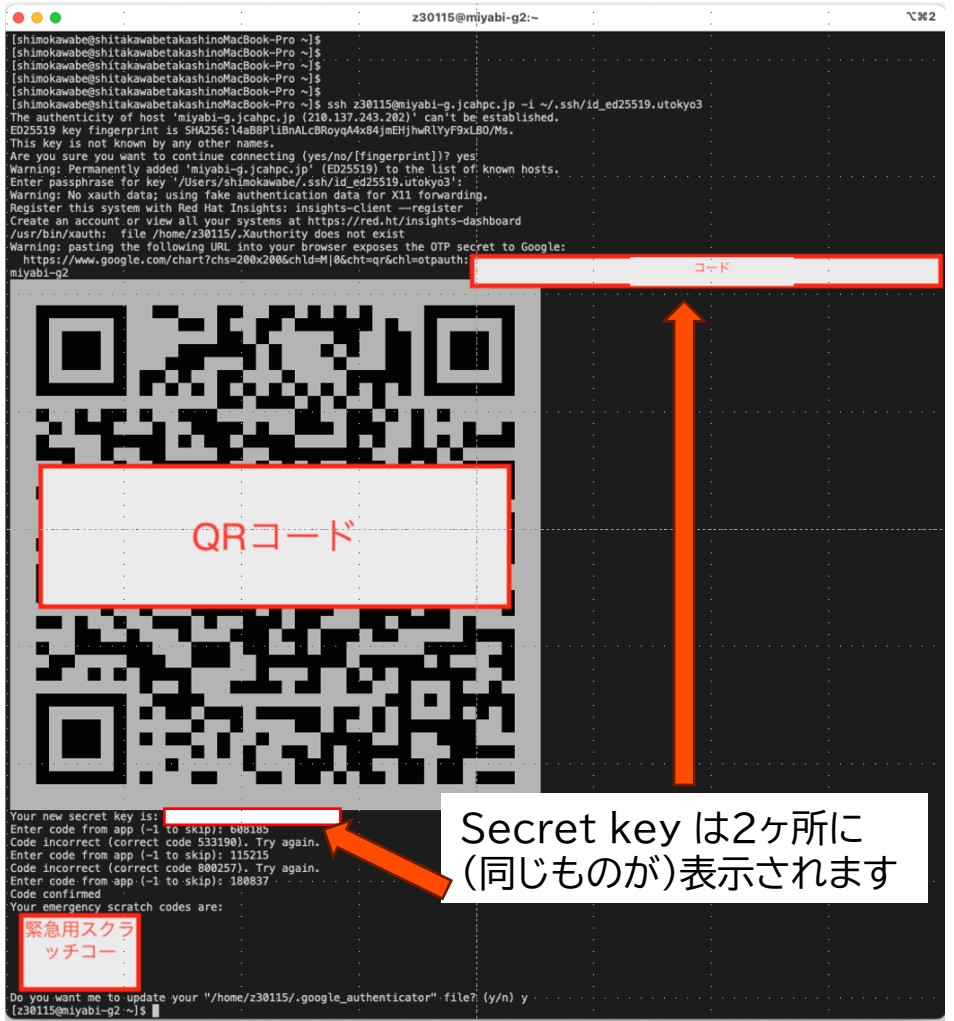
Miyabi 利用チュートリアル（Miyabi-G編）

JCAHPC
（筑波大学 計算科学研究センター・東京大学 情報基盤センター）

Miyabi システムへのアクセス(初回SSHログイン時)

- 利用支援ポータルにログインし, SSH公開鍵を登録
  - パスワード認証とワンタイムパスコード認証(OTP認証)を併用する2要素認証
  - OTP認証については, webブラウザ(拡張機能含む)であればコピペで入力できます
- `$ ssh USERNAME@miyabi-g.jcahpc.jp`
- (次回以降ログイン時の)OTP認証設定用のQRコードなどが表示されるので, 設定(設定方法はポータルと同様のため割愛)
  - QRコードをスキャンするか,
  - secret key を登録
- OTP認証設定を完了させないと次回以降ログインできなくなります
  - (現時点では, システム管理者に連絡してOTP認証を再設定する必要があります)
- 別ターミナルから正常にログインできることを確認できるまで, 初回ログイン時の画面を表示し続けておくことを推奨します



Miyabi システムへのアクセス(2回目以降)

