

# 豊橋技科大HPCクラスタの使い方

豊橋技術科学大学

情報メディア基盤センター



# はじめに

- 豊橋技科大には共用のHPCクラスタが設置されています  
HPC = High Performance Computing (高性能計算)



- 豊橋技科大の教員，学生は誰でも利用可能です
- 全国の国立高専など連携機関からも利用可能です  
<https://hpcportal.imc.tut.ac.jp/>

# HPCクラスタの用途

---

- Linux環境での科学技術計算, シミュレーション, 深層学習などの実行
- 科学技術計算, シミュレーションのプログラム開発, プログラミングの学習にも利用可能です
- プログラミング環境
  - C/C++, Fortran に適しています
  - Intelコンパイラ, GNUコンパイラ
  - MPIライブラリ : Intel MPI, OpenMPI
  - 数値演算ライブラリ (BLAS/LAPACK) : Intel MKL
- データ保存
  - /home/[数字]/[ID] **5GB** (学生) または **20GB**  
(教員, 研究利用登録した学生)
  - /work/[ID] 制限なし

# HPCクラスタの用途

- 計算機利用申込（登録料1,000円）により研究レベルの計算の実行，研究用アプリケーションの利用が可能です  
<https://imc.tut.ac.jp/research/form>

- HPCクラスタで利用可能な研究用アプリケーション（2024年度）

	用途
ABAQUS	有限要素解析
COMSOL Multiphysics	有限要素解析
Gaussian	量子化学計算
Materials Studio	第一原理計算，量子化学計算

# 窓口サーバと計算サーバ

- HPCクラスタを使用するには、まず**窓口サーバ**に接続します。窓口サーバではプログラムのコンパイル等が可能です。大規模な計算は**計算サーバ**で行います。
- 計算サーバ（1ノード）の仕様は以下の通りです

項目	仕様
機種名	DELL PowerEdge R740
OS	RedHat Enterprise Linux 7.7
CPU	Intel Xeon Gold 6132 2.60 GHz 14コア × 2
メモリ	192 GB
GPU	NVIDIA Tesla V100 × 2

- 計算サーバは全部で15ノードあります。共用の計算資源であることに留意して教育・研究にご活用ください

# 窓口サーバへの接続方法

---

- 学内ネットワークからの接続  
パスワード認証または公開鍵認証によるSSH接続
- 学外ネットワークからの接続  
公開鍵認証によるSSH接続
- 情報メディア基盤センターを利用するユーザ名とパスワードがあれば、誰でもログインし利用できます
- 本資料では Windows PC からTeraTermとWinSCPを用いて公開鍵認証で接続する例を説明します
- その他の接続方法はこちらを参照  
<https://hpcportal.imc.tut.ac.jp/wiki/SSHClient/>

# TeraTermによる鍵生成と公開鍵の登録

1. TeraTermを起動し"新しい接続"ウィンドウ右上の×印をクリックして閉じる
2. 設定 → SSH鍵生成
3. RSA, ビット数2048で"生成"をクリック
4. パスフレーズを設定して公開鍵, 秘密鍵を保存

TTSSH: 鍵生成

鍵の種類

RSA1    RSA    DSA

ECDSA-256    ECDSA-384

ECDSA-521    ED25519

ビット数(B): 2048

生成(G)

閉じる(C)

ヘルプ(H)

鍵のパスフレーズ:

パスフレーズの確認:

コメント(C):

bcrypt KDF形式(K)

ラウンド数(N): 16

公開鍵の保存(I)

秘密鍵の保存(P)

