

東北大学金属材料研究所
アクセラレータサーバマニュアル

2021年5月27日

東北大学金属材料研究所
計算材料学センター

目次

1	アクセラレータサーバ概要	1-3
1.1	構成・スペック	1-4
1.2	ノード構成	1-4
2	ログイン方法	2-5
2.1	SSH 鍵の作成	2-6
2.2	フロントエンドへのログイン方法	2-6
2.3	パスワード変更方法	2-6
3	ストレージの構成と使用方法	3-7
3.1	ストレージの構成と使用方法	3-8
4	ジョブの投入・管理	4-9
4.1	ジョブ投入コマンド	4-10
4.1.1	ジョブの投入コマンド(qsub コマンド)	4-10
4.1.2	実行スクリプトの書式	4-12
4.1.3	インタラクティブモード	4-14
4.1.4	共有キューCA_001・CA_001g へのジョブ投入方法	4-15
4.2	ジョブ管理コマンド	4-16
4.2.1	ユーザー自身のジョブの状態を確認	4-16
4.2.2	ジョブの状態を確認	4-17
4.2.3	キュー状態を確認	4-19
4.2.4	サーバ状態を確認	4-20
4.2.5	ジョブの強制終了	4-21
4.3	利用実績確認コマンド	4-22
4.4	ジョブ投入・スクリプト関連資料	4-23
4.4.1	MPI ジョブの実行方法	4-23
4.5	キュー一覧	4-24
5	コンパイラ・ライブラリ使用方法	5-26
5.1	コンパイラ使用方法	5-27
5.1.1	Intel コンパイラ	5-27
5.1.2	PGI コンパイラ	5-30
5.1.3	GNU コンパイラ	5-32
5.1.4	nvcc コンパイラ	5-34
5.2	ライブラリ使用方法	5-35
5.2.1	Intel MKL	5-35

5.2.2	cuBLAS.....	5-36
5.2.3	cuDNN.....	5-37
6	アプリケーション使用方法.....	6-38
6.1	アプリケーション一覧.....	6-39
6.2	VASP.....	6-41
6.3	QUANTUM ESPRESSO.....	6-46
6.4	LAMMPS.....	6-49
6.5	Gaussian16.....	6-52
6.6	CRYSTAL.....	6-54
6.7	WIEN2k.....	6-55
6.8	SIESTA.....	6-57
6.9	ABINIT.....	6-58
6.10	CPMD.....	6-59
6.11	MaterialsStudio.....	6-60
6.11.1	ライセンスサーバ設定方法.....	6-61
6.11.2	Gateway 設定方法.....	6-62
6.11.3	実行方法.....	6-64
6.11.4	CASTEP の実行方法.....	6-64
6.11.5	Dmol3 の実行方法.....	6-67
6.11.6	ジョブの実行確認.....	6-69
6.11.7	ジョブのキャンセル.....	6-69
6.12	Wannier90.....	6-71
7	機械学習環境 使用方法.....	7-72
7.1	機械学習環境一覧.....	7-73
7.2	Chainer.....	7-74
7.3	Keras.....	7-75
7.4	Caffe.....	7-76
7.5	Jupyter Notebook.....	7-77
7.6	DIGITS.....	7-79
8	Python 使用方法.....	8-80
8.1	Python の利用について.....	8-81
8.2	pyenv 環境の構築.....	8-81
8.3	環境変数の設定.....	8-81
8.4	動作確認.....	8-81
8.5	基本的な使い方.....	8-81
8.6	実行方法.....	8-82

1

1 アクセラレータサーバ概要

[1.1 構成・スペック](#)

[1.2 ノード構成](#)

1.1 構成・スペック

アクセラレータサーバのスペック

サーバ名	アクセラレータサーバ	フロントエンドサーバ	並列計算サーバ
機種名	Cray CS-Storm 500GT	Cray CS500	HPE ProLiant DL360 Gen 10
サーバ台数	29 台	2 台	17 台
CPU	Intel Xeon Gold 6150 ・周波数 : 2.7 GHz ・CPU コア数 : 18 Core ・搭載数 : 2 基/サーバ	Intel Xeon E5-2695v4 ・周波数 : 2.1 GHz ・CPU コア数 : 18 Core ・搭載数 : 2 基/サーバ	Intel Xeon Gold 6154 ・周波数 : 3.0 GHz ・CPU コア数 : 18 Core ・搭載数 : 2 基/サーバ
アクセラレータ	NVIDIA Tesla V100 for PCIe ・演算性能 : 7.0 TFlops ・GPU コア数 : 5,120 Core ・搭載数 : 10 基/サーバ	-	-
主記憶容量	768 GiB/サーバ	128 GiB/サーバ	576 GiB/サーバ

1.2 ノード構成

アクセラレータサーバのノード構成

ノード種別	用途	ノード数	設置場所
フロントエンドノード	ジョブ投入用ノード	2 ノード	計算材料学センター 101 室
計算ノード (キュー: CA_001, A_004)	計算を行うノード	23 ノード	計算材料学センター 101 室
計算ノード (キュー: IA_001g, CA_001g, DA_002g)	計算を行うノード	6 ノード	2 号館 713 室
計算ノード (キュー: IC_001, DC_002, C_002, C_004)	計算を行うノード	17 ノード	計算材料学センター 101 室

2

2 ログイン方法

[2.1 SSH 鍵の作成](#)

[2.2 フロントエンドへのログイン方法](#)

[2.3 パスワード変更方法](#)

2.1 SSH 鍵の作成

初めてシステムへログインする方は、事前に[公開鍵登録システムへ](#)ログインし、SSH 鍵を作成する必要があります。

2.2 フロントエンドへのログイン方法

ssh リレーサーバ `cms-ssh.sc.imr.tohoku.ac.jp` にログインします。

```
$ ssh -l username cms-ssh.sc.imr.tohoku.ac.jp
```

アクセラレータサーバのフロントエンドサーバである `gpu` へログインします。

```
$ ssh gpu
```

詳細は[ログインのページ](#)をご覧ください。

2.3 パスワード変更方法

`passwd` コマンドでフロントエンドサーバへのログインパスワードを変更することができます。

Current password には現在のパスワードを入力し、New password に以下のルールに従ったパスワードを設定してください。

- (1) 文字数は 10 文字以上
- (2) 小文字の英字を 1 文字以上使用
- (3) 大文字の英字を 1 文字以上使用
- (4) 数字を 1 文字以上使用
- (5) 特殊文字 (!, #, \$ など) を 1 文字以上使用

```
$ passwd
Current Password: [現在のパスワード]
New password: [新しいパスワード]
Retype new password: [新しいパスワード]
```

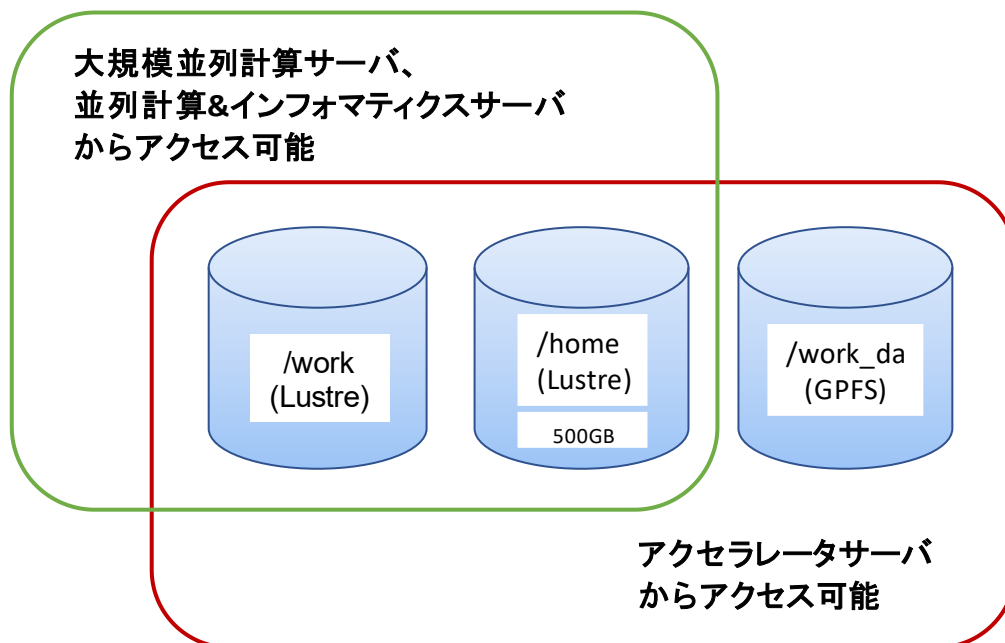
3

3 ストレージの構成と使用方法

[3.1 ストレージの構成と使用方法](#)

3.1 ストレージの構成と使用方法

スーパーコンピューティングシステムのストレージ構成を以下に示します。



ストレージの構成と使用方法

アクセス可能なマシン	領域名	quota	説明
①大規模並列計算サーバ ②アクセラレータサーバ ③並列計算&インフォマティクスサーバ	home/UID	500GB	ユーザーのホームディレクトリ。スパコンシステムのデータ全般を保存します。
①大規模並列計算サーバ ②アクセラレータサーバ ③並列計算・インフォマティクスサーバ	work/xxx	なし	高速な Lustre 領域です。出力ファイルの合計が 500GB 以上となる場合は scratch を利用してください。データは home 領域に移し、不要なデータは削除するようにしてください。
	work/scratch/xxx	なし	Lustre 領域です。Gaussian などの強烈な IO が発生する一時ファイルを保存するための領域です。1ヶ月間アクセスがないファイルは自動的に削除されます。
アクセラレータサーバ	work_da	なし	GPFS 領域です。キューIA_001g、DA_002g を利用する場合はこの領域からジョブを投入してください。アクセラレータサーバから Lustre 領域へジョブ投入する前のデバッグ領域として利用してください。データは home 領域に移し、不要なデータは削除するようにしてください。

(*)UID: ユーザーアカウント名

xxx: ユーザーが作成した任意のディレクトリまたはファイル名

(*)/work 以下の scratch 領域は 1ヶ月間アクセスがないファイルは自動的に削除されます。

4

4 ジョブの投入・管理

[4.1 ジョブの投入コマンド](#)

[4.2 ジョブ管理コマンド](#)

[4.3 利用実績確認コマンド](#)

[4.4 ジョブ投入・スクリプト関連資料](#)

[4.5 キュー一覧](#)

4.1 ジョブ投入コマンド

4.1.1 ジョブの投入コマンド(qsub コマンド)

アクセラレータサーバのキューにジョブを投入します。

なお、オプションは実行するスクリプトファイルにおいて#PBS の PBS 指示文でも指定可能です。

詳細は各詳細マニュアルを参照して下さい。

注: フロントエンドノードでは直接プログラムを実行せず、qsub コマンドでジョブとしてキューに投入してください。直接プログラムが実行されていた場合、他のユーザーへ影響があるため、管理者によりキャンセルされる場合がありますのでご了承ください。

(1) 書式

```
$ qsub [-q キュー名] [-l select=ノード数] [-N ジョブ名] [-M 電子メールアドレス] [-m 電子メール通知の指定] [-l walltime=経過時間上限] [実行するスクリプトファイル]
```

(2) オプション一覧

オプション	設定値
-q キュー名	キュー名を指定します。 キュー一覧を参照して下さい。
-l select=ノード数	使用するノード数を指定します。 省略した場合のノード数はキューのデフォルト値となります。(4.5 キュー一覧参照)
-N ジョブ名	ジョブ名を指定します。 ジョブ名は最大 236 文字まで指定できます。 リアルタイムジョブ参照システムでは 64 文字まで表示されます。 省略した場合はシステムが割り当てます。
-M 電子メールアドレス	受信する電子メールアドレスを指定します。 メールを受信する場合は-m オプションの指定が必須です。
-m 電子メール通知の指定	電子メール送信のポイントを指定します。 メール受信する場合は-M オプションの指定が必須です。
-l walltime=経過時間上限	ジョブの経過時間上限を指定します。 省略した場合の経過時間上限はキューのデフォルト値となります。(4.5 キュー一覧参照) 適切な値を設定することでキュー待ちのジョブが実行しやすくなります。
-l ライセンス種類=使用ライセンス	ライセンス管理対象のアプリケーション使用時に使用ライセンス数を指定します。

数	省略した場合はライセンス管理対象アプリケーションを使用しないとします。ライセンスの指定についてはアプリケーションの実行方法を参照して下さい。
---	--

(3) 使用例

・キューA_004を使用して、ノード2ノード使用、経過時間上限を1時間、スクリプトファイルはhello.sh

```
$ qsub -q A_004 -l select=2 -l walltime=1:00:00 hello.sh
```

スクリプトで指定する場合

```
#!/bin/sh
#PBS -q A_004
#PBS -l select=2
#PBS -l walltime=1:00:00
:
:
:
```

・キューDA_002gを使用して、ジョブ開始及び終了時にuserA@test.comに送信、スクリプトファイルはhello.sh

```
$ qsub -q DA_002g -M userA@test.com -m be hello.sh
```

スクリプトで指定する場合

```
#!/bin/sh
#PBS -q DA_002g
#PBS -M userA@test.com
#PBS -m be
:
:
:
```

・キューA_004を使用して、ジョブ名をTEST、スクリプトファイルはhello.sh

```
$ qsub -q A_004 -N TEST hello.sh
```