- $ ssh USERNAME@miyabi-g.jcahpc.jp
  - ここでもOTP認証が必要(注:利用支援ポータルのOTP認証とは別のキー)
    - 1ページ前に設定したOTP認証です

```
[ymiki@draco ~]$ ssh z30118@miyabi-g.jcahpc.jp
Enter passphrase for key '/home/ymiki/.ssh/id_ed25519':
(z30118@miyabi-g.jcahpc.jp) Verification code: 398756
Register this system with Red Hat Insights: insights-client --register
Create an account or view all your systems at https://red.ht/insights-dashboard
[z30118@miyabi-g1 ~]$
```

- 利用手引書中のシェルスクリプト等は Miyabi のログインノードの /usr/local/example 配下に格納されている
- Miyabi-C についても同様にログインできます(OTP認証などはGと共通)
  - $ ssh USERNAME@miyabi-c.jcahpc.jp

簡単な動作テスト(NVHPC単体動作編)

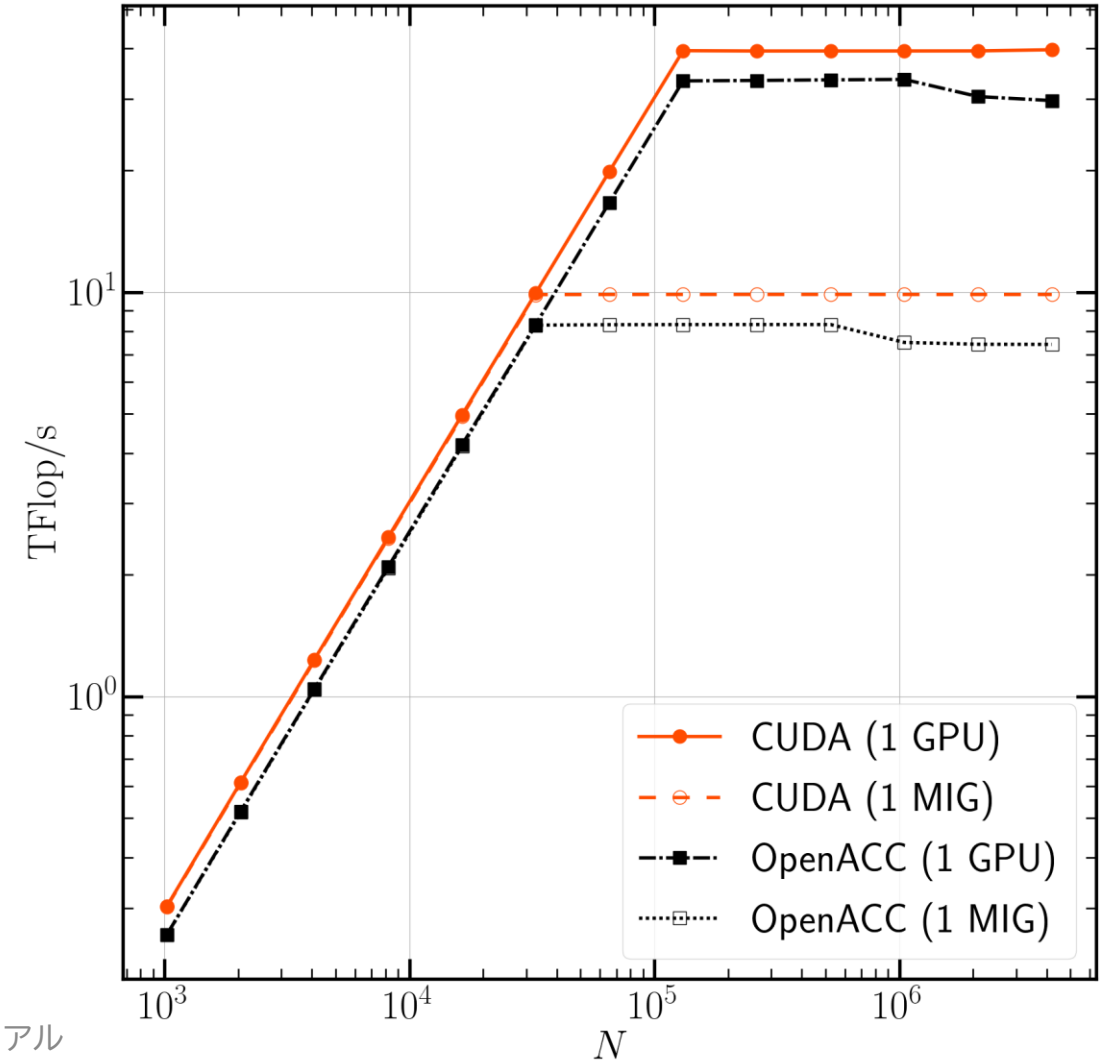
- $ cd /work/YOUR_GROUP/$USER
  - 注: YOUR_GROUP については皆様の所属グループ名(例えばshow_token コマンド出力のProjectに表示されます)に書き換えてください
- $ git clone https://github.com/ymiki-repo/nbody.git
- $ cd nbody
- $ module load nvidia hdf5
- $ cmake -S . -B build_nvidia -DBENCHMARK_MODE=ON
  - -DRELAX_RSQRT_ACCURACY=ON もつけないと遅いです(N体向け準必須オプション)
- $ cd build_nvidia
- $ ninja
  - コンパイル時メッセージに nvlink warning が出ますが, 気にしなくて良いです
- $ qsub sh/miyabi/run_nvidia.sh
- $ qsub sh/miyabi/run_nvidia_mig.sh
  - 注:ジョブスクリプト中の #PBS -W group_list= の行は皆様の所属グループ名(例えばshow_token コマンド出力のProjectに表示されます)に書き換えてください

