PRECISION,
    0, 19, MPI_COMM_WORLD, req, ierr)
        CALL MPI_RECV(b, buf_size, MPI_DOUBLE_PRECISION,
    0, 17, MPI_COMM_WORLD, status, ierr)
    ENDIF

    CALL  MPI_WAIT(req, status, ierr)

    CALL  MPI_FINALIZE(ierr)
END
```

```
$ mpicc  -o  deadlock_nonblocking .x –Wall  deadlock_nonblocking.f90
$ mpirun  -np  2  –hostfile  hosts  ./deadlock_nonblocking.x
```

MPI_Barrier

C	`int MPI_Barrier(MPI_Comm comm)`
Fortran	`MPI_BARRIER(comm, ierr)`

- Prevent the further progression of the processes till all the processes in the communicator will call MPI_Barrier

MPI_Barrier Example

Fortran

```fortran
PROGRAM barrier
IMPLICIT NONE
INCLUDE 'mpif.h'

        INTEGER nRank, nProcs, iErr

        CALL MPI_Init(iErr)
        CALL MPI_Comm_size(MPI_COMM_WORLD, nProcs, iErr)
        CALL MPI_Comm_rank(MPI_COMM_WORLD, nRank, iErr)

        IF (nRank == 0) THEN
                WRITE (*, *) '############'
                CALL SLEEP(3)
        END IF

        CALL MPI_Barrier(MPI_COMM_WORLD, iErr)

        WRITE (*, *) 'nRank : ', nRank, 'Hello World.'

        CALL MPI_FINALIZE(iErr)
END
```

$ mpif90 –o barrier.x barrier.f90
$ mpirun –np 3 –hostfile hosts ./barrier.x

C

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

int main (int argc, char *argv[])
{
    int nRank, nProcs;

    MPI_Init(&argc, &argv);                    // MPI start
    MPI_Comm_rank(MPI_COMM_WORLD, &nRank);     // Get current
     processor rank id
    MPI_Comm_size(MPI_COMM_WORLD, &nProcs); // Get number of
     processors

    if (nRank == 0)
    {
        printf(" ############₩n");
        sleep(3);
    }

    //  MPI_Barrier(MPI_COMM_WORLD);

    printf("nRank(%d) : Hello World.₩n", nRank);

    MPI_Finalize();                            // MPI End

    return 0;
}
```

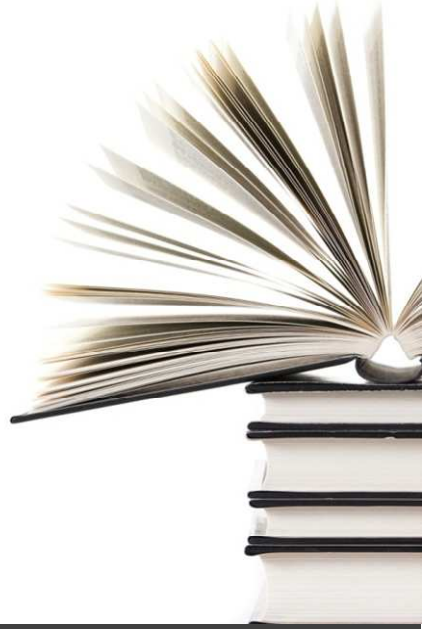
$ mpicc –o barrier barrier.c
$ mpirun –np 3 –hostfile hosts ./barrier

MPI Timing

- MPI Wtime() function returns the wall clock time
  - double s_time, e_time, t_time
  - MPI_Barrier(MPI_COMM_WORLD)
  - s_time = MPI_Wtime()
  - ..
  - ..
  - MPI_Barrier(MPI_COMM_WORLD)
  - e_time = MPI_Wtime()
  - t_time = e_time - s_time

Break!!