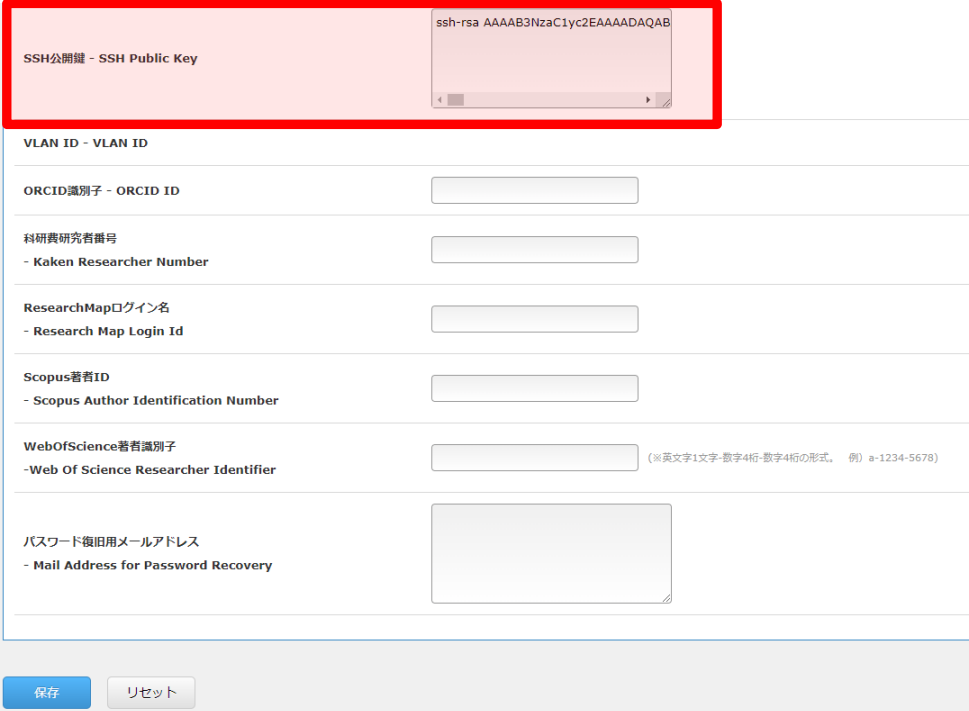
# TeraTermによる鍵生成

5. 以下のサイトの「プロフィールメンテナンス」→  
「SSH公開鍵 - SSH Public Key」に公開鍵の内容を  
(コピーアンドペーストで) 登録

<https://imc.tut.ac.jp/config/>

豊橋技科大以外の利用者は  
<https://hpcportal.imc.tut.ac.jp/>  
より“プロフィール変更”

秘密鍵は他人に知られない  
よう十分注意してください

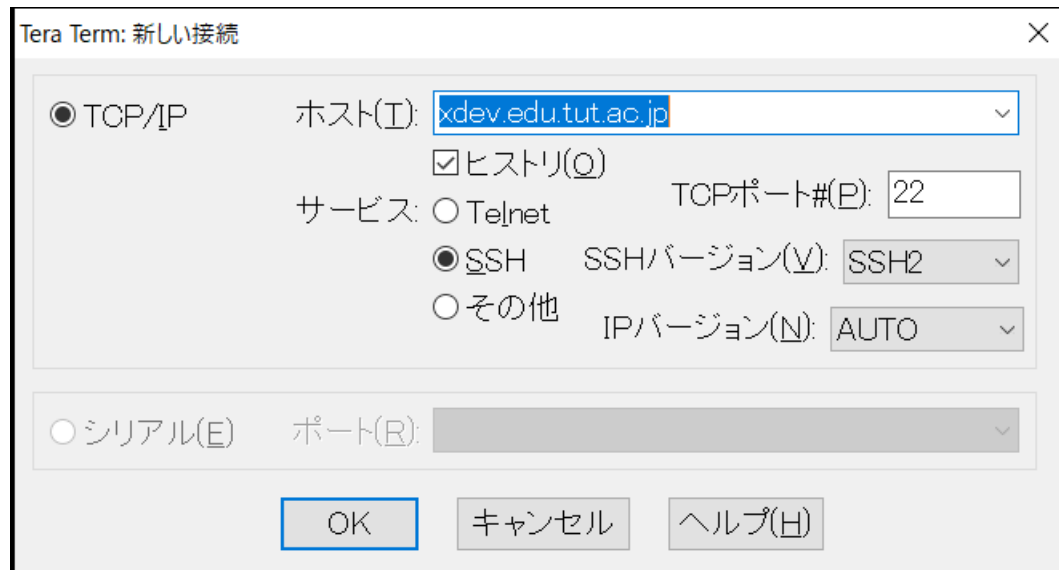


The screenshot shows a web form for profile maintenance. At the top, there is a red-bordered box containing the text "SSH公開鍵 - SSH Public Key" and a text area with the public key content: "ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB". Below this, the form has several input fields for various identifiers: "VLAN ID - VLAN ID", "ORCID識別子 - ORCID ID", "科研費研究者番号 - Kaken Researcher Number", "ResearchMapログイン名 - Research Map Login Id", "Scopus著者ID - Scopus Author Identification Number", "WebOfScience著者識別子 - Web Of Science Researcher Identifier" (with a note: "(\*英文字1文字-数字4桁-数字4桁の形式。例) a-1234-5678"), and "パスワード復旧用メールアドレス - Mail Address for Password Recovery". At the bottom, there are two buttons: "保存" (Save) and "リセット" (Reset).



