Computational Science Alliance
The University of Tokyo

Summer School 2018, Computational Science Alliance, UTokyo
Hands on Lecture on Aug. 22-24, 2018

How to use Reedbush Supercomputer System

Masaharu MATSUMOTO
Project Lecturer, Department of Computer Science

matsumoto@is.s.u-tokyo.ac.jp

The Purpose of This Lecture

- To login to Reedbush supercomputer system
- To execute C/Fortran programs on Reedbush
  - ✓ MPI/OpenMP with CPUs
  - ✓ CUDA C/CUDA Fortran with GPUs

- If you already know how to use Reedbush, please proceed with the work yourself.
- If you don't know what to do while at today's work, please ask me or someone who knows well.
- Please also refer to "https://www.cc.u-tokyo.ac.jp/supercomputer/reedbush/service/QuickStartGuide-en.pdf"

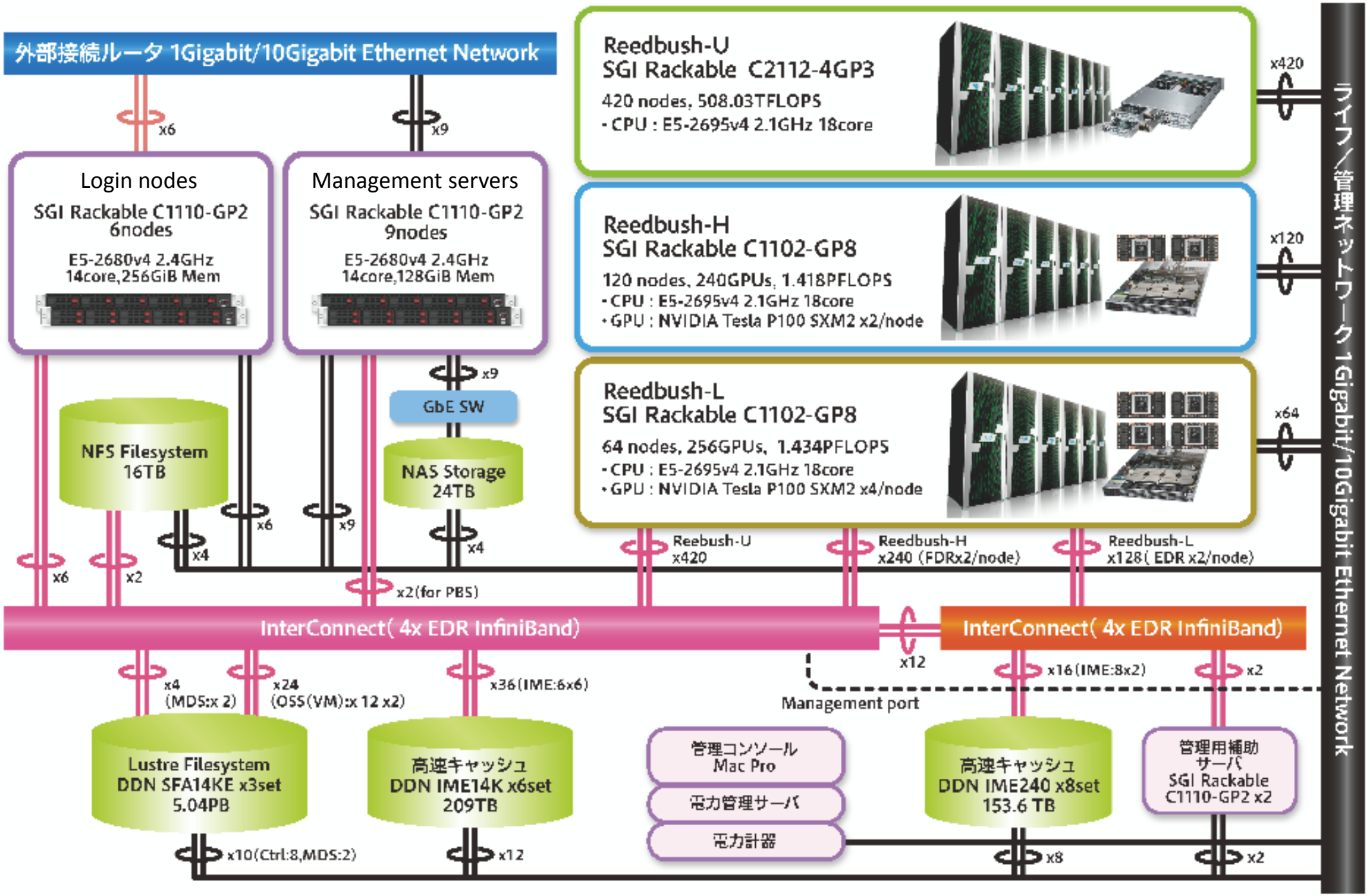
Reedbush Supercomputer System

Reedbush is installed in Information Technology Center, The University of Tokyo. Reedbush system has three types of groups of compute nodes,

1. Reedbush-U (with CPU only)
   Each Node: Intel Xeon E5-2695v4 (Broadwell-EP 2.1GHz 18core) x 2 socket, 256GB Mem
   Total 420 Nodes

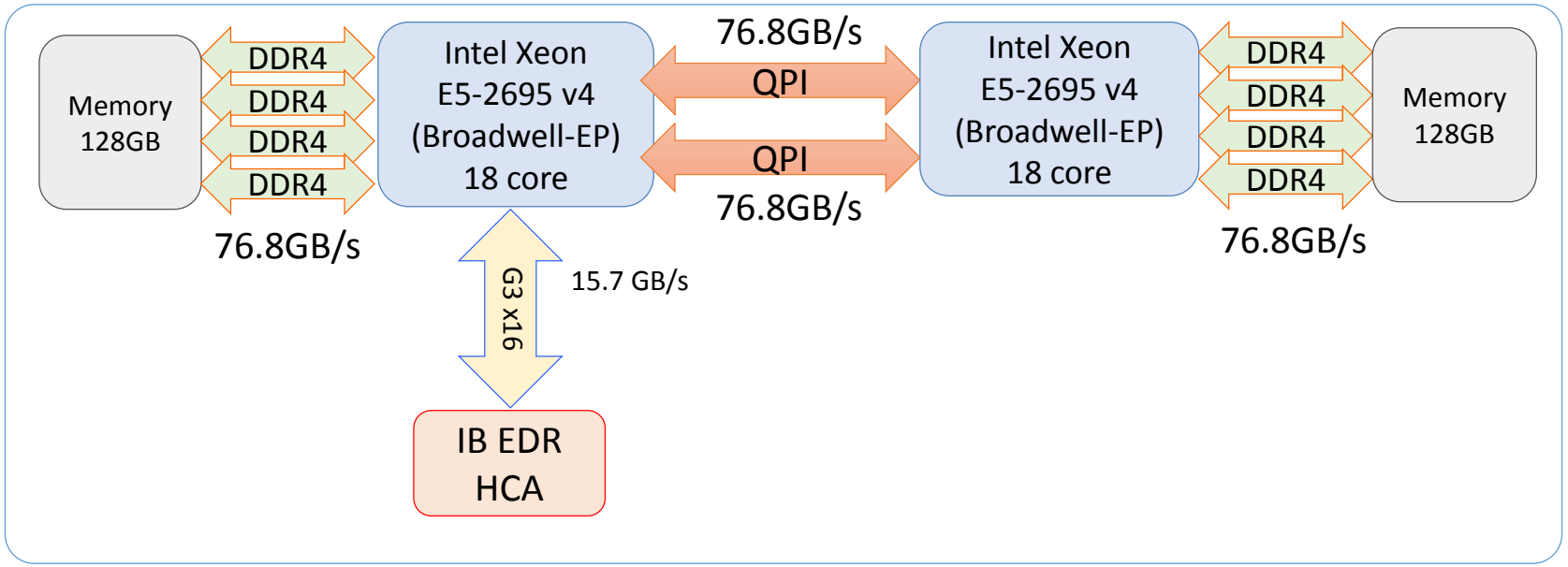
2. Reedbush-H (with GPU)
   Host CPU: Same as Reedbush-U
   Accelerators: NVIDIA Tesla P100 x 2, For each: 4.8-5.3 Tflops, 16 GB Mem
   Total 120 Nodes

3. Reedbush-L (for long-running jobs)
   abbreviation



https://www.cc.u-tokyo.ac.jp/system/reedbush/reedbush_intro.html

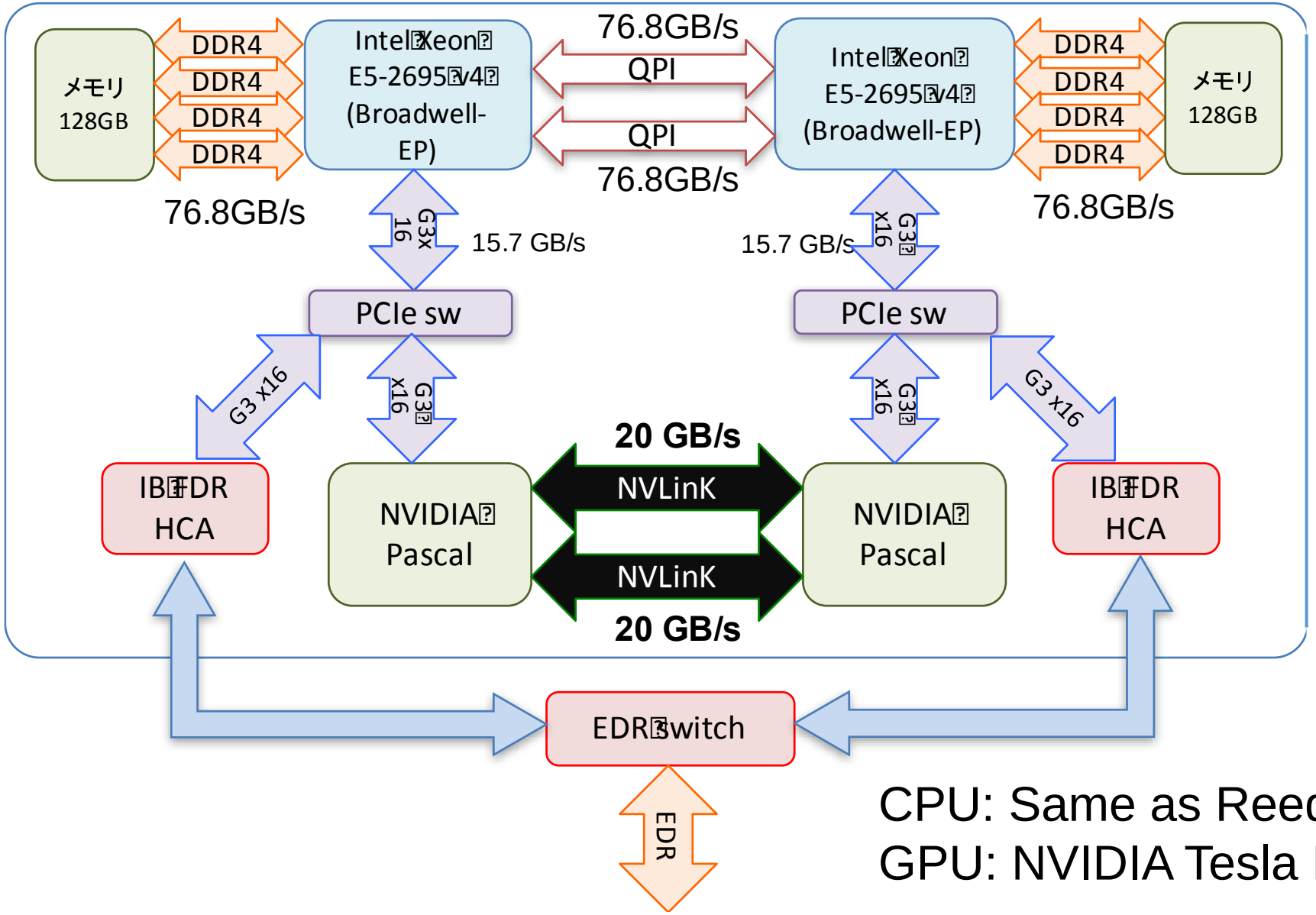
Reedbush-U compute node



×420 nodes

The number of core : 36 cores/node
The amount of memory: 256 GB/node

Reedbush-H compute node



× 120 nodes

CPU: Same as Reedbush-U

GPU: NVIDIA Tesla P100 x 2

Overview - How to Login to Reedbush -

(If you use WindowsOS, you should install and use **Cygwin**)

