Parallel Programming with MPI
MPI I Tutorial/Exercise (1)

It is God's privilege to conceal things, but the kings' pride is to research them.

(Proverbs 25:2; ascribed to King Solomon of Israel, BC 1000)

The 2nd KIAS CAC Summer School

June 30-July 2, 2011

Joo, Keehyoung (주기형) (newton@kias.re.kr)
Center for Advanced Computation (CAC)
Korea Institute for Advanced Study

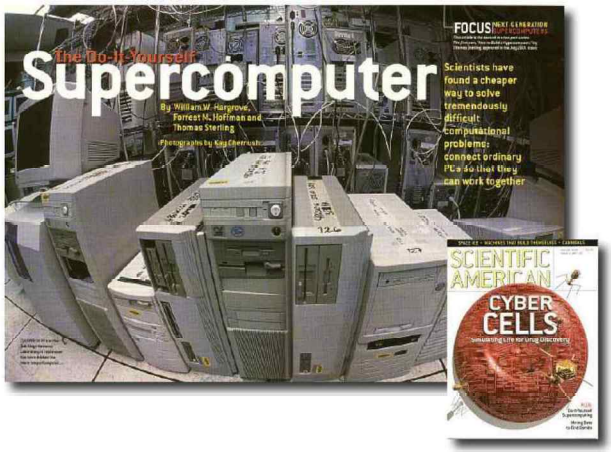
Contents (MPI I)

- Background and Getting Started (1 hour)
- MPI Basic (1 hour)
- Intermediate MPI (1 hour)
- Applications / Comments (1 hour)

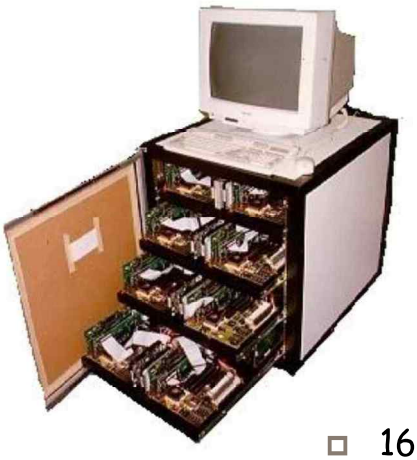
Scientific American : The Do-It-Yourself Supercomputer (August 2001)

Scientists have found a cheaper way to solve tremendously difficult computational problems:

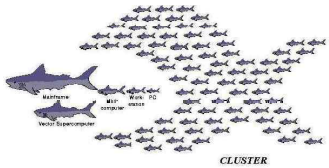
connect ordinary PCs so that they can work together.



The First Beowulf cluster



- Thomas Sterling and Donald Becker CESDIS, Goddard Space Flight Center
- Summer 1994: built an experimental cluster
- Called their cluster "Beowulf"

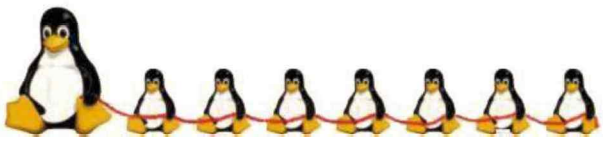


- 16 x 486DX4, 100MHz processors
- 16MB of RAM each, 256MB in total
- Channel bonded Ethernet (2 x 10Mbps)
- Not that different from our cluster system

Parallel Programming with MPI I

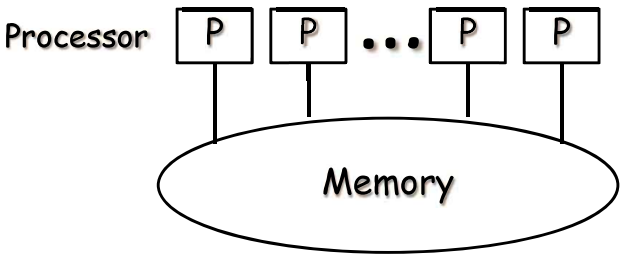
Background and Getting Started (1 hour)

- Parallel Computing System
- Parallel Programming Concept
- What is MPI?
- Hello World !



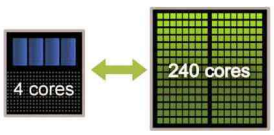
Parallel Computing Systems

- Multi Processor System (MP)



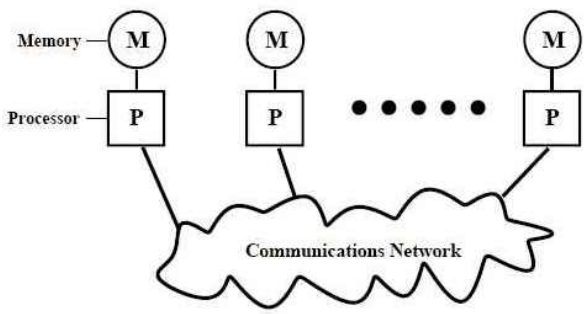
*Limted Scalability

*High Speed communication



GPU Computing

- Cluster System connected in Network



*High performance

*Large capacity

*High availability

*Incremental growth

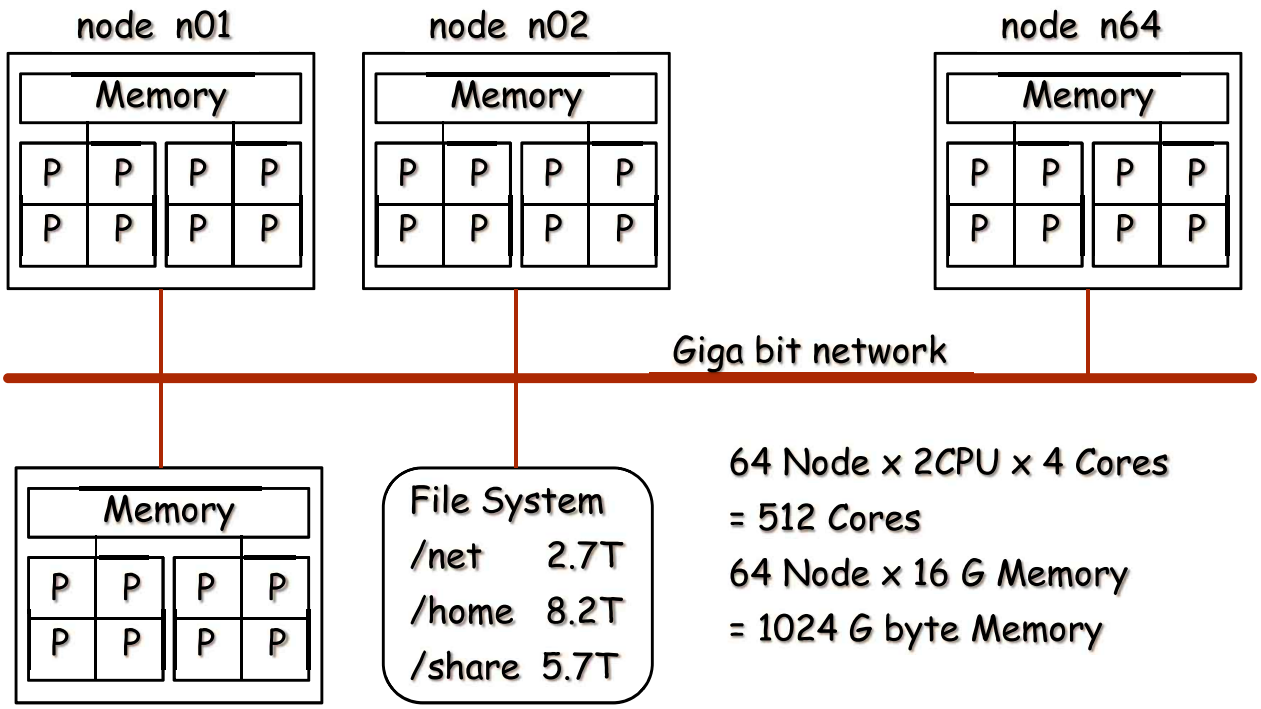
*Network dependent communication

Network Comparison

	Fast Ethernet	Gigabit Ethernet	Myrinet	Infiniband
Latency	~100μs	~100 μs	~2 μs	~0.5 μs ~2 μs (mpi)
Bandwidth	~100Mbps peak	~1Gbps peak	~1.98Gbps real	~10 Gbyte/s real
Price	~50,000 W	~200,000 W	~800,000 W	~ ???
KIAS		life, fold, etc	quest	

- Normal PC Cluster rely on TCP/IP
  - Real/MPI Latency is worse than the hardware latency
  - Efficiency for short message is less than 50%
  - Depends on Driver and NIC interaction

KIAS fold/life cluster system



64 Node x 2CPU x 4 Cores
= 512 Cores
64 Node x 16 G Memory
= 1024 G byte Memory

CPU: intel Xeon E5410 2.33 GHz

Login system life (life.kias.re.kr)

Exercise : Login

Windows :
Xmanager

Linux or UNIX machines: Terminal Programs

% ssh guest@fold.kias.re.kr

% ssh -X guest@fold.kias.re.kr

Password:

kias!guest

Exercise : Linux Commands

% less /etc/motd

% less /proc/cpuinfo

% pestat

% df -h

% free

% ps ax

% pwd

% cat /etc/passwd

% echo $SHELL

% ls -l

% man ls

% ls -al

% less .bashrc

% cd MPI1

% ls -l

% cp -r examples newton

% cd newton

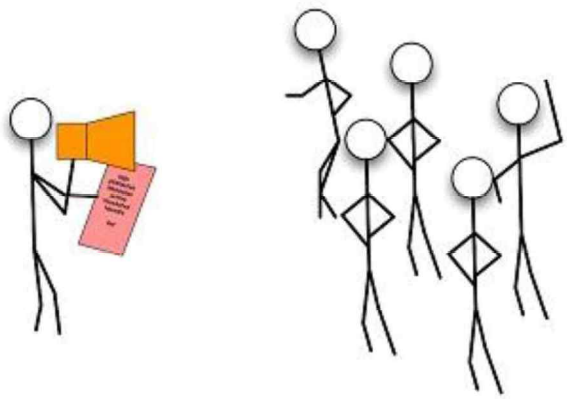
Compilers and Debuggers (serial)



- Languages : Fortran, C, C++, python, Java, etc

- GNU *http://www.gnu.org/*
  - % gcc
  - % g++
  - % gfortran
  - % gdb
- PGI *http://www.pgroup.com/*
  - % pgcc
  - % pgCC
  - % pgf77
  - % pgf90
  - % pgdbg
- Intel *http://developer.intel.com*
  - % icc
  - % ifort
  - % idb
- 실습
  - % cd MPI1/single
  - % gcc hello.c
  - % gfortran hello.f90

Parallel Computing & programming



- Separate workers or processes
- Interact by exchanging information

What is MPI ?

