

Report S1 C

Kengo Nakajima

Technical & Scientific Computing II (4820-1028)
Seminar on Computer Science II (4810-1205)
Hybrid Distributed Parallel Computing (3747-111)

Report S1

- Problem S1-1
 - Read local files <\$O-S1>/a1.0~a1.3, <\$O-S1>/a2.0~a2.3.
 - Develop codes which calculate norm $||x||$ of global vector for each case.
 - <\$O-S1>file.c, <\$O-S1>file2.c
- Problem S1-3
 - Develop parallel program which calculates the following numerical integration using “trapezoidal rule” by MPI_Reduce, MPI_Bcast etc.
 - Measure computation time, and parallel performance

$$\int_0^1 \frac{4}{1+x^2} dx$$

Copying files on Reedbush-U

Copy

```
>$ cd <$0-TOP>  
>$ cp /lustre/gt16/z30088/class_eps/C/s1r-c.tar .  
>$ tar xvf s1r-c.tar
```

Confirm directory

```
>$ ls  
    mpi  
>$ cd mpi/S1-ref
```

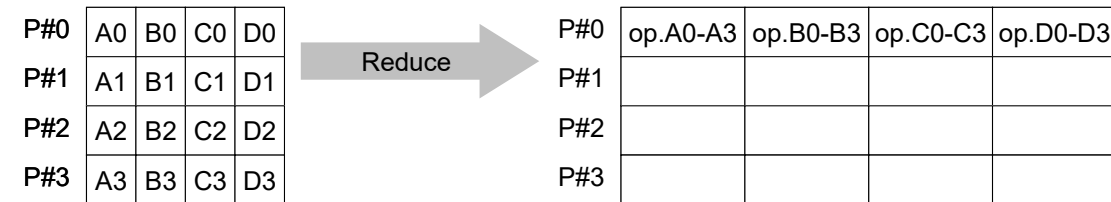
This directory is called as <\$0-S1r>.

<\$0-S1r> = <\$0-TOP>/mpi/S1-ref

S1-1 : Reading Local Vector, Calc. Norm

- Problem S1-1
 - Read local files <\$O-S1>/a1.0~a1.3, <\$O-S1>/a2.0~a2.3.
 - Develop codes which calculate norm $\|x\|$ of global vector for each case.
- Use MPI_Allreduce (or MPI_Reduce)
- Advice
 - Checking each component of variables and arrays !

MPI_Reduce



- Reduces values on all processes to a single value
 - Summation, Product, Max, Min etc.
- **MPI_Reduce (sendbuf, recvbuf, count, datatype, op, root, comm)**
 - sendbuf choice I starting address of send buffer
 - recvbuf choice 0 starting address receive buffer
 - type is defined by "datatype"
 - count int I number of elements in send/receive buffer
 - datatype MPI_Datatype I data type of elements of send/recive buffer
 - FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
 - C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc
 - op MPI_Op I reduce operation
 - MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND etc
 - Users can define operations by MPI_OP_CREATE
 - root int I rank of root process
 - comm MPI_Comm I communicator

Send/Receive Buffer (Sending/Receiving)

- Arrays of “send (sending) buffer” and “receive (receiving) buffer” often appear in MPI.
- Addresses of “send (sending) buffer” and “receive (receiving) buffer” must be different.

“op” of MPI_Reduce/Allreduce

MPI_Reduce

(sendbuf, recvbuf, count, datatype, op, root, comm)

- MPI_MAX, MPI_MIN Max, Min
- MPI_SUM, MPI_PROD Summation, Product
- MPI_LAND Logical AND

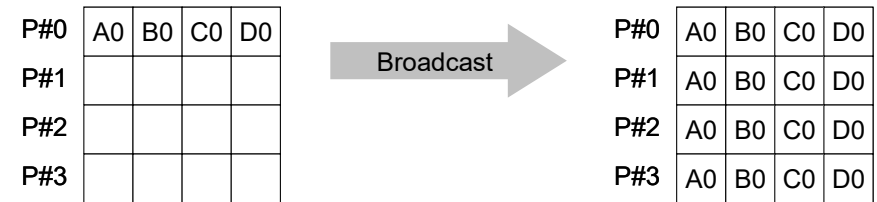
```
double X0, XSUM;
```

```
MPI_Reduce  
(&X0, &XSUM, 1, MPI_DOUBLE, MPI_SUM, 0, <comm>)
```

```
double X0[4];
```