スクリプトで指定する場合

```
#!/bin/sh
#PBS -q A_004
#PBS -N TEST
:
:
:
```

4.1.2 実行スクリプトの書式

アクセラレータサーバ上でプログラムを動作させる際に使用する実行スクリプトの書式について説明します。実行スクリプトを必要とするアプリケーションを実行する際は、事前に実行スクリプトを作成しておく必要があります。/home 領域よりも/work 領域のほうが I/O 性能が良いため、以下の例を参考に/work 領域にデータをコピーして実行し、実行後に結果をジョブ投入したディレクトリに移動するようにしてください。

詳細は各詳細マニュアルを参照して下さい。

(1) MPI を使用しないジョブの実行

```
#!/bin/sh
#PBS -l select=1
#PBS -q キュー名
#PBS -N ジョブ名
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

実行プログラム > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm
-rf $WORKDIR; fi
```

ジョブ投入したディレクトリを/work 領域にコピーし、/work 領域に移動

プログラムの実行

ジョブが終了後、結果をジョブ投入したディレクトリに移動

・例 プログラム a.out を実行

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N sample
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

./a.out > result.out 2> result.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm
-rf $WORKDIR; fi
```

(2) MPI を使用したジョブの実行

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 | -ppn ノードあたりの並列数 ] -hostfile
$PBS_NODEFILE 実行プログラム > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm
-rf $WORKDIR; fi
```

ジョブ投入したディレクトリを/work 領域にコピーし、/work 領域に移動

プログラムの実行

ジョブが終了後、結果をジョブ投入したディレクトリに移動

・例 Intel コンパイラでコンパイルしたプログラム a.out を 2 ノード、72MPI プロセスで実行

```
#!/bin/sh
#PBS -l select=2
#PBS -q A_004
#PBS -N mpi
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 72 -ppn 36 -hostfile $PBS_NODEFILE ./a.out >
result.out 2> result.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm
-rf $WORKDIR; fi
```

4.1.3 インタラクティブモード

インタラクティブモードでジョブを投入します。

qsub コマンドにオプション-I(i の大文字)を付与し、キューに IA_001g、CA_001、CA_001g、IC_001 のいずれかを指定します。

(1) 書式

```
$ qsub -I -q キュー名
```

(2) 使用例

```
$ qsub -I -q IA_001g
qsub: waiting for job 22351.gpu1 to start
qsub: job 22351.gpu1 ready

-bash-4.2$ ./a.out
```

4.1.4 共有キューCA_001・CA_001g へのジョブ投入方法

共有キューCA_001 および CA_001g にジョブを投入する方法について記載します。

共有キューはノードを他のジョブと共有して使用するキューです。

ジョブにはデフォルトで 1CPU、1GPU が割り当てられます。qsub コマンドのオプション指定により、最大で 18CPU、5GPU まで使用可能です。インタラクティブモードでの実行も可能です。

キューに CA_001 または CA_001g を指定します。

(1) 書式

```
$ qsub -q キュー名 [ -I ] [ -l select=1[:ncpus=CPU  
数] [:ngpus=GPU 数] [実行するスクリプトファイル]
```

(2) 使用例

・例 キューCA_001 を利用して、2CPU、1GPU でインタラクティブモード実行するコマンド

```
$ qsub -I -q CA_001 -l select=1:ncpus=2:ngpus=1  
qsub: waiting for job 22351.gpu1 to start  
qsub: job 22351.gpu1 ready  
  
-bash-4.2$ ./a.out
```

・例 キューCA_001g を利用して、18CPU、5GPU でプログラム a.out を実行するスクリプト
CA_001g を利用する場合は/work_da 領域からジョブを投入してください。

```
#!/bin/sh  
#PBS -l select=1:ncpus=18:ngpus=5  
#PBS -q CA_001g  
#PBS -N sample  
  
DIRNAME=`basename $PBS_O_WORKDIR`  
WORKDIR=/work/$USER/$PBS_JOBID  
mkdir -p $WORKDIR  
cp -raf $PBS_O_WORKDIR $WORKDIR  
cd $WORKDIR/$DIRNAME  
  
mpirun -np 18 -ppn 18 -hostfile $PBS_NODEFILE ./a.out >  
result.out 2> result.err  
  
cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm  
-rf $WORKDIR; fi
```


4.2 ジョブ管理コマンド

詳細は各詳細マニュアルを参照して下さい。

4.2.1 ユーザー自身のジョブの状態を確認

(1) 説明

ユーザー自身のジョブの状態を表示します。

(2) 書式

```
statj [-x] [ [job_identifier | destination] ...]
```

(3) オプション一覧

オプション	設定値
-x	終了したジョブを含めてジョブ情報の表示

(4) 使用例

```
userA@gpu2:~> statj
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Elap Time
3413.gpu1	userA	A_004	STDIN	231503	1	36	690gb	24:00	R	00:00

4.2.2 ジョブの状態を確認

(1) 説明

アクセラレータサーバのジョブの状態を表示します。

(2) 書式

```
Default format:  
qstat [-a] [-p] [-J] [-t] [-x] [ [job_identifier | destination] ...]  
Long format:  
qstat -f [-p] [-J] [-t] [-x] [ [job_identifier | destination] ...]
```

(3) オプション一覧

オプション	設定値
-a	メモリ量や経過時間、ジョブの状態の経過時間などを表示
-p	ジョブ完了割合の表示
-J	アレイジョブに関する制限表示
-t	ジョブ情報の表示
-x	終了したジョブを含めてジョブ情報の表示
-f	Long format にて表示

(4) 使用例

```
userA@gpu2:~> qstat -a
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Time	S	Time
3390.gpu1	userA	A_004	abinit	193347	4	144	2760gb	72:00	R	47:28
3401.gpu1	userA	A_004	prog9_1	121974	4	144	2760gb	72:00	R	47:26

```
userA@gpu2:~> qstat -p
```

Job id	Name	User	% done	S	Queue
3390.gpu1	abinit	userA	2	R	A_004
3401.gpu1	prog9_1	userA	0	R	A_004

```
userA@gpu2:~> qstat -t
```

Job id	Name	User	Time Use	S	Queue
3390.gpu1	abinit	userA	00:00:01	R	A_004
3401.gpu1	prog9_1	userA	00:00:01	R	A_004

```
userA@gpu2:~ > qstat -x
```

Job id	Name	User	Time Use	S	Queue
2235.gpu1	prog9_2	userA	00:00:03	F	SA_016
2236.gpu1	vasp4	userA	00:00:01	F	SA_016
2237.gpu1	prog9_1	userA	00:00:01	F	SA_016

以下略

```
userA@gpu2:~> qstat -f 3390.gpu1
```

```
Job Id: 3390.gpu1
  Job_Name = abinit
  Job_Owner = userA@gpu2
resources_used.cpubercent = 10
resources_used.cput = 00:00:01
resources_used.mem = 12836kb
resources_used.ncpus = 72
以下略
```

4.2.3 キュー状態を確認

(1) 説明

アクセラレータサーバのキューの状態を表示します。

(2) 書式

```
Default format:
statq [destination ...]
Long format:
statq -f [destination ...]
```

(3) オプション一覧

オプション	設定値
-f	Long format にて表示

(4) 使用例

```
userA@gpu2:~> statq
Queue      Max  Tot  Ena Str  Que  Run  Hld  Wat  Trn  Ext Type
-----
workq      0   0  no  yes   0   0   0   0   0   0  Exec
A_004      0   0  yes yes   0   0   0   0   0   0  Exec
MA_008     0   1  yes yes   0   1   0   0   0   0  Exec
SA_016     0   0  yes yes   0   0   0   0   0   0  Exec
DA_002g   0   0  yes yes   0   0   0   0   0   0  Exec
DC_002     0   0  yes yes   0   0   0   0   0   0  Exec
C_002     0   0  yes yes   0   0   0   0   0   0  Exec
          以下略

userA@gpu2:~> statq -f
Queue: workq
  queue_type = Execution
  total_jobs = 0
  state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0
Exiting:0 Begun
           :0
  enabled = False
  started = True
          以下略
```

4.2.4 サーバ状態を確認

(1) 説明

アクセラレータサーバのサーバの状態を表示します。

(2) 書式

```
Default format:  
qstat -B [destination ...]  
Long format:  
qstat -B -f [destination ...]
```

(3) オプション一覧

オプション	設定値
-B	サーバの表示
-f	Long format にて表示

(4) 使用例

```
userA@gpu2:~> qstat -B  
Server      Max Tot  Que  Run  Hld  Wat  Trn  Ext Status  
-----  
gpu1         0 1155   0    1    0    0    0    0 Active  
userA@gpu2:~> qstat -Bf  
Server: sdb  
  server_state = Active  
  server_host = sdb  
  scheduling = True  
  max_queued = [u:PBS_GENERIC=200]  
  以下略
```

4.2.5 ジョブの強制終了

(1) 説明

アクセラレータサーバのジョブを削除します。

(2) 書式

```
qdel [ -x ] [ -Wsuppress_email=<N> ] job_identifier  
[job_identifier ...]
```

(3) オプション一覧

オプション	設定値
-x	ジョブ履歴も含めてジョブ削除
-Wsuppress_email	削除時のメール送信数の制限

(4) 使用例

```
userA@gpu2:~/work/20180712_sample> qstat  
Job id          Name          User          Time Use S Queue  
-----  
3413.gpu1      abinit        userA         00:00:00 R A_004  
3414.gpu1      STDIN_gpu2_22 userA         00:00:00 R A_004  
userA@gpu2:~/work/20180712_sample> qdel 3414.gpu1  
userA@gpu2:~/work/20180712_sample> qstat  
Job id          Name          User          Time Use S Queue  
-----  
3413.gpu1      abinit        userA         00:00:00 R A_004  
userA@gpu2:~/work/20180712_sample>
```

4.3 利用実績確認コマンド

(1) 説明

システムの利用実績を表示します。

(2) 書式

```
jobtime
```

(3) 表示項目

項目	内容
Last Updated	更新日時
User	ユーザー名
Total	割り当てノード時間
Used	累積利用時間
Remained	残り時間

(4) 使用例

```
userA@gpu2:~ > jobtime

# Last Updated: 2018/10/01 13:45
# User      Total      Used      Remained (H)
username    500       222.32    277.68
```

4.4 ジョブ投入・スクリプト関連資料

4.4.1 MPI ジョブの実行方法

(1) 説明

MPI 環境として、IntelMPI が利用可能です。

(2) 書式

ジョブ実行には mpirun コマンドを使用します。

```
mpirun [ -np 並列数 ] [ -ppn ノードあたりの並列数 ] -hostfile  
$PBS_NODEFILE 実行プログラム
```

※ジョブのパフォーマンスを高めるために以下が成り立つよう値を指定してください。

[MPI 並列数(-np の値)] = [ノード数(#PBS -l select=の値)] × [ノードあたりの並列数(-ppn の値)]

(3) 使用例

```
#!/bin/bash  
#PBS -j oe  
#PBS -l select=1  
DIRNAME=`basename $PBS_O_WORKDIR`  
WORKDIR=/work/$USER/$PBS_JOBID  
mkdir -p $WORKDIR  
cp -raf $PBS_O_WORKDIR $WORKDIR  
cd $WORKDIR/$DIRNAME  
  
mpirun -np 36 -hostfile $PBS_NODEFILE  
/usr/local/app/ABINIT/current/src/98_main/abinit < input.files >  
result.out 2> result.err  
  
cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf  
$WORKDIR; fi
```


4.5 キュー一覧

アクセラレータサーバで使用できるキューは以下の通りです。

占有ノード

名称	占有ノード 数上限 (デフォルト)	メモリ確 保上限 [GiB]	経過時間 上限[時間] (デフォルト)	同時実行 上限	GPU 利用 上限	並列数 上限	備考
IA_001g	1(1)	690	24(1)	6	10	36	インタラクティブ用
DA_002g	2(1)	1,380	2(2)	6	20	72	デバック用
A_004	4(1)	2,760	72(24)	制限なし	40	144	
IC_001	1(1)	510	24(1)	制限なし	-	36	インタラクティブ用
DC_002	2(1)	1,020	2(2)	制限なし	-	72	デバッグ用
C_002	2(1)	1,020	72(24)	制限なし	-	72	
C_004	4(1)	2,040	72(24)	制限なし	-	72	

共有ノード

名称	CPU 利用上限 (デフォルト)	GPU 利用上限 (デフォルト)	メモリ確保 上限[GiB] (デフォルト)	経過時間 上限[時間] (デフォルト)	同時実 行上限	並列数 上限	備考
CA_001	18(1)	5(1)	345(69)	72(24)	10	18	インタラク ティブ実行 も可能
CA_001g	18(1)	5(1)	345(69)	72(24)	10	18	インタラク ティブ実行 も可能

注: キューIA_001g、CA_001g、DA_002g を利用する場合は/work_da 領域からジョブを投入してください。

本センターでは、MASAMUNE-IMR 上で超大規模計算用のアプリケーションソフトを開発し、新しい計算材料学の方法論を構築することを目標とされている研究者を応援したいと思っています。そのため、超大規模計算を行う自作アプリケーションソフトを使用される方は、申請書を提出して頂くことで、以下のキューも使用できます。ご利用になりたい方は[お問い合わせフォーム](#)からご連絡ください。

名称	占有ノード 数上限 (デフォルト)	メモリ確 保上限 [GiB]	経過時間 上限[時間] (デフォルト)	同時実行 上限	GPU 利用 上限	並列数 上限	備考
LA_004	4(4)	2,760	168(96)	1	40	144	運用中も実

MA_008	8(8)	5,520	72(24)	1	80	288	行
SA_016	16(16)	11,040	72(72)	1	160	576	定期保守後 に実行

5

5 コンパイラ・ライブラリ使用方法

[5.1 コンパイラ使用方法](#)