利用者番号	t03000	初期パスワード※1	=mXAxpK0cWcc865i		管理者
職　　名	特任講師		氏　　名	松本　正晴	
所　　属	東京大学 情報理工学系研究科 コンピュータ科学専攻 (		電話番号	03-5841-4283	
研 究 分 野	4601	計算科学	メールアドレス	matsumoto@is.s.u-tokyo.ac.jp	

- <u>Preparation for the first login</u>
  1. Get your account info. (ID and Password)
  2. Create a public and private key pair (ssh-keygen, etc.)
  3. Register the public key on Reedbush through the User Portal website (https://reedbush-www.cc.u-tokyo.ac.jp/)
- <u>Login after the above preparation</u>
  - ssh login using the private key
    - ✓ `$ ssh XXXXXX@reedbush.cc.u-tokyo.ac.jp`

Check Your Information for Login to Reedbush

Your account ID: XXXXXX

Initial password for registration of keys

利用者番号	t03000	初期パスワード※1	=mXAxpK0cWcc865i		管理者
職　　名	特任講師		氏　　名	松本　正晴	
所　　属	東京大学 情報理工学系研究科 コンピュータ科学専攻 (		電話番号	03-5841-4283	
研究分野	4601	計算科学	メールアドレス	matsumoto@is.s.u-tokyo.ac.jp	

Group name in this lecture: gi16

Creating SSH Keys on Linux/Mac/Cygwin

- Open Linux/Mac Terminal or Cygwin Terminal on Window OS
- Command for creating keys `(ssh-keygen -t rsa)`
  1. RETURN
  2. Input your favorite passphrase
  3. Input the passphrase again

***Be sure to set passphrase for the security!**
Do not use SSH key without passphrase.

Creating SSH Keys on Linux/Mac/Cygwin

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/matsumoto/.ssh/id_rsa): ← 1. RETURN
Enter passphrase (empty for no passphrase): ← 2. Input your favorite passphrase
Enter same passphrase again: ← 3. Input the passphrase again
Your identification has been saved in /home/matsumoto/.ssh/id_rsa.
Your public key has been saved in /home/matsumoto/.ssh/id_rsa.pub.
The key fingerprint is: …
The key's randomart image is:
…

$ cd ~/.ssh
$ cd ls -l
total 12
-rw-------+ 1 matsumoto  sudalab  1766 May 28 17:14 id_rsa
-rw-r--r--+ 1 matsumoto  sudalab   416 May 28 17:14 id_rsa.pub