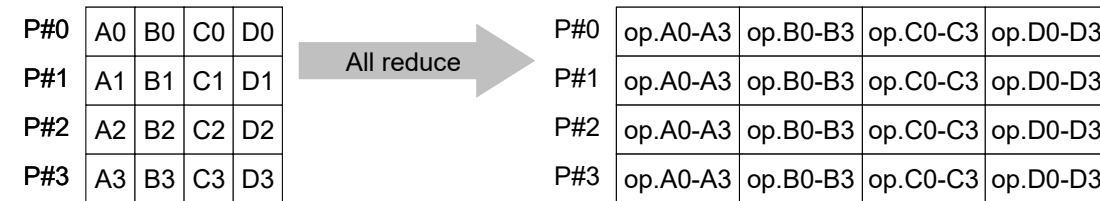
```
MPI_Reduce  
(&X0[0], &X0[2], 2, MPI_DOUBLE_PRECISION, MPI_SUM, 0, <comm>)
```

MPI_Bcast



- Broadcasts a message from the process with rank "root" to all other processes of the communicator
- **MPI_Bcast (buffer, count, datatype, root, comm)**
 - **buffer** choice I/O starting address of buffer
type is defined by "datatype"
 - **count** int I number of elements in send/receive buffer
 - **datatype** MPI_Datatype I data type of elements of send/recive buffer
 - FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
 - C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc.
 - **root** int I rank of root process
 - **comm** MPI_Comm I communicator

MPI_Allreduce



- MPI_Reduce + MPI_Bcast
- Summation (of dot products) and MAX/MIN values are likely to be utilized in each process

- call MPI_Allreduce

(sendbuf, recvbuf, count, datatype, op, comm)

- sendbuf choice I starting address of send buffer
- recvbuf choice 0 starting address receive buffer
- type is defined by "datatype"

- count int I number of elements in send/receive buffer
- datatype MPI_Datatype I data type of elements of send/recive buffer

- op MPI_Op I reduce operation
- comm MPI_Comm I communicator

S1-1 : Local Vector, Norm Calculation

Uniform Vectors (a1. *): s1-1-for_a1.c

```

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

int main(int argc, char **argv){
    int i, N;
    int PeTot, MyRank;
    MPI_Comm SolverComm;
    double vec[8];
    double sum0, sum;
    char filename[80];
    FILE *fp;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &PeTot);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyRank);

    sprintf(filename, "a1.%d", MyRank);
    fp = fopen(filename, "r");
    assert(fp != NULL);

    N=8;
    for(i=0;i<N;i++){
        fscanf(fp, "%lf", &vec[i]);
    }
    sum0 = 0.0;
    for(i=0;i<N;i++){
        sum0 += vec[i] * vec[i];
    }

    MPI_Allreduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    sum = sqrt(sum);

    if(!MyRank) printf("%27.20E\n", sum);
    MPI_Finalize();
    return 0;
}

```

S1-1: Local Vector, Norm Calculation

Non-uniform Vectors (a2.*): s1-1-for_a2.c

```

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

int main(int argc, char **argv){
    int i, PeTot, MyRank, n;
    MPI_Comm SolverComm;
    double *vec, *vec2;
    int * Count, CountIndex;
    double sum0, sum;
    char filename[80];
    FILE *fp;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &PeTot);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyRank);

    sprintf(filename, "a2.%d", MyRank);
    fp = fopen(filename, "r");
    assert(fp != NULL);

    fscanf(fp, "%d", &n);
    vec = malloc(n * sizeof(double));
    for(i=0; i<n; i++){
        fscanf(fp, "%lf", &vec[i]);
    }
    sum0 = 0.0;
    for(i=0; i<n; i++){
        sum0 += vec[i] * vec[i];
    }

    MPI_Allreduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    sum = sqrt(sum);

    if(!MyRank) printf("%27.20E\n", sum);
    MPI_Finalize();
    return 0;}

```

S1-1: Running the Codes

FORTRAN

```
$ cd <$0-S1r>
$ mpiifort -O3 s1-1-for_a1.f
$ mpiifort -O3 s1-1-for_a2.f

(modify "go4.sh")
$ qsub go4.sh
```

C

```
$ cd <$0-S1r>
$ mpicc -O3 s1-1-for_a1.c
$ mpicc -O3 s1-1-for_a2.c

(modify "go4.sh")
$ qsub go4.sh
```