[5.2 ライブラリ使用方法](#)

5.1 コンパイラ使用方法

アクセラレータサーバでは以下のコンパイラを提供しています。

詳細は各詳細マニュアルを参照して下さい。

コンパイラ名称	バージョン	備考
Intel コンパイラ Fortran/C/C++	19.1.3.304 19.1.0.166 19.0.2.187 18.0.3.222 17.0.4.196	デフォルト: 17.0.4.196
PGI コンパイラ Fortran/C/C++	20.4 19.10 19.1 18.10 18.5	デフォルト: 19.1
nvcc コンパイラ(CUDA Toolkit)	10.2.89 10.1.243 9.2.148 9.0.176 8.0.44	デフォルト: 9.0.176

5.1.1 Intel コンパイラ

(1) プログラム環境の設定

intel コンパイラがデフォルトで設定されます。

バージョンを切り替える場合には以下を実行して下さい。

```
$ module avail intel
intel/17.0.4(default) intel/18.0.3 intel/19.0.2
intel/19.1.0 intel/19.1.3

$ module switch intel/17.0.4 intel/18.0.3
```

(2) コンパイル方法

(ア) コマンド

・serial

言語	コマンド	実行形式
Fortran	ifort	ifort [オプションの並び] ファイルの並び

C	icc	icc [オプションの並び] ファイルの並び
C++	icpc	icpc [オプションの並び] ファイルの並び

・MPI

言語	コマンド	実行形式
Fortran	mpiifort	mpiifort [オプションの並び] ファイルの並び
C	mpiicc	mpiicc [オプションの並び] ファイルの並び
C++	mpiicpc	mpiicpc [オプションの並び] ファイルの並び

(イ) オプション

・最適化オプション他

オプション名	説明
-o outfile	出力ファイル名を指定します。省略時には a.out が設定されます。
-llibrary_name	リンクするライブラリ名を指定します。
-Llibrary_path	ライブラリの検索パスを指定します。
-O0 -O1 -O2 -O3	最適化オプションを指定します。デフォルトは-O2 です。
-fast	プログラム全体の速度を最大限にします。次のオプションが内部的に設定されます。 -ipo、-O3、-no-prec-div、-static、-fp-model fast=2、 -xHost
-parallel	自動並列化を有効にしてコンパイルする場合に指定します。
-openmp	OpenMP 指示文を有効にしてコンパイルする場合に指定します。
-xcore-avx512	インテル AVX-512 命令をターゲットに指定します。当該オプションの指定を推奨します。

・Fortran 専用オプション

オプション名	説明
-free fixed	プログラムが自由形式(free)であるか固定形式(fixed)であるかを指定します。

・デバッグ用オプション

オプション名	説明
-g	デバッグ情報を出力します。
-traceback	エラーが発生した場合にトレースバックを表示します。
-fpe[0-3]	例外処理をトラップします。(-fpe0 が最も詳細)

(3) 使用例

- ・固定形式の Fortran のソースプログラム hello.f から hello.out という実行モジュールを作成

```
$ ifort -xcore-avx512 -fixed -o hello.out hello.f
```

- ・自由形式の Fortran のソースプログラム hello.f90 から hello.out という実行モジュールを作成

```
$ ifort -xcore-avx512 -free -o hello.out hello.f90
```

- ・固定形式の Fortran のソースプログラム hello.f から自動並列化した hello.out という実行モジュールを作成

```
$ ifort -xcore-avx512 -fixed -parallel -o hello.out hello.f
```

- ・自由形式の Fortran のソースプログラム hello.f90 から自動並列化した hello.out という実行モジュールを作成

```
$ ifort -xcore-avx512 -free -parallel -o hello.out hello.f90
```

- ・C のソースプログラム hello.c から hello.out という実行モジュールを作成

```
$ icc -xcore-avx512 -o hello.out hello.c
```

- ・C のソースプログラム hello.c から自動並列化した hello.out という実行モジュールを作成

```
$ icc -xcore-avx512 -parallel -o hello.out hello.c
```

- ・C++のソースプログラム hello.cpp から hello.out という実行モジュールを作成

```
$ icpc -xcore-avx512 -o hello.out hello.cpp
```

- ・C++のソースプログラム hello.cpp から自動並列化した hello.out という実行モジュールを作成

```
$ icpc -xcore-avx512 -parallel -o hello.out hello.cpp
```

5.1.2 PGI コンパイラ

(1) プログラム環境の設定

以下コマンドを実行してください。

```
$ module avail PrgEnv-pgi
PrgEnv-pgi/18.5 PrgEnv-pgi/18.10 PrgEnv-pgi/19.1(default)
PrgEnv-pgi/19.10 PrgEnv-pgi/20.4

$ module switch intel PrgEnv-pgi
```

(2) コンパイル方法

(ア) コマンド

・serial

言語	コマンド	実行形式
Fortran	pgf90	pgf90 [オプションの並び] ファイルの並び
C	pgcc	pgcc [オプションの並び] ファイルの並び
C++	pgc++	pgc++ [オプションの並び] ファイルの並び

・MPI

言語	コマンド	実行形式
Fortran	mpif90	mpif90[オプションの並び] ファイルの並び
C	mpicc	mpicc [オプションの並び] ファイルの並び
C++	mpic++	mpic++ [オプションの並び] ファイルの並び

(イ) オプション

・最適化オプション他

オプション名	説明
-o outfile	出力ファイル名を指定します。省略時には a.out が設定されます。
-llibrary_name	リンクするライブラリ名を指定します。
-Llibrary_path	ライブラリの検索パスを指定します。
-O0 -O1 -O2 -O3 -O4	最適化オプションを指定します。デフォルトは-O2 です。
-fast	一般的な最適化フラグセットが有効になります。
-Mconcur	自動並列化を有効にしてコンパイルする場合に指定します。
-mp	OpenMP 指示文を有効にしてコンパイルする場合に指定します。

・Fortran 専用オプション

オプション名	説明
-Mfree -Mfixed	プログラムが自由形式(free)であるか固定形式(fixed)であるかを指定します。

・デバッグ用オプション

オプション名	説明
-g -gopt	デバッグ情報を出力します。

(3) 使用例

- ・固定形式の Fortran のソースプログラム hello.f から hello.out という実行モジュールを作成

```
$ pgf90 -Mfixed -o hello.out hello.f
```

- ・固定形式の Fortran のソースプログラム hello.f から自動並列化した hello.out という実行モジュールを作成

```
$ pgf90 -Mfixed -Mconcur -o hello.out hello.f
```

- ・固定形式の Fortran のソースプログラム hello.f から OpenMP で hello.out という実行モジュールを作成

```
$ pgf90 -mp -Mfixed -o hello.out hello.f
```

- ・自由形式の Fortran のソースプログラム hello.f90 から hello.out という実行モジュールを作成

```
$ pgf90 -Mfree -o hello.out hello.f90
```

- ・自由形式の Fortran のソースプログラム hello.f90 から自動並列化した hello.out という実行モジュールを作成

```
$ pgf90 -Mfree -Mconcur -o hello.out hello.f90
```

- ・自由形式の Fortran のソースプログラム hello.f90 から OpenMP で hello.out という実行モジュールを作成

```
$ pgf90 -mp -Mfree -o hello.out hello.f90
```

- ・C のソースプログラム hello.c から hello.out という実行モジュールを作成

```
$ pgcc -o hello.out hello.c
```

- ・C のソースプログラム hello.c から自動並列化した hello.out という実行モジュールを作成

```
$ pgcc -Mconcur -o hello.out hello.c
```


- ・ C のソースプログラム hello.c から OpenMP で hello.out という実行モジュールを作成

```
$ gcc -mp -o hello.out hello.c
```

- ・ C++ のソースプログラム hello.cpp から hello.out という実行モジュールを作成

```
$ g++ -o hello.out hello.cpp
```

- ・ C++ のソースプログラム hello.cpp から自動並列化した hello.out という実行モジュールを作成

```
$ g++ -Mconcur -o hello.out hello.cpp
```

- ・ C++ のソースプログラム hello.cpp から OpenMP で hello.out という実行モジュールを作成

```
$ g++ -mp -o hello.out hello.cpp
```

5.1.3 GNU コンパイラ

(1) プログラム環境の設定

特に設定は必要ありません。

(2) コンパイル方法

(ア) コマンド

・serial

言語	コマンド	実行形式
Fortran	gfortran	gfortran [オプションの並び] ファイルの並び
C	cc	cc [オプションの並び] ファイルの並び
C++	g++	g++ [オプションの並び] ファイルの並び

・MPI

言語	コマンド	実行形式
Fortran	mpif90	mpif90 [オプションの並び] ファイルの並び
C	mpicc	mpicc [オプションの並び] ファイルの並び
C++	mpicxx	mpicxx [オプションの並び] ファイルの並び

(イ) オプション

・最適化オプション他

オプション名	説明
-o outfile	出力ファイル名を指定します。省略時には a.out が設定されます。
-llibrary_name	リンクするライブラリ名を指定します。
-Llibrary_path	ライブラリの検索パスを指定します。

-O0 -O1 -O2 -O3 -O4	最適化オプションを指定します。デフォルトは-O2 です。
-fopenmp	OpenMP 指示文を有効にしてコンパイルする場合に指定します。

・Fortran 専用オプション

オプション名	説明
-ffree-form -ffixed-form	プログラムが自由形式(free)であるか固定形式(fixed)であるかを指定します。

・デバッグ用オプション

オプション名	説明
-g	デバッグ情報を出力します。
-g0 -g1 -g2 -g3	デバッグレベルを指定します。(-g2 = -g)

(3) 使用例

- ・固定形式の Fortran のソースプログラム hello.f から hello.out という実行モジュールを作成

```
$ gfortran -ffixed-form -o hello.out hello.f
```

- ・自由形式の Fortran のソースプログラム hello.f90 から hello.out という実行モジュールを作成

```
$ gfortran -ffree-form -o hello.out hello.f90
```

- ・C のソースプログラム hello.c から hello.out という実行モジュールを作成

```
$ cc -o hello.out hello.c
```

- ・C++のソースプログラム hello.cpp から hello.out という実行モジュールを作成

```
$ g++ -o hello.out hello.cpp
```

5.1.4 nvcc コンパイラ

(1) プログラム環境の設定

デフォルトで cuda が使用できます。

バージョンを切り替える場合には以下を実行してください。

```
$ module switch cudatoolkit/9.0.176 cudatoolkit/10.1.243
```

・Intel コンパイラをバックエンドとして使う場合、デフォルトでセットアップされています。

・PGI コンパイラをバックエンドとして使う場合、PGI コンパイラをセットアップしてください。

```
$ module switch intel PrgEnv-pgi
```

(2) コンパイル方法

(ア) オプション

・バックエンドコンパイラ、及び最適化オプション他

オプション名	説明				
-ccbin コンパイラ	バックエンドコンパイラを指定します。 <table border="1"><tr><td>Intel コンパイラの場合</td><td>lcpc</td></tr><tr><td>PGI コンパイラの場合</td><td>pgc++</td></tr></table>	Intel コンパイラの場合	lcpc	PGI コンパイラの場合	pgc++
Intel コンパイラの場合	lcpc				
PGI コンパイラの場合	pgc++				
-O0 1 ...	最適化オプション(バックエンドコンパイラに渡されます)				
-Xcompiler options	最適化オプション以外のコンパイルオプション(バックエンドコンパイラに渡されます)				
-gencode options	生成するコードの CUDA バージョンを指定します。				
--machine {32 64} (-m)	32 ビット 64 ビットを指定します。				
-I include_path	インクルードヘッダの検索パスを指定します。				
-L library_path	ライブラリの検索パスを指定します。				
-l library_name	リンクするライブラリ名を指定します。				
--help (-h)	利用可能なオプションの一覧と説明を表示します。				
--version (-V)	バージョン情報を表示します。				

(3) 使用例

・バックエンドコンパイラにインテルコンパイラ(icc)を指定する場合

```
$ nvcc -ccbin icpc -m64 -gencode arch=compute_70,code=compute_70  
-o simple simple.cpp
```

・バックエンドコンパイラに PGI コンパイラ(pgc++)を指定する場合

```
$ nvcc -ccbin pgc++ -m64 -gencode arch=compute_70,code=compute_70  
-o simple simple.cpp
```

5.2 ライブラリ使用方法

アクセラレータサーバでは以下のライブラリを提供しています。

詳細は各詳細マニュアルを参照して下さい。

ライブラリ名称	バージョン	リンク可能なコンパイラ	備考
Intel MKL(インテル マス・カーネル・ライブラリー)	19.1.3.304 19.1.0.166 19.0.2.187 18.0.3.222 17.0.4.196	Intel コンパイラ	
cuBLAS	9.0 8.0	Intel コンパイラ PGI コンパイラ	
cuDNN	7.6.5 for CUDA10.2 7.6.3 for CUDA10.1 7.6.5 for CUDA9.2 7.6.3 for CUDA9.0	Intel コンパイラ PGI コンパイラ	

5.2.1 Intel MKL

Intel MKL(インテル マス・カーネル・ライブラリー)は、BLAS, LAPACK, SparseBLAS, PARDISO, Iterative Sparse Solver, FFT, 乱数生成などを含むライブラリです。

(1) プログラム環境の設定

Intel MKL はデフォルトで環境設定されます。

バージョンを切り替える場合には以下を実行してください。

```
$ module switch intel/17.0.4 intel/18.0.3
```

(2) 使用例

・固定形式の BLAS を使用した Fortran のソースプログラム hello.f から hello.out という実行モジュールを作成

```
$ ifort -mkl -o hello.out -fixed hello.f
```

