π-computer

How to use FX10 supercomputer @Kobe University

Yohei Miyake, Hideyuki Usui, Koji Morishita
Kobe University

Outline

- About FX10 supercomputer at Kobe university
  - System overview
  - How to use
    - Login
    - Compile
    - Execute

FX10 at Kobe university (π-computer)

- Fujitsu PRIMEHPC FX10
  - SPARC64™ IXfx processor x 96 node
  - Total peak performance: 20.2 TFLOPS
  - Total main memory: 3 TByte
- Node specifications (in comparison with K-Computer)

	FX10 (SPARC64™ IXfx)	K (SPARC64™ VIIIfx)
Number of cores	16	8
L1 cache (core)	32 KB(D)/32 KB(I)	32 KB(D)/32 KB(I)
L2 cache (shared)	12 MB	6 MB
Clock frequency	1.65 GHz	2.0 GHz
Peak performance	211.2 GFlops	128 GFlops
Memory capacity	32 GB	16 GB

How to login π-computer

Using public key authentication

Procedures (see subsequent slides for details)

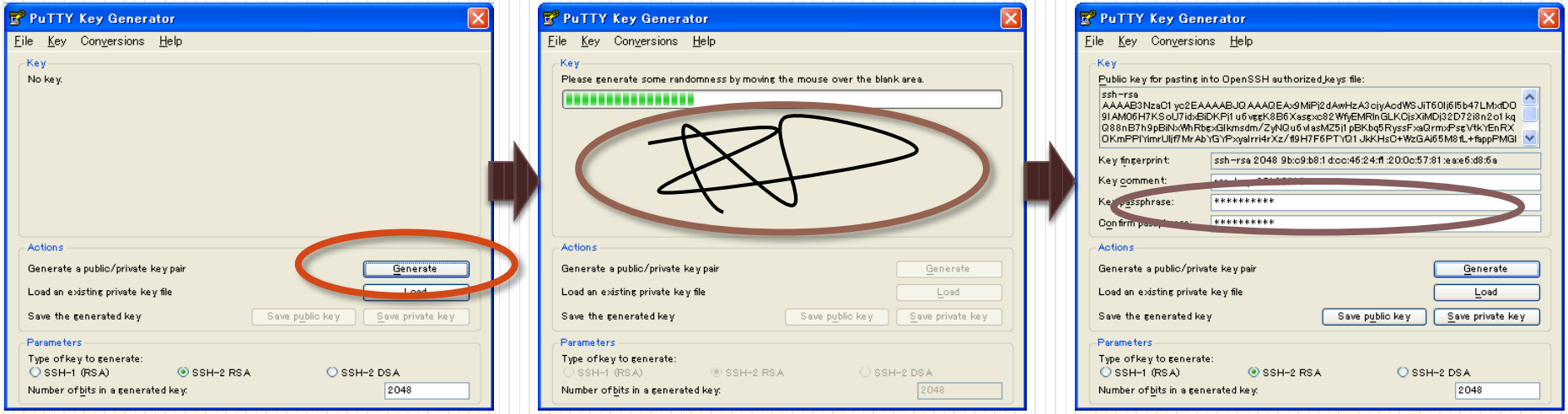
1. Generate a public/private key pair
   → Using PuTTYgen
2. Register the public key at π-computer
3. Login to π-computer
   → Using PuTTY