S1-1 : Local Vector, Calc. Norm Results

Results using one core

```
a1.* 1.620882475690325900000E+03  
a2.* 1.222184928723963600000E+03
```

```
$> ifort -O3 dot-a1.f  
$> qsub go1.sh
```

```
$> ifort -O3 dot-a2.f  
$> qsub go1.sh
```

Results

```
a1.* 1.620882475690325900000E+03  
a2.* 1.222184928723963600000E+03
```

go1.sh

```
#!/bin/sh  
#PBS -q u-lecture  
#PBS -N test  
#PBS -l select=1:mpiprocs=1  
#PBS -Wgroup_list=gt16  
#PBS -l walltime=00:05:00  
#PBS -e err  
#PBS -o test.lst  
  
cd $PBS_O_WORKDIR  
. /etc/profile.d/modules.sh  
  
mpirun ./impimap.sh ./a.out
```

S1-1 : Local Vector, Calc. Norm

If SENDBUF=RECVBUF, what happens ?

True

```
MPI_Allreduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM,  
              MPI_COMM_WORLD)
```

False

```
MPI_Allreduce(&sum0, &sum0, 1, MPI_DOUBLE, MPI_SUM,  
              MPI_COMM_WORLD)
```

S1-1 : Local Vector, Calc. Norm

If SENDBUF=RECVBUF, what happens ?

True

```
MPI_Allreduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM,  
              MPI_COMM_WORLD)
```

False

```
MPI_Allreduce(&sum0, &sum0, 1, MPI_DOUBLE, MPI_SUM,  
              MPI_COMM_WORLD)
```

True

```
MPI_Allreduce(&sumK[1], &sumK[2], 1, MPI_DOUBLE, MPI_SUM,  
              MPI_COMM_WORLD)
```

SENDBUF .ne. RECVBUF

S1-3: Integration by Trapezoidal Rule

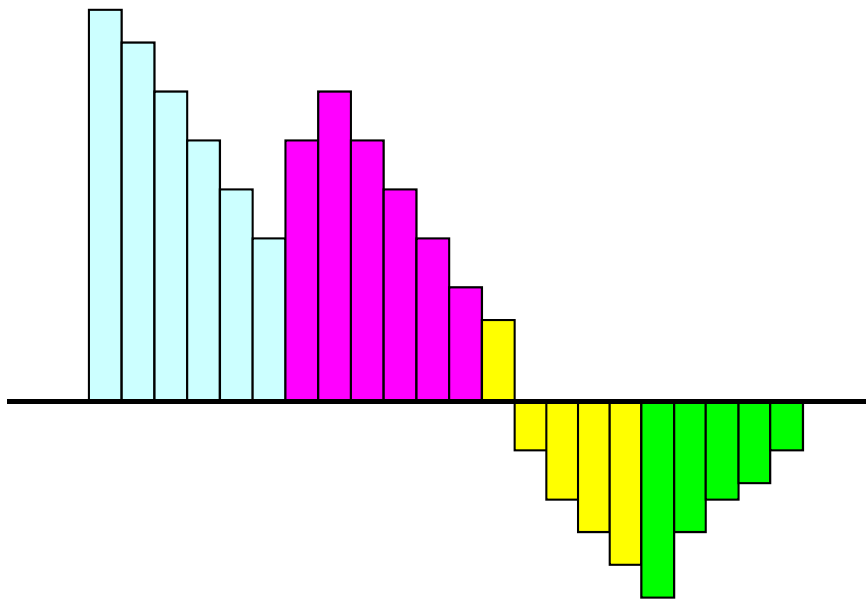
- Problem S1-3
 - Develop parallel program which calculates the following numerical integration using “trapezoidal rule” by MPI_Reduce, MPI_Bcast etc.
 - Measure computation time, and parallel performance