$ cat id_rsa.pub

(copy & paste to the User Portal)
```

Registration of the Public Key

1. Launch your web browser and browse the user portal (https://reedbush-www.cc.u-tokyo.ac.jp/)
2. Login to the user portal with your account ID and initial password*

*The printed password is NOT a collect password

利用者番号	t03000	初期パスワード※1	=mXAxpK0cWcc865i		管理者
職　　名	特任講師		氏　　名	松本　正晴	
所　　属	東京大学 情報理工学系研究科 コンピュータ科学専攻 (		電話番号	03-5841-4283	
研 究 分 野	4601	計算科学	メール アドレス	matsumoto@is.s.u-tokyo.ac.jp	

3. Change your initial password to your new own**

**For the security, you should use a different password from your SSH key's passphrase created previously.

Registration of the Public Key

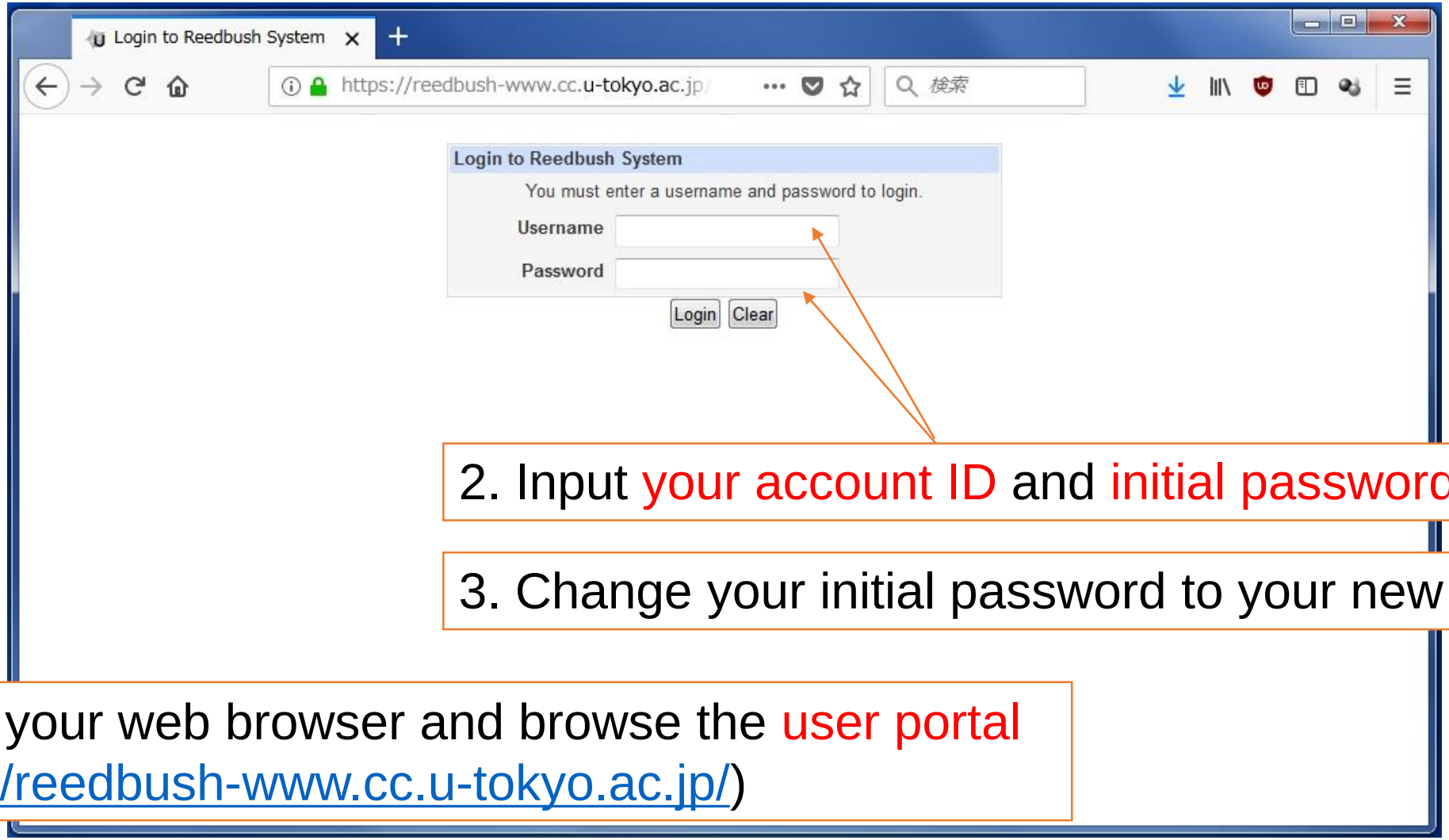
4. Login to the user portal again with new password

5. Select “SSH Configulation”

6. Copy and paste the previously created public key to the user portal

7. Click “Create” button

User Portal

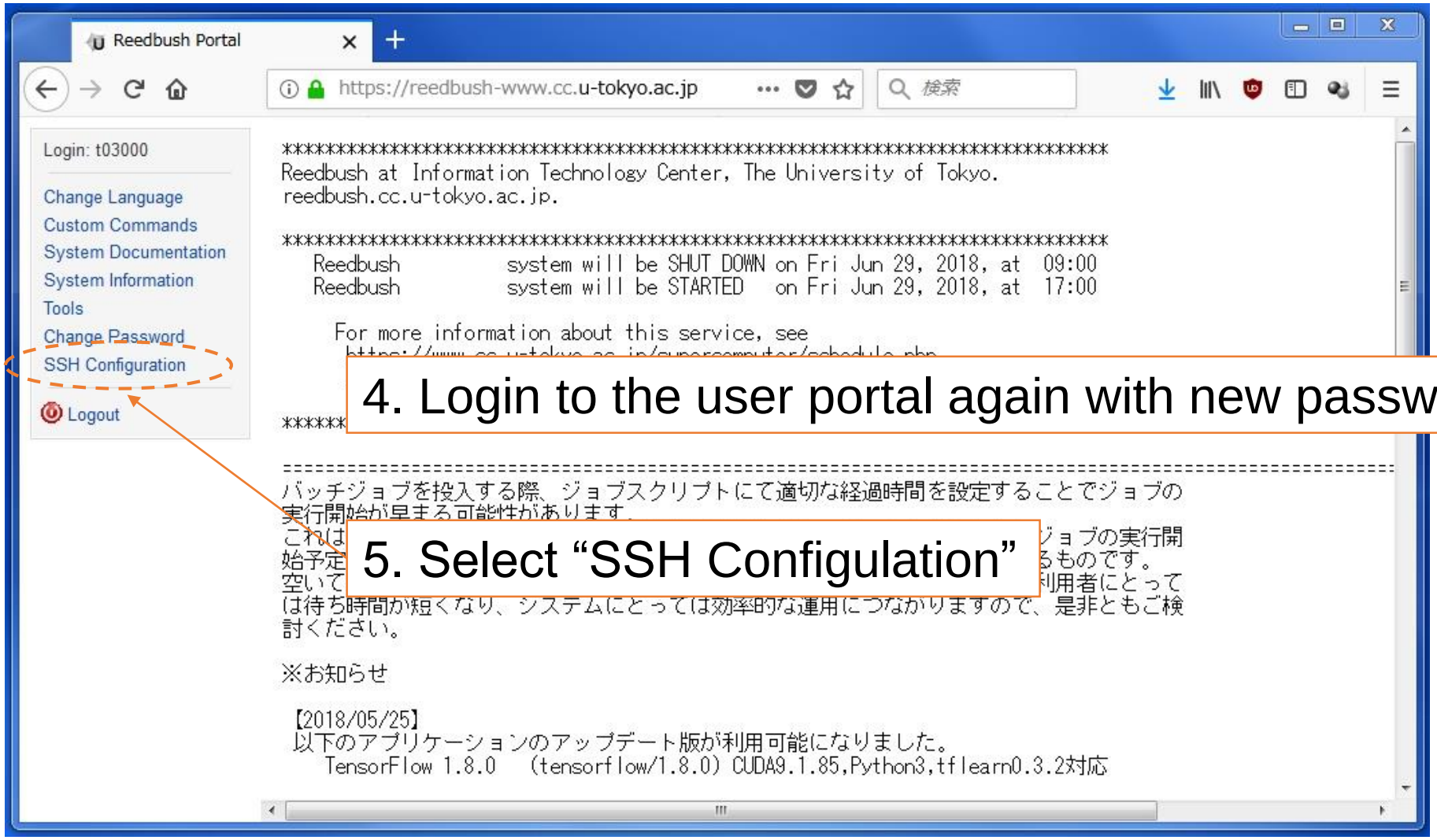


2. Input your account ID and initial password

3. Change your initial password to your new own

1. Launch your web browser and browse the user portal (https://reedbush-www.cc.u-tokyo.ac.jp/)

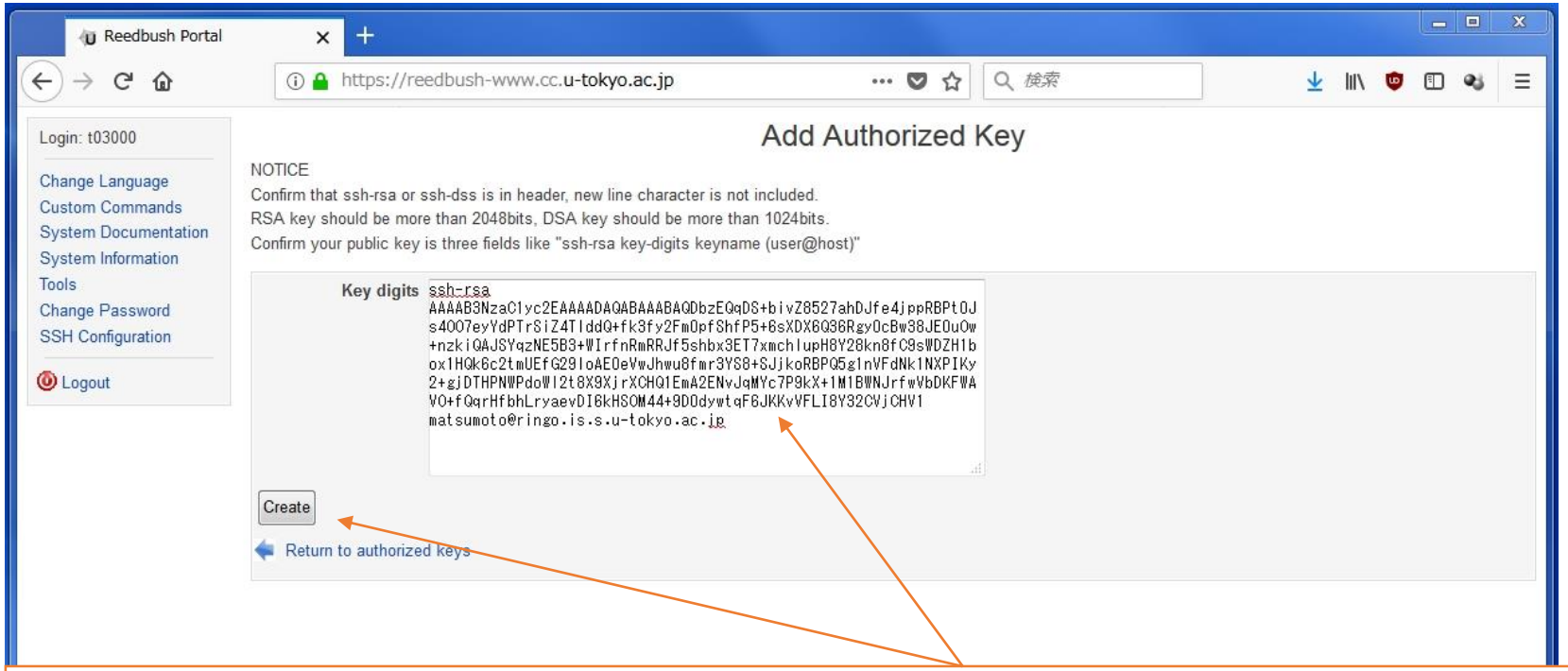
User Portal



4. Login to the user portal again with new password

5. Select “SSH Configulation”

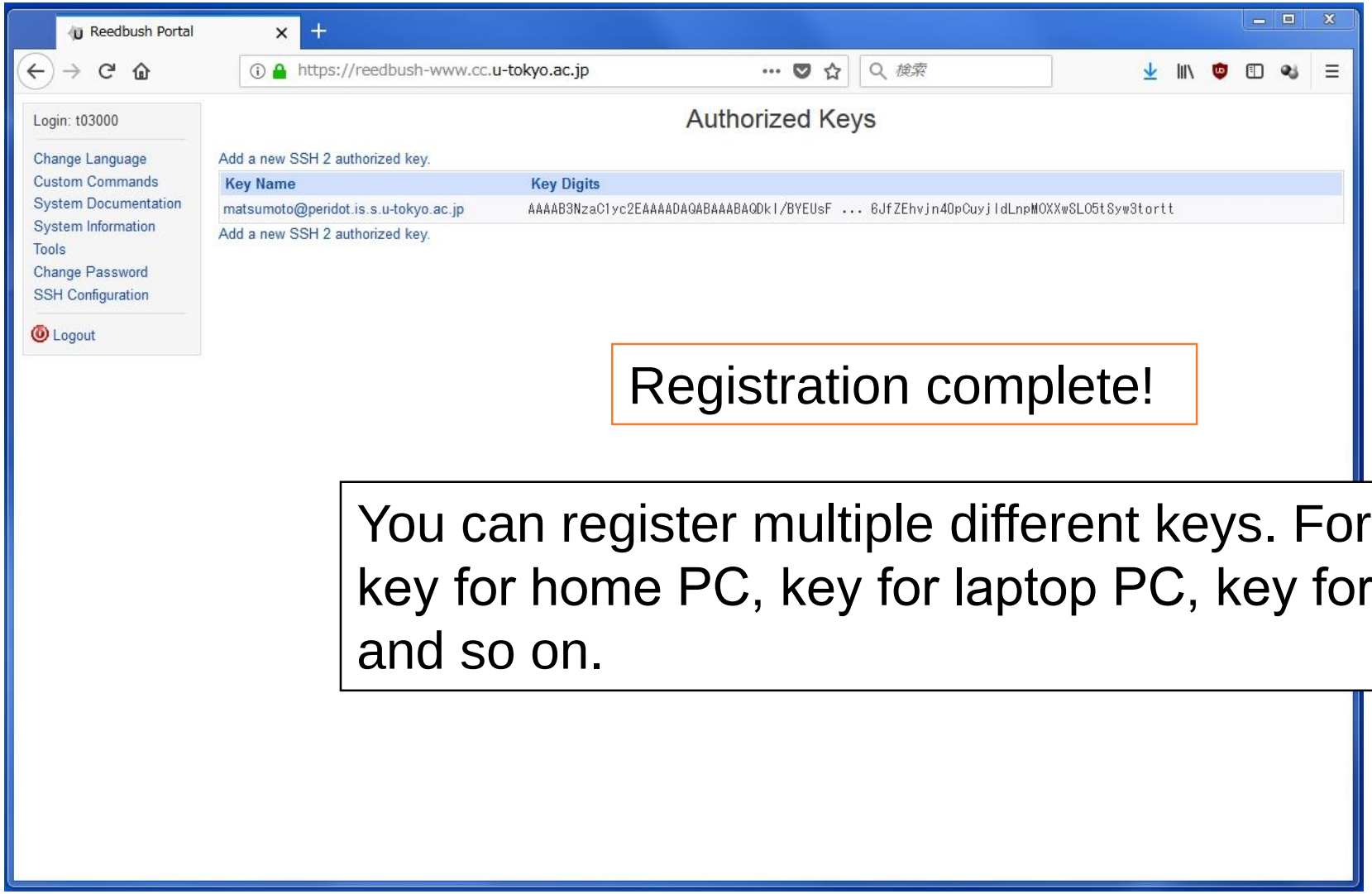
User Portal



6. Copy and paste the previously created public key to the user portal

7. Click “Create” button

User Portal



Registration complete!

You can register multiple different keys. For example, key for home PC, key for laptop PC, key for lab's PC… and so on.

Login to Reedbush

- Open a terminal on local PC and enter the following

```
$ ssh XXXXXX@reedbush.cc.u-tokyo.ac.jp
```

or

```
$ ssh reedbush.cc.u-tokyo.ac.jp -l XXXXXX
```

```
matsumoto@local:~$ ssh t09XXX@reedbush.cc.u-tokyo.ac.jp
The authenticity of host 'reedbush.cc.u-tokyo.ac.jp (130.69.241.12)' can't be established.
ECDSA key fingerprint is SHA256:IW94eJSo+B4nI/tpFd/17xfawCof4d36SISZyt7tgHY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'reedbush.cc.u-tokyo.ac.jp' (ECDSA) to the list of known hosts.
Enter passphrase for key '/home/matsumoto/.ssh/id_rsa':(Enter SSH key's passphrase created previously)
Last login: Thu May 31 14:34:05 2018 from 133.11.33.188
*********************************************************************
   Reedbush          system will be SHUT DOWN on Fri Jun 29, 2018, at  09:00
   Reedbush          system will be STARTED   on Fri Jun 29, 2018, at  17:00

     For more information about this service, see
      https://www.cc.u-tokyo.ac.jp/supercomputer/schedule.php
      https://www.cc.u-tokyo.ac.jp/guide/hpc/rbh/
*********************************************************************
[t09XXX@reedbush-u2 ~]$
```

Login successful

Home Directory and Working Directory on Reedbush

- `/home/gi16/XXXXXX`
  - ✓ Default directory when you login
  - ✓ `$ cd`
  - ✓ Max. 2GB
  - ✓ Home directory is unavailable to run batch job because compute node doesn't mount it.
- `/lustre/gi16/XXXXXX`
  - ✓ Working directory
  - ✓ `$ cdw`
  - ✓ 8TB can be used in `gi16` group
  - ✓ Build and run your program at this directory

Data Download from Reedbush to Local PC

- Use a "scp" command on a terminal <u>on local PC</u>

space

```
$ scp XXXXXX@reedbush.cc.u-tokyo.ac.jp:~/filename ./
```

- `XXXXXX` is your account ID
- A file "*`filename`*" on home directory (`~/`) of Reedbush can be downloaded to a current directory (`./`) of local PC.
- Any directory can be specified as follows

```
$ scp XXXXXX@reedbush.cc.u-tokyo.ac.jp:/lustre/gi16/XXXXXX/filename ./home/hoge/
```

- "`-r`" option have to be specified to download a directory "*`dirname`*"

```
$ scp -r XXXXXX@reedbush.cc.u-tokyo.ac.jp:~/dirname ./
```

Data Upload from Local PC to Reedbush

- Also, use a “scp” command on a terminal <u>on local PC</u>

```
$ scp ./filename XXXXXX@reedbush.cc.u-tokyo.ac.jp:
```

- A file “*filename*” on current directory (`./`) of local PC can be uploaded to a home directory of Reedbush.
- Any directory can be specified in the same manner to download
- “`-r`” option have to be specified to upload a directory “*dirname*”

```
$ scp -r ./dirname XXXXXX@reedbush.cc.u-tokyo.ac.jp:
```

Building Program on Reedbush-U (CPU only)

- Default development environment
  - ✓ Intel C, C++ and Fortran Compiler
  - ✓ Intel MPI

- Examples of compile command
  - ✓ Serial job (Intel compiler)
    - Fortran : `$ ifort file.f90`
    - C : `$ icc file.c`
    - C++ : `$ icpc file.cpp`
  - ✓ Parallel job (OpenMP only)
    - Fortran : `$ ifort -qopenmp file.f90`
    - C : `$ icc -qopenmp file.c`
    - C++ : `$ icpc -qopenmp file.cpp`

Building Program on Reedbush-U (CPU only)

- ✓ Parallel job (MPI only)
  - Fortran : `$ mpiifort file.f90`
  - C : `$ mpiicc file.c`
  - C++ : `$ mpiicpc file.cpp`
- ✓ Parallel job (MPI + OpenMP)
  - Fortran : `$ mpiifort -qopenmp file.f90`
  - C : `$ mpiicc -qopenmp file.c`
  - C++ : `$ mpiicpc -qopenmp file.cpp`

If you success the compiling of source files, you can obtain an executable file "a.out".

Building Program on Reedbush-H (with GPU)

- You can switch compiler and MPI environment with `module` command. When you use GPU, you have to load CUDA and PGI compiler environment.