# TeraTermによるログイン

1. TeraTermを起動 または ファイル → 新しい接続
2. ホストに `xdev.edu.tut.ac.jp` を指定  
(豊橋技科大以外の利用者は `lark.imc.tut.ac.jp`)  
サービスはSSH, TCPポート#は22でOKをクリック



3. 初回のログインでは警告が表示されるが“続行”

# TeraTermによるログイン

4. ユーザ名, パスフレーズを入力, 秘密鍵を指定  
→ “OK”をクリックしてログイン

SSH認証

ログイン中: xdev.edu.tut.ac.jp

認証が必要です。

ユーザ名(N):

パスフレーズ(P):

パスワードをメモリ上に記憶する(M)

エージェント転送する(Q)

認証方式

プレーンパスワードを使う(L)

RSA/DSA/ECDSA/ED25519鍵を使う

秘密鍵(K):

hosts (SSH)を使う

ローカルのユーザ名(U):

ホスト鍵(E):

キーボードインタラクティブ認証を使う(I)

Pageantを使う

OK 接続断(D)

秘密鍵を指定

## □ ユーザ名

情報メディア基盤センターから発行されたユーザ名を入力

## □ パスフレーズ

秘密鍵のパスフレーズ

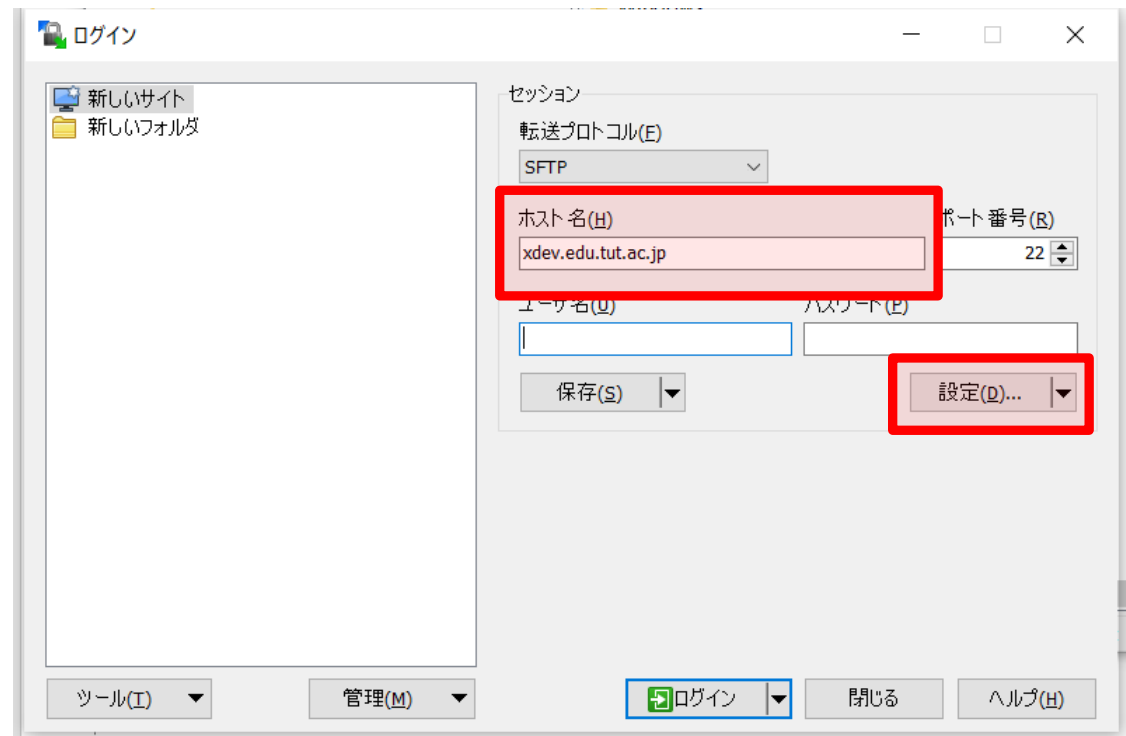
## □ 認証方式

“RSA/DSA/ECDSA/ED25519鍵を使う”から秘密鍵のファイルを指定