- M P I = Message Passing Interface
- Powerful, efficient, and *portable* way to express parallel *programs*
- Standard (specification)
  - Many implementations
  - Collection of communication library (subroutines)

- Two phases and… :
  - MPI 1: Traditional message-passing
  - MPI 2: Remote memory, parallel I/O, dynamic process
  - MPI 3: ???

Recommended Reading

- IBM RedBook
  http://www.redbooks.ibm.com/abstracts/sg245380.html
- MPI를 이용한 병렬 프로그래밍 (KISTI)
- MPI 병렬프로그래밍 (이홍석 et. al. , 어드북스)
- LAM MPI Tutorial
  http://www.lam-mpi.org/tutorials/nd/
- WWW
  http://www.cs.usfca.edu/mpi/
  http://www.citutor.org/ Education site (Search MPI)

- Use Google ! with keyword "MPI programming", "Parallel Programming", …

Message Passing Libraries

- OpenMPI
  - http://www.open-mpi.org/
  - Very easy to use !
- MPICH1 and 2
  - http://www.mcs.anl.gov/research/projects/mpich2/
  - Easy to use
  - High portability

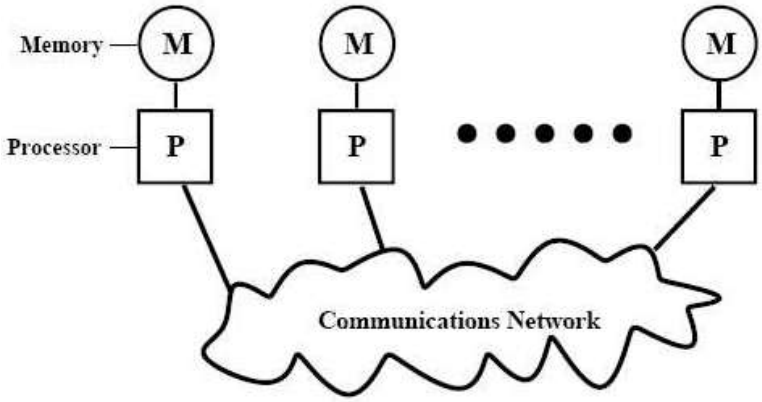
- LAM MPI
  - http://www.lam-mpi.org/
  - High availability

Math library, Parallel Library

- Linpack/Lapack, ScaLapack
- Atlas (Automatic Tuned Linear Algebra software)
- Goto Library
- MKL (intel Math Kernel Library)
- GSL (GNU Scientific Library)
- FFT (Fast Fourier Transform)
- ETC …

Parallel Programming Concept

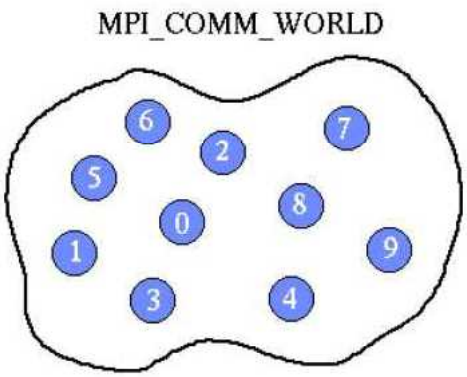
- Message Passing Interface



(1) All processes execute the same program
but each process is assigned a unique process id= 0,1,2,3,.....,(Nprocs-1)

(2) Each process performs its own tasks assigned according to its process id

Simple Example #1 (Hello World)




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

int main (int argc, char **argv)
{
    int myrank, nprocs;

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

    printf("Hello world! I am %d of %d\n",myrank, nprocs);

    MPI_Finalize ();
    return 0;
}
```

Simple Example #1 (Hello World)

Fortran 77 / 90

```
program hello

 include 'mpif.h'          ------------>   use mpi        ! Fortran 90
 integer myrank, nprocs
 integer ierr

 call MPI_INIT (ierr)
 call MPI_COMM_RANK (MPI_COMM_WORLD, myrank, ierr)
 call MPI_COMM_SIZE (MPI_COMM_WORLD, nprocs, ierr)

 print *, "Hello world! I am ",myrank," of ",nprocs

 call MPI_FINALIZE(ierr)

end program hello
```

Running your MPI programs

- Compile :

  % cd newton/
  % cat hello.c
  % mpicc –o hello hello.c
  % mpif90 –o hello hello.f90

- Running MPI Programs (OpenMPI):

  % mpiexec –np 8 hello
  % mpiexec –machinefile hosts –np 8 hello

- CPU time and Wall clock Time (Example #3 pi.f90)

  % mpif90 –o pi pi.f90
  % mpiexec –np 4 pi

Portable Batch System (PBS)

- Process and Resource Management
- Torque, openpbs
  - http://www.clusterresources.com/products/torque-resource-manager.php
- Commands (실습)
  - % qstat (Status)
  - % man qstat
  - % qstat –a
  - % qstat –u newton
  - % qstat –an
  - % qstat –q
  - % qstat –s
  - % qstat -f
  - % pestat (Node Status)
  - % qsub
  - % qdel
- PBS script (실습과 함께)

PBS Scripts

```
#!/bin/sh
### Job name
#PBS -N JOB_NAME
### Declare job non-rerunable
#PBS -r n
### Output files
#PBS -j oe
### Mail to user
#PBS -m ae
### Queue name (n2, n4, n16, n32, n64)
#PBS -q n32
### Walltime limit (hh:mm:ss)
#PBS -l walltime=168:00:00

# This job's working directory
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`
# Define number of processors
NPROCS=`wc -l < $PBS_NODEFILE`
```

```
## your parallel job

mpiexec -np $NPROCS a.out
```

```
* PBS interactive mode
% qsub –I –q n8 –N job_name
% qsub –I –q n16 –N job_name

% echo $PBS_O_WORKDIR
% echo $PBS_NODEFILE
% cat $PBS_NODEFILE
% export | grep PBS

* PBS commands
% cat start.mat
% qsub start.mat
% qdel 57480
```

Learning MPI

- MPI is large (about 125 functions)
  - MPI's extensive functionality requires many functions
  - Number of functions not necessarily a measure of complexity
- MPI is small (6 functions + sum of them~)
  - Many parallel programs can be written with just 8 basic functions
- MPI is just right
  - One can access flexibility when it is required
  - One need not master all parts of MPI to use it.

Basic MPI Functions (C)

- Init

```
MPI_Init(&argc, &argv)
MPI_Comm_size(MPI_COMM_WORLD,&nprocs)
MPI_Comm_rank(MPI_COMM_WORLD,&rank)
```

- Message Passsing

```
MPI_Send(...)          point to point communication
MPI_Recv(...)

MPI_Bcast(...)         collective communication
MPI_Reduce(...)
```

- Finalize

```
MPI_Finalize()
```

Basic MPI Functions (Fortran)

- Init

```
Call MPI_Init(ierr)
Call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierr)
```

- Message Passsing

```
Call MPI_Send(…)          point to point communication
Call MPI_Recv(…)

Call MPI_Bcast(…)         collective communication
Call MPI_Reduce(…)
```

- Finalize