Advanced Exercise

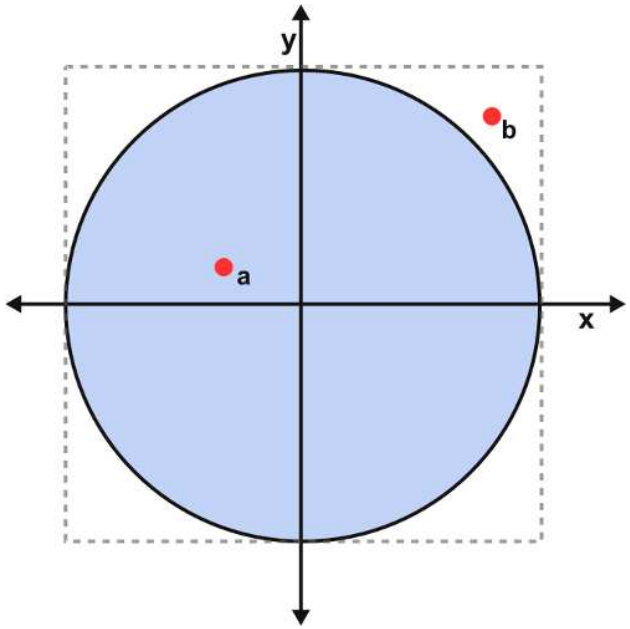


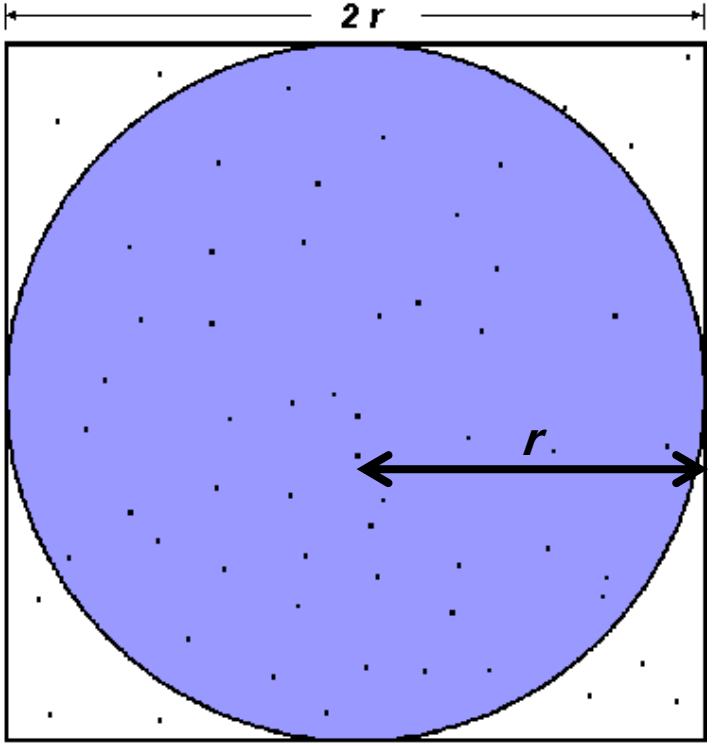
Advanced Lab #1

Advanced Lab #1 PI (Monte Carlo Simulation)

- <Problem>
  - Monte carlo simulation
  - Random number use
  - ***PI = 4 x Ac/As***

- <Requirement>
  - N's processor(rank) use
  - P2p communication





$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

Advanced Lab #1 PI MC Simulation – C (1/3)

```
/*
     Example Name    : pi_monte.c
     Compile         : $ mpicc -g -o pi_monte -Wall pi_monte.c
     Run             : $ mpirun -np 4 -hostfile hosts pi_monte
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <mpi.h>

#define   SCOPE     100000000

/*
** PI (3.14) Monte Carlo Method
*/
int main(int argc, char *argv[])
{
     int nProcs, nRank, proc, ROOT = 0;
     MPI_Status status;
     int nTag = 55;

     int i, nCount = 0, nMyCount = 0;
     double x, y, z, pi, z1;
```

Advanced Lab #1 PI MC Simulation – C (2/3)

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

srand(time(NULL)*(nRank+1));

for (i = 0; i < SCOPE; i++) {
      x = (double)rand() / (double)(RAND_MAX);
      y = (double)rand() / (double)(RAND_MAX);

      z = x*x + y*y;
      z1 = sqrt(z);

      if (z1 <= 1)        nMyCount++;
}

if (nRank == ROOT) {
      nCount = nMyCount;

      for (proc=1; proc<nProcs; proc++) {
                MPI_Recv(&nMyCount, 1, MPI_REAL, proc, nTag, MPI_COMM_WORLD,
                         &status);
                nCount += nMyCount;
      }

      pi = (double)4*nCount/(SCOPE * nProcs);
```

Advanced Lab #1 PI MC Simulation – C (3/3)

```
            printf("Processor %d sending results = %d to ROOT processor\n", nRank,
                   nMyCount);
            printf("\n # of trials (cpu#: %d, time#: %d) = %d, estimate of pi is %f\n",
                                 nProcs, SCOPE, SCOPE*nProcs, pi);

      }
      else {
            printf("Processor %d sending results = %d to ROOT processor\n", nRank,
                   nMyCount);
            MPI_Send(&nMyCount, 1, MPI_REAL, ROOT, nTag, MPI_COMM_WORLD);
      }

      printf("\n");