Login server name: **pi.ircpi.kobe-u.ac.jp**

Generate a key pair

- Run PuTTYgen
- Generate by following procedures

1. Click “generate”
2. Move mouse pointer
3. Enter passphrase



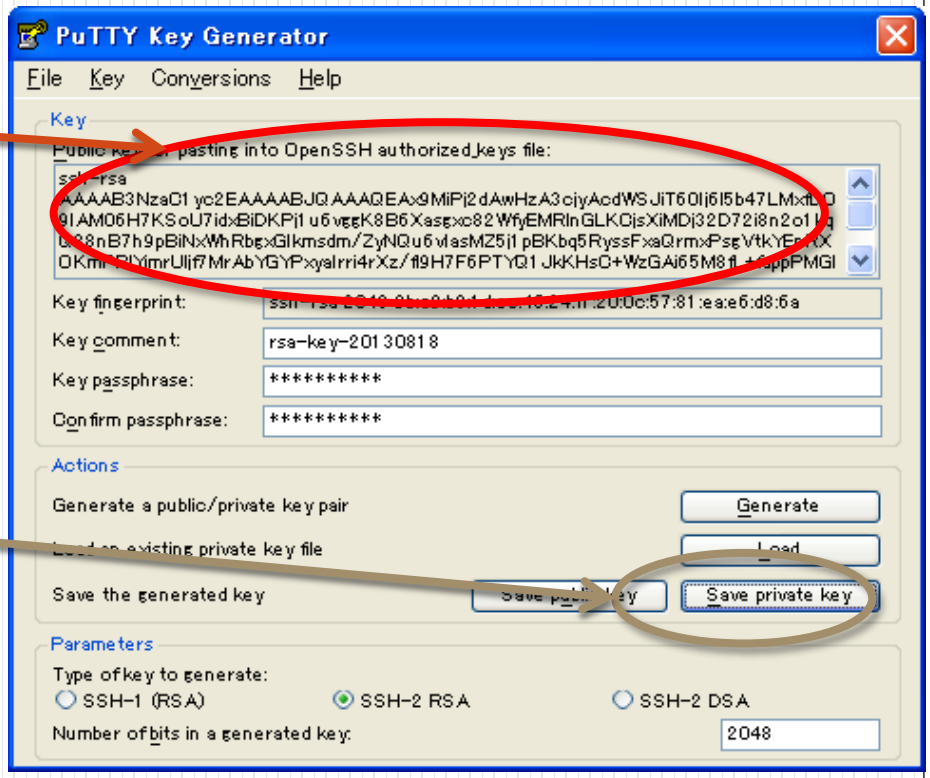
Save the key pair

Public key

- Copy & paste in the body of e-mail to me

Private key

- Click “Save private key”
- Save to a file (.ppk)



Create your account & register the public key

- This task is processed by the system administrator of π-computer. Please wait for completion.