```
$ module load cuda
$ module load pgi
```

  - CUDA C

    ```
    $ nvcc -gencode arch=compute_60,code=sm_60 [options] file.cu
    ```

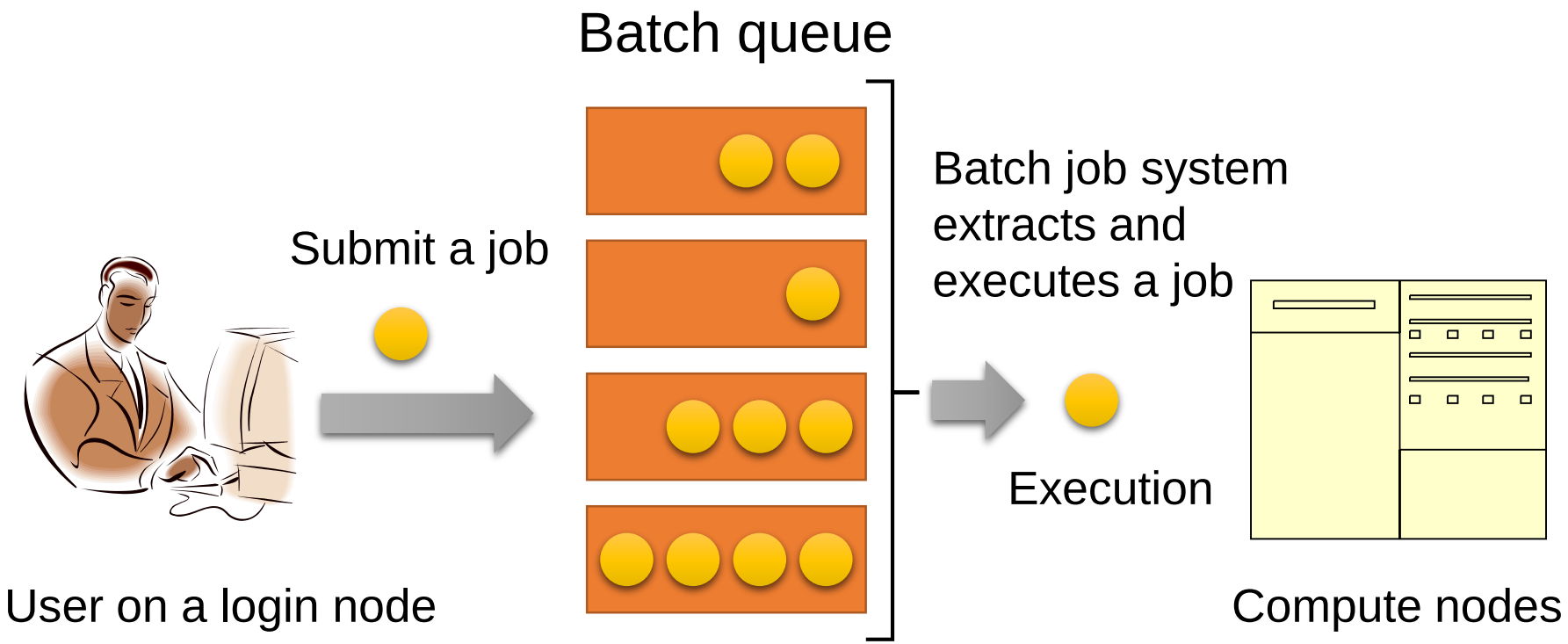
  - CUDA Fortran

    ```
    $ pgfortran -Mcuda=cc60 [options] file.cuf
    ```

  - OpenACC
    - Fortran: `$ pgfortran -acc -ta=tesla,cc60 [options] file.f90`
    - C: `$ pgcc -acc -ta=tesla,cc60 [options] file.c`
    - C++: `$ pgc++ -acc -ta=tesla,cc60 [options] file.cpp`

Batch Job System

A supercomputer is generally crowded because many people use simultaneously. In order to execute an executable file (= a job) on Reedbush, you have to submit the job to the compute nodes (–U or –H).



Batch Script Sample

In order to submit a job to compute nodes, you have to make a batch script file (e.g., `run.sh`) as follows.

- For a serial job

```
$ cd /lustre/gi16/XXXXXX
$ cat ./run.sh

#!/bin/sh
#PBS -q u-debug
#PBS -W group_list=gi16
#PBS -l select=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

./a.out
```

Queue name

Group name

Time limit

Computational Resources you request

`select`:
The number of nodes

`mpiprocs`:
The number of MPI processes per node

`ompthreads`:
The number of OpenMP threads per MPI process

Batch Script Sample

In order to submit a job to compute nodes, you have to make a batch script file (e.g., `run.sh`) as follows.

- For a serial job

```
$ cd /lustre/gi16/XXXXXX
$ cat ./run.sh

#!/bin/sh
#PBS -q u-debug
#PBS -W group_list=gi16
#PBS -l select=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

./a.out
```

Queue name

Queue name	Number of nodes	Walltime
u-debug	1-24	30min
(u-interactive)	-	-
u-interactive_1	1	30min
u-interactive_4	2-4	10min
u-short	8	4H
(u-regular)	-	-
u-small	4-16	48H
u-medium	17-32	48H
u-large	33-64	48H
u-x-large	65-128	24H

Batch Script Sample

- For 288 MPI job
  (8 nodes x 36 processes/node)

```
$ cd /lustre/gi16/XXXXXX
$ cat ./run.sh

#!/bin/sh
#PBS -q u-debug
#PBS -W group_list=gi16
#PBS -l select=8:mpiprocs=36:ompthreads=1
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

mpirun ./a.out
```

- For hybrid parallel job
  (8 nodes x 2 processes/node x
  18 threads/proc)

```
$ cd /lustre/gi16/XXXXXX
$ cat ./run.sh

#!/bin/sh
#PBS -q u-debug
#PBS -W group_list=gi16
#PBS -l select=8:mpiprocs=2:ompthreads=18
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

mpirun ./a.out
```

How to submit a job (1/2)

If you compile source files and make a batch script on /lustre directory, you can submit the job (batch script) by "`qsub`" command.

```
$ qsub run.sh
XXXXXXX.reedbush-pdsadmin0
```

Job ID (7-digit) is returned.

You can confirm the submitted job status by "`rbstat`" command.

```
$ rbstat
JOB_ID    JOB_NAME STATUS   PROJECT QUEUE     START_DATE     ELAPSE   TOKEN NODE
1234567   run.sh   RUNNING  gt09    u-lecture 06/27 19:56:41 00:00:30 0.0   1
1234568   run.sh   QUEUED   gt09    u-lecture 06/27 21:00:00 00:00:00 0.0   1
```

How to submit a job (2/2)

You can delete unnecessary jobs by “`qdel`” command.

```
$ rbstat
JOB_ID    JOB_NAME STATUS   PROJECT QUEUE      START_DATE      ELAPSE    TOKEN NODE
1234567   run.sh   RUNNING  gt09    u-lecture  06/27 19:56:41  00:00:30  0.0   1
1234568   run.sh   QUEUED   gt09    u-lecture  06/27 21:00:00  00:00:00  0.0   1

$ qdel 1234568

$ rbstat
JOB_ID    JOB_NAME STATUS   PROJECT QUEUE      START_DATE      ELAPSE    TOKEN NODE
1234567   run.sh   RUNNING  gt09    u-lecture  06/27 19:56:41  00:00:38  0.0   1
```

Execution of Sample Program on Reedbush-U (CPU only)

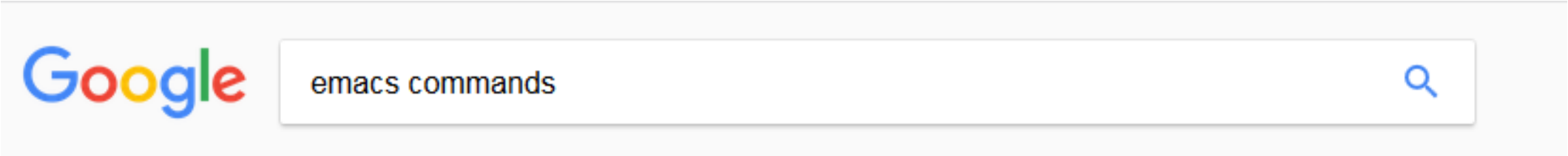
Basic Unix Commands

Command	Example	Description
ls	`ls` `ls -alF`	Lists files in current directory List in long format
cd	`cd tempdir` `cd ..`	Change directory to tempdir Move back one directory
mkdir	`mkdir graphics`	Make a directory called graphics
rmdir	`rmdir emptydir`	Remove directory (must be empty)
cp	`cp file1 web-docs` `cp file1 file1.bak`	Copy file into directory Make backup of file1
rm	`rm file1.bak` `rm *.tmp`	Remove or delete file Remove all file
mv	`mv old.html new.html`	Move or rename files
less	`less index.html`	Look at file, one page at a time. `Space key` to scroll, `q` to quit.
man	`man ls`	Online manual (help) about command

Emacs Text Editor

`$ emacs filename`	start emacs editor
• `<Ctrl>-x <Ctrl>-s`	save the file
• `<Ctrl>-x <Ctrl>-c`	close down emacs

For more information, please look it up yourself.



Of course, you can use other editors.

Compiling of Parallelized Hello Program (1/2)

1. Move to /lustre directory.
   ```
   $ cd /lustre/gi16/XXXXXX/
   ```
2. Copy Samples.tar on /lustre/gi16/c26050 to your own directory.
   ```
   $ cp /lustre/gi16/c26050/Samples.tar ./
   ```
3. Extract files from Samples.tar
   ```
   $ tar xvf Samples.tar
   ```
4. Move to Samples/ directory
   ```
   $ cd Samples
   ```
   for C users : `$ cd C`

   for Fortran users : `$ cd F`
5. Move to Hello/ directory
   ```
   $ cd Hello
   ```

Compiling of Parallelized Hello Program (2/2)

6. Compile the source file

   for C :`$ mpiicc hello.c`

   for Fortran :`$ mpiifort hello.f90`

7. Confirm the executable file (a.out)

   `$ ls`

Execution of Parallelized Hello Program

1. Submit the job in Hello/ directory
   `$ qsub run.sh`
2. Confirm the status of submitted job
   `$ rbstat`
3. After the execution, the following files are generated.
   `run.sh.exxxxxx`
   `run.sh.oxxxxxx` (`xxxxxx` is Job ID)
4. See the standard output file
   `cat run.sh.oxxxxxx`
5. If 288 lines (= 8 nodes x 36 processes) of "`Hello parallel world!`" are written in run.sh.oxxxxxx, the execution is successful.

Standard Output and Standard Error

When the execution of a batch job is finished, a standard output file and a standard error file are generated. Standard output and standard error during the job are written in each file.

Name of batch script.oXXXXXX --- standard output file
Name of batch script.eXXXXXX --- standard error file
(XXXXXX is Job ID)

Parallelized Hello Program（C）

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

int main(int argc, char *argv[]){
  int myid,nprc,ierr;

  ierr=MPI_Init(&argc,&argv);
  ierr=MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  ierr=MPI_Comm_size(MPI_COMM_WORLD,&nprc);

  printf("Hello parallel world! MyID: %d %d¥n",myid, nprc);

  ierr=MPI_Finalize();
  return 0;
}
```