      MPI_Finalize();

      return 0;
}
```

Advanced Lab #1 PI MC Simulation - Compile & Run

```
$ mpicc -g -o pi_monte -Wall pi_monte.c
$ mpirun -np 4 -hostfile hosts pi_monte
Processor 2 sending results = 78541693 to ROOT processor
Processor 1 sending results = 78532183 to ROOT processor
Processor 3 sending results = 78540877 to ROOT processor
Processor 0 sending results = 78540877 to ROOT processor

 # of trials (cpu#: 4, time#: 100000000) = 400000000, estimate of pi is 3.141516
```

Advanced Lab #1 PI MC Simulation – Fortran (1/3)

```
! Example Name  : pi_monte.f90
! Compile   : $ mpif90 -g -o pi_monte.x -Wall pi_monte.f90
! Run       : $ mpirun -np 4 -hostfile hosts pi_monte.x

PROGRAM pi_monte
IMPLICIT NONE
INCLUDE 'mpif.h'
    INTEGER nRank, nProcs, iproc, iErr
    INTEGER :: ROOT = 0, scope = 100000000
    INTEGER :: i, nMyCount = 0, nCount = 0
    REAL :: x, y, z, pi, z1

    INTEGER status(MPI_STATUS_SIZE)

    CALL MPI_INIT(iErr)
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)
    CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)

    ! Get Random #
    CALL INIT_RANDOM_SEED(nRank+1)

    DO i=0, scope-1
        CALL RANDOM_NUMBER(x)
        CALL RANDOM_NUMBER(y)
```

Advanced Lab #1 PI MC Simulation – Fortran (2/3)

```
        z = x*x + y*y
        z1 = SQRT(z)

        IF (z1 <= 1) nMyCount = nMyCount + 1
    END DO

    IF (nRank == ROOT) THEN
        nCount = nMyCount

        DO iproc=1, nProcs-1
            CALL MPI_RECV(nMyCount, 1, MPI_INTEGER, iproc, 55, MPI_COMM_WORLD,
                          status, iErr)
            nCount = nCount + nMyCount
        END DO

        pi = 4 * REAL(nCount) / (scope * nProcs)

        WRITE (*, '(A, I2, A, I10, A)') 'Processor=', nRank, ' sending results = ',
               nMyCount, ' to ROOT process'
        WRITE (*, '(A, I2, A, F15.10)') '# of trial is ', nProcs, ' estimate of PI is ',
              pi

    ELSE
        CALL MPI_SEND(nMyCount, 1, MPI_INTEGER, ROOT, 55, MPI_COMM_WORLD, iErr)
        WRITE (*, '(A, I2, A, I10)') 'Processor=', nRank, ' sending results = ',
               nMyCount
    END IF
```

Advanced Lab #1 PI MC Simulation – Fortran (3/3)

```
    CALL MPI_FINALIZE(iErr)

CONTAINS
SUBROUTINE INIT_RANDOM_SEED(rank)
IMPLICIT NONE
INTEGER rank
    INTEGER :: i, n, clock
    INTEGER, DIMENSION(:), ALLOCATABLE :: seed

    CALL RANDOM_SEED(size = n)
    ALLOCATE(seed(n))

    CALL SYSTEM_CLOCK(COUNT=clock)

    seed = clock + 37 * (/ (i - 1, i = 1, n) /)
    seed = seed * rank * rank

    CALL RANDOM_SEED(PUT = seed)

    DEALLOCATE(seed)
END SUBROUTINE

END
```

Advanced Lab #1 PI MC Simulation - Compile & Run

```
$ mpif90 -g -o pi_monte.x -Wall pi_monte.f90
$ mpirun -np 4 -hostfile hosts pi_monte.x
Processor= 3    sending results =   78540734
Processor= 2    sending results =   78537818
Processor= 0 sending results =   78540734       to ROOT process
Processor= 1    sending results =   78542358