$$\int_0^1 \frac{4}{1+x^2} dx$$

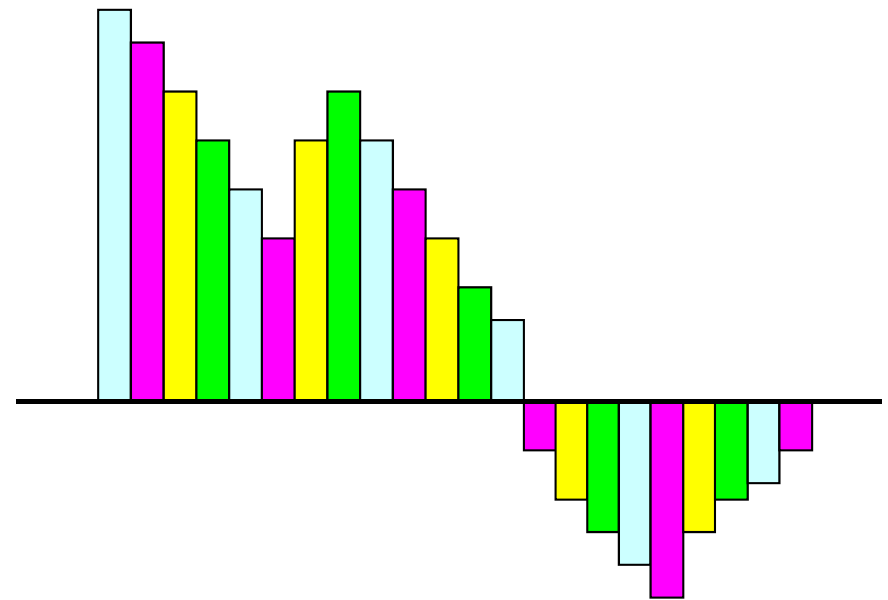
S1-3: Integration by Trapezoidal Rule

Two Types of Load Distribution

Type-A



Type-B



$$\frac{1}{2} \Delta x \left(f_1 + f_{N+1} + \sum_{i=2}^N 2 f_i \right) \text{ corresponds to "Type-A".}$$

S1-3: Integration by Trapezoidal Rule

TYPE-A (1/2): s1-3a.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include "mpi.h"

int main(int argc, char **argv){
    int i;
    double TimeStart, TimeEnd, sum0, sum, dx;
    int PeTot, MyRank, n, int *index;
    FILE *fp;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &PeTot);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyRank);

    index = calloc(PeTot+1, sizeof(int));
    fp = fopen("input.dat", "r");
    fscanf(fp, "%d", &n);
    fclose(fp);
    if(MyRank==0) printf("%s%8d\n", "N=", n);
    dx = 1.0/n;

    for(i=0; i<=PeTot; i++){
        index[i] = ((long long)i * n)/PeTot;
    }
}
```

“N (number of segments) “ is specified in “input.dat”

PE#0

PE#1

PE#2

● ● ●

PE#(PETOT-1)

index[0]

index[1]

index[2]

index[3]

index[PETOT-1]

index[PeTot]
=N

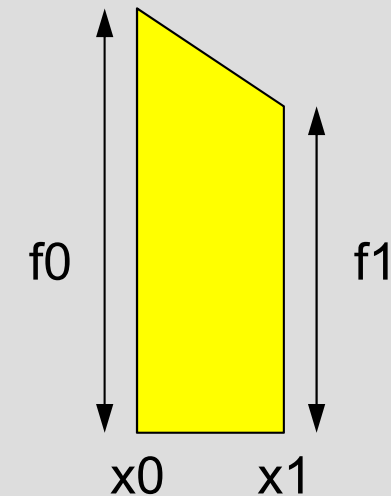
S1-3: Integration by Trapezoidal Rule

TYPE-A (2/2): s1-3a.c

```

TimeS = MPI_Wtime();
sum0 = 0.0;
for(i=index[MyRank]; i<index[MyRank+1]; i++)
{
    double x0, x1, f0, f1;
    x0 = (double)i * dx;
    x1 = (double)(i+1) * dx;
    f0 = 4.0/(1.0+x0*x0);
    f1 = 4.0/(1.0+x1*x1);
    sum0 += 0.5 * (f0 + f1) * dx;
}

```



```

MPI_Reduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
TimeE = MPI_Wtime();

```

```

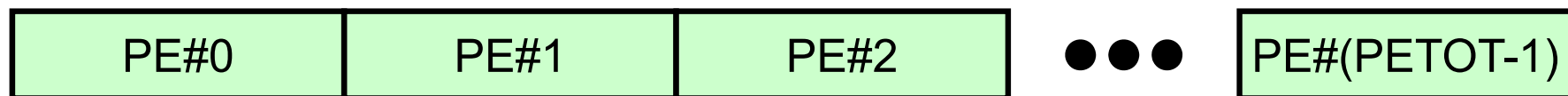
if(!MyRank) printf("%24.16f%24.16f%24.16f¥n", sum, 4.0*atan(1.0), TimeE - TimeS);

```

```

MPI_Finalize();
return 0;

```



index[0]

index[1]

index[2]

index[3]

index[PETOT-1]

index[PeTot]
=N

S1-3: Integration by Trapezoidal Rule

TYPE-B: s1-3b.c