MPI init

Get my rank ID

Get the number of total processes

MPI finilize

Parallelized Hello Program（Fortran）

```
program main
  implicit none
  include "mpif.h"

  integer::myid,nprc,ierr

  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,myid,ierr)
  call mpi_comm_size(mpi_comm_world,nprc,ierr)

  print *, "Hello parallel world! MyID:", myid, nprc

  call mpi_finalize(ierr)

  stop
end program main
```

MPI init

Get my rank ID

Get the number of total processes

MPI finilize

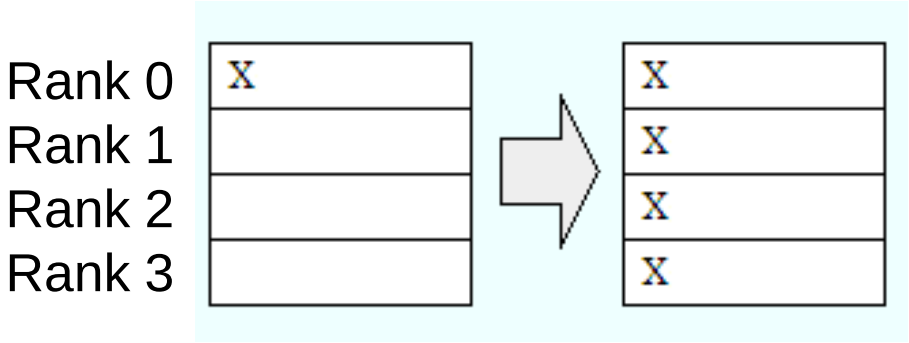
Sample programs using various MPI functions

Source files are in `MPI_func/` directory in `Samples.tar`
You can compile and execute these programs in the same way as the parallelized hello program.

1. MPI_Bcast(C)

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

int main(int argc,char *argv[]) {
  int i,rank,size;
  int data[4]={0,0,0,0};
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  if(rank==0){
    for(i=0;i<4;i++) data[i]=i+1;
  }
  printf("rank%d: before [%d %d %d %d]¥n",
         rank,data[0],data[1],data[2],data[3]);
  MPI_Bcast(data,4,MPI_INT,0,MPI_COMM_WORLD);
  printf("rank%d: after [%d %d %d %d]¥n",
         rank,data[0],data[1],data[2],data[3]);
  MPI_Finalize();
}
```

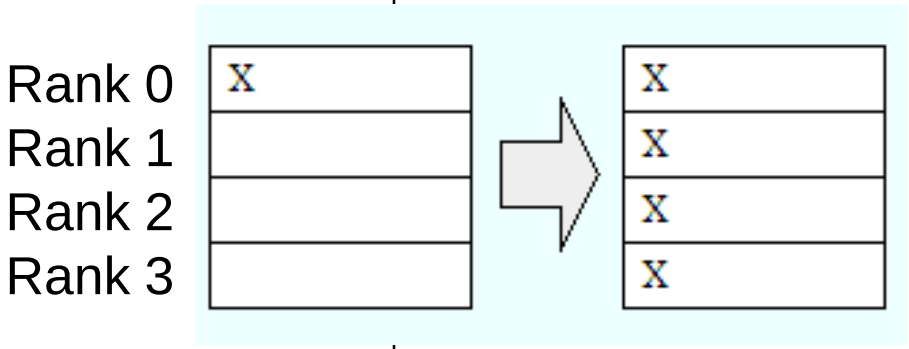


Result (4 MPI processes)

```
rank0: before [1 2 3 4]
rank0: after [1 2 3 4]
rank1: before [0 0 0 0]
rank1: after [1 2 3 4]
rank2: before [0 0 0 0]
rank2: after [1 2 3 4]
rank3: before [0 0 0 0]
rank3: after [1 2 3 4]
```

1. MPI_Bcast(fortran)

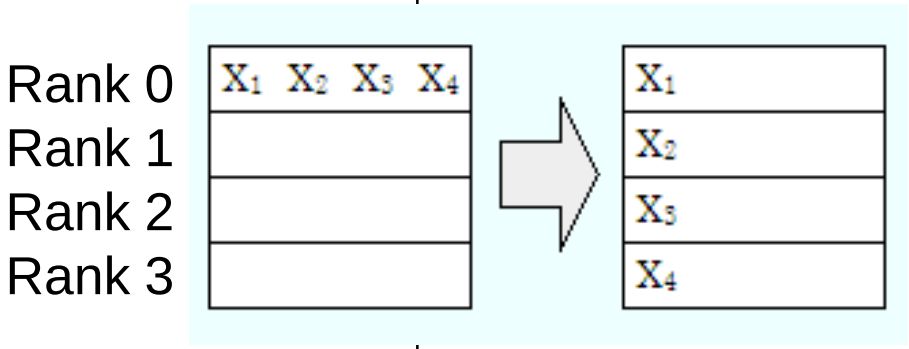
```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::data(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  data(:)=0.0d0
  if(rank==0) then
     do i=1,4
        data(i)=i
     enddo
  endif
  write(*,'("rank",i1,": before [",i1,1x,i1,1x,i1,1x,i1,"]")') &
       rank,data(1),data(2),data(3),data(4)
  call mpi_bcast(data,4,mpi_integer,0,mpi_comm_world,ierr)
  write(*,'("rank",i1,": after [",i1,1x,i1,1x,i1,1x,i1,"]")') &
       rank,data(1),data(2),data(3),data(4)
  call mpi_finalize(ierr)
end program main
```



2. MPI_Scatter(C)

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

int main(int argc,char *argv[]){
  int i,rank,size;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  if(rank==0){
    for(i=0;i<4;i++) send[i]=i+1;
  }
  MPI_Scatter(send,1,MPI_INT,recv,1,MPI_INT,0,MPI_COMM_WORLD);
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",rank,
         send[0],send[1],send[2],send[3],
         recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

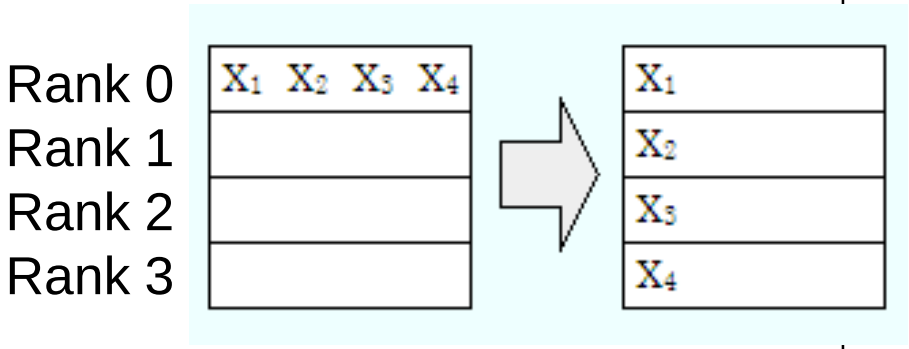


Result (4 MPI processes)

```
rank0: send=[1 2 3 4], recv=[1 0 0 0]
rank1: send=[0 0 0 0], recv=[2 0 0 0]
rank2: send=[0 0 0 0], recv=[3 0 0 0]
rank3: send=[0 0 0 0], recv=[4 0 0 0]
```

2. MPI_Scatter(fortran)

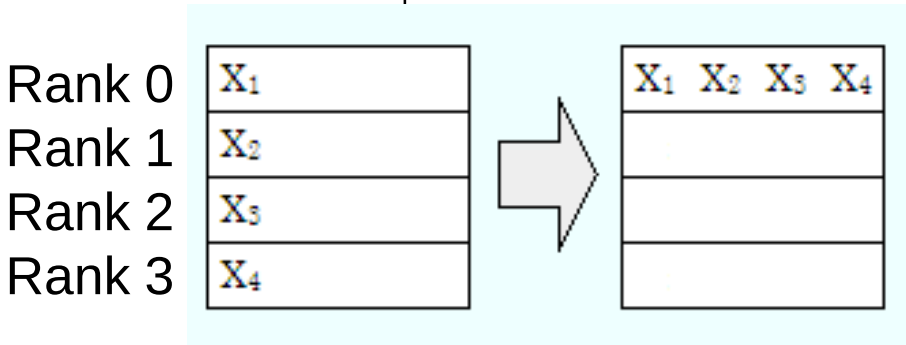
```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::send(4),recv(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  if(rank==0) then
     do i=1,4
        send(i)=i
     enddo
  endif
  call mpi_scatter(send,1,mpi_integer,recv,1,mpi_integer,0,mpi_comm_world,ierr)
  write(*,'("rank",i1,": send=[",i1,1x,i1,1x,i1,1x,i1,"], recv=[",&
       i1,1x,i1,1x,i1,1x,i1,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



3. MPI_Gather(C)

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

int main(int argc,char *argv[]){
  int rank,size;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  send[0]=rank+1;
  MPI_Gather(send,1,MPI_INT,recv,1,MPI_INT,0,MPI_COMM_WORLD);
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",rank,
         send[0],send[1],send[2],send[3],
         recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

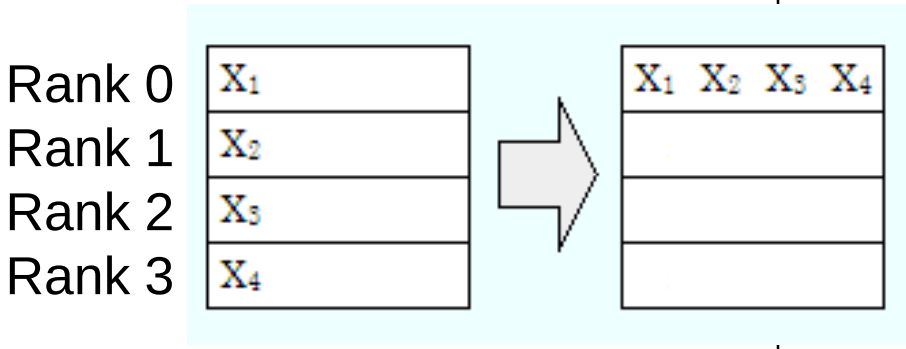


Result (4 MPI processes)

```
rank0: send=[1 0 0 0], recv=[1 2 3 4]
rank1: send=[2 0 0 0], recv=[0 0 0 0]
rank2: send=[3 0 0 0], recv=[0 0 0 0]
rank3: send=[4 0 0 0], recv=[0 0 0 0]
```