# of trial is  4 estimate of PI is    3.1415631771
```

Advanced Lab #2

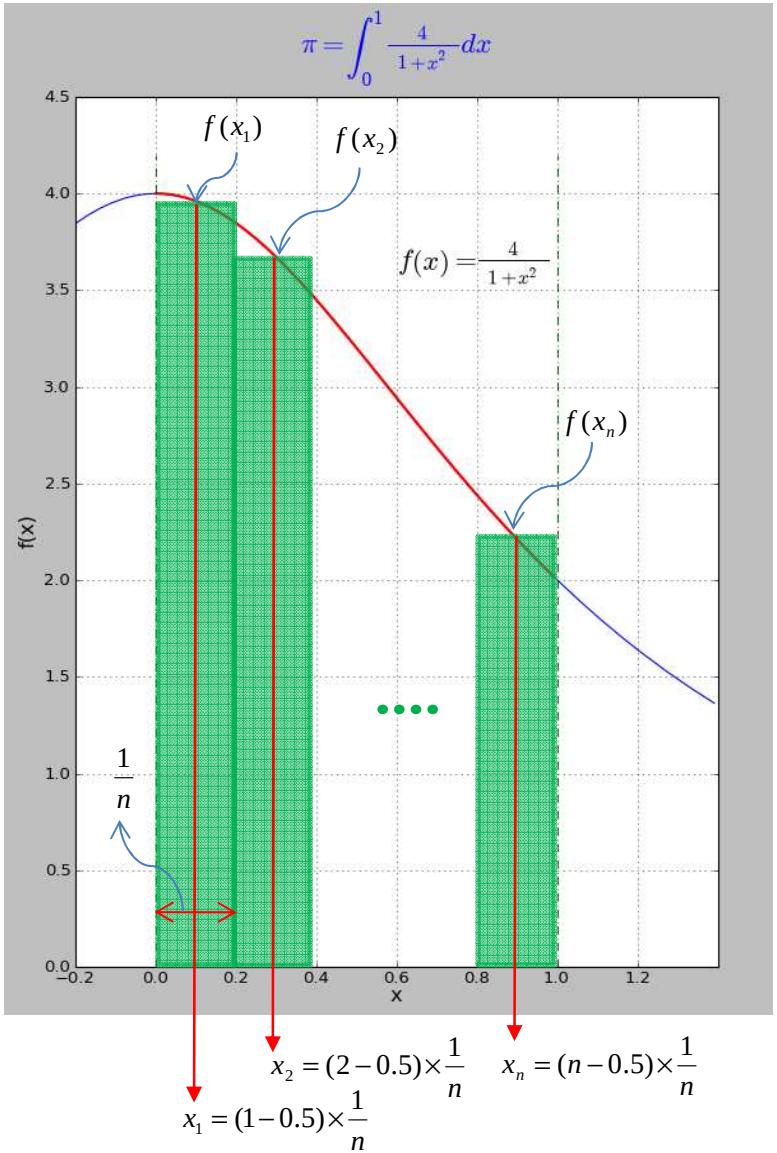
Advanced Lab #2 PI (Numerical Integration)

- <Problem>
  - Get PI using Numerical integration

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

- <Requirement>
  - Point to point communication

$$\pi \approx \sum_{i=1}^{n} \frac{4}{1+((i-0.5)\times\frac{1}{n})^2}\times\frac{1}{n}$$



Advanced Lab #2 PI Numerical Integration – C(1/3)

```
/*
     Example Name    : pi_integral.c
     Compile         : $ mpicc -g -o pi_integral -Wall pi_integral.c
     Run             : $ mpirun -np 4 -hostfile hosts pi_integral
*/

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

#define   SCOPE     100000000

int main(int argc, char *argv[])
{
     int i, n = SCOPE;
     double sum, step, pi, x, tsum, ROOT = 0;
     int nRank, nProcs;
     MPI_Status status;

     MPI_Init(&argc, &argv);
     MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
     MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

     if (nRank == ROOT) {
          for (i = 1; i < nProcs; i++)
```

Advanced Lab #2 PI Numerical Integration – C(2/3)

```
                MPI_Send(&n, 1, MPI_INT, i, 55, MPI_COMM_WORLD);
    }
    else
          MPI_Recv(&n, 1, MPI_INT, ROOT, 55, MPI_COMM_WORLD, &status);

    step = 1.0 / (double)n;

    sum = 0.0;
    tsum = 0.0;

    for (i = nRank; i < n; i += nProcs) {
          x = ((double)i-0.5)*step;
          sum = sum + 4 /(1.0 + x*x);
    }

    if (nRank == ROOT) {
          tsum = sum;

          for (i = 1; i < nProcs; i++) {
                MPI_Recv(&sum, 1, MPI_DOUBLE, i, 56, MPI_COMM_WORLD, &status);
                tsum = tsum + sum;
          }

          pi = step * tsum;
```

Advanced Lab #2 PI Numerical Integration – C(3/3)

```
        printf("---------------------------------------------\n");
        printf("PI = %.15f (Error = %E)\n", pi, fabs(acos(-1.0) - pi));
        printf("---------------------------------------------\n");
    }
    else
        MPI_Send(&sum, 1, MPI_DOUBLE, ROOT, 56, MPI_COMM_WORLD);

