東北大学金属材料研究所
大規模並列計算サーバマニュアル

2021年5月17日

東北大学金属材料研究所
計算材料学センター

目次

1

1 大規模並列計算サーバ概要

1.1 構成・スペック

大規模並列計算サーバのスペック

サーバ名	大規模並列計算サーバ	フロントエンドサーバ
機種名	Cray XC50-LC	Cray XC50-LC
サーバ台数	293 台 ＋ 3 台(予備機)	2 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 : 1 基/サーバ
主記憶容量	768 GiB/サーバ	768 GiB/サーバ

1.2 ノード構成

大規模並列計算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	計算を行うノード	293 ノード	計算材料学センター 101 室
計算ノード(予備機)	計算ノードに障害が発生した際の予備機	3 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

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
```

ssh リレーサーバから、大規模並列計算サーバのフロントエンドサーバ super へログインします。

```
$ ssh super
```

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

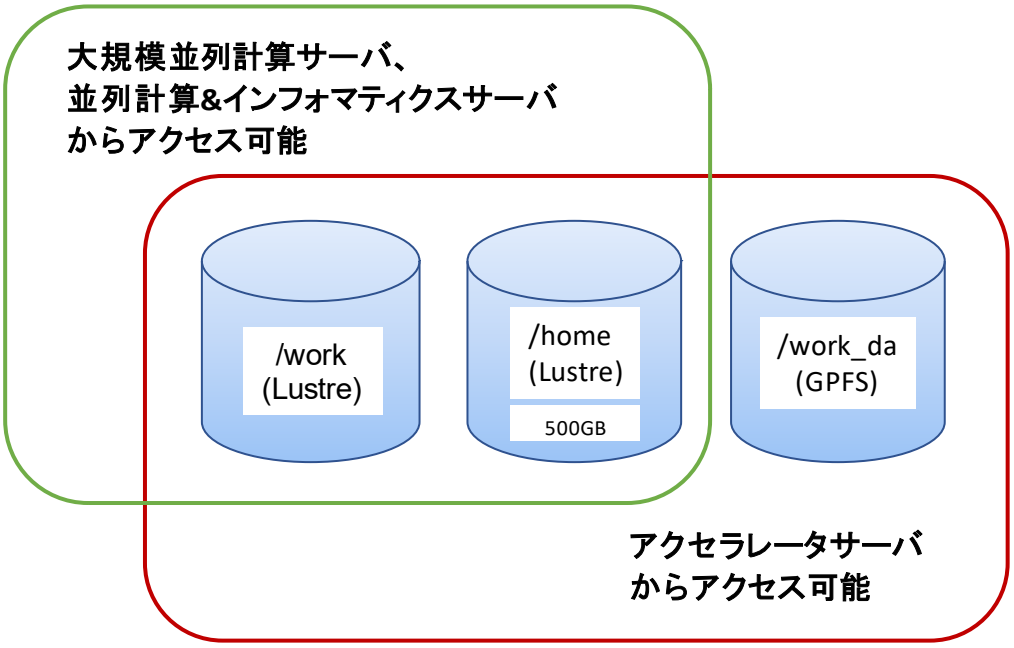
```
$ passwd
Current Password:[現在のパスワード]
New password:[新しいパスワード]
Retype new password:[新しいパスワード]
```

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

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	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。
③並列計算・インフォマティクスサーバ	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 ジョブ投入コマンド

4.1.1 ジョブの投入コマンド(qsub コマンド)

スーパーコンピュータ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] [-l ライセンス種類=使用ライセンス数] [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=使用ライセンス数	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。 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DP_002 を使用して、ノード 2 ノード使用、経過時間上限を 10 分、スクリプトファイルは hello.sh

```
$ qsub -q DP_002 -l select=2 -l walltime=00:10:00 hello.sh
```

スクリプトで指定する場合

```
#!/bin/sh
#PBS -q DP_002
#PBS -l select=2
#PBS -l walltime=00:10:00
 :
 :
 :
```

・キューP_016 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 P_016 -M userA@test.com -m be hello.sh
```