3. MPI_Gather(fortran)

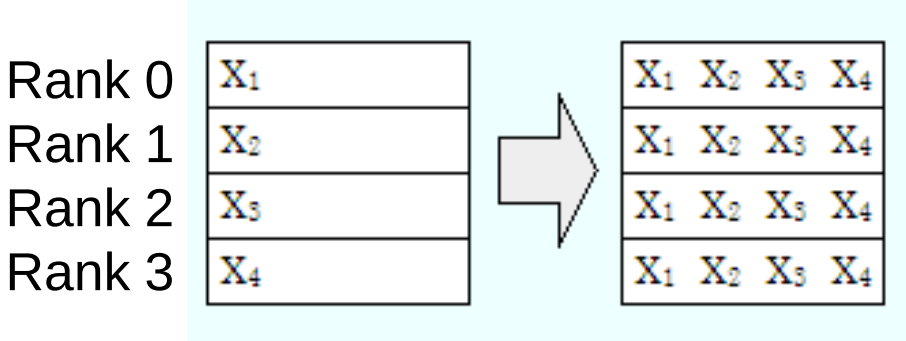
```
program main
  implicit none
  include"mpif.h"
  integer::rank,size,ierr
  integer::send(4),recv(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  send(1)=rank+1
  call mpi_gather(send,1,mpi_integer,recv,1,mpi_integer,0,mpi_comm_world,ierr)
  write(*,'("rank",i1,": send=[",i1,1x,i1,1x,i1,1x,i1,"], recv=[",&
       i1,1x,i1,1x,i1,1x,i1,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



4. MPI_Allgather(C)

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

int main(int argc, char *argv[]){
  int rank,size;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  send[0]=rank+1;
  MPI_Allgather(send,1,MPI_INT,recv,1,MPI_INT,MPI_COMM_WORLD);
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",rank,
         send[0],send[1],send[2],send[3],
         recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

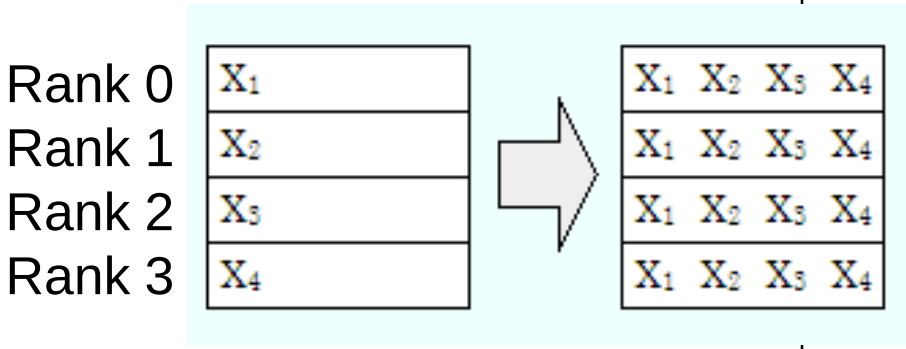


Result (4 MPI processes)

```
rank0: send=[1 0 0 0], recv=[1 2 3 4]
rank1: send=[2 0 0 0], recv=[1 2 3 4]
rank2: send=[3 0 0 0], recv=[1 2 3 4]
rank3: send=[4 0 0 0], recv=[1 2 3 4]
```

4. MPI_Allgather(fortran)

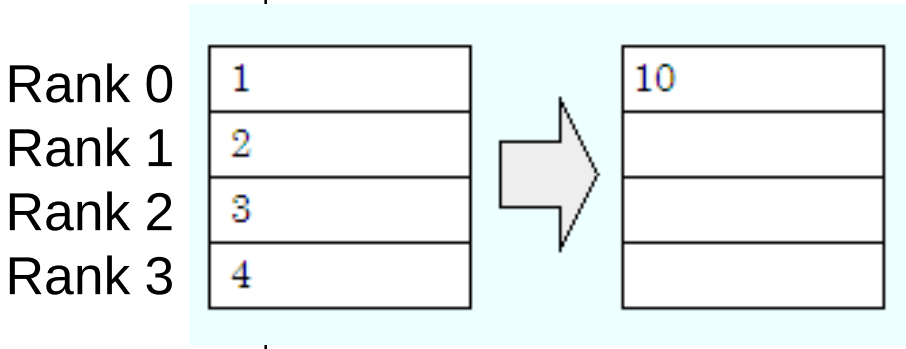
```
program main
  implicit none
  include"mpif.h"
  integer::rank,size,ierr
  integer::send(4),recv(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  send(1)=rank+1
  call mpi_allgather(send,1,mpi_integer,recv,1,mpi_integer,mpi_comm_world,ierr)
  write(*,'("rank",i1,": send=[",i1,1x,i1,1x,i1,1x,i1,"], recv=[",&
       i1,1x,i1,1x,i1,1x,i1,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



5. MPI_Reduce(C)

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

int main(int argc, char *argv[]){
  int i,rank,size;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for(i=0;i<4;i++) send[i]=rank+i;
  MPI_Reduce(send,recv,4,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",
         rank,send[0],send[1],send[2],send[3],
         recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

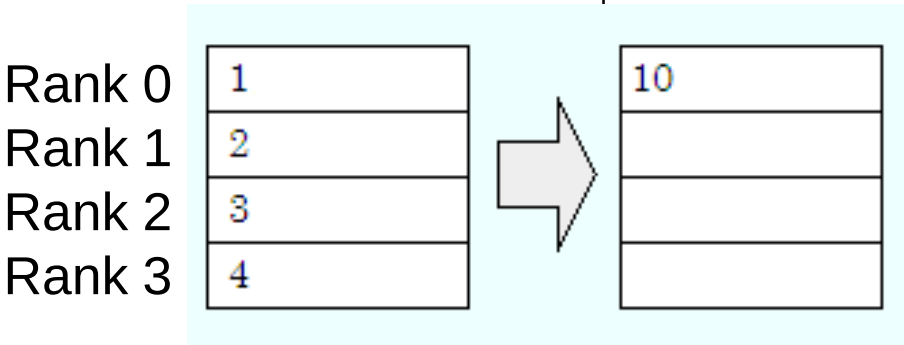


Result (4 MPI processes)

```
rank0: send=[0 1 2 3], recv=[6 10 14 18]
rank1: send=[1 2 3 4], recv=[0 0 0 0]
rank2: send=[2 3 4 5], recv=[0 0 0 0]
rank3: send=[3 4 5 6], recv=[0 0 0 0]
```

5. MPI_Reduce（fortran）

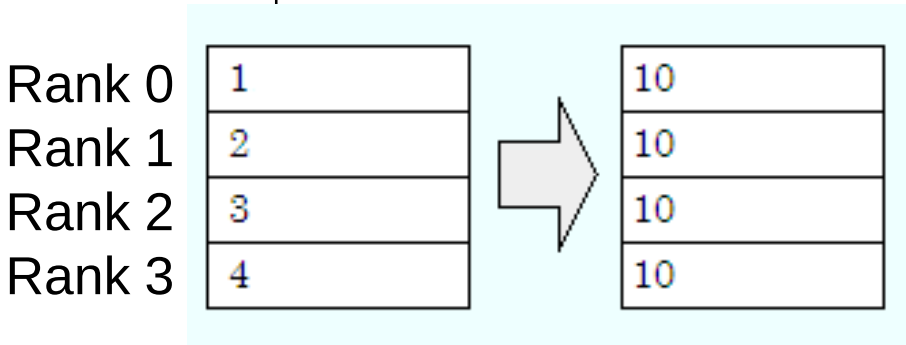
```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::send(4),recv(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  do i=1,4
     send(i)=rank+1
  enddo
  call mpi_reduce(send,recv,4,mpi_integer,mpi_sum,0,mpi_comm_world,ierr)
  write(*,'("rank",i1,": send=[",i1,1x,i1,1x,i1,1x,i1,"], recv=[",&
       i2,1x,i2,1x,i2,1x,i2,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



6. MPI_Allreduce(C)

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

int main(int argc,char *argv[]){
  int i,rank,size;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for(i=0;i<4;i++) send[i]=rank+i;
  MPI_Allreduce(send,recv,4,MPI_INT,MPI_SUM,MPI_COMM_WORLD);
  printf("rank%d: send=[%d %d %d %d], recv[%d %d %d %d]¥n",
         rank,send[0],send[1],send[2],send[3],
         recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

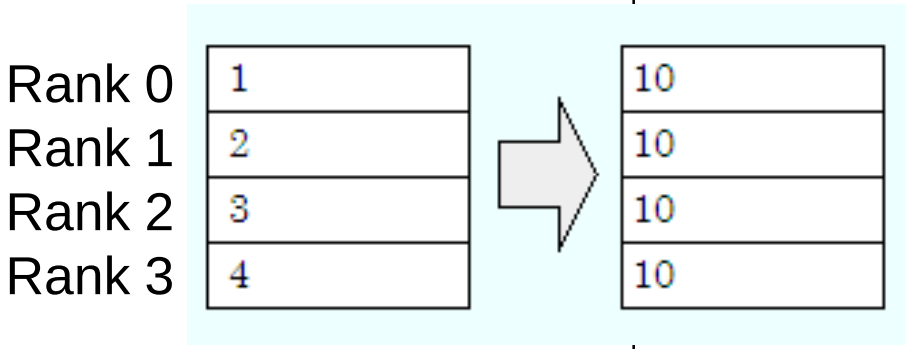


Result (4 MPI processes)

```
rank1: send=[1 2 3 4], recv[6 10 14 18]
rank2: send=[2 3 4 5], recv[6 10 14 18]
rank3: send=[3 4 5 6], recv[6 10 14 18]
rank0: send=[0 1 2 3], recv[6 10 14 18]
```

6. MPI_Allreduce（fortran）

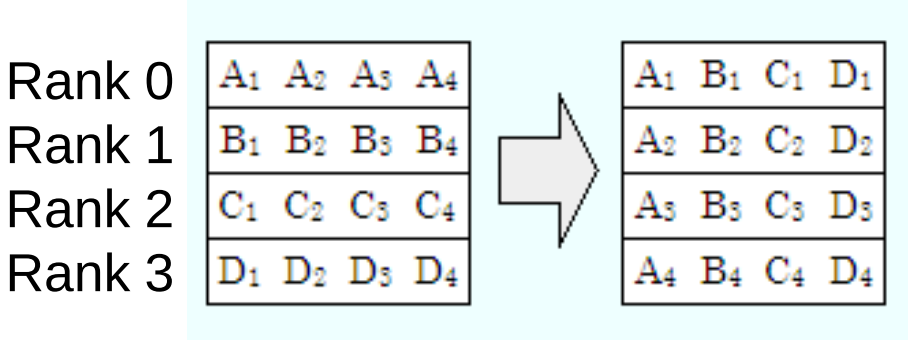
```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::send(4),recv(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  do i=1,4
     send(i)=rank+1
  enddo
  call mpi_allreduce(send,recv,4,mpi_integer,mpi_sum,mpi_comm_world,ierr)
  write(*,'("rank",i1,": send=[",i1,1x,i1,1x,i1,1x,i1,"], recv=[",&
       i2,1x,i2,1x,i2,1x,i2,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



7. MPI_Alltoall(C)

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

int main(int argc,char *argv[]){
  int i,rank,size;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for(i=0;i<4;i++) send[i]=rank+10*i;
  MPI_Alltoall(send,1,MPI_INT,recv,1,MPI_INT,MPI_COMM_WORLD);
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",rank,
         send[0],send[1],send[2],send[3],
         recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