    MPI_Finalize();

    return 0;
}
```

Advanced Lab #2 PI Compile & Run

```
$ mpicc -g -o pi_integral -Wall pi_integral.c

$ mpirun -np 4 -hostfile hosts pi_integral
---------------------------------------------
PI = 3.141592673590217 (Error = 2.000042E-08)
---------------------------------------------
```

Advanced Lab #2 PI Numerical Integration – Fortran(1/2)

```
! Example Name      : pi_monte.f90
! Compile           : $ mpif90 -g -o pi_integral.x -Wall pi_integral.f90
! Run               : $ mpirun -np 4 -hostfile hosts pi_integral.x

PROGRAM pi_integral
IMPLICIT NONE
INCLUDE "mpif.h"

     INTEGER*8 ::              i, n = 1000000, ROOT = 0
     DOUBLE PRECISION          sum, step, mypi, x, tsum
     INTEGER                   nRank, nProcs, iErr
     INTEGER                   status(MPI_STATUS_SIZE)

     CALL MPI_INIT(iErr)
     CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)
     CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)

     IF (nRank .EQ. ROOT) THEN
          DO i=1, nProcs-1
                    CALL MPI_SEND(n, 1, MPI_INTEGER8, i, 55, MPI_COMM_WORLD, iErr)
          END DO
     ELSE
          CALL MPI_RECV(n, 1, MPI_INTEGER8, ROOT, 55, MPI_COMM_WORLD, status, iErr)
     END IF

     step = (1.0d0 / dble(n))
     sum = 0.0d0
```

Advanced Lab #2 PI Numerical Integration – Fortran(2/2)

```
      tsum = 0.0d0

      DO i=nRank+1, n, nProcs
            x = (dble(i) - 0.5d0) * step
            sum = sum + 4.d0 / (1.d0 + x*x)
      END DO

      IF (nRank .EQ. ROOT) THEN
            tsum = sum

            DO i=1, nProcs-1
                        CALL MPI_RECV(sum, 1, MPI_DOUBLE_PRECISION, i, 56, MPI_COMM_WORLD,
                                      status, iErr)
                        tsum = tsum + sum
            END DO

            mypi = step * tsum
            WRITE (*, 400)
            WRITE (*, 100) mypi, dabs(dacos(-1.d0) - mypi)
            WRITE (*, 400)
100 FORMAT ('mypi = ', F17.15, ' (Error = ', E11.5, ')')
400 FORMAT ('----------------------------------------')
      ELSE
            CALL MPI_SEND(sum, 1, MPI_DOUBLE_PRECISION, ROOT, 56, MPI_COMM_WORLD, iErr)
      END IF

      CALL MPI_FINALIZE(iErr)
END
```

Advanced Lab #2 PI Compile & Run

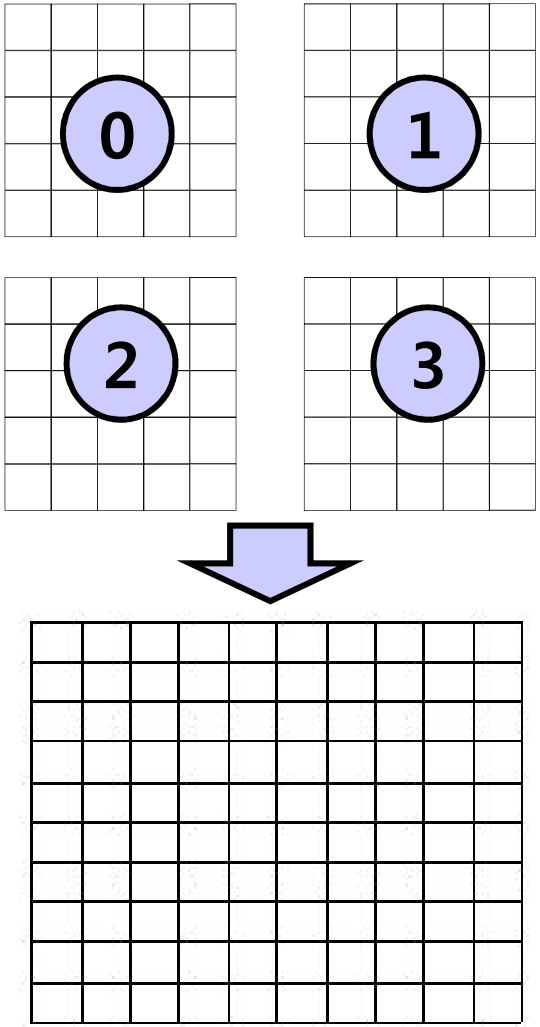
```
$ mpif90 -g -o pi_integral.x -Wall pi_integral.f90