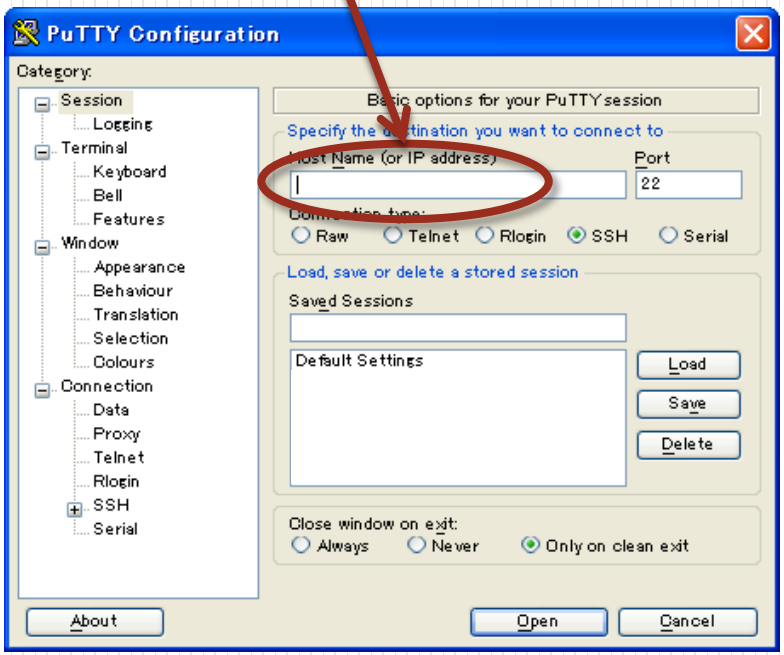
Login to π-computer

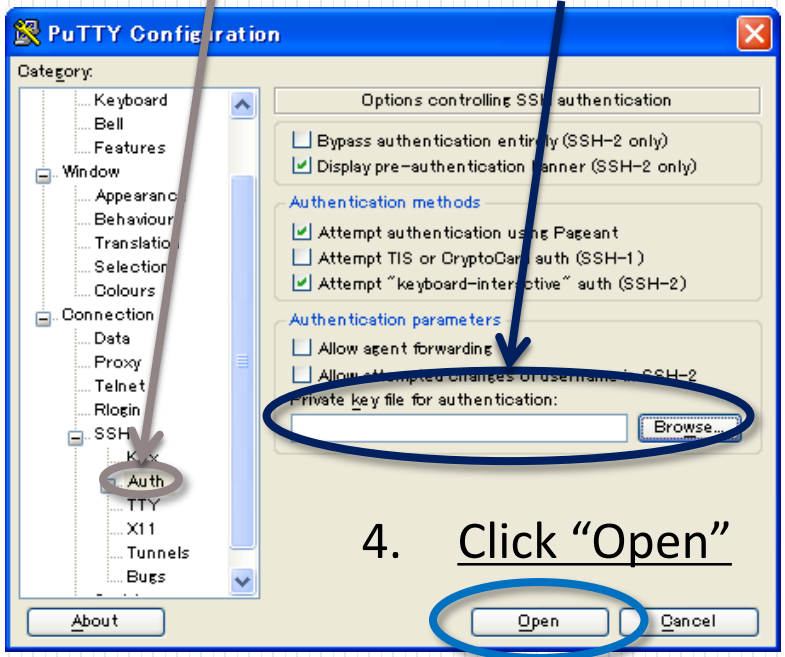
- Run PuTTY

1. Enter the host name
   pi.ircpi.kobe-u.ac.jp

2. Select [Connection]-[SSH]-[Auth] in the left menu

3. Set your private key





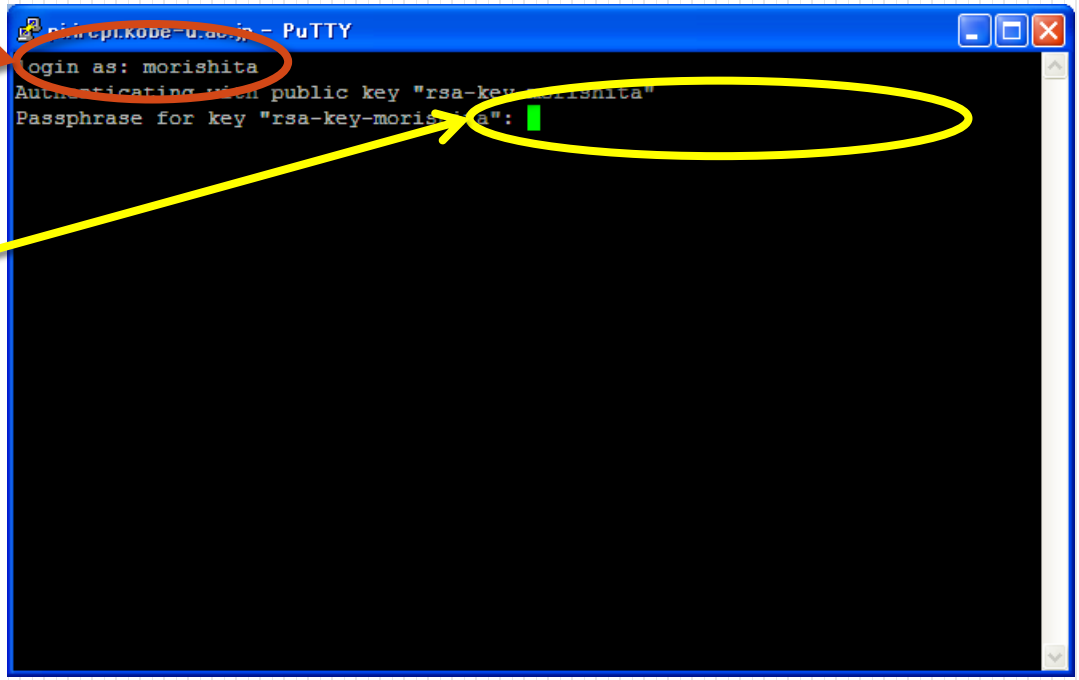
4. Click "Open"

Login to π-computer (cntd.)

- Login prompt
  - If a security alert dialog appears, then click “yes”

1. Enter your login ID

2. Enter your passphrase
   Note that entered characters aren’t displayed



How to compile/run your programs

Compiler: use Fujitsu Technical Computing Languages

Program run: use batch queueing system

Compilers

C++ : C++

F : Fortran C : C

- Serial program

```
$ frtpx sample.f90        F
$ fccpx sample.c          C
$ FCCpx sample.cpp        C++
```

- MPI program

```
$ mpifrtpx sample.f90     F
$ mpifccpx sample.c       C
$ mpiFCCpx sample.cpp     C++
```

Compilers

C++ : C++

F : Fortran C : C

- Open-MP program

```
$ frtpx -Kopenmp sample.f90      F
$ fccpx -Kopenmp sample.c        C
$ FCCpx -Kopenmp sample.cpp      C++
```