- ・BLAS を使用した C のソースプログラム hello.c から hello.out という実行モジュールを作成

```
$ icc -mkl -o hello.out hello.c
```

5.2.2 cuBLAS

cuBLAS は、cuda 対応の BLAS ライブラリです。

(1) プログラム環境の設定

- ・デフォルトで cuda (含む cuBLAS) が使用できます。
- バージョンを切り替える場合には以下を実行してください。

```
$ module switch cudatoolkit/9.0.176 cudatoolkit/8.0.44
```

- ・Intel コンパイラをバックエンドとして使う場合、デフォルトでセットアップされています。

- ・PGI コンパイラをバックエンドとして使う場合、PGI コンパイラをセットアップしてください。

```
$ module switch intel PrgEnv-pgi
```

(2) 使用例

- ・インテルコンパイラ(icc)をバックエンド指定し、cuBLAS ライブラリをリンクする場合

```
$ nvcc -ccbin icpc -I.././common/inc -m64 -gencode  
arch=compute_70,code=compute_70 -o simpleCUBLAS  
simpleCUBLAS.cpp -l cublas
```

- ・PGI コンパイラ(pgc++)をバックエンド指定し、cuBLAS ライブラリをリンクする場合

```
$ nvcc -ccbin pgc++ -I.././common/inc -m64 -gencode  
arch=compute_70,code=compute_70 -o simpleCUBLAS  
simpleCUBLAS.cpp -l cublas
```

5.2.3 cuDNN

(1) プログラム環境の設定

・デフォルトで cuda (含む cuDNN) が使用できます。

バージョンを切り替える場合には以下を実行してください。

```
$ module switch cudatoolkit/9.0.176 cudatoolkit/10.1.243
```

・Intel コンパイラをバックエンドとして使う場合、デフォルトでセットアップされています。

・PGI コンパイラをバックエンドとして使う場合、PGI コンパイラをセットアップしてください。

```
$ module switch intel PrgEnv-pgi
```

(2) 使用例

・インテルコンパイラ(icc)をバックエンド指定し、cuDNN ライブラリをリンクする場合

```
$ nvcc -ccbin icpc -I../..common/inc -m64 -gencode  
arch=compute_70,code=compute_70 -o simpleCUBLAS  
simpleCUBLAS.cpp -l cudnn
```

・PGI コンパイラ(pgc++)をバックエンド指定し、cuDNN ライブラリをリンクする場合

```
$ nvcc -ccbin pgc++ -I../..common/inc -m64 -gencode  
arch=compute_70,code=compute_70 -o simpleCUBLAS  
simpleCUBLAS.cpp -l cubdnn
```

6

6 アプリケーション使用方法

[6.1 アプリケーション一覧](#)

[6.2 VASP](#)

[6.3 QUANTUM ESPRESSO](#)

[6.4 LAMMPS](#)

[6.5 Gaussian16](#)

[6.6 CRYSTAL](#)

[6.7 WIEN2k](#)

[6.8 SIESTA](#)

[6.9 ABINIT](#)

[6.10 CPMD](#)

[6.11 MaterialsStudio](#)

[6.12 Wannier90](#)

6.1 アプリケーション一覧

アクセラレータサーバでは以下のアプリケーションが利用可能です。

#	アプリケーション名称	バージョン	動作種別	利用可能キュー
1	VASP	5.4.4 (gpu) 6.1.0 (gpu) 6.1.1 (gpu) 6.1.2 (gpu) 6.2.0 (gpu)	MPI	A_004 CA_001 CA_001g
		4.6.36 5.4.4 6.1.0 6.1.1 6.1.2	MPI	C_002 C_004
2	QUANTUM ESPRESSO	6.1 (gpu tag v1.0) 6.4.1 (gpu) 6.5 (gpu) 6.6 (gpu) 6.7 (gpu)	MPI	A_004
		6.2.1 6.4.1	MPI	C_002 C_004
3	LAMMPS	31 Mar 17 5 Jun 19	MPI	A_004 CA_001 CA_001g C_002 C_004
		12 Dec 18 7 Aug19 3 Mar 20 29 Oct 20	MPI	A_004 CA_001 CA_001g

4	Gaussian 16	Rev B.01	SMP	C_002 C_004
		Rev C.01	SMP	A_004 C_002 C_004
5	CRYSTAL	17	MPI SMP	C_002 C_004
6	WIEN2k	17.1 19.1 19.2	SMP	C_002 C_004
7	SIESTA	4.0 4.1.5	MPI	C_002 C_004
8	ABINIT	8.8.2 8.10.3	MPI	C_002 C_004
9	CPMD	4.1 4.3	MPI	C_002 C_004
10	Materials Studio	2020 2019	MPI	C_002 C_004
11	Wannier90	1.2 2.1.0	Serial	A_004
		3.1.0	MPI	

6.2 VASP

VASP はライセンスをお持ちでない方は利用できません。利用希望者は[こちら](#)までその旨お問合せ下さい。ライセンスを当センターにて確認させて頂いた後利用可能となります。

以下のバージョンが利用可能です。

*注: アクセラレータサーバにて VASP6.1.1 または VASP6.1.2 を利用する場合は、モジュールを「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください。

*注: アクセラレータサーバにて VASP6.2.0 を利用する場合は、モジュールを「intel 19.0.2」「CUDA 10.2.89」へ切り替えてください。

*注: 並列計算・インフォマティクスサーバにて VASP6.1.1、VASP6.1.2 または non-collinear 版を利用する場合は、モジュールを「intel 19.1.0」へ切り替えてください。

実行モジュールの種類	実行モジュールのパス	実行キュー
VASP5.4.4 gpu 版	/usr/local/app/VASP5/current/bin/vasp_gpu	A_004 CA_001 CA_001g
VASP5.4.4 gpu+ non-collinear 版	/usr/local/app/VASP5/current/bin/vasp_gpu_ncl	A_004 CA_001 CA_001g
VASP6.1.0 gpu 版	/usr/local/app/VASP6/vasp.6.1.0/bin/vasp_gpu	A_004 CA_001 CA_001g
VASP6.1.0 gpu+ non-collinear 版	/usr/local/app/VASP6/vasp.6.1.0/bin/vasp_gpu_ncl	A_004 CA_001 CA_001g
VASP6.1.1 gpu 版	/usr/local/app/VASP6/current/bin/vasp_gpu *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.1.1 gpu+ non-collinear 版	/usr/local/app/VASP6/current/bin/vasp_gpu_ncl *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.1.2 gpu 版	/usr/local/app/VASP6/vasp.6.1.2/bin/vasp_gpu *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.1.2 gpu+ non-collinear 版	/usr/local/app/VASP6/vasp.6.1.2/bin/vasp_gpu_ncl *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.2.0 gpu 版	/usr/local/app/VASP6/vasp.6.2.0/bin/vasp_gpu *「intel 19.0.2」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g

VASP6.2.0 gpu+ non-collinear 版	/usr/local/app/VASP6/vasp.6.2.0/bin/vasp_gpu_ncl *「intel 19.0.2」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP4.6.36	/usr/local/app/VASP4/current/vasp	C_002 C_004
VASP4.6.36 Gamma 点 版	/usr/local/app/VASP4/vasp.4.6_gamma/vasp	C_002 C_004
VASP5.4.4 Standard 版	/usr/local/app/VASP5/current/bin/vasp_std	C_002 C_004
VASP5.4.4 Gamma 点 版	/usr/local/app/VASP5/current/bin/vasp_gam	C_002 C_004
VASP5.4.4 non-collinear 版	/usr/local/app/VASP5/current /bin/vasp_ncl *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.0 Standard 版	/usr/local/app/VASP6/vasp.6.1.0/bin/vasp_std	C_002 C_004
VASP6.1.0 Gamma 点 版	/usr/local/app/VASP6/vasp.6.1.0/bin/vasp_gam	C_002 C_004
VASP6.1.0 non-collinear 版	/usr/local/app/VASP6/vasp.6.1.0/bin/vasp_ncl *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.1 Standard 版	/usr/local/app/VASP6/current/bin/vasp_std *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.1 Gamma 点 版	/usr/local/app/VASP6/current/bin/vasp_gam *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.1 non-collinear 版	/usr/local/app/VASP6/current /bin/vasp_ncl *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.2 Standard 版	/usr/local/app/VASP6/vasp.6.1.2/bin/vasp_std *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.2 Gamma 点 版	/usr/local/app/VASP6/vasp.6.1.2/bin/vasp_gam *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.2 non-collinear 版	/usr/local/app/VASP6/vasp.6.1.2/bin/vasp_ncl *「intel 19.1.0」へ切り替えてください	C_002 C_004

Wannier90 をリンクした VASP 6.1.1 および VASP6.1.2 も利用可能です。

実行モジュール の種類	実行モジュールのパス	実行キュー
VASP6.1.1 gpu (Wannier90) + Standard 版	/usr/local/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_gpu /usr/local/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_gpu *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g

VASP6.1.1 gpu (Wannier90) + non-collinear 版	/usr/local/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_gpu_ncl /usr/local/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_gpu_ncl *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.1.1 (Wannier90) Standard 版	/usr/local/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_std /usr/local/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_std *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.1 (Wannier90) Gamma 点版	/usr/local/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_gam /usr/local/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_gam *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.1 (Wannier90) non-collinear 版	/usr/local/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_ncl /usr/local/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_ncl *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.2 gpu (Wannier90) + Standard 版	/usr/local/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_gpu /usr/local/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_gpu *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.1.2 gpu (Wannier90) + non-collinear 版	/usr/local/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_gpu_ncl /usr/local/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_gpu_ncl *「intel 18.0.3」「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g
VASP6.1.2 (Wannier90) Standard 版	/usr/local/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_std /usr/local/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_std *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.2 (Wannier90) Gamma 点版	/usr/local/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_gam /usr/local/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_gam *「intel 19.1.0」へ切り替えてください	C_002 C_004
VASP6.1.2 (Wannier90) non-collinear 版	/usr/local/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_ncl /usr/local/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_ncl *「intel 19.1.0」へ切り替えてください	C_002 C_004

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ] [ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/VASP5/current/bin/vasp_gpu > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例)アクセラレータサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N vasp

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 10 -ppn 10 -hostfile $PBS_NODEFILE
/usr/local/app/VASP5/current/bin/vasp_gpu > vasp.out 2> vasp.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例)並列計算・インフォマティクスサーバへの投入 (non-collinear 版以外)

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N vasp

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE /usr/local/app/VASP4/current/vasp
> vasp.out 2> vasp.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入 (non-collinear 版)

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N vasp

module switch intel intel/19.1.0

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/VASP5/current/bin/vasp_ncl > vasp.out 2> vasp.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.3 QUANTUM ESPRESSO

以下のバージョンが利用可能です。

バージョン	実行モジュールのパス	実行キュー
6.1 (gpu tag v1.0)	/usr/local/app/QuantumESPRESSO/current	A_004
6.2.1	/usr/local/app/QuantumESPRESSO/current	C_002 C_004
6.4.1(gpu)	/usr/local/app/QuantumESPRESSO/qe-gpu-6.4.1	A_004
6.4.1	/usr/local/app/QuantumESPRESSO/qe-6.4.1	C_002 C_004
6.5(gpu)	/usr/local/app/QuantumESPRESSO/qe-gpu-6.5	A_004
6.6(gpu)	/usr/local/app/QuantumESPRESSO/qe-gpu-6.6 *「pgi 20.4」「CUDA 10.1.243」へ切り替えてください	A_004
6.7(gpu)	/usr/local/app/QuantumESPRESSO/qe-gpu-6.7 *「pgi 20.4」「CUDA 10.2.89」へ切り替えてください *環境変数 LD_LIBRARY_PATH に 「/opt/intel/mkl/lib/intel64」を追加してください	A_004

アクセラレータサーバへのジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

module switch intel PrgEnv-pgi/18.5

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ] [ -N ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/QuantumESPRESSO/current/bin/pw.x < 入力ファイル > 出力ファイル 2> エラ
ー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) アクセラレータサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N espresso

module switch intel PrgEnv-pgi/18.5
```

```

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 10 -N 10 -hostfile $PBS_NODEFILE
/usr/local/app/QuantumESPRESSO/current/bin/pw.x < cluster4.in > qe.out 2>
pe.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi

```

並列数は 10 までを指定可能です。

例) アクセラレータサーバへの投入(6.7(gpu))

```

#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N espresso

module switch intel PrgEnv-pgi/20.4
module switch cudatoolkit cudatoolkit/10.2.89

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/mkl/lib/intel64

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 10 -N 10 -hostfile $PBS_NODEFILE /usr/local/app/QuantumESPRESSO/
qe-gpu-6.7/bin/pw.x < cluster4.in > qe.out 2> pe.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi

```

並列数は 10 までを指定可能です。

並列計算・インフォマティクスサーバへのジョブ投入用のスクリプトは以下の通りです。