$ mpirun -np 4 -hostfile hosts pi_integral.x
----------------------------------------
mypi = 3.141592653589903 (Error = 0.11013E-12)
----------------------------------------
```

Advanced Lab #3

Advanced Lab #3 2D Array Msg Communication

- <Problem>
  - Global 2D Array (10 X 10)
  - Local 2D Array (5 x 5)
  - ***Use 4 Processor***

- <Requirement>
  - Initialize the 2D array with number
    - 0# RANK : 101 ~ 125
    - 1# RANK : 201 ~ 225
    - 2# RANK : 301 ~ 325
    - 3# RANK : 401 ~ 425
  - Collecting four local 2d arrays(5x5) into one
  - Global 2d array(10x10)
  - ***Use MPI_Isend & MPI_Recv***
  - Print Global 2d array(10x10) in the screen



Advanced Lab #3 2D Array Msg Communication – C (1/4)

```
/*
     Example Name    : lab5.c
     Compile         : $ mpicc -g -o lab5 -Wall lab5.c
     Run             : $ mpirun -np 4 -hostfile hosts lab5
*/
#include <stdio.h>
#include <math.h>
#include <mpi.h>

#define   LOCAL_ROW 5
#define   LOCAL_COL 5

#define   GLOBAL_ROW 10
#define   GLOBAL_COL 10

#define MAX_PROCS 16

int main (int argc, char *argv[])
{
     int nRank, nProcs, ROOT = 0;
     int i, j, row, col;
     int nLocalGrid[LOCAL_ROW][LOCAL_COL];
     int nGlobalGrid[GLOBAL_ROW][GLOBAL_COL];
     MPI_Status status[MAX_PROCS];
     MPI_Request req[MAX_PROCS];
```

Advanced Lab #3 2D Array Msg Communication – C (2/4)

```
MPI_Init(&argc, &argv);                    // MPI start
MPI_Comm_rank(MPI_COMM_WORLD, &nRank);     // Get current processor rank id
MPI_Comm_size(MPI_COMM_WORLD, &nProcs);    // Get number of processors

// Init Local Array
for (i = 0; i < LOCAL_ROW; i++)
      for (j = 0; j < LOCAL_COL; j++)
                  nLocalGrid[i][j] = (nRank+1)*100 + i*LOCAL_COL + j + 1;
MPI_Barrier(MPI_COMM_WORLD);
printf("nRank : %d\n", nRank);
printf("-------------------\n");
for (i = 0; i < LOCAL_ROW; i++) {
      for (j = 0; j < LOCAL_COL; j++) {
                  printf("%d ", nLocalGrid[i][j]);
      }
      printf("\n");
}
printf("-------------------\n");

if (nRank != ROOT) {
      MPI_Isend(nLocalGrid[0], LOCAL_COL, MPI_INTEGER, ROOT, 11, MPI_COMM_WORLD, &req[0]);
      MPI_Isend(nLocalGrid[1], LOCAL_COL, MPI_INTEGER, ROOT, 12, MPI_COMM_WORLD, &req[1]);
      MPI_Isend(nLocalGrid[2], LOCAL_COL, MPI_INTEGER, ROOT, 13, MPI_COMM_WORLD, &req[2]);
      MPI_Isend(nLocalGrid[3], LOCAL_COL, MPI_INTEGER, ROOT, 14, MPI_COMM_WORLD, &req[3]);
      MPI_Isend(nLocalGrid[4], LOCAL_COL, MPI_INTEGER, ROOT, 15, MPI_COMM_WORLD, &req[4]);
```

Advanced Lab #3 2D Array Msg Communication – C (3/4)

```
        for (i = 0; i < 5; i++)
                MPI_Wait(&req[i], &status[i]);
    }
    else {
        for (i = 0; i < LOCAL_ROW; i++)
                for (j = 0; j < LOCAL_COL; j++)
                        nGlobalGrid[i][j] = nLocalGrid[i][j];

        for (i = 1; i < nProcs; i++) {
                row = i / (int)sqrt(nProcs);
                col = i % (int)sqrt(nProcs);

                MPI_Recv(&nGlobalGrid[row*LOCAL_ROW][col*LOCAL_COL], LOCAL_COL,
                        MPI_INT, i, 11, MPI_COMM_WORLD, &status[0]);
                MPI_Recv(&nGlobalGrid[row*LOCAL_ROW+1][col*LOCAL_COL], LOCAL_COL,
                        MPI_INT, i, 12, MPI_COMM_WORLD, &status[1]);
                MPI_Recv(&nGlobalGrid[row*LOCAL_ROW+2][col*LOCAL_COL], LOCAL_COL,
                        MPI_INT, i, 13, MPI_COMM_WORLD, &status[2]);
                MPI_Recv(&nGlobalGrid[row*LOCAL_ROW+3][col*LOCAL_COL], LOCAL_COL,
                        MPI_INT, i, 14, MPI_COMM_WORLD, &status[3]);
                MPI_Recv(&nGlobalGrid[row*LOCAL_ROW+4][col*LOCAL_COL], LOCAL_COL,
                        MPI_INT, i, 15, MPI_COMM_WORLD, &status[4]);
        }
```

Advanced Lab #3 2D Array Msg Communication – C (4/4)