スクリプトで指定する場合

```
#!/bin/sh
#PBS -q P_016
#PBS -M userA@test.com
#PBS -m be
 :
 :
 :
```

・キューDP_002 を使用して、ライセンス管理対象の QuantumATK を実行、スクリプトファイルは atk.sh

```
$ qsub -q DP_002 -l atk=1 -l atkdp=35 atk.sh
```

スクリプトで指定する場合

```
#!/bin/sh
#PBS -q DP_002
#PBS -l atk=1 -l atkdp=35
 :
 :
 :
```

4.1.2 プログラム起動コマンド(aprun コマンド)

ジョブでプログラムを起動するには aprun コマンドを利用します。

注:aprun コマンドでプログラムを起動しないと、計算ノードで実行されません。計算ノードで実行されていない場合、他のユーザーのジョブへ影響があるため、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

```
$ aprun  [-n 並列数] [-d 並列数][-N ノードあたりの並列数] [-S CPU ソケットあたりの並列数] [-j 0|1|N] [--cc 配置方法] 実行プログラム
```

(2)オプション一覧

オプション名	説明
-n 並列数	MPI 並列数を指定する。
-d 並列数	OpenMP 並列数を指定する。 (合わせて OMP_NUM_THREADS の指定も必要)
-N ノードあたりの並列数	ノードごとに配置する MPI プロセス数を指定する。
-S CPU ソケットあたりの並列数	CPU ソケットごとに配置する MPI プロセス数を指定する。
-j 0|1|N	CPU コアごとに配置するスレッド数を指定する。 0: HyperThreading を使用する(デフォルト) 1: HyperThreading を使用しない N: HyperThreading を使用、コア毎に N スレッドを配置
--cc 配置方法	プロセス／スレッドの配置方法を指定する。 depth: プロセスが割り当てられた CPU コアに近接するようにスレッドをバインド(OpenMP, MPI+OpenMP のプログラムを実行する際に有効)

※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(-n の値)] = [ノード数(#PBS -l select=の値)] × [ノードあたりの並列数(-N の値)]

4.1.3 実行スクリプトの書式

スーパーコンピュータ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

aprun 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 P_016
#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

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

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

Cray XC の高速通信(ESM モード)を使用した MPI 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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] 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　1 ノードで 2MPI プロセスのプログラム a.out を実行

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N mpi1
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

aprun -n 2 -N 2 ./a.out > result.out  2> result.err

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

・例　2 ノードで 2MPI プロセスのプログラム a.out を実行

```
#!/bin/sh
#PBS -l select=2
#PBS -q P_016
#PBS -N mpi2
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

aprun -n 2 -N 1 ./a.out > result.out  2> result.err

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

4.1.4 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 IP_001 を指定します。

（1）書式

```
$ qsub -I -q IP_001
```

（2）使用例

```
$ qsub -I -q IP_001
qsub: waiting for job 220331.sdb to start
qsub: job 220331.sdb ready