学内ネットワークからは“プレーンパスワードを使う”でもログインできます

# WinSCPによるファイル転送

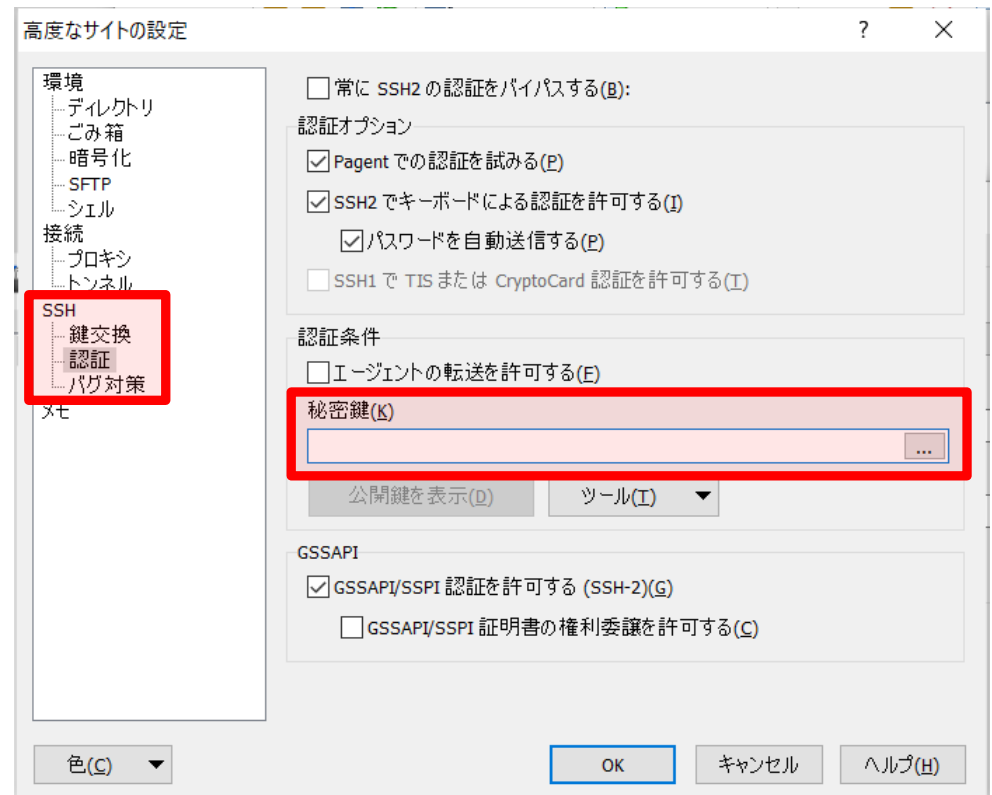
1. WinSCPを起動
2. 新しいサイト → ホスト名に `xdev.edu.tut.ac.jp` (`lark.imc.tut.ac.jp`) を入力  
ユーザ名, パスワードは空欄でよい
3. “設定”を  
クリック



# WinSCPによるファイル転送

4. SSH → 認証 から秘密鍵を指定
5. OK → ログイン でWindows PCとファイルのやりとりができます

ログインの前に  
“保存”をクリックして  
ログイン情報を登録  
しておくとう便利です



# Linuxのコマンド

- TeraTermなどでログインしてからはLinuxのコマンドで操作します

コマンド	説明
ls [ファイル/ディレクトリ]	ファイル/ディレクトリ情報の表示
cd [ディレクトリ]	ディレクトリの移動
cd ..	一つ上のディレクトリに移動
cd ~	ホームディレクトリに移動
mkdir [ディレクトリ]	ディレクトリの作成
rmdir [ディレクトリ]	ディレクトリの削除 ※ 中身が空になっている必要あり
mv [変更前ディレクトリ] [変更後ディレクトリ]	ディレクトリ名の変更
vi [ファイル名]	ファイルの編集 ※ Escape → :q! → Enter で終了
emacs [ファイル名]	ファイルの編集 ※ Ctrl+x → Ctrl+c で終了