```
        printf("nRank : %d (Total)\n", nRank);
        printf("-----------------------------------------\n");

        for (i = 0; i < GLOBAL_ROW; i++) {
                for (j = 0; j < GLOBAL_COL; j++) {
                        printf("%d ", nGlobalGrid[i][j]);
                        if (j == (GLOBAL_COL/2)-1)
                                printf(" |  ");
                }

                if (i == (GLOBAL_ROW/2)-1) {
                        printf("\n");
                        for (j = 0; j <= GLOBAL_COL; j++) {
                                printf(" -  ");
                        }
                }
                printf("\n");
        }
        printf("-----------------------------------------\n");
    }

    MPI_Finalize();
    return 0;
}
```

Advanced Lab #3 2D Array Compile & Run - C

```
$ mpicc -g -o lab lab5.c
$ mpirun -np 4 -hostfile hosts ./lab5

nRank : 0
-------------------
101 102 103 104 105
106 107 108 109 110
111 112 113 114 115
116 117 118 119 120
121 122 123 124 125
-------------------
nRank : 0 (Total)
-------------------------------------------
101 102 103 104 105  |  201 202 203 204 205
106 107 108 109 110  |  206 207 208 209 210
111 112 113 114 115  |  211 212 213 214 215
116 117 118 119 120  |  216 217 218 219 220
121 122 123 124 125  |  221 222 223 224 225
 -   -   -   -   -   -   -   -   -   -   -
301 302 303 304 305  |  401 402 403 404 405
306 307 308 309 310  |  406 407 408 409 410
311 312 313 314 315  |  411 412 413 414 415
316 317 318 319 320  |  416 417 418 419 420
321 322 323 324 325  |  421 422 423 424 425
-------------------------------------------
```

Advanced Lab #3 2D Array Msg – Fortran (1/4)

```
! Example Name : lab5.f90
! Compile      : $ mpif90 -g -o lab5.x -Wall lab5.f90
! Run          : $ mpirun -np 4 -hostfile hosts lab5.x

PROGRAM lab5
IMPLICIT NONE
INCLUDE 'mpif.h‘
     INTEGER nRank, nProcs, nProcs_Sqrt, iErr
     INTEGER status(16), req(16)
     INTEGER :: ROOT = 0, i, j, row, col
     INTEGER nLocalROW, nLocalCOL, nGlobalROW, nGlobalCOL
     PARAMETER (nLocalROW=5, nLocalCOL=5, nGlobalROW=10, nGlobalCOL=10)

     INTEGER nLocalGrid(0:nLocalROW-1, 0:nLocalCOL-1)
     INTEGER nGlobalGrid(0:nGlobalROW-1, 0:nGlobalCOL-1)

     CALL MPI_Init(iErr)
     CALL MPI_Comm_size(MPI_COMM_WORLD, nProcs, iErr)
     CALL MPI_Comm_rank(MPI_COMM_WORLD, nRank, iErr)

     DO i = 0, nLocalROW-1
          DO j = 0, nLocalCOL-1
                    nLocalGrid(i, j) = ((nRank+1) * 100) + i*nLocalCOL + j + 1
          END DO
     END DO
```

Advanced Lab #3 2D Array Msg – Fortran (2/4)

```
     CALL MPI_Barrier(MPI_COMM_WORLD, iErr)
     WRITE(*, 100) 'nRank: ', nRank
     WRITE(*, *) '------------------'
     DO i = 0, nLocalROW-1
          DO j = 0, nLocalCOL-1
                    WRITE (*, 200, advance='no') nLocalGrid(i, j)
          END DO
          WRITE(*, *)
     END DO
     WRITE(*, *) '------------------'
100 FORMAT(A, I2)
200 FORMAT(I3, X)