簡単な動作テスト(CUDA単体動作編)

- $ cd /work/YOUR_GROUP/$USER
  - 注: YOUR_GROUP については皆様の所属グループ名(例えばshow_token コマンド出力のProjectに表示されます)に書き換えてください
- $ git clone https://github.com/ymiki-repo/nbody.git
- $ cd nbody
- $ module purge
- $ module load cuda
- $ module use /work/share/opt/modules/lib
  - システムで提供しているHDF5はNVHPCビルド版なので, ここでは個別ビルド版を使用
- $ module load hdf5
- $ cmake -S . -B build_cuda -DUSE_CUDA=ON -DBENCHMARK_MODE=ON
- $ cd build_cuda
- $ ninja
- $ qsub sh/miyabi/run_cuda.sh
- $ qsub sh/miyabi/run_cuda_mig.sh
  - 注:ジョブスクリプト中の #PBS -W group_list= の行は皆様の所属グループ名(例えばshow_token コマンド出力のProjectに表示されます)に書き換えてください

動作テスト(N体計算)の性能測定結果

- 性能測定結果は log/collapse_run.csv に出力されている
  - 詳細は https://github.com/ymiki-repo/nbody を参照
  - この測定では全て単精度(FP32)を使用
  - GPU自体が並列計算機
    - 全ての演算器が埋まるまでは
      性能は粒子数Nに比例して増加
  - OpenACC版はCUDA版の3/4程度
  - MIG実行時は1 GPU使用時の1/4程度
    - 使用できるSM数が約1/4のため,
      性能が飽和する粒子数も1 GPU版の1/4程度



簡単な動作テスト(複数ノード(=複数GPU)使用編)

- `$ cd /work/YOUR_GROUP/$USER`
  - 注: YOUR_GROUP については皆様の所属グループ名(例えばshow_token コマンド出力のProjectに表示されます)に書き換えてください
- `$ cp /work/share/samples/openacc_mpi_miyabi.tar.gz .`
  - https://github.com/hoshino-UTokyo/lecture_openacc_mpi がベースです
  - オリジナル版に含まれていたサンプルコードのうち一部のみを抜粋しています
- `$ tar -xvf openacc_mpi_miyabi.tar.gz`
- `$ cd lecture_openacc_mpi/[C F]` # CかFortranか好きな方を選択
- `$ cd openacc_mpi_diffusion/02_openacc`
  - C言語の場合には 03_openacc_overlap もあり, 03の方がはるかに高性能です
- `$ module purge`
- `$ module load nvidia nv-hpcx`
- `$ make`
- `$ qsub run_[1 2 4]gpu.sh`
  - 注:ジョブスクリプト中の #PBS -W group_list= の行は皆様の所属グループ名(例えばshow_token コマンド出力のProjectに表示されます)に書き換えてください

