2022/6/15

豊橋技術科学大学HPCクラスタ（研究用クラスタシステム）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)

項目	仕様
ノード数	15
総CPUコア数	420
総メモリ容量	2880 GB
総ストレージ容量	2 PB
ノード間通信	InfiniBand 4x EDR

- 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で科学技術計算，シミュレーション，
  機械学習などを実行
  OpenMP, MPI によるノード内/ノード間並列計算
  GPUも利用可能
- プログラミング
  C/C++, Fortran, Python など
  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
- 科学技術計算，シミュレーションのプログラム開発
  プログラム開発手法の学習

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（2022年度）

	用途
ABAQUS	有限要素解析
ANSYS HFSS	電磁界解析
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
MATLAB	プログラミング，数値計算

窓口サーバへの接続方法

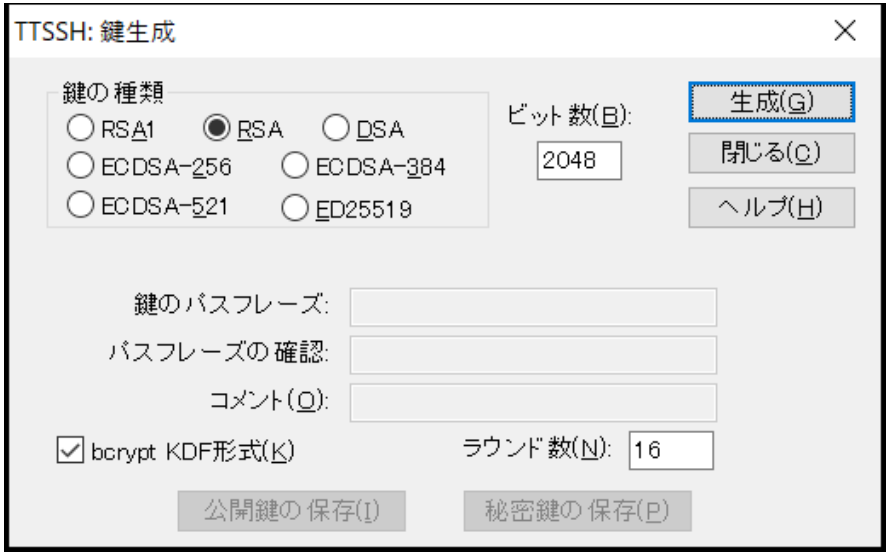
- 学内ネットワークからの接続
  パスワード認証または公開鍵認証によるSSH接続
- 学外ネットワークからの接続
  公開鍵認証によるSSH接続
- 情報メディア基盤センターを利用するユーザ名と
  パスワードがあれば，誰でもログインし利用できます
- 本資料では 学外のWindows PC から
  TeraTermとWinSCPを用いて接続する例を説明します
- その他の接続方法はこちらを参照
  https://hpcportal.imc.tut.ac.jp/wiki/SSHClient/