```
Call MPI_Finalize(ierr)
```

Types of parallel programming

- Coarse-Grained Parallel Applications:
  Monte Carlo Simulations, 3D Graphics rendering,
  Database Search Problems (Web Search Engine),
  Embarrassingly Parallel Programs.

- Medium-grained Parallel Applications:
  Divide & Conquer style analysis,
  Domain decomposition Methods.

- Fine-grained parallel applications:

Parallel Mandelbrot set

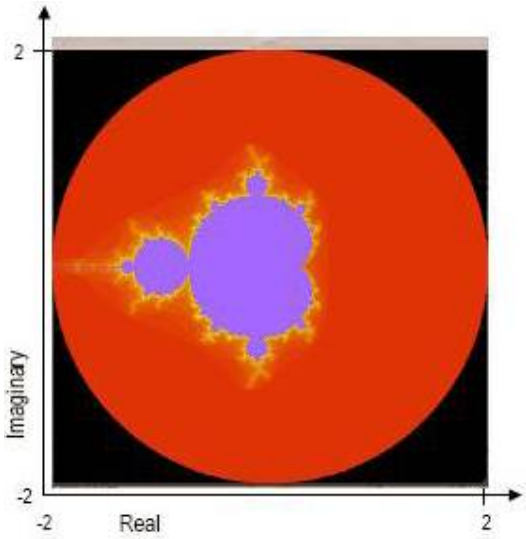
- A Mandelbrot set is a set of points in the complex plane
- Computed by the recursive function

$$z_{k+1} = z_k^2 + c$$

c is a complex number which gives the position of a point in the complex plane

z_{k+1} is the *(k+1)*th iteration of the complex number $z = a+bi$

the initial value for *z* is 0



실습:

% cd Mandel

% ./run.sh 8

Povray on Linux Cluster http://www.povray.com

- POVRAY (Persistence Of Vision RAYtracer) is a high-quality tool for creating three-dimensional graphics.
- Raytraced images are publication-quality and 'photo-realistic'.
- but are computationally expensive so that large images can take many hours to create.

실습:

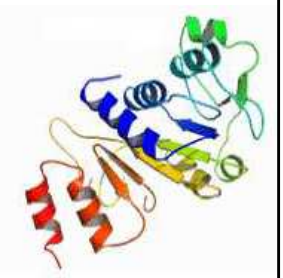
% cd povray/example

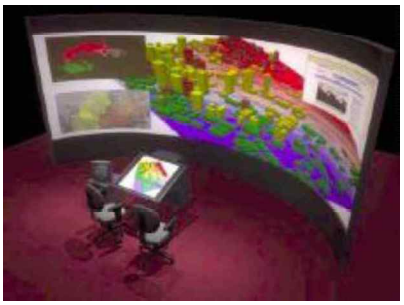
% ./povray.single woodbox.pov

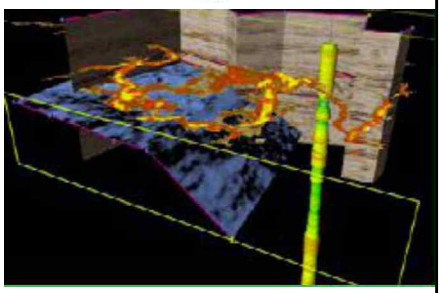
% ./run.sh

Other Applications…

- Visualize / Weather
- Media Encoding (Image compress, DVD, MP3, ETC…)
- Internet Server (Web, Database)
- Science, etc…



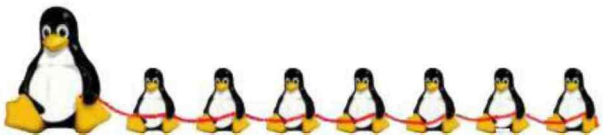




Parallel Programming with MPI I

MPI Basics (1 hour)

- Point to Point Communication (blocking Send/Recv)
- Some of collective communication



Basic MPI Functions (Fortran)

- Init

Call MPI_Init(ierr)
Call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierr)

- Message Passsing

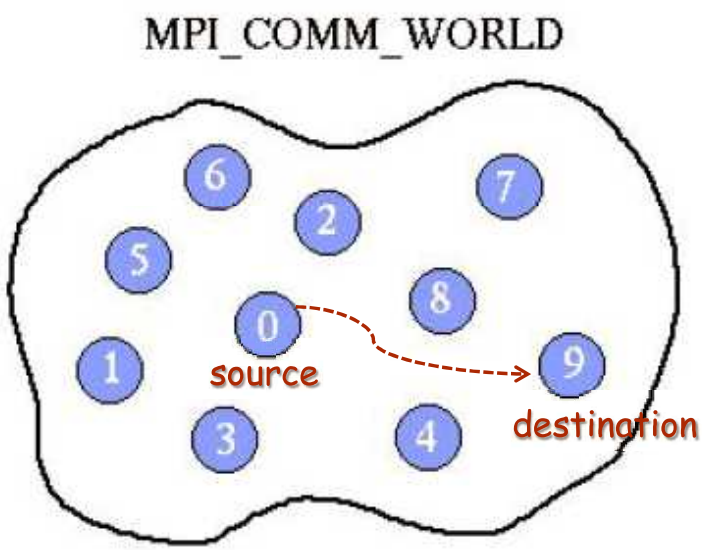
Call MPI_Send(…) point to point communication
Call MPI_Recv(…)

Call MPI_Bcast(…) collective communication
Call MPI_Reduce(…)

- Finalize

Call MPI_Finalize(ierr)

Point to Point Communication



Message

MPI Send / Recv

*Data:
buffer, count, datatype
*Envelope:
process identifier (src/dest)
message tag
communicator

Point to Point Communication (C)

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

- buf : Send buffer
- count : Number of sent elements
- datatype : MPI datatype

- dest : Destination Process ID
- tag : Message tag
- comm : Communicator

MPI_Recv(buf, count,datatype,source, tag, comm,&status);

- buf : Receive Buffer
- source : Source Process ID
- status : Receive status

Point to Point Communication (Fortran)

Call MPI_SEND(buf, count, datatype, dest, tag, comm, ierr);

- buf : Send buffer
- count : Number of sent elements
- datatype : MPI datatype

- dest : Destination Process ID
- tag : Message tag
- comm : Communicator

Call MPI_RECV(buf, count,datatype,source, tag, comm,status, ierr);

- buf : Receive Buffer
- source : Source Process ID
- status : Receive status

Fortran

```
program hello2
   use mpi
   implicit none
   integer irank, nprocs, ierr, i
   integer tag
   integer istatus(MPI_STATUS_SIZE)
   character*100 msg

   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD,  irank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)

   tag = 10
   if (irank == 0) then
      write(msg,'(A,2x,I2)') 'Hello! node 2'
      idest = 1
      call MPI_SEND(msg,100, MPI_CHARACTER, idest, tag, MPI_COMM_WORLD, ierr)
   else
      isrc  = 0
      call MPI_RECV(msg,100, MPI_CHARACTER, isrc , tag, MPI_COMM_WORLD, istatus, ierr)
      write(6,'(A)') trim(msg)
      write(6,'(A)') 'Hi ~'
   endif

   call MPI_FINALIZE(ierr)
end program hello2
```

* Example (hello2.f90)

% mpicc hello2.f90

% mpiexec –np 2 ./a.out

% mpiexec –np 4 ./a.out (dead lock?)

C

```
#include <stdio.h>
#include "mpi.h"
// hello2
int main (int argc, char **argv)
{
    int rank, size;
    int tag;
    int dest, src;
    char msg[100];
    MPI_Status status;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);

    tag = 123;
    if (rank == 0) {
        sprintf(msg,"Hello! node 2");
        dest= 1;
        MPI_Send(msg,100,MPI_CHAR, dest, tag, MPI_COMM_WORLD);
    }
    else {
        src = 0;
        MPI_Recv(msg,100,MPI_CHAR, src , tag, MPI_COMM_WORLD,&status);
        printf("%s\n", msg);
        printf("=> Hi ~\n");
    }

    MPI_Finalize ();
}
```

* Example (hello2.c)

% mpicc hello2.f90

% mpiexec –np 2 ./a.out

% mpiexec –np 4 ./a.out (dead lock?)

Fortran

```
program hello3
   use mpi
   implicit none
   integer ierr, irank, nprocs
   integer tag, num, next, prev
   integer istatus(MPI_STATUS_SIZE)
   integer i
   character*100 msg

   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD,  irank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)

   tag = 10
   if (irank == 0) then
      do i=1, nprocs-1
        write(msg,'(A,2x,I2)') 'Hello! node',i
        call MPI_SEND(msg,100, MPI_CHARACTER, i, tag, MPI_COMM_WORLD, ierr)
      end do
   else
      call MPI_RECV(msg,100, MPI_CHARACTER, 0, tag, MPI_COMM_WORLD, istatus, ierr)
      write(6,'(A)') trim(msg)
   endif

   call MPI_FINALIZE(ierr)
end program hello3
```

* hello3: for all nodes

C

```
#include <stdio.h>
#include "mpi.h"
// hello4
int main (int argc, char **argv)
{
    int rank, size;
    int tag;
    int dest, src, man;
    char msg[100];
    MPI_Status status;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);

    tag = 123;
    if (rank == 0) {
        for (dest=1; dest < size; dest++) {
            sprintf(msg,"Hello! node %d",dest);
            MPI_Send(msg,100,MPI_CHAR, dest, tag, MPI_COMM_WORLD);
        }
    }
    else {
        src = 0;
        MPI_Recv(msg,100,MPI_CHAR, src , tag, MPI_COMM_WORLD,&status);
        printf("%s\n", msg);
        man = status.MPI_SOURCE;
        printf("=> Hi, master %d ~\n",man);
    }

    MPI_Finalize ();
}
```

* hello4: with MPI_SOURCE

MPI Data Type

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

Call MPI_Recv (buf, count, datatype,source, tag, comm, status)

C Data Types		Fortran Data Types	
MPI_CHAR	signed char	MPI_CHARACTER	character(1)
MPI_SHORT	signed short int		
MPI_INT	signed int	MPI_INTEGER	integer
MPI_LONG	signed long int		
MPI_UNSIGNED_CHAR	unsigned char		
MPI_UNSIGNED_SHORT	unsigned short int		
MPI_UNSIGNED	unsigned int		
MPI_UNSIGNED_LONG	unsigned long int		
MPI_FLOAT	float	MPI_REAL	real
MPI_DOUBLE	double	MPI_DOUBLE_PRECISION	double precision
MPI_LONG_DOUBLE	long double		
		MPI_COMPLEX	complex
		MPI_DOUBLE_COMPLEX	double complex
		MPI_LOGICAL	logical
MPI_BYTE	8 binary digits	MPI_BYTE	8 binary digits
MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack	MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack

Retrieving further information

- Status is a data structure allocated in the user's program.
- in C