```

#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ][ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/QuantumESPRESSO/current/bin/pw.x < 入力ファイル > 出力ファイル 2> エラ
ー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi

```


例) 並列計算・インフォマティクスサーバへのジョブ投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N espresso

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/QuantumESPRESSO/current/bin/pw.x < cluster4.in > qe.out 2> qe.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.4 LAMMPS

以下のバージョンが利用可能です。

*注:アクセラレータサーバにて 7Aug19 または 3Mar2020 を利用する場合は、モジュールを「CUDA 10.1.243」へ切り替えてください。

*注:アクセラレータサーバにて 29 Oct 20 を利用する場合は、モジュールを「CUDA 10.2.89」へ切り替えてください。

バージョン	実行モジュールのパス	実行キュー
31 Mar 17	/usr/local/app/LAMMPS/current	A_004 CA_001 CA_001g C_002 C_004
12 Dec 18	/usr/local/app/LAMMPS/lammps-12Dec18	A_004 CA_001 CA_001g
5 Jun 19	/usr/local/app/LAMMPS/lammps-5Jun19	A_004 CA_001 CA_001g C_002 C_004
5 Jun 19 -DFFT_SINGLE OFF	/usr/local/app/LAMMPS/lammps-5Jun19_wo_single	A_004 CA_001 CA_001g
7 Aug 19	/usr/local/app/LAMMPS/lammps-7Aug19 *「CUDA 10.1.243」へ切り替えてください	A_004 CA_001 CA_001g
3 Mar 20	/usr/local/app/LAMMPS/lammps-3Mar20 *「CUDA 10.1.243」へ切り替えてください	A_004 CA_001 CA_001g
29 Oct 20	/usr/local/app/LAMMPS/lammps-29Oct20 *「CUDA 10.2.89」へ切り替えてください	A_004 CA_001 CA_001g

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME
```

```
mpirun [ -np 並列数 ] [ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/LAMMPS/current/src/lmp_gpu -sf gpu -pk gpu ノードあたりの利用 GPU 数 <
入力ファイル > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) アクセラレータサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N lammps

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 30 -ppn 30 -hostfile $PBS_NODEFILE
/usr/local/app/LAMMPS/current/src/lmp_gpu -sf gpu -pk gpu 10 < in.ij > lammps.out
2> lammps.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) アクセラレータサーバへの投入(7 Aug 19)

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N lammps

module switch cudatoolkit/9.0.176 cudatoolkit/10.1.243

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 30 -ppn 30 -hostfile $PBS_NODEFILE
/usr/local/app/LAMMPS/lammps-7Aug19/src/lmp_gpu -sf gpu -pk gpu 10 < in.ij >
lammps.out 2> lammps.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N lammps

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
```

```
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/LAMMPS/current/src/lmp_intel_cpu_intelmpi < in.ij > lammeps.out
2> lammeps.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.5 Gaussian16

以下のバージョンが利用可能です。

バージョン	環境設定方法	実行キュー
B.01	source /usr/local/app/Gaussian/g16.profile	C_002 C_004
C.01	source /usr/local/app/Gaussian/C.01/g16.profile	A_004 C_002 C_004

/work に作成したディレクトリに Gaussian 16 の入力ファイル(***.com)を準備します。
ヘキサカルボニルクロニウムの構造最適化を行なう入力ファイルが以下にありますので、ご覧ください。この入力ファイルでは Hartree-Fock 法を用い、3-21G 基底で構造最適化計算を行います。
(例)

```
$ ls -l /work/app/Gaussian/example.com  
-rw-r--r-- 1 root root 420 Jul 12 16:33 /work/app/Gaussian/example.com
```

・並列数の指定

GPU を利用するためには、コア数、GPU 数を指定する必要があります。
以下は 36 コア、10GPU を使用する例です。

入力ファイルでの指定例

```
%CPU=0-35
```

```
%GPUCPU=0-9=0-4,18-22
```

環境変数での指定例

```
export GAUSS_CDEF="0-35"
```

```
export GAUSS_GDEF="0-9=0-4,18-22"
```

全ての計算に対して有効なわけではございませんので、途中結果の確認や経過時間上限の設定などをご検討ください。

以下のページもご覧ください。

<http://gaussian.com/gpu/>

・一時ファイル出力先の設定

/work/scratch 以下にご自身のアカウント名でディレクトリを作成してください。
入力ファイルには下記のように記述し、一時ファイルの出力先を指定します。

(例) %Chk=example_app.chk

ジョブ投入用のスクリプトを作成します。

```
#!/bin/sh
#PBS -l select=1
#PBS -q キュー名
#PBS -N ジョブ名

source /usr/local/app/Gaussian/g16.profile
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

g16 入力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) アクセラレータサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N g16

source /usr/local/app/Gaussian/g16.profile
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

g16 test0000.com 2> g16.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N g16

source /usr/local/app/Gaussian/g16.profile
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

g16 test0000.com 2> g16.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.6 CRYSTAL

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

source /usr/local/app/Crystal/current/utils17/cry17.bashrc
runmpi17 並列数 入力ファイル > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N crystal

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

source /usr/local/app/Crystal/current/utils17/cry17.bashrc
runmpi17 36 test11 > crystal.out 2> crystal.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.7 WIEN2k

WIEN2kはライセンスをお持ちでない方は利用できません。利用希望者は[こちら](#)までその旨お問合せ下さい。ライセンスを当センターにて確認させて頂いた後利用可能となります。

以下のバージョンが利用可能です。

バージョン	実行モジュールのパス	実行キュー
17.1	/usr/local/app/WIEN2k/current	C_002 C_004
19.1	/usr/local/app/WIEN2k/WIEN2k_19.1	C_002 C_004
19.2	/usr/local/app/WIEN2k/WIEN2k_19.2 *「intel 19.0.1」へ切り替えてください	C_002 C_004

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME
export SCRATCH=$WORKDIR/$DIRNAME
export TMPDIR=$WORKDIR/$DIRNAME
export WIENROOT=/usr/local/app/WIEN2k/current
export PATH=$WIENROOT:$PATH

wien2k 実行スクリプト オプションパラメータ > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N wien2k

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME
export SCRATCH=${PBS_O_WORKDIR}
export TMPDIR=$WORKDIR/$DIRNAME
export WIENROOT=/usr/local/app/WIEN2k/current
export PATH=$WIENROOT:$PATH
```



```
run_lapw -p -cc 0.0001 -NI > wien2k.out 2> wien2k.err  
cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

並列実行させるには、実行スクリプトの中で-p オプションを指定し、.machines ファイルを実行ディレクトリに用意します。

```
$ cat .machines  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
1:localhost  
granularity:1  
extrafine:1
```

6.8 SIESTA

以下のバージョンが利用可能です。

バージョン	実行モジュールのパス	実行キュー
4.0	/usr/local/app/SIESTA/current	C_002 C_004
4.1.5	/usr/local/app/SIESTA/siesta-4.1.5	C_002 C_004

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ][ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/SIESTA/current/Obj/siesta < 入力ファイル > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N siesta

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/SIESTA/current/Obj/siesta < input.fdf > siesta.out 2> siesta.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.9 ABINIT

以下のバージョンが利用可能です。

バージョン	実行モジュールのパス	実行キュー
8.8.2	/usr/local/app/ABINIT/current/src/98_main/abinit	C_002 C_004
8.10.3	/usr/local/app/ABINIT/abinit-8.10.3/src/98_main/abinit	C_002 C_004

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ][ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/ABINIT/current/src/98_main/abinit < 入力ファイル > 出力ファイル 2> エ
ラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N abinit

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/ABINIT/current/src/98_main/abinit < input.files > abinit.out 2>
abinit.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.10 CPMD

CPMD を使用するためには、利用者自身が CPMD のライセンスを取得する必要があります。
CPMD の利用を希望される場合は、[CPMD のライセンスを取得し、計算材料学センター](#)までご連絡ください。

以下のバージョンが利用可能です。

バージョン	実行モジュールのパス	実行キュー
4.1	/usr/local/app/CPMD/current	C_002 C_004
4.3	/usr/local/app/CPMD/CPMD4.3	C_002 C_004

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ][ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/CPMD/current/bin/cpmd.x 入力ファイル > 出力ファイル 2> エラー出力ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q C_002
#PBS -N cpmd

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/CPMD/current/bin/cpmd.x inp-1 > cpmd.out > cpmd.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

6.11 MaterialsStudio

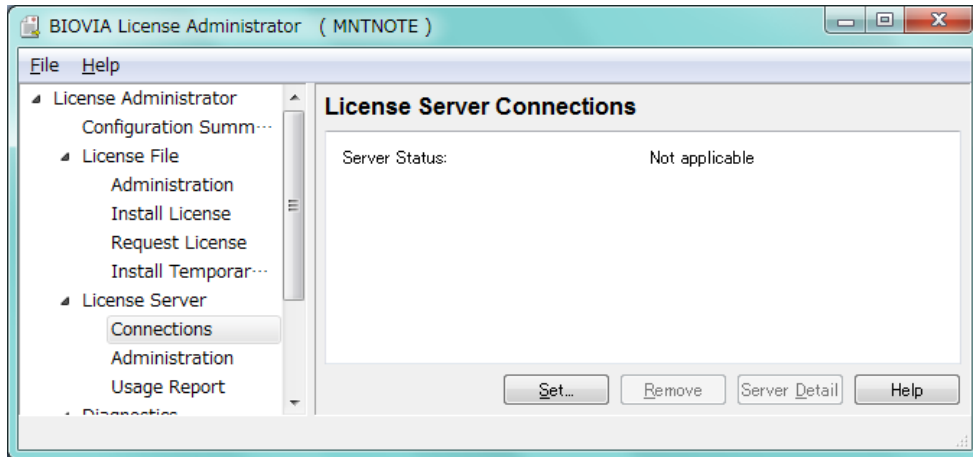
各自の PC にインストールして使用することができます。使用したい方は[こちら](#)までご連絡ください。
以下のモジュールが利用可能です。

モジュール名	ライセンス数	主な機能
Visualizer	8	構造モデルの作成とシミュレーションへの入力、計算結果、グラフ、表などの表示・作成
CASTEP_Interface	2	CASTEP 実行のための入力ファイルの作成、結果の解析
CASTEP	16	セラミックス、半導体、および金属を含む物質科学分野における固体、界面、および表面における広範囲な物性をシミュレート
DMol3_Interface	2	DMol3 実行のための入力ファイルの作成、結果の解析
DMol3-Solid_State	16	量子力学に基づいた高精度計算を高速で実行可能な事により信頼性の高い物性を高速に予測
Forcite Plus	3	構造と分子の性質の関係、分子間相互作用の理解、および固体、液体、気体の性質を予測
DFTB+	1	物質中の電子特性を研究するための量子シミュレーションソフトウェア
Sorption	1	ローディング曲線、Henry 定数などの基本物性を予測

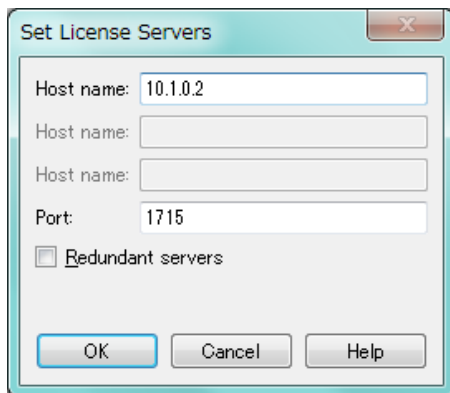
6.11.1 ライセンスサーバ設定方法

スタートメニューのプログラムから「BIOVIA」-「Licensing」を選択し、「License Administrator」を選択して起動します。

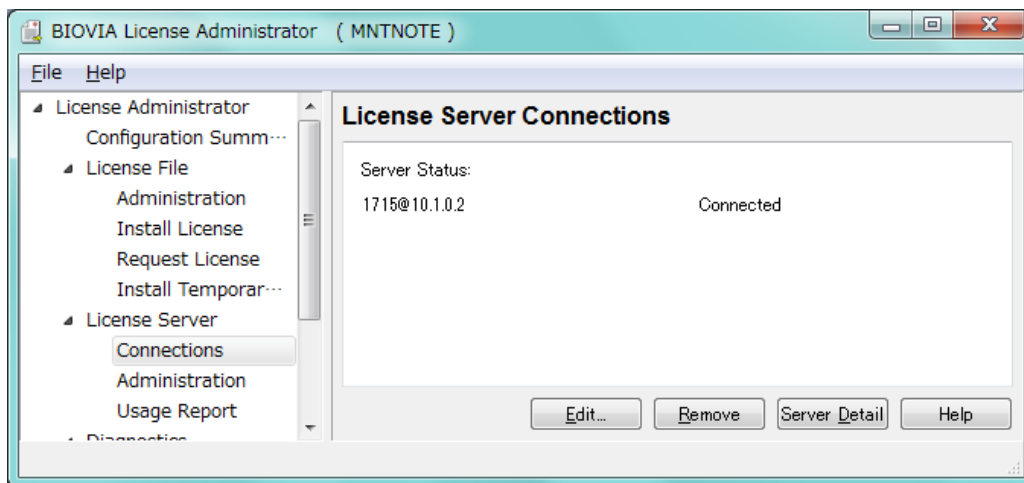
「License Server」-「Connections」を選択し、「Set」または「Edit」を押下します。