TeraTermによる鍵生成

1. TeraTermを起動し"新しい接続"ウインドウ右上の×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
5. 以下のサイトで公開鍵の１行目を登録

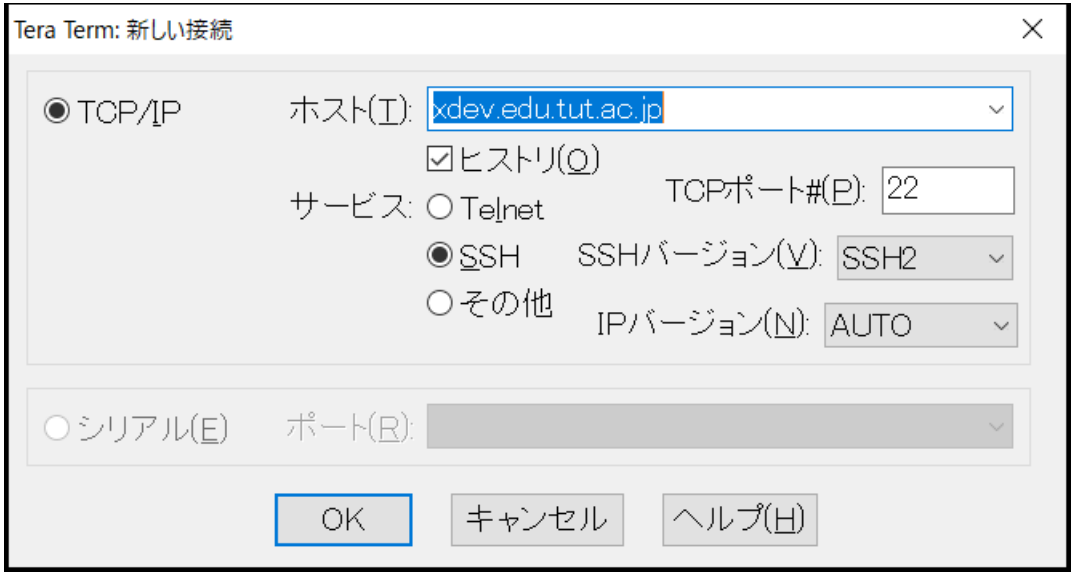
https://imc.tut.ac.jp/config/

豊橋技科大以外の利用者は
https://hpcportal.imc.tut.ac.jp/
より"プロファイル変更"



TeraTermによるログイン

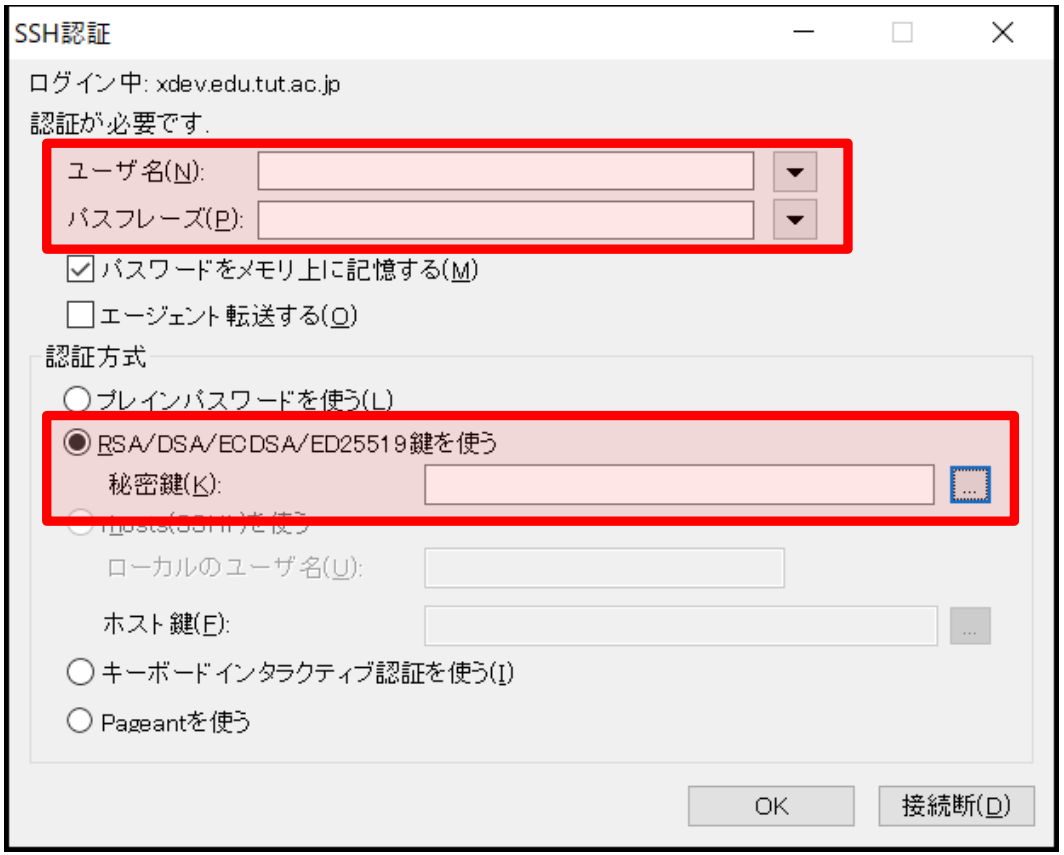
1. TeraTermを起動　または　ファイル → 新しい接続
2. ホストに xdev.edu.tut.ac.jp を指定
   ※ 豊橋技科大以外の利用者は lark.imc.tut.ac.jp
   サービスはSSH，TCPポート#は22でOKをクリック