     IF (nRank /= ROOT) THEN
          CALL MPI_Isend(nLocalGrid(0, 0), nLocalROW, MPI_INTEGER, ROOT, 11,
                         MPI_COMM_WORLD, req(1), ierr)
          CALL MPI_Isend(nLocalGrid(0, 1), nLocalROW, MPI_INTEGER, ROOT, 12,
                         MPI_COMM_WORLD, req(2), ierr)
          CALL MPI_Isend(nLocalGrid(0, 2), nLocalROW, MPI_INTEGER, ROOT, 13,
                         MPI_COMM_WORLD, req(3), ierr)
          CALL MPI_Isend(nLocalGrid(0, 3), nLocalROW, MPI_INTEGER, ROOT, 14,
                         MPI_COMM_WORLD, req(4), ierr)
          CALL MPI_Isend(nLocalGrid(0, 4), nLocalROW, MPI_INTEGER, ROOT, 15,
                         MPI_COMM_WORLD, req(5), ierr)
```

Advanced Lab #3 2D Array Msg – Fortran (3/4)

```
        DO i = 1, 5
                CALL MPI_WAIT(req(i), status(i), ierr)
        END DO
    ELSE
        DO i = 0, nLocalCOL-1
                DO j = 0, nLocalROW-1
                        nGlobalGrid(j, i) = nLocalGrid(j, i)
                END DO
        END DO

        DO i = 1, nProcs-1
                nProcs_Sqrt = SQRT(REAL(nProcs))
                row = i / nProcs_Sqrt
                col = MOD(i, nProcs_Sqrt)

                CALL MPI_Recv(nGlobalGrid(row*nLocalROW, col*nLocalCOL), nLocalROW,
                        MPI_INTEGER, i, 11, MPI_COMM_WORLD, status(1), ierr)
               CALL MPI_Recv(nGlobalGrid(row*nLocalROW, col*nLocalCOL+1),nLocalROW,
                        MPI_INTEGER, i, 12, MPI_COMM_WORLD, status(2), ierr)
               CALL MPI_Recv(nGlobalGrid(row*nLocalROW, col*nLocalCOL+2),nLocalROW,
                        MPI_INTEGER, i, 13, MPI_COMM_WORLD, status(3), ierr)
               CALL MPI_Recv(nGlobalGrid(row*nLocalROW, col*nLocalCOL+3),nLocalROW,
                        MPI_INTEGER, i, 14, MPI_COMM_WORLD, status(4), ierr)
               CALL MPI_Recv(nGlobalGrid(row*nLocalROW, col*nLocalCOL+4),nLocalROW,
                        MPI_INTEGER, i, 15, MPI_COMM_WORLD, status(5), ierr)
        END DO
```

Advanced Lab #3 2D Array Msg – Fortran (4/4)

```
        WRITE(*, 100) 'nRank(Total): ', nRank
        WRITE(*, *) '----------------------------------------'

        DO i = 0, nGlobalROW-1
                DO j = 0, nGlobalCOL-1
                        WRITE (*, 200, advance='no') nGlobalGrid(i, j)
                END DO
                WRITE(*, *)
        END DO

        WRITE(*, *) '----------------------------------------'
    END IF

    CALL MPI_FINALIZE(iErr)

END
```

Advanced Lab #3 2D Array Msg – Compile & Run

```
$ mpif90 -g -o lab5.x lab5.f90
$ mpirun -np 4 -hostfile hosts ./lab5.x

nRank(Total):  0
 --------------------------------------
101 102 103 104 105 201 202 203 204 205
106 107 108 109 110 206 207 208 209 210
111 112 113 114 115 211 212 213 214 215
116 117 118 119 120 216 217 218 219 220
121 122 123 124 125 221 222 223 224 225
301 302 303 304 305 401 402 403 404 405
306 307 308 309 310 406 407 408 409 410
311 312 313 314 315 411 412 413 414 415
316 317 318 319 320 416 417 418 419 420
321 322 323 324 325 421 422 423 424 425
 --------------------------------------
```

Summary

- MPI is best for distributed-memory parallel systems
- There are several MPI implementations
- The MPI library is large but the core is small
- MPI Communication:
  - Point-to-point, blocking
  - Point-to-point, non-blocking
  - Collective

Core MPI

- 11 Out of 125 Total Functions
- Program startup and shutdown
  - `MPI_Init, MPI_Finalize`
  - `MPI_Comm_size, MPI_Comm_rank`
- Point-to-point communication
  - `MPI_Send, MPI_Recv`
- Non-blocking communication
  - `MPI_Isend, MPI_Irecv, MPI_Wait`
- Collective communication
  - `MPI_Bcast, MPI_Reduce`

Course Outline (KISTI)

- Part 1 – Introduction
  - Basics of Parallel Computing
  - Six-function MPI
  - Basic Communication
  - Point-to-Point Communications
  - Non-blocking Communication
- Part 2 – High-Performance MPI
  - Review of MPI Basics
  - Collective Communication
  - User-defined datatypes
  - Topologies
  - The Rest of MPI

Q&A

Thank you!