```
MPI_Recv(..., MPI_ANY_SOURCE, MPI_ANY_TAG, ..., &status);
tag = status.MPI_TAG
src = status.MPI_SOURCE
MPI_Get_count (&status, datatype, &cnt);
```

- in Fortran

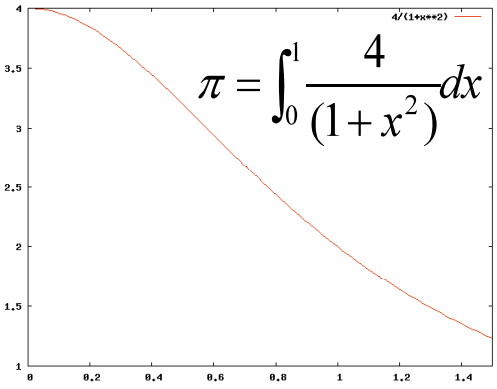
```
call MPI_RECV(...,MPI_ANY_SOURCE, MPI_ANY_TAG,...status,...)
itag = status (MPI_TAG)
isrc = status (MPI_SOURCE)
call MPI_GET_COUNT(status,datatype,icnt, ierr)
```

* retrieve.f90

Fortran:
retriev.f90

```
program main
 use mpi

 integer irank, nproc, itag, i, ierr
 integer isrc, idst, source
 integer my_src, my_tag, my_cnt
 integer istatus (MPI_STATUS_SIZE)
 real*8  data(10)

 call MPI_INIT (ierr)
 call MPI_COMM_RANK (MPI_COMM_WORLD, irank, ierr)
 call MPI_COMM_SIZE (MPI_COMM_WORLD, nproc, ierr)

 print *, 'Process ', irank, '  of ', nproc, ' is alive'

 if (irank == 0) then
     do i = 1, 10
        data(i) = i
     enddo
     idst = 1
     call MPI_SEND (data,10,MPI_REAL8, idst, 2011,MPI_COMM_WORLD,ierr)
 else if (irank == 1) then
     itag    = MPI_ANY_TAG
     source = MPI_ANY_SOURCE
     call MPI_RECV (data,10,MPI_REAL8, source, itag, MPI_COMM_WORLD,istatus,ierr)

     my_src = istatus(MPI_SOURCE)
     my_tag = istatus(MPI_TAG)

     call MPI_GET_COUNT (istatus,MPI_REAL8, my_cnt, ierr)

     print *, 'status info: source = ', my_src
     print *, '             tag    = ', my_tag
     print *, '             count  = ', my_cnt
 else
     continue
 endif

 call MPI_FINALIZE (ierr)
end program main
```

* PI Calculation



$$\pi = \int_0^1 \frac{4}{(1+x^2)} dx$$

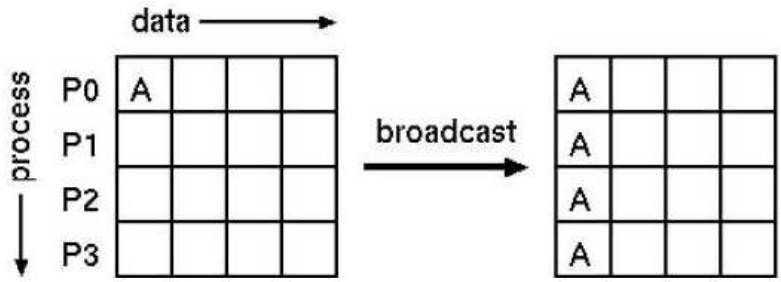
```
program pi_calculation
 use mpi

 real*8, parameter :: PI25DT = 3.14159265358979323846264338327950288419716939937510d0
 real*8  pi_node, pi, h, sum, x, f, a
 integer n, irank, nprocs, i, rc
 integer istat(MPI_STATUS_SIZE)
 real*4  time_start, time_end, elapsed_time, total_time
 real*8  stime, etime

 ! function to integrate
 f(x) = 4.d0 / (1.d0 + x*x)

 call MPI_INIT( ierr )
 call MPI_COMM_RANK( MPI_COMM_WORLD, irank, ierr )
 call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )

 if ( irank == 0 ) then
     write(6,'(A,$)') 'Enter the number of intervals = '
     read *, n
     do i=1, nprocs-1
       call MPI_SEND(n,1,MPI_INTEGER,i,123,MPI_COMM_WORLD,ierr)
     enddo
 else
     call MPI_RECV(n,1,MPI_INTEGER,0,123,MPI_COMM_WORLD,istat,ierr)
 endif

 stime= MPI_WTIME ()
 call cpu_time (time_start)

 ! calculate the interval size and pi_node
 h     = 1.0d0/n
 sum   = 0.0d0

 do i = irank+1, n, nprocs
    x = h * (dble(i) - 0.5d0)
    sum = sum + f(x)
 enddo
 pi_node = h * sum

 call cpu_time(time_end)
 elapsed_time = time_end-time_start
```

* pi_comm.f90

```
! collect all the partial sums
if ( irank == 0 ) then
   pi = pi_node
   do i=1, nprocs-1
     call MPI_RECV(pi_node,1,MPI_REAL8,i,345,MPI_COMM_WORLD,istat,ierr)
     pi = pi + pi_node
     call MPI_RECV(elapsed_time,1,MPI_REAL,i,678,MPI_COMM_WORLD,istat,ierr)
     total_time = total_time + elapsed_time
   enddo
else
   call MPI_SEND(pi_node,1,MPI_REAL8,0,345,MPI_COMM_WORLD,ierr)
   call MPI_SEND(elapsed_time,1,MPI_REAL,0,678,MPI_COMM_WORLD,ierr)
endif

etime= MPI_WTIME ()

write(6, *) 'My elapsed time (',irank,') =',elapsed_time, ' seconds'
if (irank .eq. 0) then
    write(6,  *)
    write(6, 97) pi, abs(pi - PI25DT)
    97 format('PI is approximately: ', F18.16,'  Error is: ', F18.16)
    write(6, *) 'Wall Clock Time =', etime-stime, ' seconds'
    write(6, *) 'Total Cpu Time  =', total_time,  ' seconds'
    write(6, *) 'Speed up        =', total_time/(etime-stime)
    write(6, *) 'The number of Processors =', nprocs
endif

 call MPI_FINALIZE(ierr)
end program pi_calculation
```

Some of Collective Communication

- Broadcast



In C:

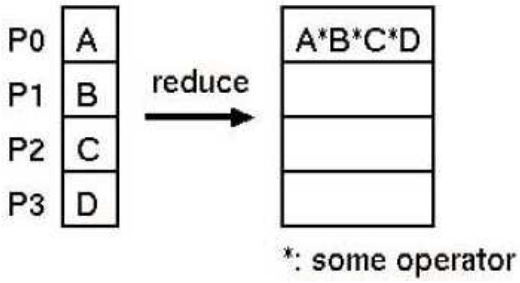
```
MPI_Bcast (&buf,count,datatype,root, comm);
```

In Fortran:

```
Call MPI_BCAST (buf,count,datatype,root, comm,ierr)
```

Some of Collective Communication

Reduce



In C:

```
MPI_Reduce (&sendbuf,&recvbuf,count,datatype,operator,root, comm);
```

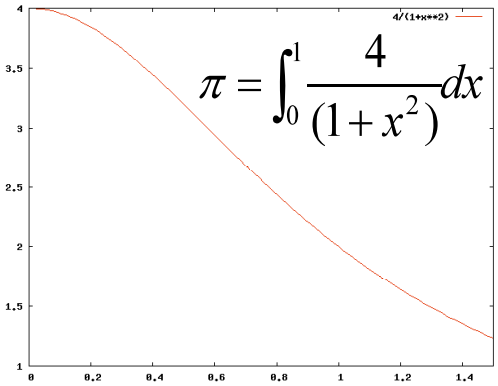
In Fortran:

```
Call MPI_BCAST (sendbuf,recvbuf,count,datatype,operator,root, comm,ierr)
```

Barrier : create a barrier synchronization

MPI_Barrier (comm); Call MPI_BARRIER(comm,ierr)

* PI Calculation



* pi_reduce.f90

```
program pi_calculation
  use mpi

  real*8, parameter :: PI25DT = 3.14159265358979323846264338d0
  real*8  pi_node, pi, h, sum, x, f, a
  integer n, irank, nprocs, i, rc
  integer istat(MPI_STATUS_SIZE)
  real*4  time_start, time_end, elapsed_time, total_time
  real*8  stime, etime

  ! function to integrate
  f(x) = 4.d0 / (1.d0 + x*x)

  call MPI_INIT( ierr )
  call MPI_COMM_RANK( MPI_COMM_WORLD, irank, ierr )
  call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )

  if ( irank == 0 ) then
      write(6,'(A,$)') 'Enter the number of intervals = '
      read *, n
  endif
  call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

  stime= MPI_WTIME ()
  call cpu_time (time_start)
```

```
if ( irank == 0 ) then
    write(6,'(A,$)') 'Enter the number of intervals = '
    read *, n
    do i=1, nprocs-1
      call MPI_SEND(n,1,MPI_INTEGER,i,123,MPI_COMM_WORLD,ierr)
    enddo
else
    call MPI_RECV(n,1,MPI_INTEGER,0,123,MPI_COMM_WORLD,istat,ierr)
endif
```