Host name に 10.1.0.2 、Port に 1715 を入力し、「OK」を押下します。

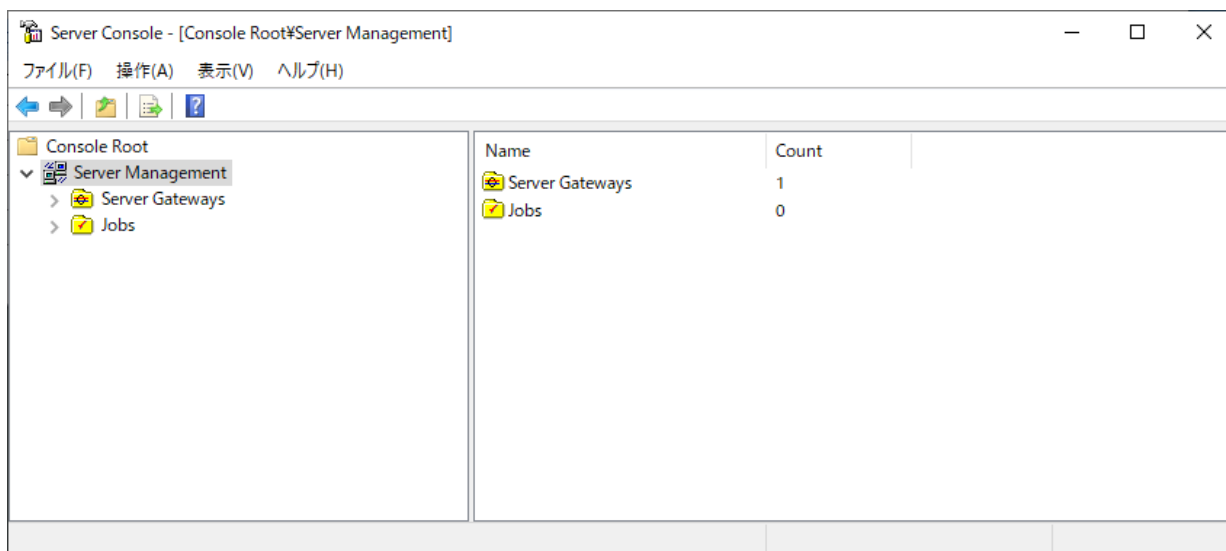


Server Status が Connected と表示されることを確認します。

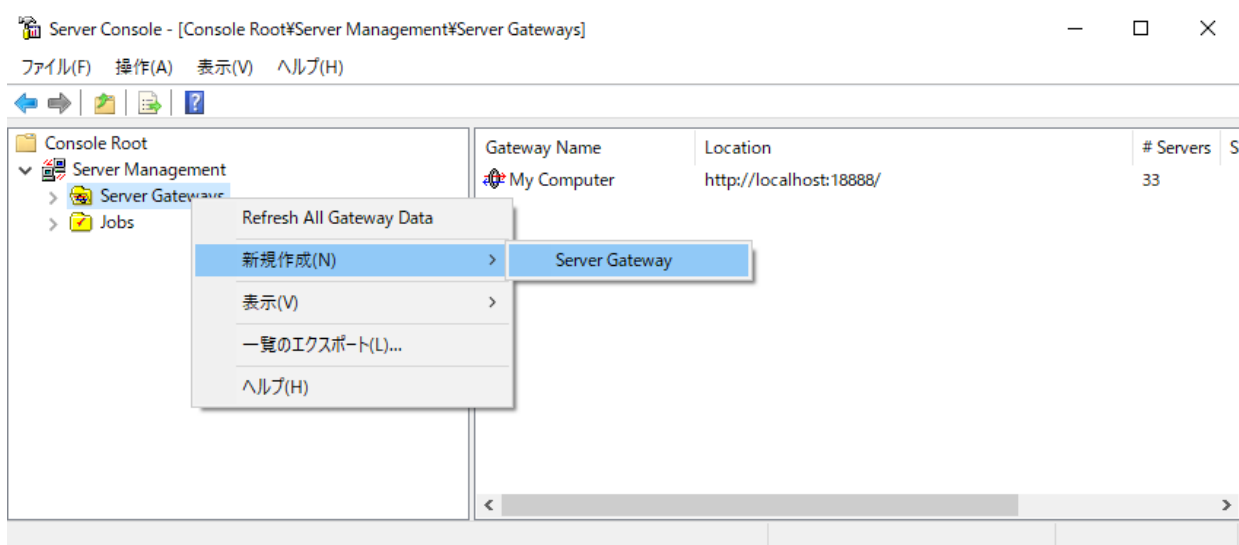


6.11.2 Gateway 設定方法

スタートメニューのプログラムから「BIOVIA」を選択し、「Server Console」を選択して起動します。

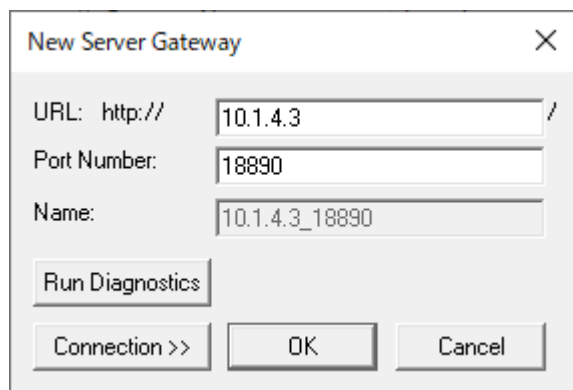


「Server Gateways」を右クリックし、新規作成の「Server Gateway」を選択します。



URL に 10.1.4.3、Port Number に 以下を入力します。

バージョン	Port Number
2019	18889
2020	18890



New Server Gateway

URL: http:// 10.1.4.3 /

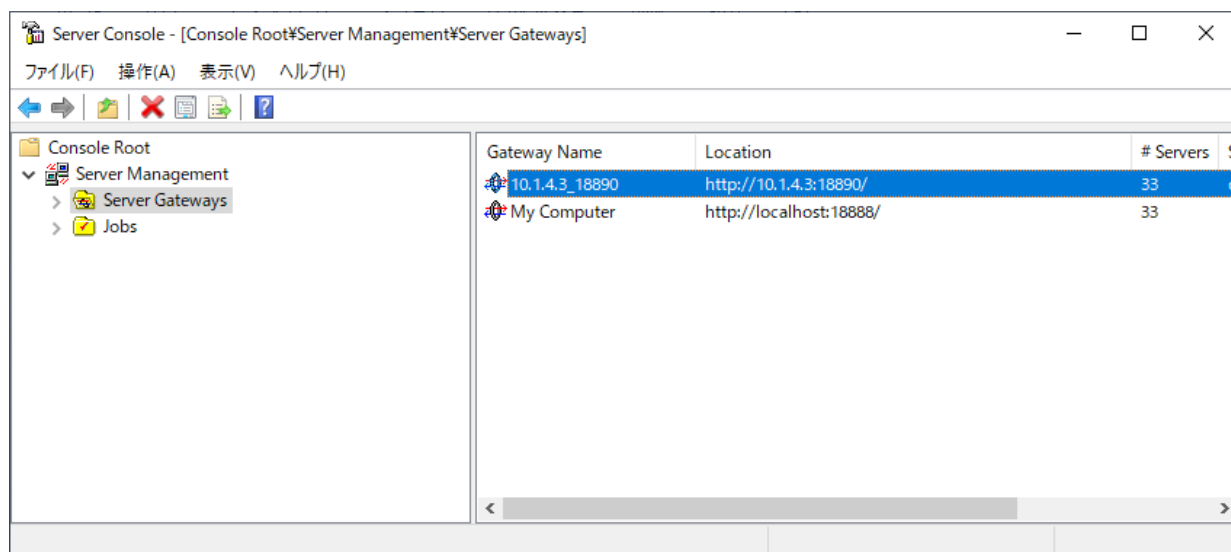
Port Number: 18890

Name: 10.1.4.3_18890

Run Diagnostics

Connection >> OK Cancel

Server Gateway に追加されたことを確認します。



Server Console - [Console Root*Server Management*Server Gateways]

ファイル(F) 操作(A) 表示(V) ヘルプ(H)

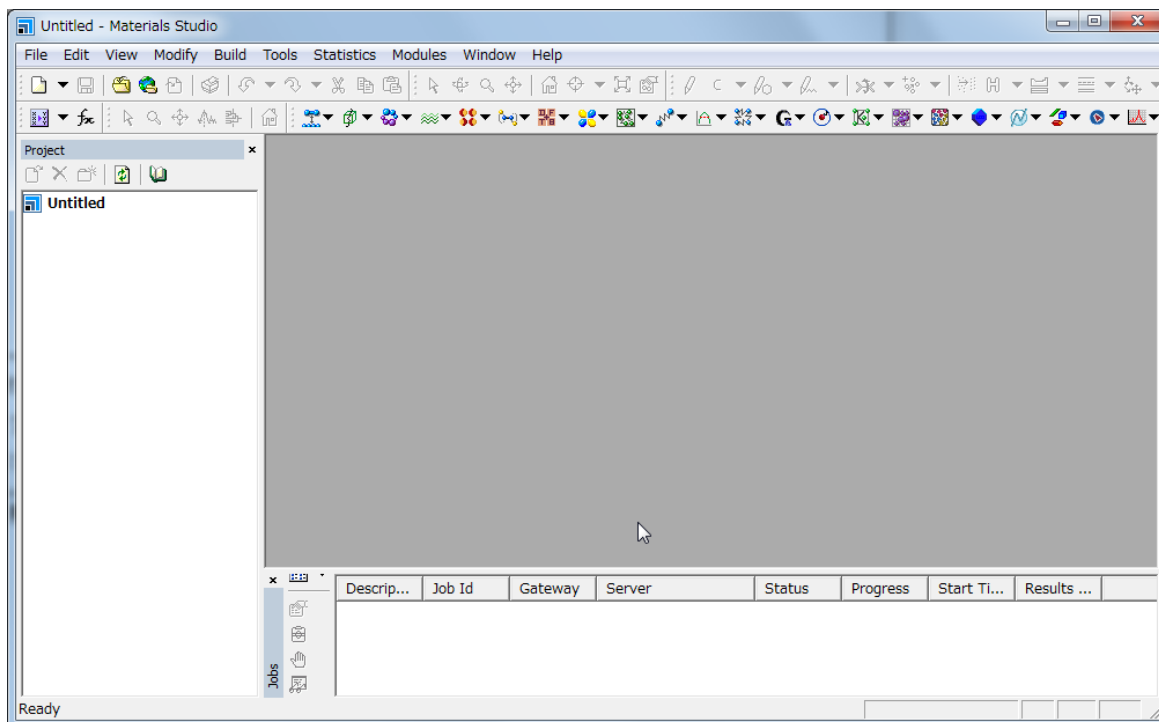
Console Root

- Server Management
 - Server Gateways
 - Jobs

Gateway Name	Location	# Servers	S
10.1.4.3_18890	http://10.1.4.3:18890/	33	
My Computer	http://localhost:18888/	33	

6.11.3 実行方法

スタートメニューのプログラムから「BIOVIA」を選択し、「Materials Studio」を選択して起動します。

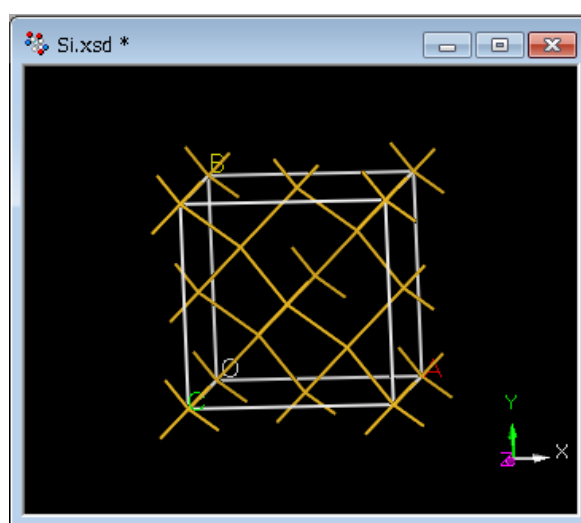


以下では CASTEP と DMol3 を実行する例を記述します。

6.11.4 CASTEP の実行方法

Si の例を示します。

- ① モデルの構築



② パラメータの設定

メニューバーの「Modules」から「CASTEP」、「Calculation」を選択し、パラメータを設定します。

③ ジョブの投入

【Materials Studio からジョブを投入する場合】

Job Control タブで実行先を指定します。

Gateway location	実行するマシン
My Computer	Materials Studio をインストールした自分の PC
10.1.4.3_18889	並列計算・インフォマティクスサーバ(Materials Studio 2019)
10.1.4.3_18890	並列計算・インフォマティクスサーバ(Materials Studio 2020)

【コマンドラインからジョブを投入する場合】

「CASTEP Calculation」ダイアログにおいて、「Files」、「Save Files」を選択し、入力ファイルを作成します。コマンドラインからのジョブの投入は/work 領域から行いますので、/work 領域に実行ディレクトリを作成し、作成された入力ファイルをスパコンシステムの実行ディレクトリへ転送します。必要なファイルは以下です。

- ・ *.params
- ・ *.cell

*.cell は隠しファイルです。エクスプローラでこの設定を解除してください。

これらのファイルはすべてテキストモードで転送してください。また、ファイル名に空白やカッコを使用しないでください。

次に、以下のコマンドでスクリプトを実行ディレクトリにコピーします。

```
$ cp /work/app/MaterialsStudio_cs/current/etc/CASTEP/bin/RunCASTEP.sh ./
```