Directory: /home/userA
Mon Sep 23 01:03:04 JST 2019

userA@mom1:~> cd $PBS_O_WORKDIR
userA@mom1: /work/userA/testdir> aprun -n 32 -j 1 ./a.out
```

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@super2:~> statj
                                                          Req'd   Req'd   Elap
Job ID     Username Queue  Jobname  SessID  NDS TSK Memory Time  S Time
--------  ---------------- ------ -------- ----- ------- --- ---- -----
3413.sdb    userA     P_016  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

スーパーコンピュータ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@super2:~> qstat -a

                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
3390.sdb        userA    P_016    abinit      193347   4 144 2760gb 72:00 R 47:28
3401.sdb        userA    P_016    prog9_1     121974   4 144 2760gb 72:00 R 47:26

userA@super2:~> qstat -p
Job id           Name             User              % done  S Queue
---------------  ---------------- ----------------  -------- - -----
3390.sdb         abinit           userA               2  R P_016
3401.sdb         prog9_1          userA               0  R P_016

userA@super2:~> qstat -t
Job id           Name             User              Time Use S Queue
---------------  ---------------- ----------------  -------- - -----
3390.sdb         abinit           userA          00:00:01 R P_016
3401.sdb         prog9_1          userA          00:00:01 R P_016

userA@super2:~ > qstat -x
Job id           Name             User              Time Use S Queue
---------------  ---------------- ----------------  -------- - -----
2235.sdb         prog9_2          userA          00:00:03 F P_016
2236.sdb         vasp4            userA          00:00:01 F P_016
2237.sdb         prog9_1          userA          00:00:01 F P_016
        以下略

userA@super2:~> qstat -f 3390.sdb
Job Id: 3390.sdb
    Job_Name = abinit
    Job_Owner = userA@nid00204
    resources_used.cpupercent = 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

スーパーコンピュータ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@super2:~> 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
DP_002       0     0  yes  yes     0     0     0     0     0     0 Exec
P_016        0     1  yes  yes     0     1     0     0     0     0 Exec
P_032        0     0  yes  yes     0     0     0     0     0     0 Exec
P_064        0     0  yes  yes     0     0     0     0     0     0 Exec
LP_032       0     0  yes  yes     0     0     0     0     0     0 Exec
LP_064       0     0  yes  yes     0     0     0     0     0     0 Exec
               以下略

userA@super2:~> 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

スーパーコンピュータ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@super2:~> qstat -B
Server            Max   Tot   Que   Run   Hld   Wat   Trn   Ext Status
---------------- ----- ----- ----- ----- ----- ----- ----- ----- ----------
sdb                0  1155     0     1     0     0     0     0 Active
userA@super2:~> 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

スーパーコンピュータ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	ジョブヒストリー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@super2:~/work/20180712_sample> statj
                                                    Req'd   Req'd   Elap
Job ID     Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
---------- -------- -------- ---------- ------ --- --- ------ ----- - -----
3413.sdb   userA    P_016    abinit     3710   3   216 2304gb 72:00 R 00:00
3414.sdb   userA    DP_002   STDIN      13588  1    72  768gb 00:10 R 00:00
userA@super2:~/work/20180712_sample> qdel 3414.sdb
userA@super2:~/work/20180712_sample> statj
                                                    Req'd   Req'd   Elap
Job ID     Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
---------- -------- -------- ---------- ------ --- --- ------ ----- - -----
3413.sdb   userA    P_016    abinit     3710   3   216 2304gb 72:00 R 00:00
userA@super2:~/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@super2:~ >  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 ジョブ実行性能に関する指定

(1) 説明

スーパーコンピュータは Hyper-Threading を有効にしているため、aprun コマンドでコアあたりのプロセスの割当を指定することにより性能改善する可能性があります。

(2) 書式

物理コアあたりのスレッド数で 1 を指定します。

```
aprun -j 1 実行プログラム
```

(3) 使用例

```
#!/bin/bash
#PBS -j oe
#PBS -l select=1
#PBS -q P_016

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

aprun -n 36 -N 36 -j 1 ./xhpl_skl_diag_cray_opt > 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

スーパーコンピュータで使用できるキューは以下の通りです。

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	並列数上限※	備考
IP_001	1(1)	768	24(1)	制限なし	72	インタラクティブ用
DP_002	2(1)	1,536	0.5(0.5)	制限なし	144	デバック用
P_016	16(1)	12,288	72(24)	制限なし	1152	1 から 16 ノード使用
P_032	32(32)	24,576	72(24)	4	2304	17 から 32 ノード使用
P_064	64(64)	49,152	72(24)	2	4608	33 から 64 ノード使用

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	並列数上限※	備考
LP_064	64(64)	49,152	168(96)	1	4608	運用中も実行
MP_096	96(96)	73,728	72(24)	1	6912	
SP_064	64(64)	49,152	336(336)	1	4608	定期保守後に実行
SP_128	128(128)	98,304	168(168)	1	9216	
SP_293	293(293)	225,024	24(24)	1	21096	