```
! calculate the interval size and pi_node
h    = 1.0d0/n
sum  = 0.0d0

do i = irank+1, n, nprocs
   x = h * (dble(i) - 0.5d0)
   sum = sum + f(x)
enddo
pi_node = h * sum

call cpu_time(time_end)
elapsed_time = time_end-time_start

! collect all the partial sums
call MPI_REDUCE(pi_node,pi,1,MPI_REAL8,MPI_SUM,0,MPI_COMM_WORLD,ierr)
call MPI_REDUCE(elapsed_time,total_time,1,MPI_REAL4,MPI_SUM,0,MPI_COMM_WORLD,ierr)

etime= MPI_WTIME ()

write(6, *) 'My elapsed time (',irank,') =',elapsed_time, ' seconds'
if (irank .eq. 0) then
    write(6,  *)
    write(6, 97) pi, abs(pi - PI25DT)
    97 format('PI is approximately: ', F18.16,'  Error is: ', F18.16)
    write(6, *) 'Wall Clock Time =', etime-stime, ' seconds'
    write(6, *) 'Total Cpu Time  =', total_time,  ' seconds'
    write(6, *) 'Speed up        =', total_time/(etime-stime)
    write(6, *) 'The number of Processors =', nprocs
endif

 call MPI_FINALIZE(ierr)
end program pi calculation
```

MPI Reduce Operators

MPI Reduction Operation		C Data Types	Fortran Data Type
MPI_MAX	maximum	integer, float	integer, real, complex
MPI_MIN	minimum	integer, float	integer, real, complex
MPI_SUM	sum	integer, float	integer, real, complex
MPI_PROD	product	integer, float	integer, real, complex
MPI_LAND	logical AND	integer	logical
MPI_BAND	bit-wise AND	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LOR	logical OR	integer	logical
MPI_BOR	bit-wise OR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer	logical
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double	real, complex,double precision
MPI_MINLOC	min value and location	float, double and long double	real, complex, double precision

Basic MPI Functions (Fortran)

- Init

Call MPI_Init(ierr)
Call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
Call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierr)

- Message Passsing

Call MPI_Send(…) point to point communication
Call MPI_Recv(…)

Call MPI_Bcast(…) collective communication
Call MPI_Reduce(…)

- Finalize

Call MPI_Finalize(ierr)

```
program workshare
 use mpi
 integer irank, nprocs, ierr
 integer n, istart, iend
 integer*8 isum, itotal

 call MPI_INIT (ierr)
 call MPI_COMM_RANK (MPI_COMM_WORLD, irank, ierr)
 call MPI_COMM_SIZE (MPI_COMM_WORLD, nprocs, ierr)
 if (irank == 0) then
    write(6,'(A,$)') ' Enter the number of your jobs = '
    read *, n
 endif
 call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call para_range(1,n, nprocs, irank, istart, iend)

 isum = 0
 do i= istart, iend
    isum = isum + i
 enddo
 write(6,10) 'I am node ', irank, ' my range = ', istart,iend, 'my sum =',isum
 10 format(A,I3,A,2I12,2x,A,I14)

 call MPI_REDUCE (isum,itotal,1,MPI_INTEGER8,MPI_SUM,0,MPI_COMM_WORLD,ierr)
 if (irank == 0) then
     write(6,'(A,2x,I14)') 'Total sum = ', itotal
 endif
 call MPI_FINALIZE(ierr)
end program workshare

subroutine para_range(n1, n2, nprocs, irank, ista, iend)
  integer n1,n2,nprocs,irank,ista,iend
  integer iwork1, iwork2
  iwork1 = (n2 - n1 + 1) / nprocs
  iwork2 = MOD(n2 - n1 + 1, nprocs)
  ista = irank * iwork1 + n1 + MIN(irank, iwork2)
  iend = ista + iwork1 - 1
  IF (iwork2 > irank) iend = iend + 1
end
```

* workshare.f90
-load balencing
-fine grain…

```c
#include <stdio.h>
#include "mpi.h"
#define MIN(a,b) ((a) < (b) ? (a) : (b))
// workshare
void para_range(int n1,int n2,int nprocs,int rank,int *start,int *end);

int main (int argc, char **argv)
{
    int rank, nprocs;
    int i, n, start, end;
    unsigned long int sum, total;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &nprocs);
    if (rank == 0) {
        printf(" Enter the number of your jobs = \n");
        scanf("%d",&n);
    }
    MPI_Bcast (&n,1,MPI_INT,0,MPI_COMM_WORLD);
    para_range(1,n,nprocs, rank, &start, &end);

    sum = 0;
    for (i=start; i<= end; i++) {
        sum += i;
    }
    printf("I am node %d  my range = %d  %d  my sum = %ld\n",
            rank, start, end, sum);

    MPI_Reduce(&sum,&total,1,MPI_UNSIGNED_LONG,MPI_SUM,0,MPI_COMM_WORLD);
    if (rank == 0) {
        printf("Total sum = %ld\n",total);
    }
    MPI_Finalize ();
}

void para_range(int n1,int n2,int nprocs,int rank,int *start,int *end){
    int iwork1, iwork2;
    iwork1 = (n2 - n1 + 1) / nprocs;
    iwork2 = (n2 - n1 + 1) % nprocs;
    *start  = rank * iwork1 + n1 + MIN (rank, iwork2);
    *end    = *start + iwork1 - 1 ;
    if (iwork2 > rank) *end = *end + 1;
}
```

* workshare.c

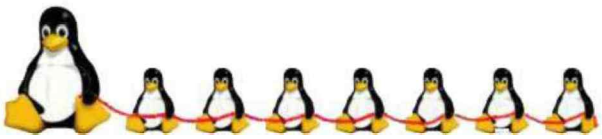
-load balencing

-fine grain...

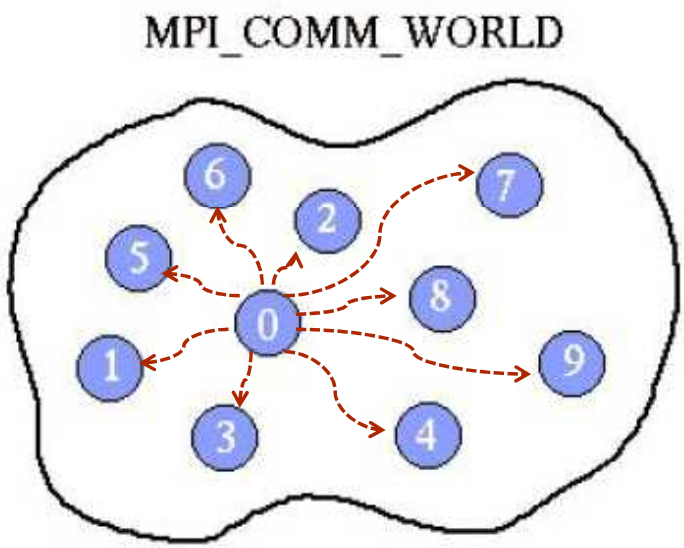
Parallel Programming with MPI I

Intermediate MPI and ... (2 hour)

- non-Blocking Send/Recv
- Collective Communications
- Applications / Comments



non-Blocking Send/Recv (Ex)



Non-blocking operations allow overlapping computation and communication.

Blocking/non-Blocking Send/Recv

In C :

MPI_request request;

- Blocking (Send/Recv)

```
MPI_Send (buf, count, datatype, dest, tag, comm);
MPI_Recv (buf, count,datatype,source, tag, comm, &status);
```

- non-Blocking (Isend/Irecv)

```
MPI_Isend (buf, count, datatype, dest, tag, comm, &request);
MPI_Irecv (buf, count,datatype,source, tag, comm, &request);

MPI_Wait (&request, &status);
```

* hello5, 6, 7

Blocking/non-Blocking Send/Recv

In Fortran :

integer request

- Blocking (Send/Recv)

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

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

- non-Blocking (Isend/Irecv)

  Call MPI_ISEND (buf, count, datatype, dest, tag, comm, request)

  Call MPI_IRECV (buf, count,datatype,source, tag, comm, request)

  Call MPI_WAIT (request, status)

* hello5, 6, 7

* hello5.c : MPI_Isend

```c
#include <stdio.h>
#include "mpi.h"
// hello5
int main (int argc, char **argv)
{
    int rank, size;
    int tag;
    int dest, src, man;
    char msg[100];
    MPI_Status status;
    MPI_Request req;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);

    tag = 123;
    if (rank == 0) {
        for (dest=1; dest < size; dest++) {
            sprintf(msg,"Hello! node %d",dest);
            MPI_Isend(msg,100,MPI_CHAR, dest, tag, MPI_COMM_WORLD,&req);
        }
    }
    else {
        src = 0;
        MPI_Recv(msg,100,MPI_CHAR, src , tag, MPI_COMM_WORLD,&status);
        printf("%s\n", msg);
        man = status.MPI_SOURCE;
        printf("=> Hi, master %d ~\n",man);
    }

    MPI_Finalize ();
}
```

* hello6.c

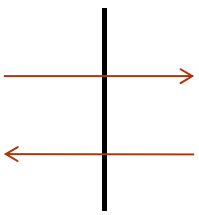
MPI_Isend, MPI_Irecv

MPI_Wait