# Linuxのコマンド

コマンド	説明
ls [ファイル or ディレクトリ]	ファイルまたはディレクトリ情報の表示
less [ファイル]	ファイルの表示 ※ space, b, カーソルキーで操作, q で終了
cp [ファイル] [ディレクトリ]	ファイルを指定したディレクトリにコピー
cp [ファイル1] [ファイル2]	ファイル1と同内容のファイル2を生成
mv [ファイル] [ディレクトリ]	ファイルを指定したディレクトリに移動
mv [変更前ファイル] [変更後ファイル]	ファイル名の変更
rm [ファイル]	ファイルの削除
cat *.f90 > temp.f90	ファイルの出力 この場合, 拡張子がf90のすべてのファイルをtemp.f90に出力する
grep "temp" *.f90	ファイル中の文字列の検索 この場合, 拡張子がf90のすべてのファイルを対象として"temp"を含む行を表示
logout	ログアウト ※ Ctrl + d でも可

# Emacsの操作

コマンド	説明
Ctrl+x → Ctrl+s	上書き保存
Ctrl+x → Ctrl+w	別名で保存
Ctrl+x → Ctrl+c	Emacsの終了
Ctrl+x → u	元に戻す
Ctrl+r	前方検索
Ctrl+s	後方検索
Meta (Escape) → <	冒頭へ移動
Meta (Escape) → >	末尾へ移動
Meta (Escape) → gg → Enter	指定した行番号へ移動
Ctrl+space	領域選択
Ctrl+w	指定した領域を切り取り
Meta (Escape) → w	指定した領域をコピー
Ctrl+y	貼り付け

# module コマンド

---

- コンパイラや研究用アプリケーションを使用するために `module` コマンドが用意されています
- 利用可能なモジュールを表示  
`$ module avail`
- loadしたモジュールを表示  
`$ module list`
- `module` コマンドによる設定を解除  
`$ module unload`