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

スーパーコンピュータ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
Cray コンパイラ Fortran/C/C++	8.7.10 8.6.5	デフォルト：8.6.5
Intel コンパイラ Fortran/C/C++	19.1.3.304 19.1.0.166 19.0.2.187 18.0.2.199 17.0.4.196	デフォルト：18.0.2.199
PGI コンパイラ Fortran/C/C++	19.1-0 18.5-0	デフォルト：19.1-0

5.1.1 プログラミング環境

（1）コマンド

Fortran, C, C++コンパイラのコマンド名は ftn, cc, CC に統一されています。
プログラミング環境を切り替えることで、コマンドが内部で呼び出すコンパイラが自動的に切り替わります。

コンパイラ	コマンド	Cray コンパイラ	Intel コンパイラ	PGI コンパイラ	gnu コンパイラ
Fortran	ftn	crayftn	ifort	pgf90	gfortran
C	cc	craycc	icc	pgcc	gcc
C++	CC	crayCC	icpc	pgc++	g++

MPI プログラムをコンパイルする場合も ftn, cc, CC コマンドを利用します。
MPI ライブラリは自動的にリンクされるので、明示的にリンクオプションを付ける必要はありません。

（2）プログラム環境の切り替え

各プログラミング環境に対応する module ファイルは以下です。

Cray コンパイラ	Intel コンパイラ	PGI コンパイラ	gnu コンパイラ
PrgEnv-cray	PrgEnv-intel	PrgEnv-pgi	PrgEnv-gnu

プログラミング環境は module switch コマンドで切り替えます。
デフォルトでは Cray コンパイラ PrgEnv-cray がロードされています。
例として、Intel コンパイラに切り替える場合には以下のように実行します。

```
$ module switch PrgEnv-cray PrgEnv-intel
```

（3）共通オプション

以下オプションが全プログラミング環境に共通で指定可能です。

オプション名	説明
-craype-verbose	コンパイラ・リンカに渡すオプションの表示
-static	リンカに静的リンクを指示
-dynamic	リンカに動的リンクを指示
-shared	実行時に動的にリンクするライブラリを作成
-help	プログラミング環境個別のオプションを表示

5.1.2 Cray コンパイラ

（1）プログラム環境の設定

super では Cray コンパイラをデフォルトで設定しています。

他の環境に切り替えていた場合は下記のように環境を切り替えて下さい。

・例　intel コンパイラの環境からの切り替え

```
$ module switch PrgEnv-intel PrgEnv-cray
```

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。

・例　バージョンの切り替え

```
$ module avail cce
cce/8.6.5(default)
cce/8.7.10

$ module switch cce cce/8.7.10
```

（2）コンパイル方法

オプション

・最適化オプション

オプション名	説明
-o outfile	出力ファイル名を指定する。省略時には a.out が設定される。
-llibrary_name	リンクするライブラリ名を指定する
-Llibrary_path	ライブラリの検索パスを指定する
-h autothread	ノード内自動並列化を適用する(デフォルト: 無効)
-h omp/noomp	OpenMP 指示行による並列化を有効／無効にする(デフォルト: 有効)
-h thread[0-3]	OpenMP 最適化レベルの指定(3:最高水準、デフォルトは 2)
-O [0-3]	自動最適化レベルの指定(3:最高水準、デフォルトは 2)
-h ipa[0-5]	関数のインライン展開最適化レベルの指定

・Fortran 専用オプション

オプション名	説明
-e0	未定義のローカルスタック変数をゼロ初期化する
-ev	変数を static に割り当てる（例外条件あり）
-ez	allocate()文で確保した領域をゼロ初期化する
-f [free|fixed]	ソースファイルの形式

・C 専用オプション

オプション名	説明
-h c99	C99 仕様に準拠してコンパイルする
-h noc99	C99 仕様に準拠しないでコンパイルする
-h zero	未定義のローカルスタック変数をゼロ初期化する