```c
#include <stdio.h>
#include "mpi.h"
// hello6
int main (int argc, char **argv)
{
    int rank, size;
    int tag;
    int dest, src, man;
    char msg[100];
    MPI_Status stat1, stat2;
    MPI_Request req1, req2;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    tag = 123;
    if (rank == 0) {
        for (dest=1; dest < size; dest++) {
            sprintf(msg,"Hello! node %d",dest);
            MPI_Isend(msg,100,MPI_CHAR, dest, tag, MPI_COMM_WORLD,&req1);
        }
    }
    else {
        src = 0;
        MPI_Irecv(msg,100,MPI_CHAR, src , tag, MPI_COMM_WORLD,&req2);
    }
    MPI_Wait(&req1, &stat1);
    MPI_Wait(&req2, &stat2);

    if (rank != 0) {
        printf("%s\n", msg);
        man = status.MPI_SOURCE;
        printf("=> Hi, master %d ~\n",man);
    }
    MPI_Finalize ();
}
```

* bi-directional communication

* hello7.f90



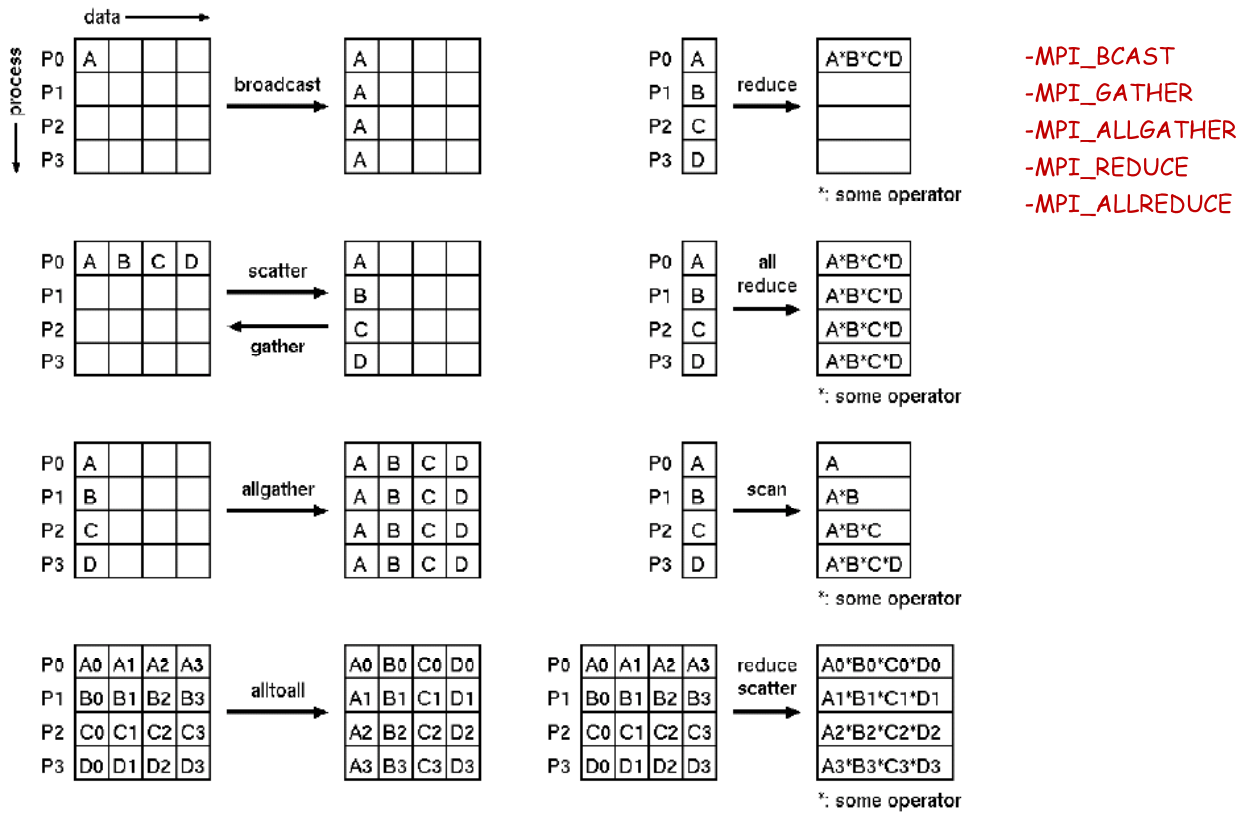
```fortran
program hello7
   use mpi
   implicit none
   integer ierr, irank, nprocs
   integer tag, num, next, prev
   integer ireq1, ireq2
   integer istat1(MPI_STATUS_SIZE), istat2(MPI_STATUS_SIZE)
   integer man, i
   character*100 msg, msg2, msgs(500)
   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD,  irank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
   tag = 123
   if (irank == 0) then
      do i=1, nprocs-1
        write(msg,'(A,2x,I2)') 'Hello! node',i
        call MPI_ISEND(msg,100, MPI_CHARACTER, i, tag, MPI_COMM_WORLD,ireq1,ierr)
      end do
      do i=1, nprocs-1
        call MPI_IRECV(msgs(i),100, MPI_CHARACTER, i, tag, MPI_COMM_WORLD,ireq2,ierr)
      enddo
   else
      call MPI_IRECV(msg,100, MPI_CHARACTER, 0, tag, MPI_COMM_WORLD, ireq2, ierr)
      write(msg2,'(A,2x,I2)') "Hi, master~ I'm node ", irank
      call MPI_ISEND(msg2,100, MPI_CHARACTER, 0, tag, MPI_COMM_WORLD, ireq1, ierr)
   endif
   call MPI_WAIT (ireq1, istat1, ierr)
   call MPI_WAIT (ireq2, istat2, ierr)
   if (irank == 0) then
      do i=1, nprocs-1
         write(6,'(A)') trim(msgs(i))
      enddo
   else
      write(6,'(A)') trim(msg)
   endif
   call MPI_FINALIZE(ierr)
end program hello7
```

Synchronization

MPI_Barrier (comm) **in C**
Call MPI_Barrier (comm, ierr) **in Fortran**

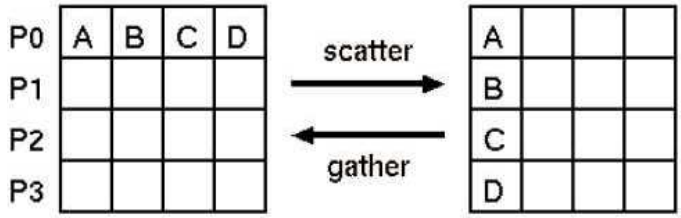
- Blocks until all processes in the group of the communicator **comm** call it.
- Almost never required in a parallel program
  - Occasionally useful in measuring performance and load balancing

Collective Communications



-MPI_BCAST
-MPI_GATHER
-MPI_ALLGATHER
-MPI_REDUCE
-MPI_ALLREDUCE

Scatter & Gather



In C:

```
MPI_Scatter (&sendbuf,sendcount,sendtype,
             &recvbuf,recvcount,recvtype,root, comm);
MPI_Gather  (&sendbuf,sendcount,sendtype,
             &recvbuf,recvcount,recvtype,root,comm);
```

In Fortran:

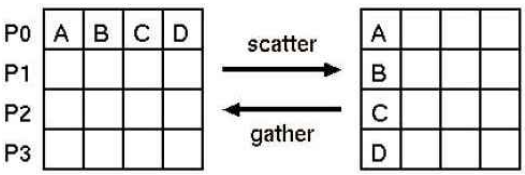
```
Call MPI_SCATTER (sendbuf,sendcount,sendtype,
                  recvbuf,recvcount,recvtype,root, comm,ierr);
Call MPI_GATHER  (sendbuf,sendcount,sendtype,
                  recvbuf,recvcount,recvtype,root,comm,ierr);
```

* gather.c

% mpicc -c nrutil.c

% mpicc gather.c nrutil.o

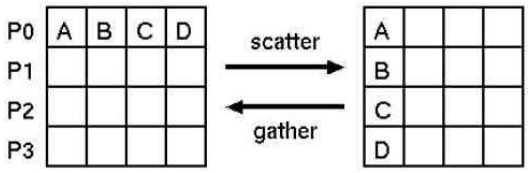
```c
#include <stdio.h>
#include "mpi.h"
#include "nrutil.h"
#define MIN(a,b) ((a) < (b) ? (a) : (b))
// gather
void para_range(int n1,int n2,int nprocs,int rank,int *start,int *end);

int main (int argc, char **argv)
{
    int rank, nprocs;
    int i, n, start, end;
    unsigned long *val;
    unsigned long sum, total;