PBS/TCS/NQSV の簡易対照表

設定内容	東大Wisteria（TCS）	Miyabi（PBS）	筑波大Pegasus（NQSV）
キュー名	#PJM -L rscgrp=キュー	#PBS -q キュー	
グループ名 （プロジェクト名）	#PJM -g グループ	#PBS -W group_list=グループ	#PBS -A グループ
実行時間	#PJM -L elapse=HH:MM:SS	#PBS -l walltime=HH:MM:SS	#PBS -l elapstim_req=HH:MM:SS
ジョブ名	#PJM -N ジョブ	#PBS -N ジョブ	
使用ノード数N	#PJM -L node=N	#PBS -l select=N	#PBS -b N
ノードあたり MPIプロセス数P	#PJM --mpi proc=P	#PBS -l select=N:mpiprocs=P （mpiexecのオプションも付与）	mpiexecのオプションで指定
プロセスあたり OpenMPスレッド数T	#PJM --omp thread=T	#PBS -l select=N:ompthreads=T （mpiprocsとの同時指定も可）	#PBS -v OMP_NUM_THREADS=T
ジョブ投入ディレクトリ	${PJM_O_WORKDIR}	${PBS_O_WORKDIR}	

- ホストファイルなどはモジュールファイル側で自動的に渡す設定になっています（デフォルト設定となっている nv-hpcx で確認済み）

ジョブスクリプト例


```
#!/bin/bash
#PBS -q debug-g
#PBS -l select=4:mpiprocs=1
#PBS -l walltime=00:05:00
#PBS -W group_list=gz00

module purge
module load nvidia nv-hpcx

cd ${PBS_O_WORKDIR}
mpiexec ./run

exit 0
```

複数GPU使用の最小構成スクリプト
- ノードあたり 1 MPIプロセス使用
  （MPIプロセスあたり 1 GPU使用）

```
#!/bin/bash
#PBS -q debug-mig
#PBS -l select=1
#PBS -l walltime=00:05:00
#PBS -W group_list=gz00

module purge
module load cuda
module use /work/share/opt/modules/lib
module load hdf5

cd ${PBS_O_WORKDIR}
bin/cuda_memcpy_shmem

exit 0
```

自作モジュールを呼ぶ際の例
- /work/ 配下で自分が読み書きできる領域にライブラリやモジュールファイルなどを配置
  （←の例では /work/share/opt/modules/lib/ 以下にモジュールファイルがある）
  - 注:皆様は /work/share/ 以下にはファイルを作成できません
- module use で自作モジュールを参照可能にする
- あとは通常通り module load する

バッチキューの設定方法

- Miyabi でのバッチ処理は, PBS Proで管理
  - Wisteria/BDEC-01 での TCS では pj* となっているものを q* に変更
  - Cygnus, Pegasus での NQSV とコマンド体系は同じ(作り込みコマンドは除く)
- 主要コマンド:
  - ジョブの投入: `qsub <ジョブスクリプト名>`
  - 自分が投入したジョブの状況確認: `qstat`
  - 投入ジョブの削除: `qdel <ジョブID>`
  - 計算ノードの込み具合を見る: `qstat --rscuse`
  - バッチキューの状態を見る: `qstat --rsc`
  - バッチキューの詳細構成を見る: `qstat --rsc -x`
  - 投げられているジョブ数を見る: `qstat --rsc -b`
  - 過去の投入履歴を見る: `qstat -H`
  - 同時に投入できる数・実行できる数を見る: `qstat --limit`

moduleの指定

- コンパイラ・ライブラリ等の環境をセットアップ

```
$ module load cuda gcc hpcx
```
または
```
$ module load nvidia nv-hpcx
```

- 現在指定済みのmoduleを確認するには

```
$ module list
```

- 困った時(module環境を整理したくなった時)は

```
$ module purge
```

moduleの一覧

- 現在利用中の環境で追加できるものを確認
  ```
  $ module avail
  ```
- 使い方の確認(例はModuleNameというmoduleのヘルプを表示)
  ```
  $ module help ModuleName
  ```
- 設定されるPATHなどの確認(例はModuleNameというmoduleの場合)
  ```
  $ module show ModuleName
  ```

- 全ての環境を確認(Miyabi 向けに提供されるコマンド)
  ```
  $ show_module
  ```

```
ApplicationName  ModuleName          NodeGroup     BaseCompiler/MPI
-------------------------------------------------------------------
Apptainer        apptainer/1.3.5     Login-G       -
Apptainer        apptainer/1.3.5     Miyabi-G      -
...
```

不具合などに遭遇した際の連絡先

- Miyabi の相談窓口にご連絡ください
  https://forms.office.com/r/rNa3izFYXr
  - 「1. システム」で Miyabi にチェックを入れてください