# module コマンド

---

- Intelコンパイラを利用する場合

```
$ module load intel/2022.1.2
```

- Intel MPIを利用する場合

```
$ module load intelmpi.intel/2022.1.2
```

- GNUコンパイラを利用する場合

```
$ module load gcc-7.3.1
```

- OpenMPIを利用する場合

```
$ module load openmpi.intel-4.0.1
```

- MPI環境は競合するためどちらかのみロードすること

# C言語プログラムのコンパイル

---

## □ Intelコンパイラ

### □ 逐次実行

```
icx -o temp.x temp.c
```

### □ OpenMP並列

```
icx -qopenmp -o temp.x temp.c
```

### □ MPI並列 (Intel MPI)

```
mpiicc -cc=icx -o temp.x temp.c
```

### □ MPI/OpenMPハイブリッド並列 (Intel MPI)

```
mpiicc -cc=icx -qopenmp -o temp.x temp.c
```

## □ GNUコンパイラ

### □ 逐次実行

```
gcc -o temp.x temp.c
```

### □ OpenMP並列

```
gcc -fopenmp -o temp.x temp.c
```

### □ MPI並列 (OpenMPI)

```
env OMPI_CC=gcc mpicc -o temp.x temp.c
```

# C++プログラムのコンパイル

---

## □ Intelコンパイラ

### □ 逐次実行

```
icpx -o temp.x temp.cpp
```

### □ OpenMP並列

```
icpx -qopenmp -o temp.x temp.cpp
```

### □ MPI並列 (Intel MPI)

```
mpiicpc -cxx=icpx -o temp.x temp.cpp
```

### □ MPI/OpenMPハイブリッド並列 (Intel MPI)

```
mpiicpc -cxx=icpx -qopenmp -o temp.x temp.cpp
```

## □ GNUコンパイラ

### □ 逐次実行

```
g++ -o temp.x temp.cpp
```

### □ OpenMP並列

```
g++ -fopenmp -o temp.x temp.cpp
```

### □ MPI並列 (OpenMPI)

```
env OMPI_CXX=g++ mpicxx -o temp.x temp.cpp
```

# Fortranプログラムのコンパイル

---

## □ Intelコンパイラ

### □ 逐次実行

```
ifx -o temp.x temp.f90
```

### □ OpenMP並列

```
ifx -qopenmp -o temp.x temp.f90
```

### □ MPI並列 (Intel MPI)

```
mpiifort -fc=ifx -o temp.x temp.f90
```

### □ MPI/OpenMPハイブリッド並列 (Intel MPI)

```
mpiifort -fc=ifx -qopenmp -o temp.x temp.f90
```

## □ GNUコンパイラ

### □ 逐次実行

```
gfortran -o temp.x temp.f90
```

### □ OpenMP並列

```
gfortran -fopenmp -o temp.x temp.f90
```

### □ MPI並列 (OpenMPI)

```
env OMPI_FC=gfortran mpif90 -o temp.x temp.f90
```

# コンパイラオプション

□ `icx --help | less` などと入力して確認できます

□ 代表的なオプションは以下の通り

オプション	説明
<code>-o</code>	実行ファイル名を指定
<code>-O0</code> , <code>-O1</code> , <code>-O2</code> , <code>-O3</code>	コンパイラによる最適化によりプログラムを高速化 <code>-O0</code> は最適化しない, <code>-O3</code> が最速
<code>-qopenmp</code> (Intel compiler) <code>-fopenmp</code> (GNU compiler)	OpenMPによる並列化
<code>-traceback -g</code> (Intel) <code>-fbacktrace -g</code> (GNU)	デバッグ用オプション デバッグ情報を生成し, 実行時にエラーが起きたときファイル名や行番号を表示する
<code>-check all</code> (Intel) <code>-fcheck=all</code> (GNU)	デバッグ用オプション 実行時にチェックを行いエラーや警告を出力する
<code>-mkl</code> or <code>-qmkl</code> (Intel)	Intel Math Kernel Library (MKL) を使用する BLAS, LAPACKなどのライブラリを利用可能
<code>-static</code>	静的リンクによりライブラリがインストールされていない環境でもバイナリを実行できるようにする。MPIとの併用不可

# GPU使用プログラムのコンパイル

---

- 計算サーバでコンパイルできます  
※ 窓口サーバではできません
- 窓口サーバにて  

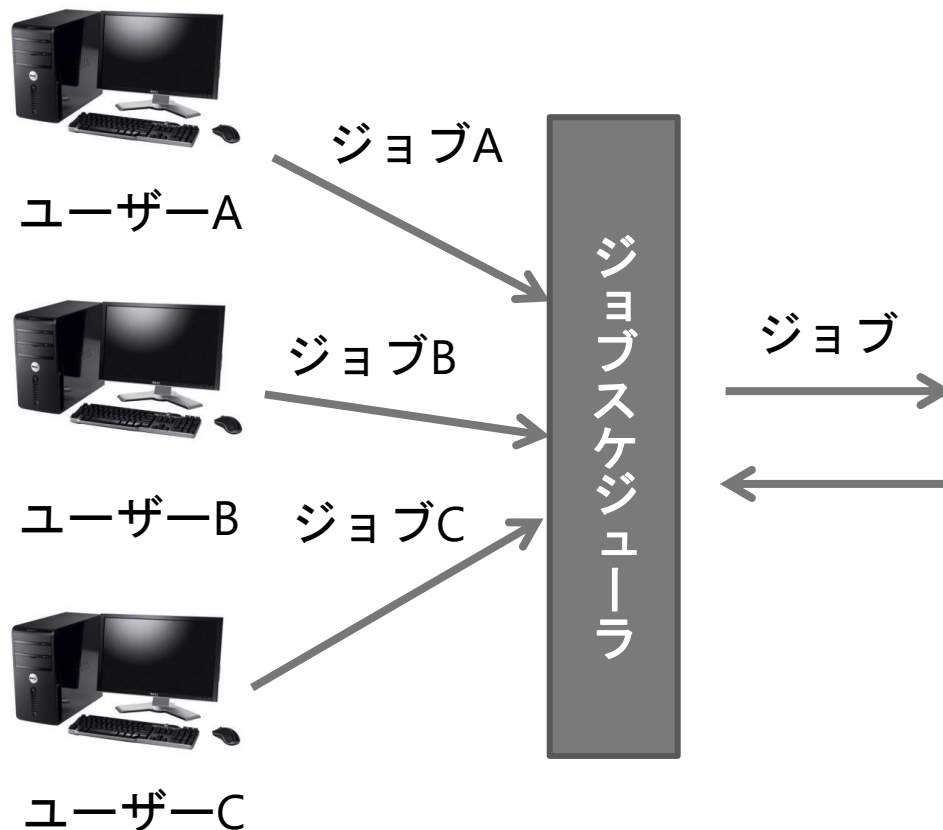
```
qsub -I -q gEduq -l select=1:ncpus=1:ngpus=1 -v  
DOCKER_IMAGE=prg-env:latest -- bash
```