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

5. 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



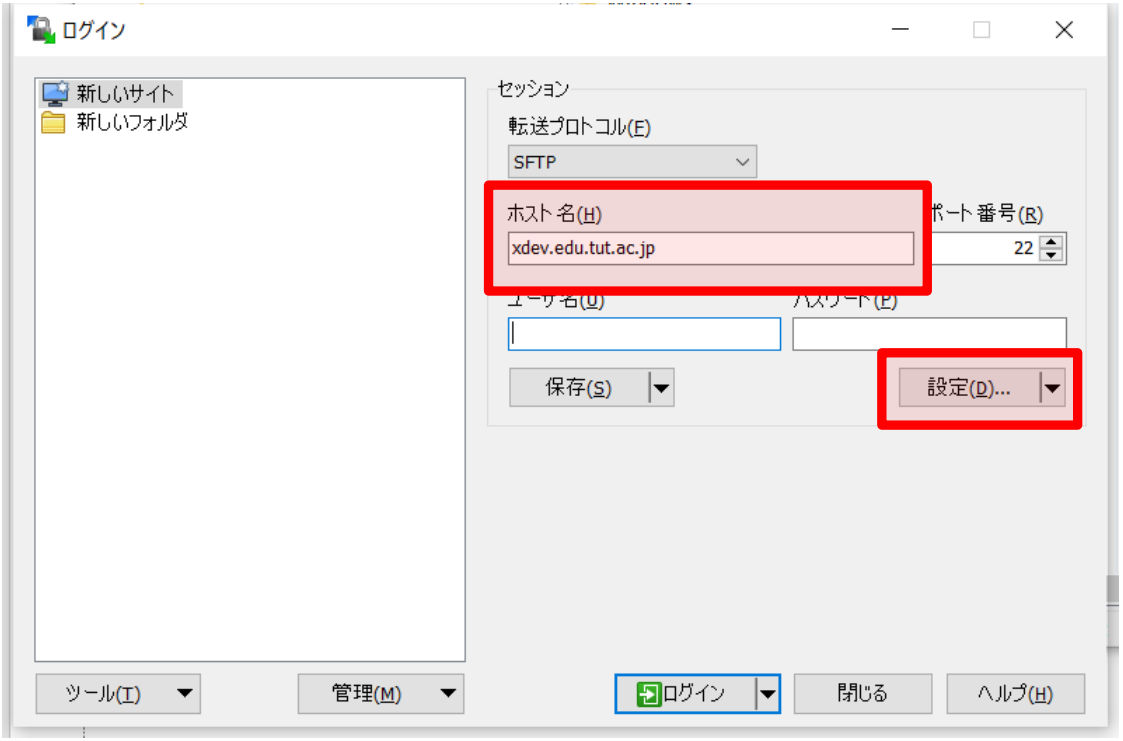
- ユーザ名
  情報メディア基盤センターで使用しているものを入力

- 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

WinSCPによるファイル転送

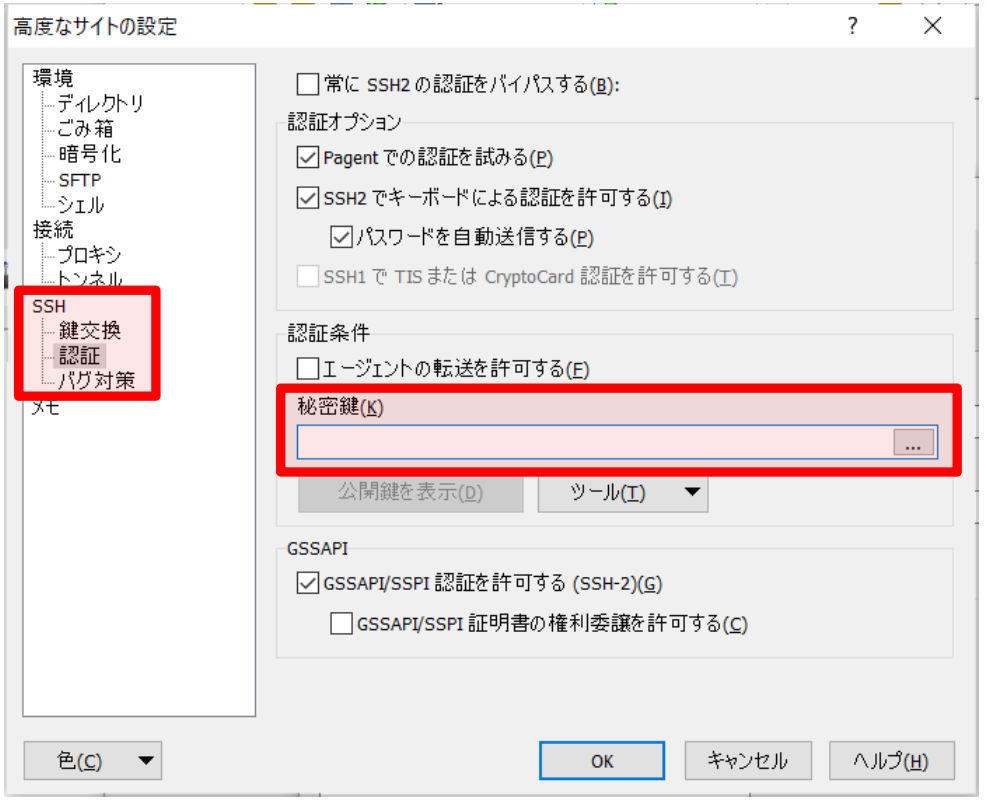
1. WinSCPを起動
2. 新しいサイト
   ホスト名に xdev.edu.tut.ac.jp (lark.imc.tut.ac.jp) を入力
   ユーザ名，パスワードは空欄でよい
3. “設定”を
   クリック



WinSCPによるファイル転送

3. SSH → 認証 から秘密鍵を指定
4. 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/ディレクトリ]	ファイル/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

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

- コンパイラや各種アプリケーションを使用する前に module コマンドを実行する必要があります
- 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/19.0.4.243　※ バージョンは省略可
- GNUコンパイラを利用する場合
  $ module load gcc-7.3.1
- Intel MPIを利用する場合
  $ module load intelmpi.intel/19.0.4.243
  ※ バージョンは省略可
- 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loadすること
- 初心者はIntelコンパイラ，Intel MPIの利用を推奨します

ログインシェル

- ログインと同時に実行されるシェルスクリプト
  初期設定では~/.bashrc
- エディタで"User specific aliases and functions"以下を編集
- 例えば右のようにすると
  メッセージの英語化
  (export LNAG=C)
  モジュールのロード
  エイリアスの設定
  ("e"でEmacsを起動)
  をログイン時に行う

```
File Edit Options Buffers Tools Sh-Script Hel
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't \
like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

export LANG=C

module load intel/19.0.4.243
module load gcc-7.3.1
module load intelmpi.intel/19.0.4.243

alias e='emacs -nw'
alias l='less'


-=--:----F1  .bashrc       All L1      (Shell
Beginning of buffer
```

C言語プログラムのコンパイル

- Intelコンパイラ
  - 逐次実行
    icc -o temp.x temp.c
  - OpenMP並列
    icc -qopenmp -o temp.x temp.c
  - MPI並列 (Intel MPI)
    mpiicc -o temp.x temp.c
  - MPI/OpenMPハイブリッド並列 (Intel MPI)
    mpiicc -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