```

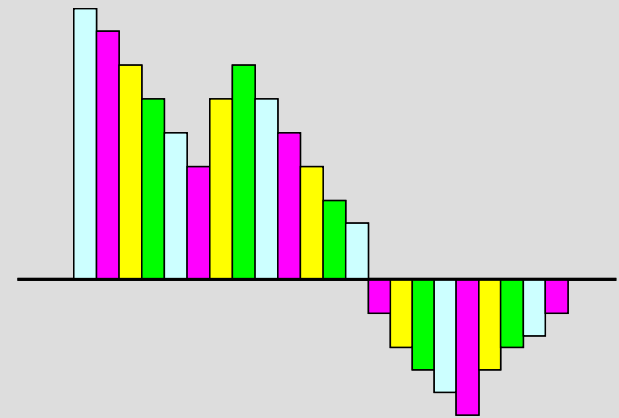
TimeS = MPI_Wtime();
sum0 = 0.0;
for(i=MyRank; i<n; i+=PeTot)
{
    double x0, x1, f0, f1;
    x0 = (double)i * dx;
    x1 = (double)(i+1) * dx;
    f0 = 4.0/(1.0+x0*x0);
    f1 = 4.0/(1.0+x1*x1);
    sum0 += 0.5 * (f0 + f1) * dx;
}

MPI_Reduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
TimeE = MPI_Wtime();

if(!MyRank) printf("%24.16f%24.16f%24.16f\n", sum, 4.0*atan(1.0), TimeE-TimeS);

MPI_Finalize();
return 0;
}

```



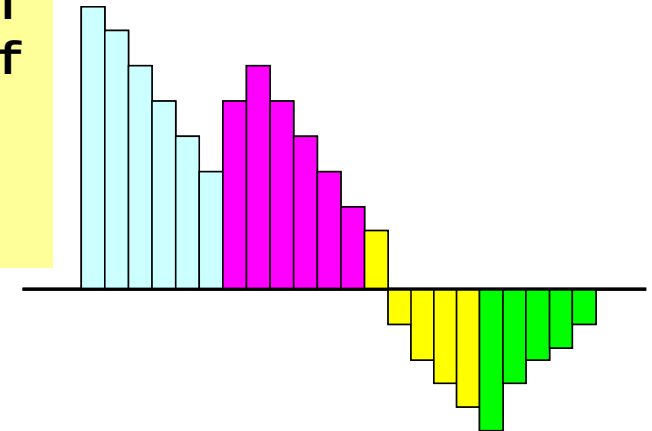
S1-3: Running the Codes

FORTTRAN

```
$ mpiifort -O3 -xCORE-AVX2 -align array32byte s1-3a.f
$ mpiifort -O3 -xCORE-AVX2 -align array32byte s1-3b.f
```

```
(modify "go.sh")
$ qsub go.sh
```

Type-A

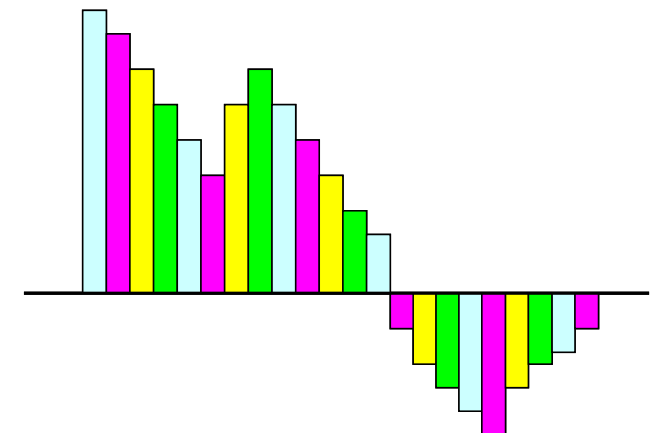


C

```
$ mpicc -O3 -xCORE-AVX2 -align s1-3a.c
$ mpicc -O3 -xCORE-AVX2 -align s1-3b.c
```

```
(modify "go.sh")
$ qsub go.sh
```

Type-B



go.sh

```
#!/bin/sh
```

```
#PBS -q u-lecture
```

```
#PBS -N test
```

```
#PBS -l select=8:mpiprocs=32
```

```
#PBS -Wgroup_list=gt16
```

```
#PBS -l walltime=00:05:00
```

```
#PBS -e err
```

```
#PBS -o hello.lst
```

```
cd $PBS_O_WORKDIR
```

```
. /etc/profile.d/modules.sh
```

```
mpirun ./impimap.sh ./a.out
```

Name of "QUEUE"

Job Name

node#, proc#/node

Group Name (Wallet)

Computation Time

Standard Error

Standard Outpt

go to current dir
(ESSENTIAL)

Exec's