と入力すると、計算サーバにて対話処理ができます
- ここでは1CPUコアと1GPUの使用を宣言しています  
詳細は以下のページの「インタラクティブジョブ」を参照  
<https://hpcportal.imc.tut.ac.jp/wiki/HowToSubmitJob>
- C言語でライブラリとしてcuBLASを利用する場合のコンパイル  

```
nvcc -ccbin gcc -I/usr/include/cublas.h -lcublas [ソースファイル名]
```

# ジョブスケジューラ

- 複数のユーザが共同で計算資源を利用  
→ ジョブスケジューラ PBS による管理
- 計算やシミュレーションを行う際には必ず使用すること



計算サーバ  
全15ノード  
xsnd00~xsnd14



# ジョブスケジューラの利用方法

---

- 事前にジョブを実行するためのスクリプトファイル (bash, cshなど) を用意
- ジョブ投入  
\$ qsub [実行スクリプトファイル名]
- ジョブの状態, ジョブID表示  
\$ qstat -a
- ジョブの詳細を表示  
\$ qstat -f | less
- ジョブの削除  
\$ qdel [ジョブID]
- ジョブが実行されると標準出力と標準エラー出力が別々のファイルに出力されます



# ジョブスケジューラの利用方法

## □ qsubコマンドのオプション

オプション	使用例	意味
-q	-q wEduq	ジョブを投入するキューを指定
-v	-v DOCKER_ IMAGE=<image>	指定したDockerイメージ上でジョブを実行
-v	-v SINGULARITY_ IMAGE=<image>	指定したSingularityイメージ上でジョブを実行
-l	-l mem=1g	使用するCPUコア数, メモリ上限などを設定
-o	-o filename	標準出力の内容を指定されたファイルに出力
-e	-e filename	標準エラー出力の内容を指定されたファイルに出力
-j oe		標準エラー出力を標準出力にマージ

□ これらは実行スクリプトファイル内の指示文でも指定できます

# 利用可能なキュー

キュー	ノード数 最大/標準	CPUコア数 最大/標準	メモリ 最大/標準	GPU数 最大/標準	経過時間 上限	備考
wEduq	4/1	4/1	32GiB/6GiB	0/0	8時間	教育用
gEduq	2/1	4/1	32GiB/6GiB	2/1	8時間	教育用 GPU使用

## □ 計算機利用申込により以下も利用できます

キュー	ノード数 最大/標準	CPUコア数 最大/標準	メモリ 最大/標準	GPU数 最大/標準	経過時間 上限	備考
wSrchq	1/1	16/1	160GiB/8GiB	0/0	336時間	小規模
wLrchq	13/1	364/1	2496GiB/6GiB	0/0	336時間	大規模
gSrchq	1/1	16/4	160GiB/32GiB	2/1	336時間	小規模 GPU使用
gLrchq	13/1	364/1	2496GiB/6GiB	26/1	336時間	大規模 GPU使用

# コンテナイメージ

---

- ジョブ投入時のコマンドオプションまたは実行スクリプトの指示文でコンテナイメージを指定する必要があります
- コンテナ上でジョブが実行されます
- Dockerコンテナを使用する場合のオプション  
-v DOCKER\_IMAGE=<イメージ名>
- Singularityコンテナを使用する場合のオプション  
-v SINGULARITY\_IMAGE=<イメージ名>
- 指定可能なイメージ名を表示  
\$ showimages

# 実行スクリプトファイルの例：逐次プログラム

---

```
#!/bin/bash
```

```
#PBS -q wEduq ← 教育用のキュー, GPU使用なし
```

```
#PBS -l select=1:ncpus=1 ← 1 ノード 1 CPUコア使用
```

```
#PBS -v DOCKER_IMAGE=prg-env:latest
```

(空行)

```
ulimit -c 0
```

```
source /etc/profile
```

```
./common/intel-2022/compiler/latest/env/vars.sh
```

```
./common/intel-2022/mpi/latest/env/vars.sh
```

```
./common/intel-2022/mkl/latest/env/vars.sh
```