実行ディレクトリにジョブ投入スクリプトを作成します。

```
#!/bin/sh
#PBS -l select=1
#PBS -l castep=並列数 / 18 小数点以下切り上げ の整数
#PBS -q C_002
#PBS -N castep

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

./RunCASTEP.sh -np 並列数 $i

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -l castep=2      ※ 34/18 =1.888... → 2
#PBS -q C_002
#PBS -N castep

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

./RunCASTEP.sh -np 34 Si

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

オプション-l castep の指定をしない場合、ジョブが正常に実行できません。

RunCASTEP.sh の引数で並列数の後に指定するのは、入力ファイルの拡張子の前の部分です。
Si.param/Si.cell の場合は、Si となります。

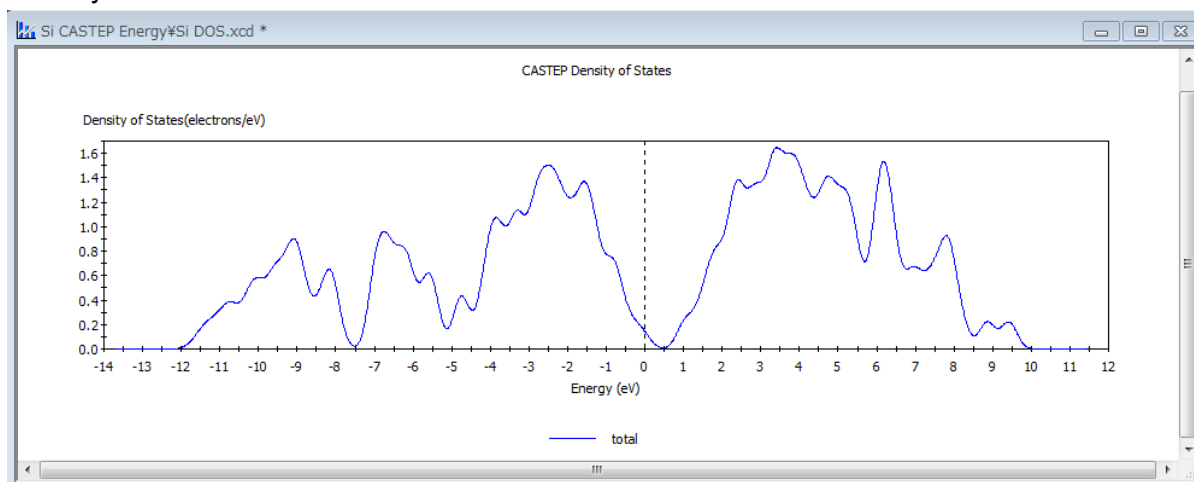
④ 結果の解析

コマンドラインから投入した場合は出力ファイルを PC に転送しておきます。その際、適切なファイルモードで転送する必要があります。

file コマンドを実行した結果が text のものはテキストモード、data のものはバイナリモードで転送します。

メニューバーの「Modules」から「CASTEP」、「Analysis」を選択します。

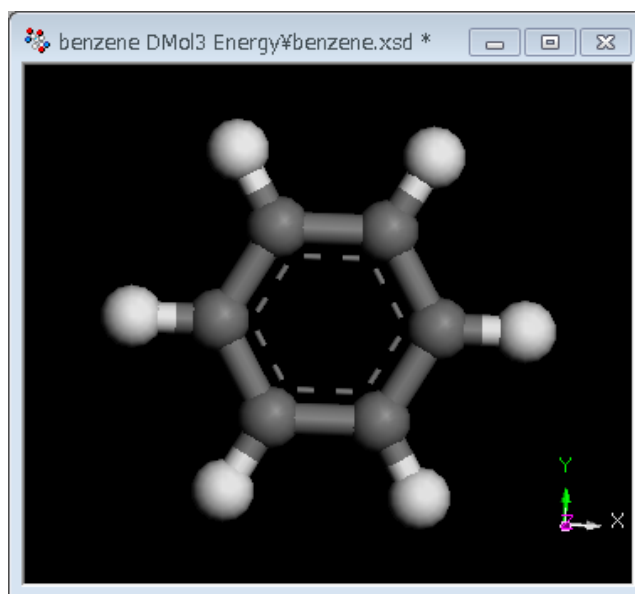
Density of States



6.11.5 Dmol3 の実行方法

ベンゼンの例を示します。

① モデルの構築



② パラメータの設定

メニューバーの「Modules」から「DMol3」、「Calculation」を選択し、パラメータを設定します。

③ ジョブの投入

【Materials Studio からジョブを投入する場合】

Job Control タブで実行先を指定します。

Gateway location	実行するマシン
My Computer	Materials Studio をインストールした自分の PC
10.1.4.3_18889	並列計算・インフォマティクスサーバ(Materials Studio 2019)
10.1.4.3_18890	並列計算・インフォマティクスサーバ(Materials Studio 2020)

【コマンドラインからジョブを投入する場合】

「DMol3 Calculation」ダイアログにおいて、「Files」、「Save Files」を選択し、入力ファイルを作成します。コマンドラインからのジョブの投入は/work 領域から行いますので、/work 領域に実行ディレクトリを作成し、作成された入力ファイルをスパコンシステムの実行ディレクトリへ転送します。必要なファイルは以下です。

- ・ *.input
- ・ *.car

*.car は隠しファイルです。エクスプローラでこの設定を解除してください。

これらのファイルはすべてテキスト形式で転送してください。また、ファイル名に空白やカッコを使用し

ないでください。

次に、以下のコマンドでスクリプトを実行ディレクトリにコピーします。

```
$ cp /work/app/MaterialsStudio_cs/current/etc/DMol3/bin/RunDMol3.sh .
```

実行ディレクトリにジョブ投入スクリプトを作成します。

```
#!/bin/sh
#PBS -l select=1
#PBS -l dmol3=並列数 / 18 小数点以下切り上げ の整数
#PBS -q C_002
#PBS -N dmol3

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

./RunDMol3.sh -np 並列数 benzene

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) 並列計算・インフォマティクスサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -l dmol3=2 ※ 32/18 =1.777... →2
#PBS -q C_002
#PBS -N dmol3

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

./RunDMol3.sh -np 32 benzene

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

オプション-l dmol3 の指定をしない場合、ジョブが正常に実行できません。

RunDMol3.sh の引数で並列数の後に指定するのは、入力ファイルの拡張子の前の部分です。
benzene.input/benzene.car の場合は、benzene となります。

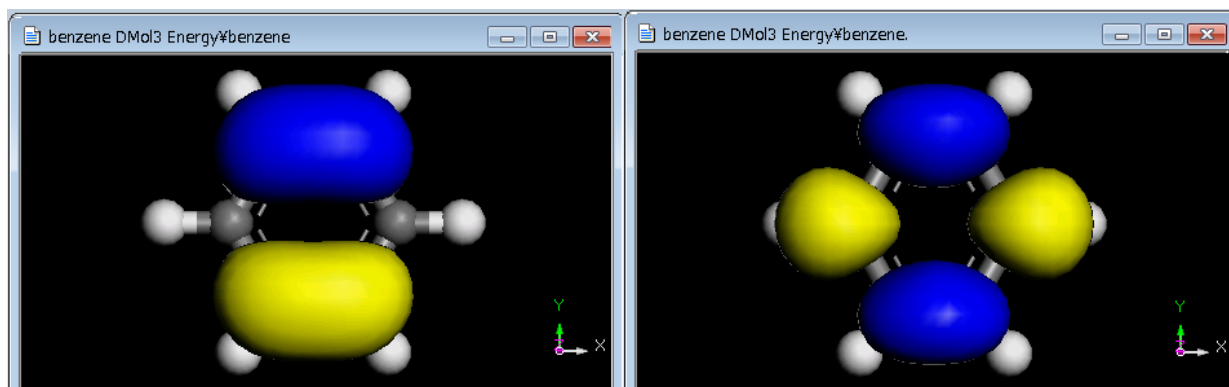
④ 結果の解析

コマンドラインから投入した場合は出力ファイルを PC に転送しておきます。その際、適切なファイルモードで転送する必要があります。

file コマンドを実行した結果が text のものはテキストモード、data のものはバイナリモードで転送します。

メニューバーの「Modules」から「DMol3」、「Analysis」を選択します。

Orbitals (HOMO: 右、LUMO: 左)



6.11.6 ジョブの実行確認

【Materials Studio からジョブを投入した場合】

ジョブの状態は、Materials Studio の画面右下の「Jobs」ペインで確認できます。

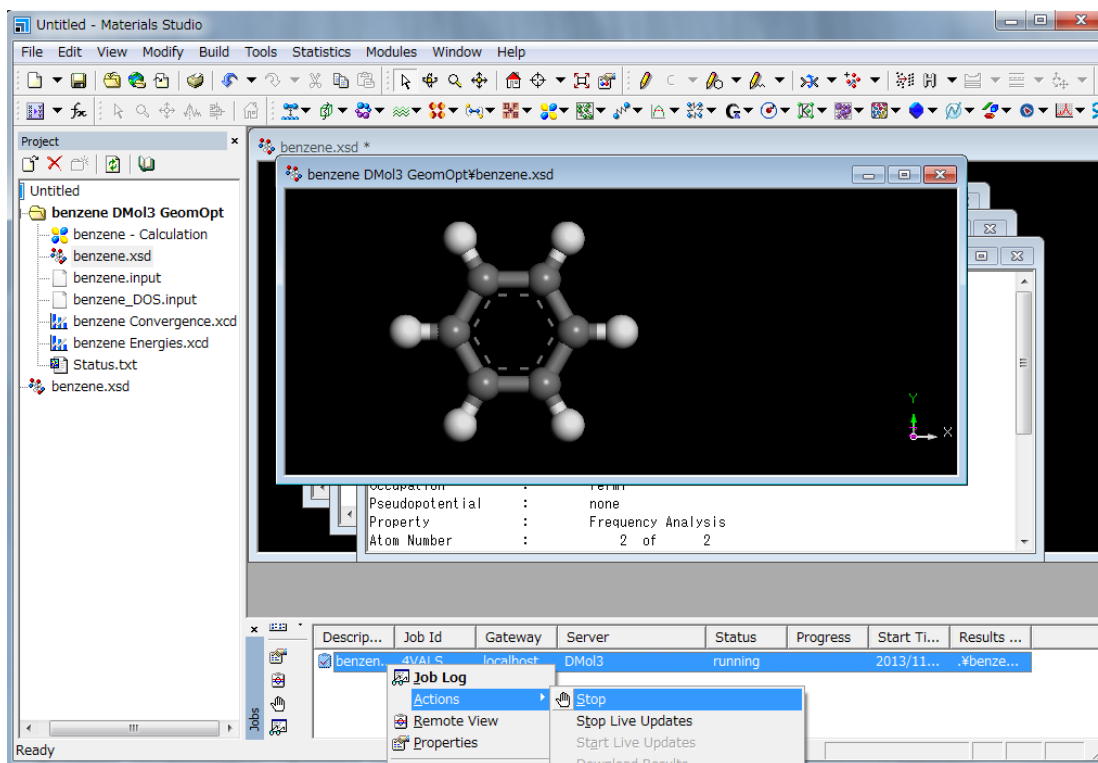
【コマンドラインからジョブを投入した場合】

ジョブの状況を含む情報表示コマンドはスーパーコンピュータと同じです。

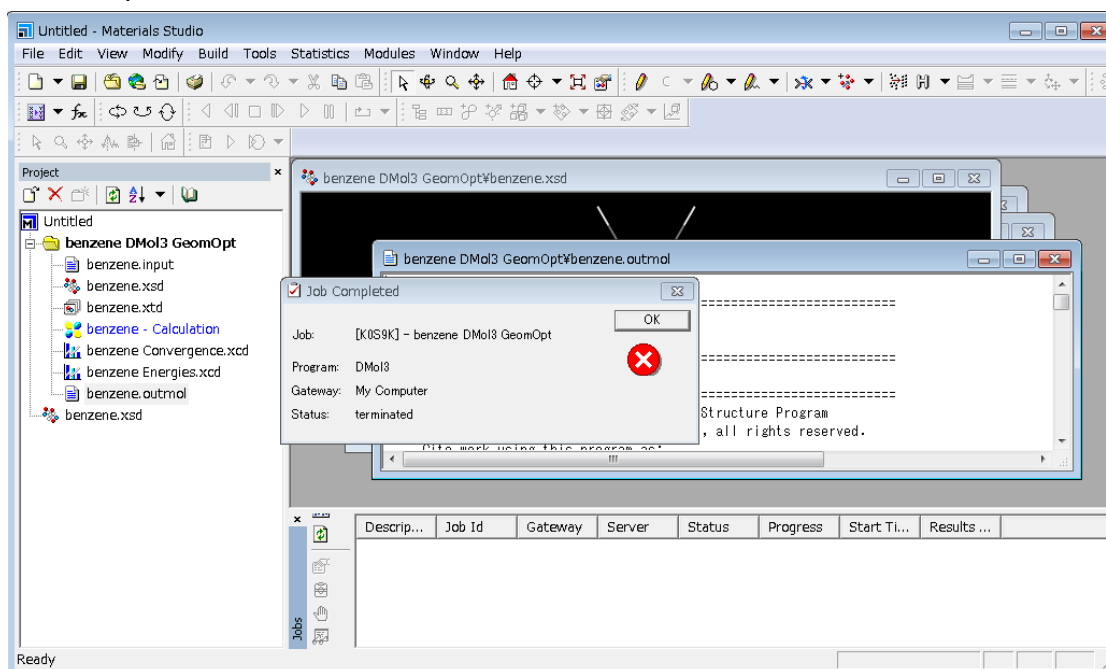
6.11.7 ジョブのキャンセル

【Materials Studio からジョブを投入した場合】

Materials Studio 画面右下の「Jobs」ペインで対象のジョブを右クリックし、プルダウンメニューの「Actions」から「Stop」を選択します。



しばらくすると、「JobStatus」が terminated となり、以下のポップアップが起動し、ジョブは停止します。
Job Completed のポップアップが最終的に起動します



6.12 Wannier90

以下のバージョンが利用可能です。

バージョン	実行モジュールのパス	実行キュー
1.2	/usr/local/app/Wannier90/wannier90-1.2	A_004
2.1.0	/usr/local/app/Wannier90/wannier90-2.1.0	A_004
3.1.0	/usr/local/app/Wannier90/current	A_004