```
#PBS -l select=1:mpiprocs=4
```

1-node, 4-proc's

```
#PBS -l select=1:mpiprocs=16
```

1-node, 16-proc's

```
#PBS -l select=1:mpiprocs=36
```

1-node, 36-proc's

```
#PBS -l select=2:mpiprocs=32
```

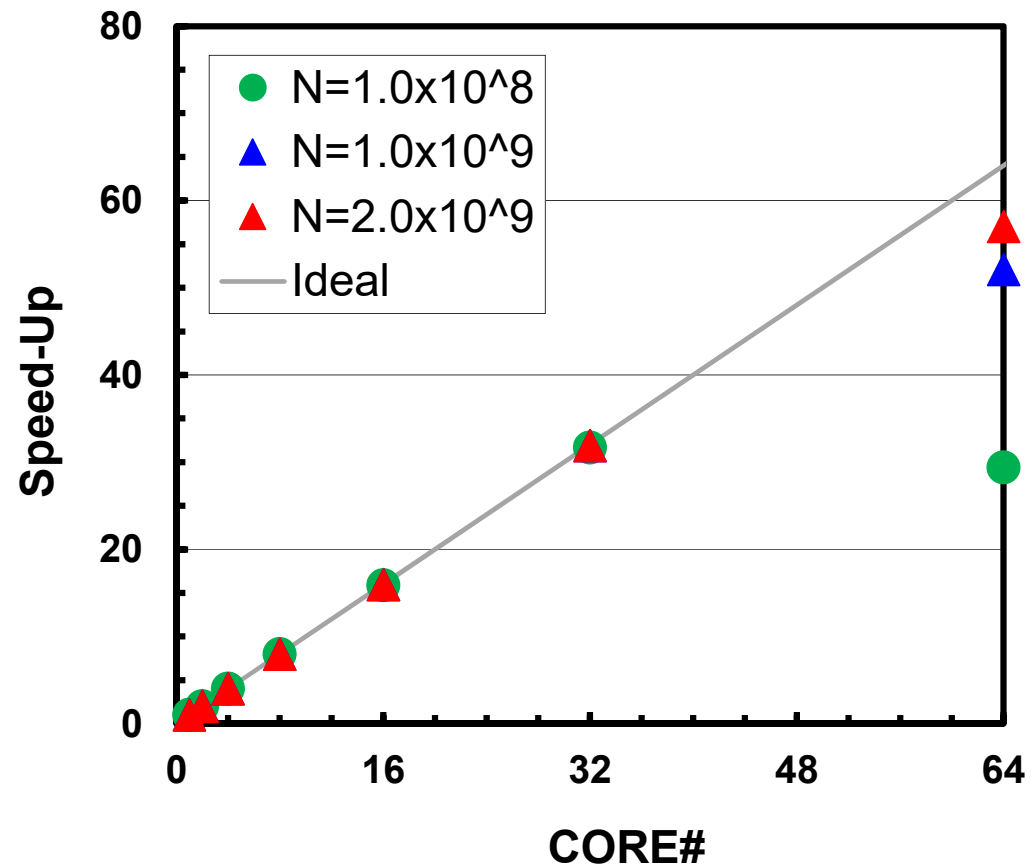
2-nodes, 32x2=64-proc's