C++プログラムのコンパイル

- Intelコンパイラ
  - 逐次実行
    icpc -o temp.x temp.c
  - OpenMP並列
    icpc -qopenmp -o temp.x temp.c
  - MPI並列 (Intel MPI)
    mpiicpc -o temp.x temp.c
  - MPI/OpenMPハイブリッド並列 (Intel MPI)
    mpiicpc -qopenmp -o temp.x temp.c
- GNUコンパイラ
  - 逐次実行
    g++ -o temp.x temp.c
  - OpenMP並列
    g++ -fopenmp -o temp.x temp.c

Fortranプログラムのコンパイル

- Intelコンパイラ
  - 逐次実行
    ifort -o temp.x temp.c
  - OpenMP並列
    ifort -qopenmp -o temp.x temp.c
  - MPI並列 (Intel MPI)
    mpiifort -o temp.x temp.c
  - MPI/OpenMPハイブリッド並列 (Intel MPI)
    mpiifort -qopenmp -o temp.x temp.c
- GNUコンパイラ
  - 逐次実行
    gfortran -o temp.x temp.c
  - OpenMP並列
    gfortran -fopenmp -o temp.x temp.c

コンパイラオプション

- icc --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

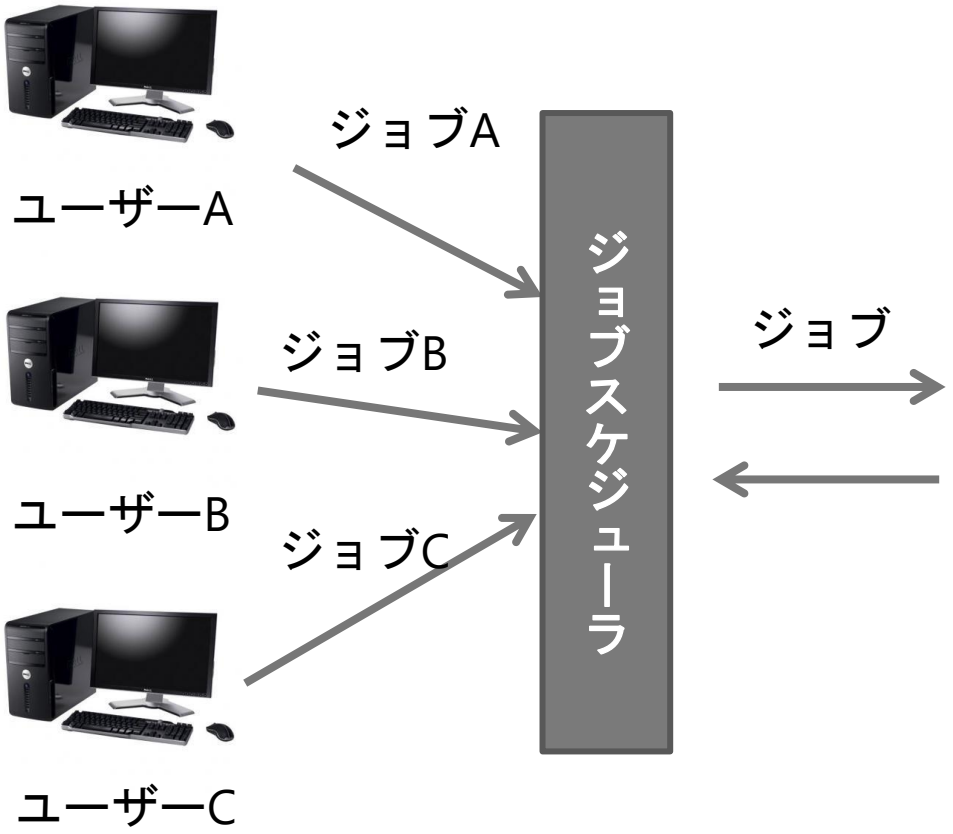
オプション	説明
-o	実行ファイル名を指定
-O0, -O1, -O2, -O3	コンパイラによる最適化によりプログラムを高速化 -O0は最適化しない，-O3が最速
-qopenmp (Intel) -fopenmp (GNU)	OpenMPによる並列化
-traceback -g (Intel) -fbacktrace -g (GNU)	実行時にエラーが起きたときソースコードのファイル名や 行番号を表示するデバッグ用オプション
-check all (Intel) -fcheck=all (GNU)	実行時にチェックを行いエラーや警告を出力する デバッグ用オプション
-mkl (Intelのみ)	Intel Math Kernel Library を使用する BLAS, LAPACKなどが利用可能