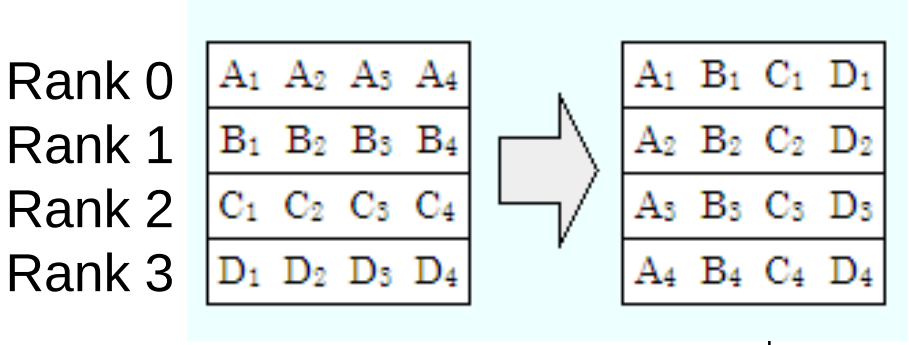


Result (4 MPI processes)

```
rank0: send=[0 10 20 30], recv=[0 1 2 3]
rank1: send=[1 11 21 31], recv=[10 11 12 13]
rank2: send=[2 12 22 32], recv=[20 21 22 23]
rank3: send=[3 13 23 33], recv=[30 31 32 33]
```

7. MPI_Alltoall(fortran)

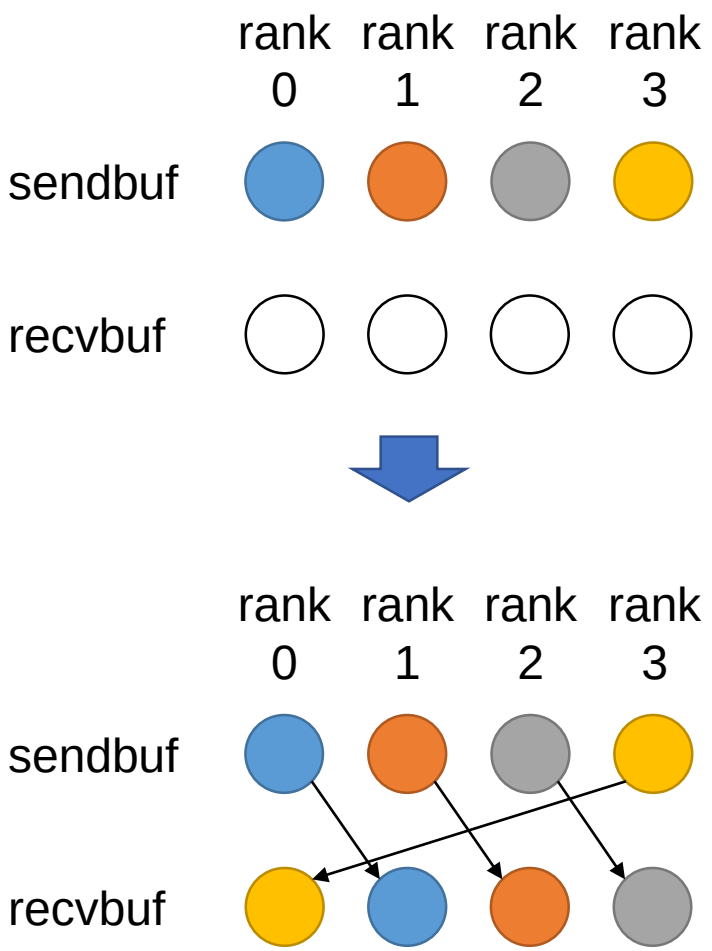
```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::send(4),recv(4)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  do i=1,4
     send(i)=rank+10*(i-1)
  enddo
  call mpi_alltoall(send,1,mpi_integer,recv,1,mpi_integer,mpi_comm_world,ierr)
  write(*,'("rank",i1,": send=[",i2,1x,i2,1x,i2,1x,i2,"], recv=[",&
       i2,1x,i2,1x,i2,1x,i2,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



8. MPI_Send/MPI_Recv(C)

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

int main(int argc,char *argv[]){
  int i,size,rank;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Status stat;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for(i=0;i<4;i++) send[i]=rank+10*i;
  if(rank==0){
    MPI_Send(send,4,MPI_INT,1,0,MPI_COMM_WORLD);
    MPI_Recv(recv,4,MPI_INT,3,3,MPI_COMM_WORLD,&stat);
  }else if(rank==1){
    MPI_Send(send,4,MPI_INT,2,1,MPI_COMM_WORLD);
    MPI_Recv(recv,4,MPI_INT,0,0,MPI_COMM_WORLD,&stat);
  }else if(rank==2){
    MPI_Send(send,4,MPI_INT,3,2,MPI_COMM_WORLD);
    MPI_Recv(recv,4,MPI_INT,1,1,MPI_COMM_WORLD,&stat);
  }else if(rank==3){
    MPI_Send(send,4,MPI_INT,0,3,MPI_COMM_WORLD);
    MPI_Recv(recv,4,MPI_INT,2,2,MPI_COMM_WORLD,&stat);
  }
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",rank,
         send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

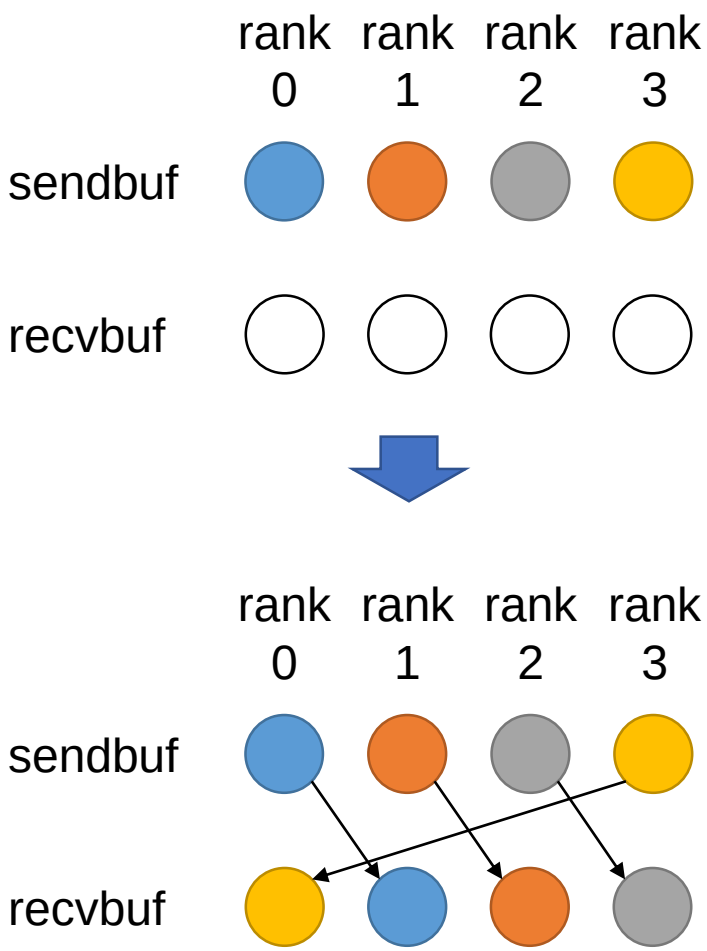


Result (4 MPI processes)

```
rank0: send=[0 10 20 30], recv=[3 13 23 33]
rank1: send=[1 11 21 31], recv=[0 10 20 30]
rank2: send=[2 12 22 32], recv=[1 11 21 31]
rank3: send=[3 13 23 33], recv=[2 12 22 32]
```

8. MPI_Send/MPI_Recv(fortran)

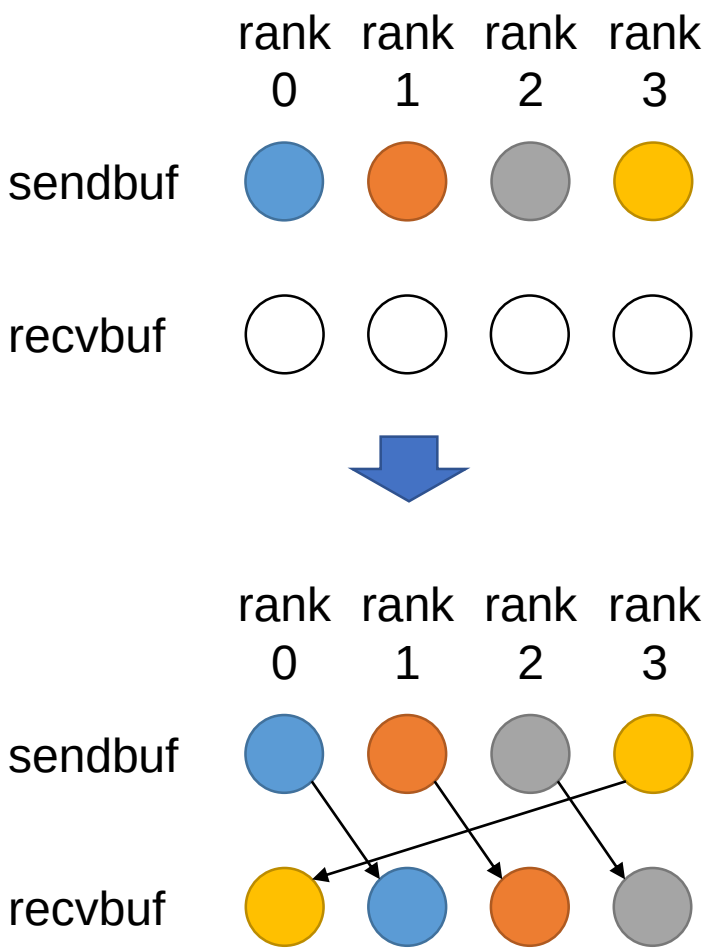
```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::send(4),recv(4)
  integer::stat(mpi_status_size)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  do i=1,4
     send(i)=rank+10*(i-1)
  enddo
  if(rank==0) then
     call mpi_send(send,4,mpi_integer,1,0,mpi_comm_world,ierr)
     call mpi_recv(recv,4,mpi_integer,3,3,mpi_comm_world,stat(:),ierr)
  else if(rank==1) then
     call mpi_send(send,4,mpi_integer,2,1,mpi_comm_world,ierr)
     call mpi_recv(recv,4,mpi_integer,0,0,mpi_comm_world,stat(:),ierr)
  else if(rank==2) then
     call mpi_send(send,4,mpi_integer,3,2,mpi_comm_world,ierr)
     call mpi_recv(recv,4,mpi_integer,1,1,mpi_comm_world,stat(:),ierr)
  else if(rank==3) then
     call mpi_send(send,4,mpi_integer,0,3,mpi_comm_world,ierr)
     call mpi_recv(recv,4,mpi_integer,2,2,mpi_comm_world,stat(:),ierr)
  endif
  write(*,'("rank",i1,": send=[",i2,1x,i2,1x,i2,1x,i2,"], recv=[",&
       i2,1x,i2,1x,i2,1x,i2,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
end program main
```



9. MPI_Isend/MPI_Irecv/MPI_Waitall(C)

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

int main(int argc,char *argv[]){
  int i,size,rank;
  int send[4]={0,0,0,0},recv[4]={0,0,0,0};
  MPI_Request ireq[2];
  MPI_Status stat[2];
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for(i=0;i<4;i++) send[i]=rank+10*i;
  if(rank==0){
    MPI_Isend(send,4,MPI_INT,1,0,MPI_COMM_WORLD,&ireq[0]);
    MPI_Irecv(recv,4,MPI_INT,3,3,MPI_COMM_WORLD,&ireq[1]);
  }else if(rank==1){
    MPI_Isend(send,4,MPI_INT,2,1,MPI_COMM_WORLD,&ireq[0]);
    MPI_Irecv(recv,4,MPI_INT,0,0,MPI_COMM_WORLD,&ireq[1]);
  }else if(rank==2){
    MPI_Isend(send,4,MPI_INT,3,2,MPI_COMM_WORLD,&ireq[0]);
    MPI_Irecv(recv,4,MPI_INT,1,1,MPI_COMM_WORLD,&ireq[1]);
  }else if(rank==3){
    MPI_Isend(send,4,MPI_INT,0,3,MPI_COMM_WORLD,&ireq[0]);
    MPI_Irecv(recv,4,MPI_INT,2,2,MPI_COMM_WORLD,&ireq[1]);
  }
  MPI_Waitall(2,ireq,stat);
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]¥n",rank,
         send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```

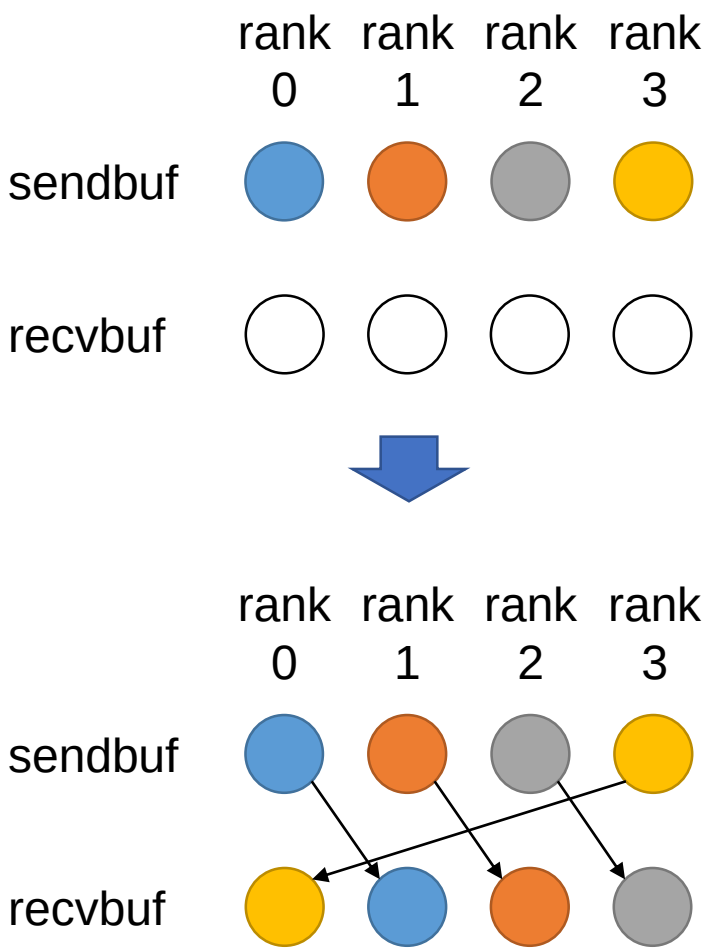


Result (4 MPI processes)