・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。
-G [0-2]	デバッグ情報の管理 (0:最多の情報がデバッグ時に得られます。 ”-G0”=“-g”)
-K trap=…	例外処理をトラップします。 [fp | divz | denorm | …]

(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

```
$ ftn -f 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

```
$ ftn -f 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

```
$ ftn -h autothread -f fixed -o hello.out hello.f
```

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

```
$ ftn -h autothread -f free -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 hello.c -o hello.out
```

・ 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

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

・ 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

```
$ CC hello.cpp -o hello.out
```

・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

```
$ CC -h autothread hello.cpp -o hello.out
```

5.1.3 Intel コンパイラ

（1）プログラム環境の設定

super では Cray コンパイラがデフォルトで設定しているため下記のように環境を切り替えて下さい。

・例　Cray コンパイラの環境からの切り替え

```
$ module switch PrgEnv-cray PrgEnv-intel
```

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。

・例　バージョンの切り替え

```
$ module avail intel
intel/17.0.4.196
intel/18.0.2.199(default)
intel/19.0.2.187
intel/19.1.0.166
intel/19.1.3.304

$ module switch intel intel/19.0.2.187
```

（2）コンパイル方法

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。

・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

```
$ ftn -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

```
$ ftn -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

```
$ ftn -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

```
$ ftn -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

```
$ cc -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

```
$ cc -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

```
$ CC -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

```
$ CC -parallel -o hello.out hello.cpp
```

5.1.4 PGI コンパイラ

（1）プログラム環境の設定

super では Cray コンパイラがデフォルトで設定しているため下記のように環境を切り替えて下さい。

・例　Cray コンパイラの環境からの切り替え

```
$ module switch PrgEnv-cray PrgEnv-pgi
```

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。

・例　バージョンの切り替え

```
$ module avail pgi
pgi/18.5
pgi/19.1(default)

$ module switch pgi pgi/18.5
```

（2）コンパイル方法

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

```
$ ftn -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

```
$ ftn -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

```
$ ftn -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

```
$ ftn -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

```
$ ftn -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

```
$ ftn -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

```
$ cc -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

```
$ cc -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

```
$ cc -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

```
$ CC -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

```
$ CC -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

```
$ CC -mp -o hello.out hello.cpp
```

5.1.5 GNU コンパイラ

（1）プログラム環境の設定

super では Cray コンパイラがデフォルトで設定しているため下記のように環境を切り替えて下さい。

・例　Cray コンパイラの環境からの切り替え

```
$ module switch PrgEnv-cray PrgEnv-gnu
```

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。

・例　バージョンの切り替え

```
$ module avail gcc
gcc/4.9.3
gcc/5.3.0
gcc/6.1.0
gcc/7.3.0(default)
gcc/8.3.0

$ module switch gcc gcc/8.3.0
```

（2）コンパイル方法

(ア) 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

```
$ ftn -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

```
$ ftn -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

```
$ CC -o hello.out hello.cpp
```

5.2 ライブラリ使用方法

スーパーコンピュータ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
CSML(Cray Scientific and Math Libraries)	19.05.5 18.03.1	各コンパイラ	
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 18.0.2.199 17.0.4.196	Intel コンパイラ	
Third Party Products	-	各コンパイラ	

5.2.1 CSML(Cray Scientific and Math Libraries)

CSML (Cray Scientific and Math Libraries)は、Cray が提供する科学数学ライブラリ群です。
以下のライブラリが利用可能です。