- MPI-OpenMP hybrid program

```
$ mpifrtpx -Kopenmp sample.f90   F
$ mpifccpx -Kopenmp sample.c     C
$ mpiFCCpx -Kopenmp sample.cpp   C++
```

Compilers

C++ : C++

F : Fortran C : C

- BLAS/LAPACK program

```
$ frtpx -SSL2 sample.f90        F
$ fccpx -SSL2 sample.c          C
$ FCCpx -SSL2 sample.cpp        C++
```

- ScaLAPACK program

```
$ mpifrtpx -SCALAPACK -SSL2 sample.f90    F
$ mpifccpx -SCALAPACK -SSL2 sample.c      C
$ mpiFCCpx -SCALAPACK -SSL2 sample.cpp    C++
```

Job execution (serial program)

- Make the job script
  - single_job.sh:

```
#!/bin/sh
#PJM -L "rscgrp=small"          ←Resource group name
#PJM -L "node=1"                ←# of requested nodes
#PJM -L "elapse=10:00"          ←Elapsed time limit (hh:mm:ss)
#PJM -j
#
./a.out                         ←Execute “a.out”
```

- Submit a job

```
$ pjsub single_job.sh
```

Job execution (Flat-MPI program)

- Make the job script
  - mpi_job.sh:

```
#!/bin/sh
#PJM -L "rscgrp=small"
#PJM -L "node=2“
#PJM --mpi ”proc=32”
#PJM -L "elapse=10:00"
#PJM -j
mpiexec -n 32 ./a.out
```

←Resource group name
←# of requested nodes
←# of MPI processes
←Elapsed time limit (hh:mm:ss)
←Execute “a.out” by MPI

1 node consists of 16 CPU-cores

- Submit a job

```
$ pjsub mpi_job.sh
```

Job execution (Open-MP program)

- Make the job script
  - omp_job.sh:

```
#!/bin/sh
#PJM -L "rscgrp=small"          ←Resource group name
#PJM -L "node=1“                ←# of requested nodes
#PJM -L "elapse=10:00"          ←Elapsed time limit (hh:mm:ss)
#PJM –j
export OMP_NUM_THREADS=16       ←# of Open-MP threads
./a.out                         ←Execute “a.out” by Open-MP
```

1 node consists of 16 CPU-cores

- Submit a job

```
$ pjsub omp_job.sh
```

Job execution (MPI-OMP hybrid program)

- Make the job script
  - hybrid_job.sh:

In case of using
4 proc. × 8 thr. = 32 CPU-cores
(requesting 2-node resource)

```
#!/bin/sh
#PJM -L "rscgrp=small"          ←Resource group name
#PJM -L "node=2“                ←# of requested nodes
#PJM --mpi ”proc=4”             ←# of MPI processes
#PJM -L "elapse=10:00"          ←Elapsed time limit (hh:mm:ss)
#PJM –j
export OMP_NUM_THREADS=8        ←# of Open-MP threads
mpiexec -n 4 ./a.out            ←Execute “a.out” by MPI
```

- Submit a job

```
$ pjsub hybrid_job.sh
```

Job control

- Displaying job states

```
$ pjstat
```

  - "-v" option: displaying detailed information

```
$ pjstat -v
```

- Deleting a job

```
$ pjdel [JOB_ID]
```

  - [JOB_ID] is displayed by "pjstat" commond
  - ex.) Deleting the job that [JOB_ID] is 12345

```
$ pjdel 12345
```

Referring to job execution results

- At the end of a job, following files are output to the current directory
  - Standard output file: [JOB_NAME].o[JOB_ID]
  - Standard error output file: [JOB_NAME].e[JOB_ID]
    - [JOB_NAME] is the same as the file name of the job script
    - If we set "`#PJM -j`" option in the job script, then the standard error output is merged into the standard output file

Resource group

- There are three resource groups

Resource group	Available num. of nodes	Max. elapsed time	Available term
small	1 ~ 12	10 minutes	Everyday
medium	1 ~ 48	24 hours	Weekday [Mon. 9 am – Fri. 9 pm (JST)]
large	48 ~ 84	12 hours	Weekend [Fri. 9 pm – Mon. 9 am (JST)]

- Please specify an appropriate resource group, which meets with your job size.

Practice

- You try to compile and execute the programs that you have been making in this school