```
rank0: send=[0 10 20 30], recv=[3 13 23 33]
rank1: send=[1 11 21 31], recv=[0 10 20 30]
rank2: send=[2 12 22 32], recv=[1 11 21 31]
rank3: send=[3 13 23 33], recv=[2 12 22 32]
```

9. MPI_Isend/MPI_Irecv/MPI_Waitall（Fortran）

```
program main
  implicit none
  include"mpif.h"
  integer::i,rank,size,ierr
  integer::send(4),recv(4)
  integer::ireq(2),stat(mpi_status_size,2)
  call mpi_init(ierr)
  call mpi_comm_rank(mpi_comm_world,rank,ierr)
  call mpi_comm_size(mpi_comm_world,size,ierr)
  send(:)=0.0d0; recv(:)=0.0d0
  do i=1,4
     send(i)=rank+10*(i-1)
  enddo
  if(rank==0) then
     call mpi_isend(send,4,mpi_integer,1,0,mpi_comm_world,ireq(1),ierr)
     call mpi_irecv(recv,4,mpi_integer,3,3,mpi_comm_world,ireq(2),ierr)
  else if(rank==1) then
     call mpi_isend(send,4,mpi_integer,2,1,mpi_comm_world,ireq(1),ierr)
     call mpi_irecv(recv,4,mpi_integer,0,0,mpi_comm_world,ireq(2),ierr)
  else if(rank==2) then
     call mpi_isend(send,4,mpi_integer,3,2,mpi_comm_world,ireq(1),ierr)
     call mpi_irecv(recv,4,mpi_integer,1,1,mpi_comm_world,ireq(2),ierr)
  else if(rank==3) then
     call mpi_isend(send,4,mpi_integer,0,3,mpi_comm_world,ireq(1),ierr)
     call mpi_irecv(recv,4,mpi_integer,2,2,mpi_comm_world,ireq(2),ierr)
  endif
  call mpi_waitall(2,ireq(1:2),stat(:,1:2),ierr)
  write(*,'("rank",i1,": send=[",i2,1x,i2,1x,i2,1x,i2,"], recv=[",&
       i2,1x,i2,1x,i2,1x,i2,"]")') &
       rank,send(1),send(2),send(3),send(4),&
       recv(1),recv(2),recv(3),recv(4)
  call mpi_finalize(ierr)
```



Execution of Sample Programs for CUDA on Reedbush-H (with GPU)

Execution of a CUDA Program (1/2)

1. Move to CUDA/Hello_CUDA/ directory in Samples.tar.
   `$ cd CUDA/Hello_CUDA/`
2. Load CUDA environments and PGI compiler
   `$ module load cuda pgi`
3. Compile the source file
   for CUDA C : `$ nvcc -gencode arch=compute_60,code=sm_60 hello.cu`
   for CUDA Fortran : `$ pgfortran -Mcuda=cc60 hello.cuf`
4. Submit the job
   `$ qsub run.sh`
5. Confirm the status of submitted job
   `$ rbstat`
6. After the execution, the following files are generated.
   `run.sh.exxxxxx`
   `run.sh.oxxxxxx` (`xxxxxx` is Job ID)
7. See the standard output file
   `cat run.sh.oxxxxxx`

Batch Script for CUDA

```
$ cd /lustre/gi16/XXXXXX
$ cat ./run.sh

#!/bin/sh
#PBS -q h-debug
#PBS -W group_list=gi16
#PBS -l select=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=00:10:00

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

module load cuda pgi

./a.out
```

`h-debug` when you use GPU

`module` command is needed

Parallelized Hello Program for CUDA C

```
#include<stdio.h>

__global__ void helloFromGPU();

int main(int argc,char *argv[]){
  printf("Hello World from CPU¥n");
  helloFromGPU<<<1,128>>>();
  cudaDeviceReset();
  return 0;
}

__global__ void helloFromGPU(){
  printf("Hello World from GPU thread %d¥n",threadIdx.x);
}
```

Parallelized Hello Program for CUDA Fortran

```
module cudakernel
contains
  attributes(global) subroutine helloFromGPU()
    implicit none
    print *,"Hello World from GPU thread",threadIdx%x
    return
  end subroutine helloFromGPU
end module cudakernel

program main
  use cudafor
  use cudakernel
  implicit none
  integer::ierr
  print *,"Hello World from CPU"
  call helloFromGPU<<<1,128>>>()
  ierr=cudaDeviceReset()
end program main
```

sumarray.cu (CUDA C)

```
#include<stdio.h>
#include<stdlib.h>
#include<cuda.h>
#include<cuda_runtime.h>

__global__ void sum_array(int nemax,double *dA,
                          double *dB,double *dC){
  int ni;
  ni=blockIdx.x*blockDim.x+threadIdx.x;
  dC[ni]=dA[ni]+dB[ni];
  return;
}

int main(int argc,char *argv[]){
  const int nemax=4096,thread=1024;
  int ni;
  double A[nemax],B[nemax],C[nemax];
  double *dA,*dB,*dC;
  // -- set initial value --
  srand(248309);
  for(ni=0;ni<nemax;ni++){
    A[ni]=(double)rand()/RAND_MAX;
    B[ni]=(double)rand()/RAND_MAX;
  }
  // -- allocate arrays on device --
  cudaMalloc((double **)&dA,nemax*sizeof(double));
  cudaMalloc((double **)&dB,nemax*sizeof(double));
  cudaMalloc((double **)&dC,nemax*sizeof(double));
  // -- copy memories from host to device --
  cudaMemcpy(dA,A,nemax*sizeof(double),
             cudaMemcpyHostToDevice);
  cudaMemcpy(dB,B,nemax*sizeof(double),
             cudaMemcpyHostToDevice);
  // -- sum --
  sum_array<<<nemax/thread,thread>>>(nemax,dA,dB,dC);
  // -- copy memories from device to host --
  cudaMemcpy(C,dC,nemax*sizeof(double),
             cudaMemcpyDeviceToHost);
  // -- output --
  for(ni=0;ni<nemax;ni++){
    printf("%d: %lf + %lf = %lf¥n",
           ni,A[ni],B[ni],C[ni]);
  }
  // -- deallocate arrays on Device --
  cudaFree(dA); cudaFree(dB); cudaFree(dC);
  cudaDeviceReset();
  return 0;
}
```

sumarray.cuf（CUDA Fortran）

```
module cudakernel
contains
  attributes(global) subroutine sum_array(nemax,dA,dB,dC)
    implicit none
    integer,value,intent(in)::nemax
    double precision,intent(in)::dA(nemax),dB(nemax)
    double precision,intent(out)::dC(nemax)
    integer::ni
    ni=(blockIdx%x-1)*blockDim%x+threadIdx%x
    dC(ni)=dA(ni)+dB(ni)
    return
  end subroutine sum_array
end module cudakernel

program main
  use cudafor
  use cudakernel
  implicit none
  integer,parameter::nemax=4096,thread=1024
  integer::ni,ierr
  double precision::A(nemax),B(nemax),C(nemax)
  double precision,device,allocatable::dA(:),dB(:),dC(:)
  ! -- set initial value --
  call set_seed(248309)
  do ni=1,nemax
     call random_number(A(ni))
     call random_number(B(ni))
  enddo
  ! -- allocate arrays on device --
  allocate(dA(nemax))
  allocate(dB(nemax))
  allocate(dC(nemax))
  ! -- copy memories from host to device --
  dA=A
  dB=B
  ! -- sum --
  call sum_array<<<nemax/thread,thread>>>(nemax,dA,dB,dC)
  ! -- copy memories from device to host --
  C=dC
  ! -- output --
  do ni=1,nemax
     write(*,'(i5,":",f10.7," +",f10.7," =",f10.7)')&
           ni,A(ni),B(ni),C(ni)
  enddo
  ! -- deallocate arrays on device --
  deallocate(dA); deallocate(dB); deallocate(dC)
  ierr=cudaDeviceReset()
end program main
```

Information for Reedbush

- If you have any questions, you can see manuals on the user portal site (https://reedbush-www.cc.u-tokyo.ac.jp/). If the problems are not solved, please ask me. Don't ask Information Technology Center (ITC) directly.

- Your account expiration will be the end of fiscal year (Mar. 2019). You have to back up the data at your own responsibility.

- You can use Reedbush freely until the account expiration, but the computational resources are limited to 172 node-hours. You can check the remaining token by `show_token` command.

```
$ show_token
```