ライブラリ名称	説明	対応ライブラリ	モジュール名	オプション
Cray LibSci	XC50 システム向けに最適化された科学技術計算ライブラリ デフォルトでロードされる	BLAS, LAPACK, BLACS, ScaLAPCK, IRT, etc	cray-libsci	
Cray PETSc (Portable, Extensible Toolkit for Scientific Computation)	線形・非線形方程式並列ソルバーライブラリ	MUMPS, SuperLU, SuperLU_dist, ParMETIS, HYPRE , etc	cray-petsc	
Cray Trilinos Packages	科学計算ライブラリのオブジェクト指向インターフェイス cray-petsc を事前にロードする必要あり	PETSc, Metis/ParMetis, SuperLU, Aztec, BLAS, LAPACK	cray-trilinos	
TPSL (Third Party Scientific Libraries)	PETSc/Trilinos と組み合わせ可能な数学ライブラリ	MUMPS, Super_LU, Super_LU_dist, ParMetis, Hypre, Sumdials, Scotch, etc	cray-tpsl	

FFTW3.3 Library	FFTW バージョン 3.3	FFTW3.3	cray-fftw	-lfftw3 (MPI 並列) -lfftw3_mpi (スレッド並列) -lfftw3_threads
FFTW2.1 Library	FFTW バージョン 2.1 ライブラリ名が単精度・倍精度によって変わるため、リンク時に選択指示が必要	FFTW2.1	fftw	(単精度) -lsrfftw_mpi -lsfftw_mpi -lsrfftw -lsfftw (倍精度) -ldrfftw_mpi -ldfftw_mpi -ldrfftw -ldfftw

(1) プログラム環境の設定

super では Cray コンパイラをデフォルトで設定しています。

必要に応じて利用するプログラム環境に切り替えて下さい。

・例 intel コンパイラの環境への切り替え

```
$ module switch PrgEnv-cray PrgEnv-intel
```

(2) 使用例

例 1: Cray コンパイラで FFTW3.3 Library を MPI 並列プログラムから利用する場合

```
$ module load cray-fftw
$ cc main.c -L${FFTW_DIR} -lfftw3_mpi  -lfftw3
```

例 2: Intel コンパイラで FFTW3.3 Library を MPI 並列とスレッド並列を組み合わせたプログラムから利用する場合

```
$ module switch PrgEnv-cray PrgEnv-intel
$ module load cray-fftw
$   cc   -qopenmp   main.c   -L${FFTW_DIR}   -lfftw3_mpi
-lfftw3_threads -lfftw3
```

例 3: FFTW2.1 Library を単精度で利用する場合

```
$ module load fftw
$ cc main.c -lsrfftw_mpi -lsfftw_mpi -lsrfftw -lsfftw
```

例 4：FFTW2.1 Library を倍精度で利用する場合

```
$ module load fftw
$ cc main.c -ldrfftw_mpi -ldfftw_mpi -ldrfftw -ldfftw
```

5.2.2 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 プログラム環境(PrgEnv-intel)をロードすることで利用可能です。

なお、競合回避のため、cray-libsci はアンロードして下さい。

・例 Cray コンパイラの環境からの切り替え

```
$ module switch PrgEnv-cray PrgEnv-intel
$ module unload cray-libsci
```