GPU使用プログラムのコンパイル

- 演算ノード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（ログインノード）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，演算ノード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，Intelコンパイラでライブラリとしてcu BLASを利用する場合のコンパイル
  nvcc -ccbin icc -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による管理
- 計算を行う際には必ず使用すること



演算ノード×15
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
  ジョブ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                    ← wEduqキューを指定
#PBS -l select=1:ncpus=1         ← 1ノード1CPUコア使用
#PBS -v DOCKER_IMAGE=prg-env:latest
source /etc/profile    ← moduleコマンドを実行するために必要
module load intel
cd $PBS_O_WORKDIR                ← 実行ディレクトリに移動
./test.x    ← プログラムの実行
```

- 上記をtest.bashとして保存すれば
  $ qsub test.bash でジョブを実行できます

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
source /etc/profile
module load intel
export OMP_NUM_THREADS=4    ← OpenMPによる 4 スレッド並列
cd $PBS_O_WORKDIR
./test.x
```

- 計算機利用申込のない場合，最大 4 CPUコアとなります

実行スクリプトファイルの例：ノード内MPI並列

```
#!/bin/bash
#PBS -q wEduq
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -v DOCKER_IMAGE=mpi-env:latest
source /etc/profile
module load intelmpi.intel
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=1
mpirun -np 4 test.x
```

1ノード，
4 CPUコア，4 MPIプロセス

MPI環境の
コンテナイメージ

OpenMPのスレッド数を1に設定
デフォルトでは ncpus と同じ数に
なります

← 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：ノード間MPI並列

```
#!/bin/bash
#PBS -q wEduq
#PBS -l select=4:ncpus=1:mpiprocs=1
#PBS -v DOCKER_IMAGE=mpi-env:latest,DOCKER_
OPTIONS="--network=overlaynw"
source /etc/profile
module load intelmpi.intel
cd $PBS_O_WORKDIR
mpirun -np 4 test.x
```

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

← MPIによるノード間並列の設定

← 4 プロセスでプログラムを実行

- 計算機利用申込のない場合，最大 4 ノードとなります

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=2:ncpus=2:mpiprocs=1
#PBS -v DOCKER_IMAGE=mpi-env:latest,DOCKER_
OPTIONS="--network=overlaynw"
source /etc/profile
module load intelmpi.intel
export OMP_NUM_THREADS=2
cd $PBS_O_WORKDIR
mpirun -np 2 test.x
```

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

← MPIによるノード間並列の設定

← OpenMPによりプロセスごとに
2 スレッド並列

← 2 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    ← gEduq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
source /etc/profile
module load intel
cd $PBS_O_WORKDIR
./test.x
```

実行スクリプトファイルの例：cshの場合

```
#!/bin/csh
#PBS -q wEduq
#PBS -l select=1:ncpus=4
#PBS -v DOCKER_IMAGE=prg-env:latest
source /etc/profile.d/modules.csh
module load intel
setenv OMP_NUM_THREADS 4
cd $PBS_O_WORKDIR
./test.x
```

← moduleコマンドを
実行するために必要

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
#PBS -l select=1:vnode=xsnd06:ncpus=4:mem=48G
#PBS -v DOCKER_IMAGE=prg-env:latest
```

← 経過時間の上限

xsnd06で4コア，
48GBメモリ確保

- 演算ノードは xsnd00～xsnd14 (wEduq, gEduq)
  その他のキューでは xsnd00～xsnd12
- 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

さいごに

- 演算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

- 研究用アプリケーションなどの利用方法は
  豊橋技術科学大学 HPCクラスタ Wiki
  https://hpcportal.imc.tut.ac.jp/wiki/ を参照してください
- CPUやディスクなどのリソースは共用であることに
  留意した上で教育・研究にご活用ください