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=ノード数
#PBS -q キュー名
#PBS -N ジョブ名

module switch intel intel/19.1.0

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun [ -np 並列数 ][ -ppn ノードあたりの並列数 ] -hostfile $PBS_NODEFILE
/usr/local/app/Wannier90/current/wannier90.x 入力ファイル > 出力ファイル 2> エラー出力
ファイル

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

例) アクセラレータサーバへの投入

```
#!/bin/sh
#PBS -l select=1
#PBS -q A_004
#PBS -N wannier90

module switch intel intel/19.1.0

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

mpirun -np 36 -ppn 36 -hostfile $PBS_NODEFILE
/usr/local/app/Wannier90/current/wannier90.x wannier90 > wannier.out 2>
wannier.err

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm -rf $WORKDIR; fi
```

実行モジュールの後に指定するのは、入力ファイルの拡張子の前の部分です。

7

7 機械学習環境 使用方法

[7.1 機械学習環境一覽](#)

[7.2 Chainer](#)

[7.3 Keras](#)

[7.4 Caffe](#)

[7.5 Jupyter Notebook](#)

[7.6 DIGITS](#)

7.1 機械学習環境一覧

アクセラレータサーバでは以下の機械学習環境が利用可能です。

機械学習環境にはキューCA_001gを利用します。

/work_da 領域よりジョブを投入してください。

#	アプリケーション	イメージ:タグ名
1	Chainer	conda3/mlenv:cuda10.1-007 (CUDA10.1 環境) conda3/mlenv:cuda9.1-006 (CUDA9.1 環境)
2	Keras	
3	Caffe	
4	Jupyter Notebook	
5	DIGITS	nvidia/digits:6.0

7.2 Chainer

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=1[:ncpus=CPU 数][:ngpus=GPU 数]
#PBS -q CA_001g
#PBS -N ジョブ名
#PBS -v DOCKER_IMAGE= conda3/mlenv:タグ名

cd $PBS_O_WORKDIR

python 入力ファイル > 出力ファイル 2> エラー出力ファイル
```

例)

```
#!/bin/sh
#PBS -l select=1
#PBS -q CA_001g
#PBS -N Chainer
#PBS -v DOCKER_IMAGE=conda3/mlenv:cuda10.1-007

cd $PBS_O_WORKDIR

python train_cifar.py > train_cifar.out 2> train_cifar.err
```

インタラクティブモード実行のコマンドは以下の通りです。

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mlenv:タグ名
```

例)

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mlenv:cuda10.1-007
qsub: waiting for job 26269.gpu1 to start
qsub: job 26269.gpu1 ready

bash-4.2$ cd $PBS_O_WORKDIR
bash-4.2$ python train_cifar.py > train_cifar.out 2> train_cifar.err
```

7.3 Keras

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=1[:ncpus=CPU 数][:ngpus=GPU 数]
#PBS -q CA_001g
#PBS -N ジョブ名
#PBS -v DOCKER_IMAGE= conda3/mлenv:タグ名

cd $PBS_O_WORKDIR

python 入力ファイル > 出力ファイル 2> エラー出力ファイル
```

例)

```
#!/bin/sh
#PBS -l select=1
#PBS -q CA_001g
#PBS -N Keras
#PBS -v DOCKER_IMAGE=conda3/mлenv:cuda10.1-007

cd $PBS_O_WORKDIR

python cifar100_resnet_multigpu.py > cifar100.out 2> cifar100.err
```

インタラクティブモード実行のコマンドは以下の通りです。

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mлenv:タグ名
```

例)

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mлenv:cuda10.1-007
qsub: waiting for job 26269.gpu1 to start
qsub: job 26269.gpu1 ready

bash-4.2$ cd $PBS_O_WORKDIR
bash-4.2$ python cifar100_resnet_multigpu.py > cifar100.out 2> cifar100.err
```

7.4 Caffe

Caffe のコマンドとして以下が利用可能です。

詳細は--help オプションでご確認ください。

caffe	ta	train
classification	detect	train_net
classify	device_query	upgrade_net_proto_binary
compute_image_mean	draw_net	upgrade_net_proto_text
convert_cifar_data	extract_features	upgrade_solver_proto_text
convert_imageset	finetune_net	
convert_mnist_data	net_speed_benchmark	
convert_mnist_siamese_data	test_net	

ジョブ投入用のスクリプトは以下の通りです。

```
#!/bin/sh
#PBS -l select=1[:ncpus=CPU 数][:ngpus=GPU 数]
#PBS -q CA_001g
#PBS -N ジョブ名
#PBS -v DOCKER_IMAGE=conda3/mlenv:タグ名

cd $PBS_O_WORKDIR

caffe コマンド オプション
```

例)

```
#!/bin/sh
#PBS -l select=1
#PBS -q CA_001g
#PBS -N Caffe
#PBS -v DOCKER_IMAGE=conda3/mlenv:cuda10.1-007

cd $PBS_O_WORKDIR

caffe train --solver=examples/cifar10/cifar10_quick_solver.prototxt
```

インタラクティブモード実行のコマンドは以下の通りです。

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mlenv:タグ名
```

例)

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mlenv:cuda10.1-007
qsub: waiting for job 26269.gpu1 to start
qsub: job 26269.gpu1 ready

bash-4.2$ cd $PBS_O_WORKDIR
bash-4.2$ caffe train --solver=examples/cifar10/cifar10_quick_solver.prototxt
```

7.5 Jupyter Notebook

Jupyter Notebook はインタラクティブモードで利用します。

コマンドは以下の通りです。

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mlenv:タグ名
$ jupyter notebook --ip=* --no-browser
```

例)

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=conda3/mlenv:cuda10.1-007
qsub: waiting for job 26322.gpu1 to start
qsub: job 26322.gpu1 ready
```

Access Port:

```
<proto>://10.1.4.28:6037/ -> container port 8888
```

```
bash-4.2$ jupyter notebook --ip=* --no-browser
```

(略)

To access the notebook, open this file in a browser:

```
file:///home/hitacse/.local/share/jupyter/runtime/nbserver-12-open.html
```

Or copy and paste one of these URLs:

```
http://cgpu28-26322-cgpu28:8888/?token=3a4f5d232de317ef49f51630ba4038e47bdd51f7d191ee2c
```

or

```
http://127.0.0.1:8888/?token=3a4f5d232de317ef49f51630ba4038e47bdd51f7d191ee2c
```

ログイン後、画面に出力される"container port 8888"に対応するポートおよびログインを控えておきます。上記の例の場合は 6037 です。(以下ポート A とします)

またログイン token を控えておきます。上記の例の場合は

3a4f5d232de317ef49f51630ba4038e47bdd51f7d191ee2c です。

ご自身の PC にて新たにターミナルを起動し、任意のポート(以下ポート B とします)を gpu2 のポート 22 にポートフォワードします。

Unix 系の場合は以下のように指定します。

```
$ ssh -L ポートB:gpu2:22 ユーザー名@cms-ssh.sc.imr.tohoku.ac.jp
```

例)

```
$ ssh -L 8022:gpu2:22 userA@cms-ssh.sc.imr.tohoku.ac.jp
```

再度ターミナルを起動し、任意のポート(以下ポート C とします)を Jupyter Notebook の IP アドレス: ポートにフォワードします。

```
$ ssh -L ポートC:10.1.4.28:ポートA -p ポートB localhost
```

例)

```
$ ssh -L 15000:10.1.4.28:6037 -p 8022 localhost
```

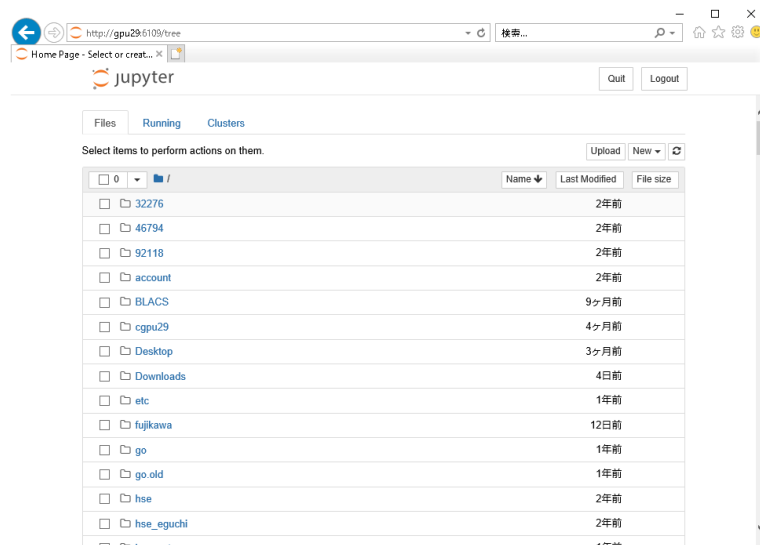
ご自身の PC にてブラウザを起動し、以下 URL にアクセスします。

```
http://localhost:ポートC/?token=ログイン token
```

例)

`http://localhost:15000/?token=3a4f5d232de317ef49f51630ba4038e47bdd51f7d191e2c`

Jupyter Notebook の画面が表示されます。



7.6 DIGITS

DIGITS はインタラクティブモードで利用します。

コマンドは以下の通りです。

```
$ qsub -I -q CA_001g -v DOCKER_IMAGE=nvidia/digits:6.0
qsub: waiting for job 26322.gpu1 to start
qsub: job 26322.gpu1 ready

Access Port:
    <proto>://10.1.4.28:6037/ -> container port 5000
    <proto>://10.1.4.28:6161/ -> container port 6006
bash-4.2$
```

ログイン後、画面に出力される"container port 5000"に対応するポートを控えておきます。上記の例の場合は 6037 です。(以下ポート A とします)

ご自身の PC にて新たにターミナルを起動し、任意のポート(以下ポート B とします)を gpu2 のポート 22 にポートフォワードします。

Unix 系の場合は以下のように指定します。

```
$ ssh -L ポート B:gpu2:22 ユーザー名@cms-ssh.sc.imr.tohoku.ac.jp
```

例)

```
$ ssh -L 8022:gpu2:22 userA@cms-ssh.sc.imr.tohoku.ac.jp
```

再度ターミナルを起動し、任意のポート(以下ポート C とします)を DIGITS の IP アドレス:ポートにフォワードします。

```
$ ssh -L ポート C:10.1.4.28:ポート A -p ポート B localhost
```

例)

```
$ ssh -L 15000:10.1.4.28:6037 -p 8022 localhost
```

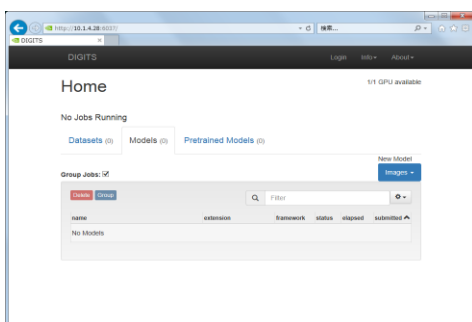
ご自身の PC にてブラウザを起動し、以下 URL にアクセスします。

```
http://localhost:ポート C
```

例)

```
http://localhost:15000
```

DIGITS の画面が表示されます。



8

8 Python 使用方法

[8.1 Python の利用について](#)

[8.2 pyenv 環境の構築](#)

[8.3 環境変数の設定](#)

[8.4 動作確認](#)

[8.5 基本的な使い方](#)

[8.6 実行方法](#)

8.1 Python の利用について

本システムでは、下記のスクリプトを実行することでユーザーの環境に pyenv をインストールすることができます。pyenv では python のバージョン管理が可能です。詳細は下記をご確認ください。

8.2 pyenv 環境の構築

下記のコマンドを実行すると super および gpu 上に pyenv がインストールされます。

```
$ bash /work/app/pyenv/pyenv-setup.bash
```

8.3 環境変数の設定

インストールスクリプト実行後にカレントディレクトリに bash_env というファイルが生成されます。pyenv をデフォルトで読み込む場合は下記のコマンドで ~/.bash_profile に内容をコピーします。

```
$ cat bash_env >> ~/.bash_profile
```

8.4 動作確認

インストール後にフロントエンドノードから一度ログアウトし、再度ログイン後、下記のコマンドを実行します。

```
$ pyenv --version  
pyenv 1.2.8-5-gec9fb549 ←バージョンは異なる場合があります。
```

8.5 基本的な使い方

・Python のインストール

```
$ pyenv install --list ←利用可能な Python のバージョンを表示  
....  
$ pyenv install 3.7.8 ←Python 3.7.8 をインストールする場合
```

・Python のバージョン切り替え

```
$ pyenv versions          ←インストールされているバージョンの確認
* system (set by /home/userA/.pyenv/version)
  3.7.8
$ pyenv global 3.7.8      ←Python 3.7.8 に切り替え
$ python --version
```

pyenv でインストールした Python では、pip を用いてパッケージの追加も可能です。
より詳細な使用方法については、pyenv のドキュメント等をご確認ください。

Simple Python version management

<https://github.com/pyenv/pyenv>

8.6 実行方法

負荷の高い Python プログラムは、フロントエンドノードではなく計算ノード上で実行してください。

・インタラクティブモードでの実行例

```
userA@gpu2:~> qsub -I -q CA_001
qsub: waiting for job 70568.gpu1 to start
qsub: job 70568.gpu1 ready

-bash-4.2$ python
Python 3.7.8 (default, Mar 25 2021, 09:54:46)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

ジョブとして投入して実行することも可能です。

・ジョブスクリプト例

```
#!/bin/sh
#PBS -l select=1
#PBS -q CA_001
#PBS -N sample
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

python program.py

cd; if cp -raf $WORKDIR/$DIRNAME $PBS_O_WORKDIR/.. ; then rm
-rf $WORKDIR; fi
```