Intelコンパイラ,  
Intel MPI, MKLを  
有効にする

```
cd $PBS_O_WORKDIR
```

```
./test.x
```

□ 上記をtest.bashとして保存し, \$ qsub test.bash で  
test.xを実行できます

# 実行スクリプトファイルの例：OpenMP並列

---

```
#!/bin/bash
#PBS -q wEduq
#PBS -l select=1:ncpus=4 ← 1 ノード 4 CPUコア使用
#PBS -v DOCKER_IMAGE=prg-env:latest
(空行)
ulimit -c 0
source /etc/profile
. /common/intel-2022/compiler/latest/env/vars.sh
. /common/intel-2022/mpi/latest/env/vars.sh
. /common/intel-2022/mkl/latest/env/vars.sh
export OMP_NUM_THREADS=4 ← OpenMPによる 4 スレッド並列
cd $PBS_O_WORKDIR
./test.x
```

# 実行スクリプトファイルの例：MPI並列

---

```
#!/bin/bash
```

```
#PBS -q wEduq
```

1ノード,  
4 CPUコア, 4 MPIプロセス

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

```
#PBS -v DOCKER_IMAGE=mpi-env:latest
```

MPI環境の  
コンテナイメージを指定

(空行)

```
ulimit -c 0
```

```
source /etc/profile
```

```
./common/intel-2022/compiler/latest/env/vars.sh
```

```
./common/intel-2022/mpi/latest/env/vars.sh
```

```
./common/intel-2022/mkl/latest/env/vars.sh
```

```
cd $PBS_O_WORKDIR
```

```
export OMP_NUM_THREADS=1 ← OpenMPのスレッド数を1に設定
```

```
mpirun -np 4 test.x ← 4 プロセスでプログラムを実行
```

## 実行スクリプトファイルの例：MPI/OpenMPハイブリッド並列

---

```
#!/bin/bash
#PBS -q wEduq
#PBS -l select=1:ncpus=4:mpiprocs=2
#PBS -v DOCKER_IMAGE=mpi-env:latest,DOCKER_
OPTIONS="--network=overlaynw"
(空行)
ulimit -c 0
source /etc/profile
. /common/intel-2022/compiler/latest/env/vars.sh
. /common/intel-2022/mpi/latest/env/vars.sh
. /common/intel-2022/mkl/latest/env/vars.sh
export OMP_NUM_THREADS=2
cd $PBS_O_WORKDIR
mpirun -np 2 test.x
```

1ノード,  
1ノードあたり4 CPUコア,  
1ノードあたり2 MPIプロセス

MPIによるノード間並列の設定

← OpenMPにより1プロセスあたり  
2 スレッド並列

← 2 MPIプロセスでプログラムを実行

# 実行スクリプトファイルの例：GPUの利用

---

```
#!/bin/bash
```

```
#PBS -q gEduq ← GPUを使用できるキューを指定
```

```
#PBS -l select=1:ncpus=1:ngpus=1 ← 1 ノード 1 CPUコア 1 GPU
```

```
#PBS -v DOCKER_IMAGE=prg-env:latest
```

(空行)

```
ulimit -c 0
```

```
source /etc/profile
```

```
cd $PBS_O_WORKDIR
```

```
./test.x
```



# 実行スクリプトファイルの例：計算資源の設定

---

```
#!/bin/bash
```

```
#PBS -q wLrchq
```

```
#PBS -l walltime=96:00:00 ← 経過時間の上限
```

```
#PBS -l select=1:vnode=xsnd06:ncpus=4:mem=48G
```

xsnd06の4コア,  
48GBメモリ確保

```
#PBS -v DOCKER_IMAGE=prg-env:latest
```

- wEduq, gEduq では 計算サーバ (vnode) として xsnd00~xsnd14 を指定できる。その他のキューでは xsnd02~xsnd14 を指定
- あらかじめ計算時間, 必要なメモリ量を見積って効率よく利用しましょう

# 謝辞について

---

□ HPCクラスタを使用して得た成果を論文，学会等で発表する際には謝辞（Acknowledgment）に記載をお願い致します

□ 例

本研究で行った計算は豊橋技術科学大学情報メディア基盤センターのHPCクラスタを利用して行いました。

□ Example

The authors would like to thank Information and Media Center (IMC) at Toyohashi University of Technology for providing computer resources and the kind support of its staff.