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

    if (rank == 0) {
        val = lvector(0,nprocs-1);
        printf(" Enter the number of your jobs = \n");
        scanf("%d",&n);
    }
    MPI_Bcast (&n,1,MPI_INT,0,MPI_COMM_WORLD);
    para_range(1,n,nprocs, rank, &start, &end);

    sum = 0;
    for (i=start; i<= end; i++) {
        sum += i;
    }
    MPI_Gather(&sum,1,MPI_UNSIGNED_LONG,
               val,1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD);
    if (rank == 0) {
        total = 0;
        for (i=0; i<nprocs ; i++) {
            printf(" node %d  sum = %ld\n",i,val[i]);
            total += val[i];
        }
        printf("Total sum = %ld\n",total);
    }
    MPI_Finalize ();
}
```



* count = 1

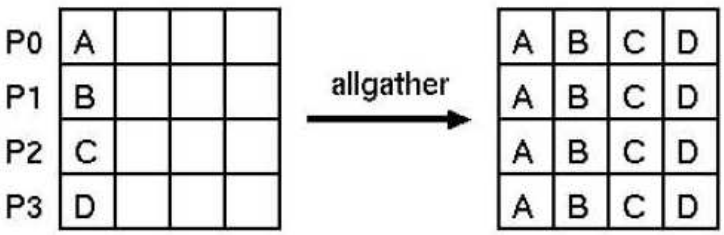
```
program gather
 use mpi
 integer irank, nprocs, ierr
 integer n, istart, iend
 integer*8  isum
 integer*8, allocatable :: ival(:)

 call MPI_INIT (ierr)
 call MPI_COMM_RANK (MPI_COMM_WORLD, irank, ierr)
 call MPI_COMM_SIZE (MPI_COMM_WORLD, nprocs, ierr)
 if (irank == 0) then
    allocate (ival(0:nprocs-1))
    write(6,'(A,$)') ' Enter the number of your jobs = '
    read *, n
 endif
 call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call para_range(1,n, nprocs, irank, istart, iend)

 isum = 0
 do i= istart, iend
    isum = isum + i
 enddo
 call MPI_GATHER(isum,1,MPI_INTEGER8, &
                 ival,1,MPI_INTEGER8,0,MPI_COMM_WORLD,ierr)

 if (irank == 0) then
     do i=0, nprocs-1
        write(6,'(A,2x,I2,2x,A,I14)') 'node ',i,' sum =',ival(i)
     enddo
     write(6,'(A,2x,I14)') 'Total sum = ', sum(ival)
 endif
 call MPI_FINALIZE(ierr)
end program gather
```

* gather.f90



* count = 1

All Gather



In C:

```
MPI_Allgather  (&sendbuf,sendcount,sendtype,
                &recvbuf,recvcount,recvtype,comm);
```

In Fortran:

```
Call MPI_ALLGATHER  (sendbuf,sendcount,sendtype,
                     recvbuf,recvcount,recvtype,comm,ierr);
```

* allgather.c

```c
#include <stdio.h>
#include "mpi.h"
#include "nrutil.h"
#define MIN(a,b) ((a) < (b) ? (a) : (b))
// allgather
void para_range(int n1,int n2,int nprocs,int rank,int *start,int *end);

int main (int argc, char **argv)
{
    int rank, nprocs;
    int i, j, n, start, end;
    unsigned long *val;
    unsigned long sum, total;

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

    val = lvector(0,nprocs-1);
    if (rank == 0) {
        printf(" Enter the number of your jobs = \n");
        scanf("%d",&n);
    }
    MPI_Bcast (&n,1,MPI_INT,0,MPI_COMM_WORLD);
    para_range(1,n,nprocs, rank, &start, &end);

    sum = 0;
    for (i=start; i<= end; i++) {
        sum += i;
    }
    MPI_Allgather(&sum,1,MPI_UNSIGNED_LONG,
                  val,1,MPI_UNSIGNED_LONG,MPI_COMM_WORLD);
    total = 0;
    for (i=0; i<nprocs ; i++) total += val[i];
    printf(" node %d  sum = %ld ",rank,total);
    for (j=0; j<nprocs; j++) printf(" %ld",val[j]); printf("\n");

    MPI_Finalize ();
}
```

* count = 1

- no root

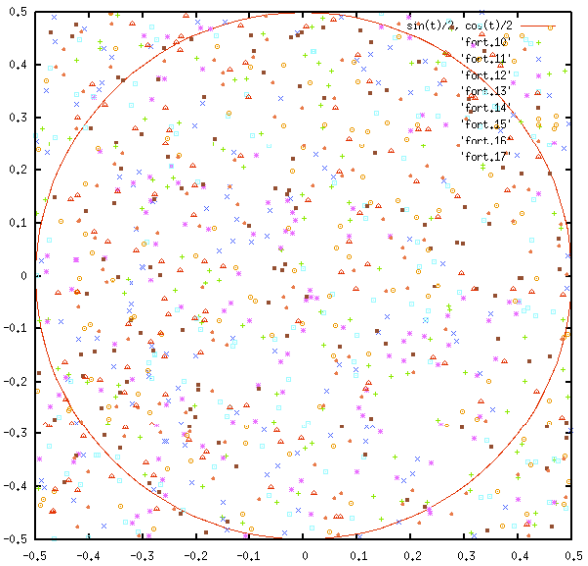
* allreduce.f90

```fortran
program allreduce
 use mpi
 implicit none
 integer irank, nprocs, ierr
 integer n, istart, iend
 integer*8  isum, itot
 integer i,j

 call MPI_INIT (ierr)
 call MPI_COMM_RANK (MPI_COMM_WORLD, irank, ierr)
 call MPI_COMM_SIZE (MPI_COMM_WORLD, nprocs, ierr)
 if (irank == 0) then
    write(6,'(A,$)') ' Enter the number of your jobs = '
    read *, n
 endif
 call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call para_range(1,n, nprocs, irank, istart, iend)

 isum = 0
 do i= istart, iend
    isum = isum + i
 enddo

 call MPI_ALLREDUCE(isum,itot,1,MPI_INTEGER8,MPI_SUM,MPI_COMM_WORLD,ierr)
 write(6,10) 'node ',irank,' itot =',itot, ' isum =',isum
 10 format(A,2x,I2,2x,A,I16,2x,A,I16)
 call MPI_FINALIZE(ierr)
end program allreduce
```

Applications

- Monte Carlo PI Calculation
- Molecule energy minimization
  - Master / Worker model

Exercise : (Monte Carlo PI calculation)



PI=3.141592…

$$\pi = 4\frac{N_{in\_circle}}{N_{tpoints}}$$

Follow exercise note !

% qsub -I -q n4 -N [your account]

* pi_mc.f90

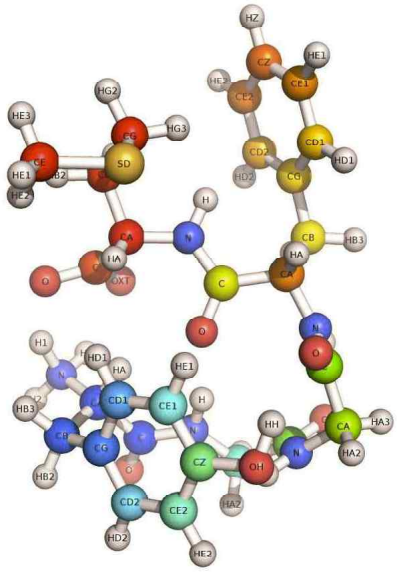
```
program pi_mc
  use mpi
  implicit none
  real*8, parameter :: PI25DT=3.14159265358979323846264 3d0
  integer tpoints
  integer istart, iend, in_circle, in_tot
  integer irank, nprocs, i, ierr
  real*8  :: pi, x,y,r
  real*4  :: time_before, time_after, time_elap,time_tot
  real*8  :: stime, etime, ttime
  integer :: iseed = 123
  call MPI_INIT( ierr )
  call MPI_COMM_RANK( MPI_COMM_WORLD, irank, ierr )
  call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
  if ( irank == 0 ) then
      write(6,'(A)') 'Monte Carlo PI Calculation...'
      write(6,'(A,$)') 'Enter the number of tpoints = '
      read *, tpoints
      print *, 'Total tpoints    =',tpoints
  endif
  call MPI_BCAST(tpoints,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
  call para_range(1,tpoints, nprocs, irank, istart, iend)
```

* pi_mc.f90