```
#PBS -l select=8:mpiprocs=36
```

8-nodes, 36x8=288-proc's

S1-3: Performance on RB-U (1/3)

- ◆ : $N=10^8$, ● : 10^9 , ▲ : 2×10^9 , — : Ideal
- Based on results (sec.) using a single core
- Strong Scaling**
 - Entire problem size fixed
 - $1/N$ comp. time using $N \times$ cores
- Weak Scaling**
 - Problem size/core is fixed
 - Comp. time is kept constant for $N \times$ scale problems using $N \times$ cores



32 cores/node, 16 cores/socket
up to 2 nodes (64 cores)

S1-3: Performance on RB-U (2/3)

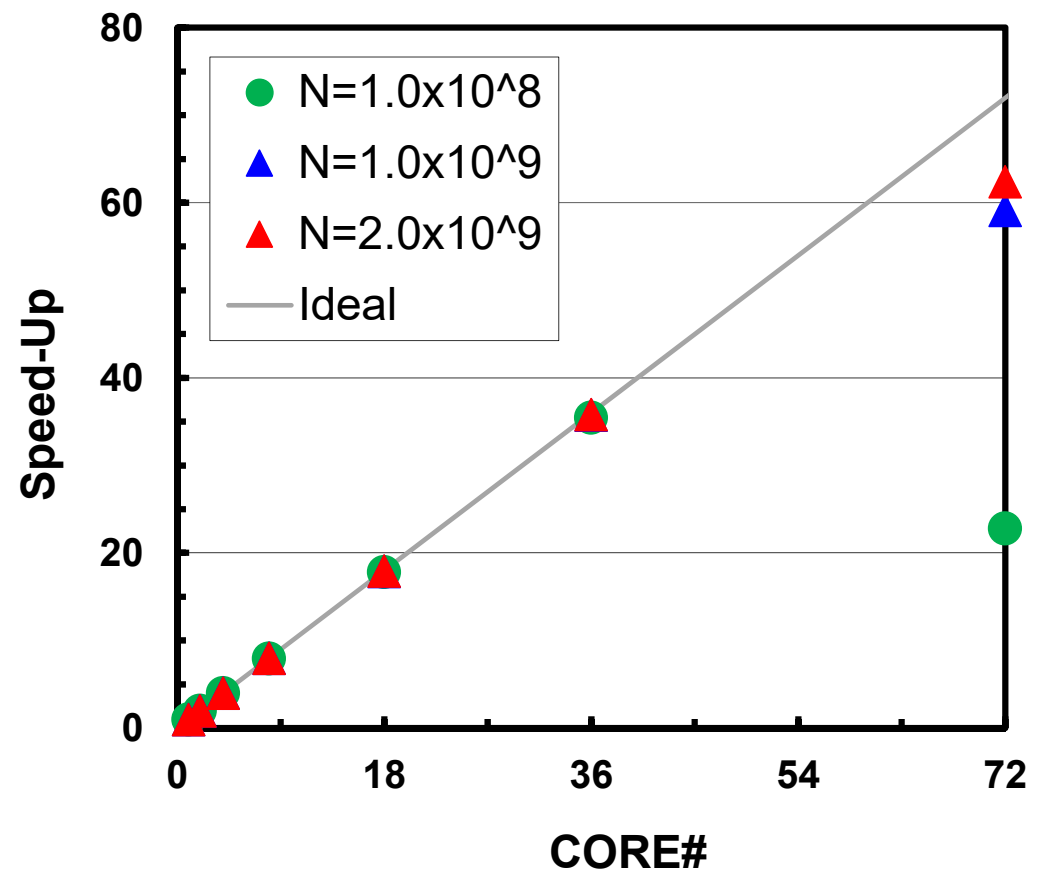
- ◆ : $N=10^8$, ● : 10^9 , ▲ : 2×10^9 , — : Ideal
- Based on results (sec.) using a single core

- Strong Scaling**

- Entire problem size fixed
- $1/N$ comp. time using $N \times$ cores

- Weak Scaling**

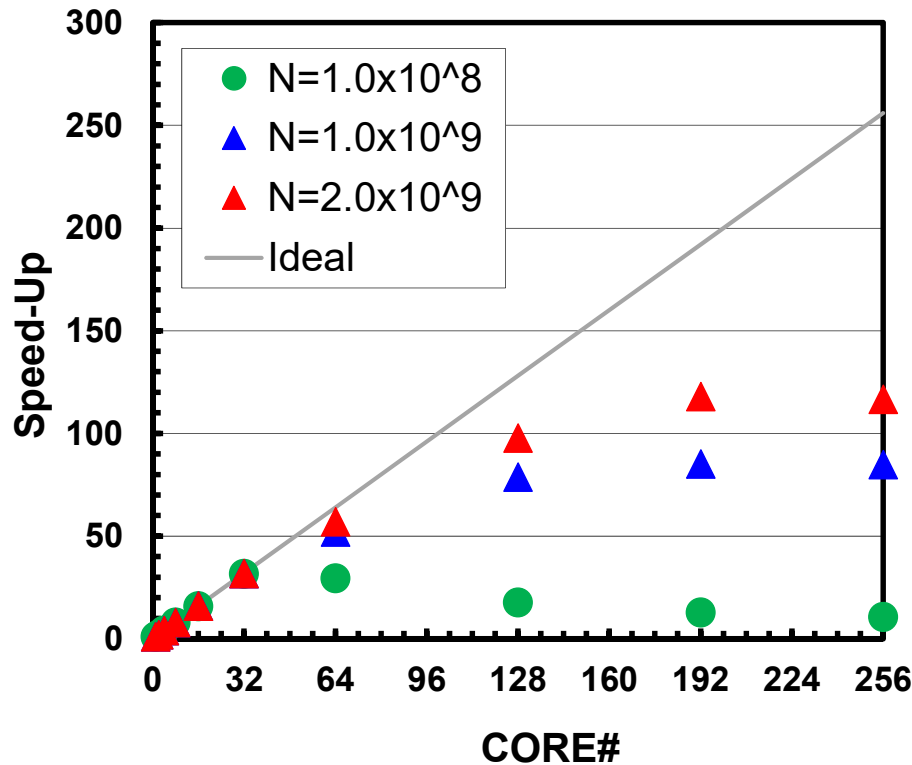
- Problem size/core is fixed
- Comp. time is kept constant for $N \times$ scale problems using $N \times$ cores



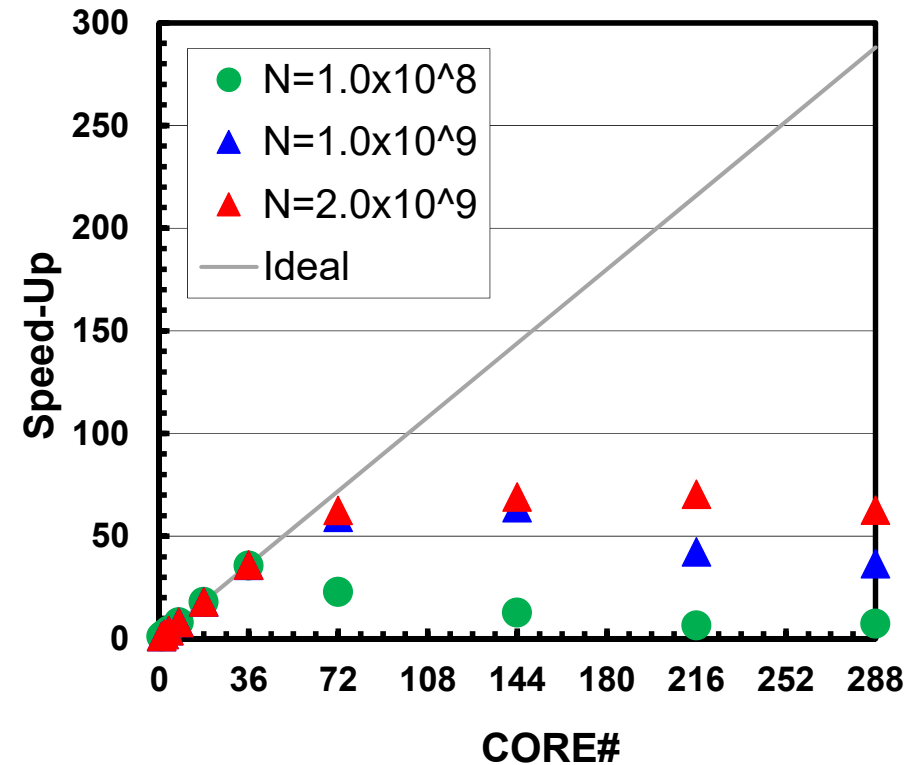
**36 cores/node, 18 cores/socket
up to 2 nodes (72 cores)**

S1-3: Performance on RB-U (3/3)

- ◆ : $N=10^8$, ● : 10^9 , ▲ : 2×10^9 , — : Ideal
- 32cores/node (16cores/socket) cases are better
 - smaller number of MPI processes



**32 cores/node, 16 cores/socket
up to 8 nodes (256 cores)**



**36 cores/node, 18 cores/socket
up to 8 nodes (288 cores)**

Performance is lower than ideal one

- Time for MPI communication
 - Time for sending data
 - Communication bandwidth between nodes
 - Time is proportional to size of sending/receiving buffers
- Time for starting MPI
 - latency
 - does not depend on size of buffers
 - depends on number of calling, increases according to process #
 - $O(10^0)$ - $O(10^1)$ μ sec.
- Synchronization of MPI
 - Increases according to number of processes

Performance is lower than ideal one (cont.)

- If computation time is relatively small (N is small in S1-3), these effects are not negligible.
 - If the size of messages is small, effect of “latency” is significant.