```
  in_circle = 0
  iseed = iseed + irank
  stime = MPI_WTIME ()
  call cpu_time (time_before)
  do i = istart, iend
    x = ran(iseed)-0.5
    y = ran(iseed)-0.5
    r = sqrt(x*x + y*y)
    if (r<0.5) in_circle = in_circle + 1
  enddo
  call cpu_time (time_after)
  etime = MPI_WTIME ()
  ttime = etime - stime
  time_elap = time_after- time_before
  call MPI_REDUCE(in_circle,in_tot,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)
  call MPI_REDUCE(time_elap,time_tot,1,MPI_REAL,MPI_SUM,0,MPI_COMM_WORLD,ierr)
  write(6,'(A,I4,A,I16)') ' irank = ',irank, ' in_circle= ', in_circle
  if (irank == 0) then
     pi = 4.0*in_tot/tpoints
     print *,'pi=',pi,' error=',dabs(PI25DT-pi)
     print *,'Total in_circle=',in_tot, ttime,'  seconds'
     print *,'Total cpu time =',time_tot, '  seconds'
     print *,'speed up       =',time_tot/ttime
  endif
  call MPI_FINALIZE (ierr)
end program pi_mc
```

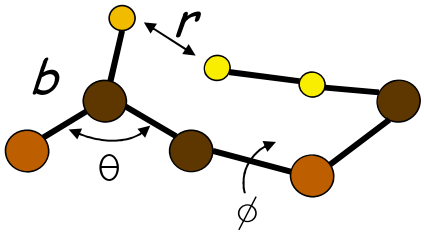
Molecule energy minimization

Met-enkephalin (TYR-GLY-GLY-PHE-MET)

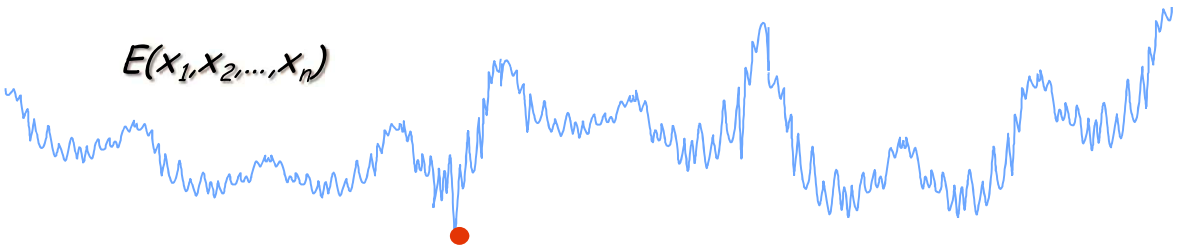


$$E = \frac{1}{2}\sum_{\text{bonds}} k_b (b - b_0)^2 + \frac{1}{2}\sum_{\substack{\text{bonds}\\ \text{angles}}} k_\theta (\theta - \theta_0)^2 + \sum_{\substack{\text{dihedral}\\ \text{angles}}} k_\phi \cos(n\phi - \delta)$$

$$+ \sum_{ij}\left(\frac{A_{ij}}{r_{ij}^6} + \frac{B_{ij}}{r_{ij}^{12}} + \frac{C_{ij}}{r_{ij}^{10}}\right) + \sum_{ij}\frac{q_i q_j}{\varepsilon r_{ij}}$$



$E(x_1, x_2, \ldots, x_n)$



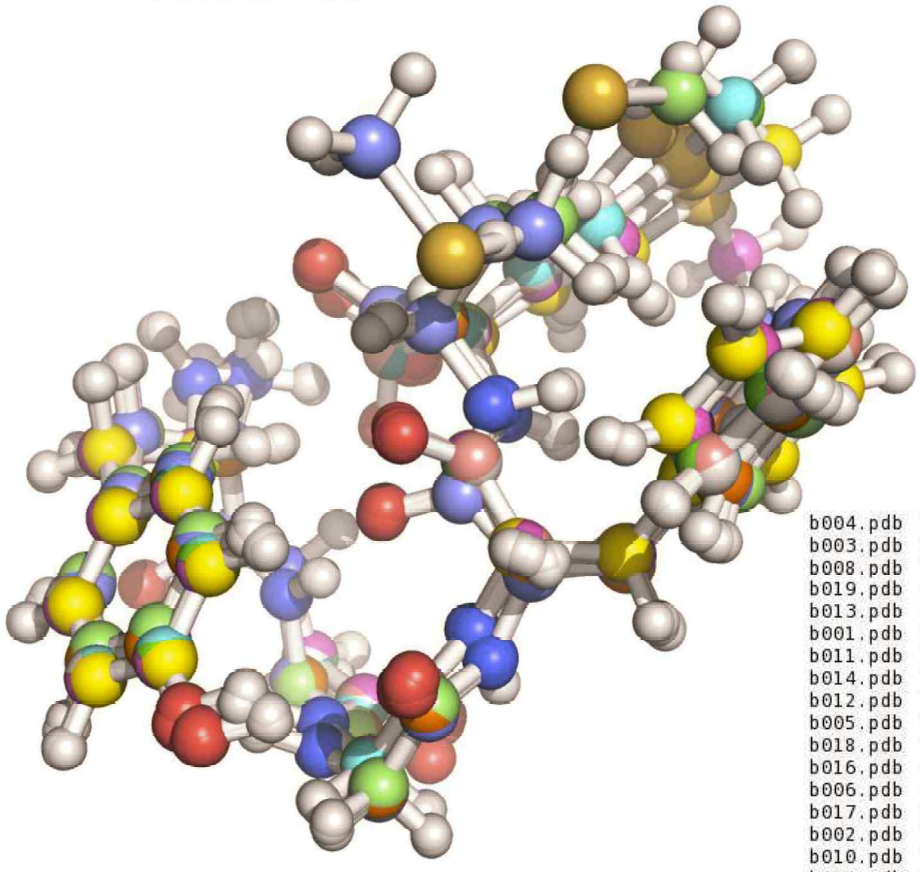
```
module inimodel
 save
 real*8, allocatable :: xyzi(:,:)
end module inimodel

program nmin
  use mpi
  use inimodel
  implicit none
  include 'sizes.i'
  include 'atoms.i'
  include 'sequen.i'
  include 'inform.i'
  include 'argue.i'
  include 'zcoord.i'
  real*8, allocatable :: ene(:)
  real*8, allocatable :: bank(:,:,:)
  integer mode
  integer ierr, irank, nprocs
  integer nconf, i,j,k

  call initial
  call getxyz
  call mechanic
  allocate (xyzi(3,n))
  forall (i=1:n)              ! copy initial model
    xyzi(:,i) = (/ x(i), y(i), z(i) /)
  end forall
  call MPI_INIT( ierr )
  call MPI_COMM_RANK( MPI_COMM_WORLD, irank, ierr )
  call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )

  if (irank == 0) then
      write(6,'(A,$)') 'Enter the number of conformation to minimize ='
      read *, nconf
      allocate(ene(nconf), bank(3,n,nconf))
  endif

  if (irank == 0) then
      call feedin (nconf)
  else
      call minim (nconf)
  endif
```

* project/tinker/source/nmin.f90

```
subroutine feedin (nconf)
  implicit none
  include 'sizes.i'
  include 'atoms.i'
  integer nconf
  integer istat(MPI_STATUS_SIZE)
  integer i,j,k,l,mm,id, ierr, man
  integer idum
  mm =0
  do id=1, min(nprocs,nconf)
     mm = mm + 1
     call mpi_send(mm,mpi_integer4,mm,123,mpi_comm_world,ierr)
  enddo

  do id=1, nconf
     call mpi_recv(k,1,mpi_integer4,mpi_any_source,123, &
                   mpi_comm_world,istat,ierr)
     man = istat(mpi_source)
     call mpi_recv(ene(k),1,mpi_real8,man,234, &
                   mpi_comm_world,istat,ierr)
     call mpi_recv(bank(:,:,k),3*n,mpi_real8,man,345, &
                   mpi_comm_world,istat,ierr)

     if (mm < nconf) then
         mm = mm + 1
         call mpi_send(mm,1,mpi_integer4,man,123, &
                       mpi_comm_world,ierr)
     else
         call mpi_send(0,1,mpi_integer4,man,123, &
                       mpi_comm_world,ierr)
     endif
  enddo
end subroutine feedin
```

```
subroutine minim (nconf)
  implicit none
  include 'sizes.i'
  include 'atoms.i'
  integer istat(mpi_status_size)
  integer id,ierr, nconf
  integer i,j, idum
  real ran1, ranum
  real*8  e
  real*8  xyz(3,n)

  idum = 123 + irank

  if (irank > nconf) return
  do
     call mpi_recv(id,1,mpi_integer4,0,123, &
                   mpi_comm_world,istat,ierr)
     if (id == 0) return
     ranum = ran1 (idum)
     call randomize (idum)
     call minimize (e)
     forall (i=1:n)
       xyz(1,i) = x(i); xyz(2,i) = y(i); xyz(3,i) = z(i)
     end forall
     call mpi_send(id,1,mpi_integer4,0,123,mpi_comm_world,ierr)
     call mpi_send(e,1,mpi_real8,0,234,mpi_comm_world,ierr)
     call mpi_send(xyz,3*n,mpi_real8,0,345,mpi_comm_world,ierr)
  enddo

end subroutine minim
```

* 20 minimized models



```
b004.pdb  Total Energy=        -56.020190
b003.pdb  Total Energy=        -56.183630
b008.pdb  Total Energy=        -56.270449
b019.pdb  Total Energy=        -56.318451
b013.pdb  Total Energy=        -56.336022
b001.pdb  Total Energy=        -56.338882
b011.pdb  Total Energy=        -56.338927
b014.pdb  Total Energy=        -56.341662
b012.pdb  Total Energy=        -56.342432
b005.pdb  Total Energy=        -56.342932
b018.pdb  Total Energy=        -56.343037
b016.pdb  Total Energy=        -56.343071
b006.pdb  Total Energy=        -56.343181
b017.pdb  Total Energy=        -56.343955
b002.pdb  Total Energy=        -56.344669
b010.pdb  Total Energy=        -56.345870
b015.pdb  Total Energy=        -56.345952
b009.pdb  Total Energy=        -56.609061
b020.pdb  Total Energy=        -57.016739
b007.pdb  Total Energy=        -57.106061
```

Advanced Course

- Derived Type
- Grouping and Virtual Topology
  - MPI provides routines to provide structure to collections of processes
- MPI-2
  - One-side communication
  - Dynamic process
  - Parallel I/O

Summary

- 6 Function of MPI library
- Point to Point Communication
- Collective Communication

MPI 프로그래밍 생각할 점

- 시리얼 프로그램: 시간이 얼마나 걸리는가?
- 병렬컴퓨터에 대해서 알아야 한다
  - CPU 구성
  - 메모리 상황
  - 네트워크 성능
- 풀고 싶은 문제의 특성을 알아야 한다
  - 문제가 쉽게 병렬로 될 수 있는 것인지?
  - Load Balencing
  - 커뮤니케이션 문제

Acknowledgement

Home page: http://cac.kias.re.kr

``*Only love has no limits. In contrast, our predictions can fail, our communication can fail, and our knowledge can fail. For our knowledge is patchwork, and our predictive power is limited.* ***But when perfection comes, all patchwork will disappear.***''

(1 Cor. 13:8-10)

CAC 거대수치계산연구센터
KIAS CENTER FOR ADVANCED COMPUTATION

- Director Professor Changbom Park
- Research Professor Juhan Kim
- Research Professor Keehyoung Joo
- Computer Administrator Kyoungsoo Kim
- Administration Staff Minjung Choi
- Assisting developer Jinsun Kang