（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

```
$ ftn -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

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

5.2.3 Third Party Products

以下のライブラリが利用可能です。

ライブラリ名称	モジュール名	備考
NetCDF(Unidata's Network Common Data Format) Library	cray-netcdf cray-parallel-netcdf	シリアル版 並列版
HDF5(Hierarchical Data Format 5) Libraries & Utilities	cray-hdf5 cray-hdf5-parallel	シリアル版 並列版

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

スーパーコンピュータ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
1	Gaussian 16	Rev B.01 Rev C.01	SMP
2	ADF	2017.113 2018.105 2019.102 2019.304 2020.101	MPI
3	QuantumATK	2019.03 2019.12 2020.09	MPI
4	CRYSTAL	17	MPI SMP
5	VASP	4.6.38 5.4.4 6.1.0 6.1.1 6.1.2 6.2.0	MPI
6	WIEN2k	17.1 18.2 19.1 19.2 21.1	SMP
7	SIESTA	4.0 4.1.5	MPI
8	ABINIT	8.8.2 8.10.2 8.10.3 9.2.2	MPI

9	CPMD	4.1 4.3	MPI
10	QUANTUM ESPRESSO	6.2.1 6.3 6.4.1 6.5 6.6 6.7	MPI
11	LAMMPS	31 Mar 17 22 Aug 18 12 Dec 18 5 Jun 19 7 Aug 19 3 Mar 20 29 Oct 20	MPI
12	OpenMX	3.8.5 3.9.1 3.9.2	MPI
13	SMASH	2.2.0	MPI
14	TOMBO	2	MPI
15	RSDFT	1.3.0	MPI
16	HPhi	3.1.2	MPI × SMP
17	mVMC	1.0.3	MPI
18	CP2K	7.0 8.1.0	MPI
19	Elk	6.3.2 6.8.4	MPI × SMP
20	ALAMODE	1.1.0	SMP
21	SALMON	1.2.1 2.0.0	MPI
22	OCTOPUS	9.1	MPI

23	Wannier90	1.2 2.1.0	serial
		3.1.0	MPI

6.2 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
B.01	source /work/app/Gaussian/g16.profile
C.01	source /work/app/Gaussian/C.01/g16.profile

/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

並列処理を行うためには、入力ファイルに'CPU=0-N'を指定してください。N は並列数で、35 以下の値にします。
（例）%CPU=0-35
'NProc=N'では正常に並列処理が行われません。

・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。
-d 並列数 の値は入力ファイルの CPU の値と合わせてください。

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

source /work/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

aprun -j 1 -d 並列数 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N g16

source /work/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

aprun -j 1 -d 36 g16 test0000.com 2> g16.err

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

6.3 ADF

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
2017.113	module load adf/2017.113
2018.105	module load adf/2018.105
2019.102	module load adf/2019.102
2019.304	module load adf/2019.304
2020.101	module load adf

ジョブ投入用のスクリプトは以下の通りです。

・入力ファイルでの実行(バージョン:2017.113～2019.304)

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

module load adf/バージョン
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

adf  -n 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) 2019.304

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N adf

module load adf/2019.304
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

adf -n 36  < in  >  adf.out  2> adf.err

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

・入力ファイルでの実行(バージョン:2020.101)

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

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

ams -n 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) 2020.101

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N adf

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

ams -n 36  < in  >  adf.out  2> adf.err

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

・run スクリプトでの実行方法

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

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

export NSCM=並列数
./run スクリプト > 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N adf

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

export NSCM=36
./H2O_HF_freq.run > adf.out 2> adf.err

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

6.4 QuantumATK

以下のバージョンが利用可能です。

バージョン	パス
2019.03	/work/app/QuantumATK/QuantumATK-P-2019.03/bin/atkpython
2019.12	/work/app/QuantumATK/QuantumATK-Q-2019.12/bin/atkpython
2020.09	/work/app/QuantumATK/current/bin/atkpython

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 -l atk=1 -l atkdp= 並列数-1
#PBS -q キュー名
#PBS -N ジョブ名

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ]  hostname  | grep -v ^Applicati > hostfile
ccmrun /work/app/QuantumATK/current/libexec/mpiexec.hydra -n 並列数
-f ./hostfile -genv I_MPI_FABRICS=shm:tcp
/work/app/QuantumATK/current/bin/atkpython 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -l atk=1 -l atkdp=35
#PBS -q P_016
#PBS -N atk

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

aprun -n 36 -N 36 hostname  | grep -v ^Applicati > hostfile
ccmrun /work/app/QuantumATK/current/libexec/mpiexec.hydra -n 36 -f ./hostfile
-genv I_MPI_FABRICS=shm:tcp /work/app/QuantumATK/current/bin/atkpython
input.py > atk.out 2> atk.err

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

オプション-l atk=1 -l atkdp= 並列数 -1 の指定をしない場合、ジョブが正常に実行できません。

6.5 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

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

source /work/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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N crystal

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

source /work/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
```

6.6 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。

実行モジュールの種類	実行モジュールのパス
VASP4.6.36	/work/app/VASP4/current/vasp
VASP4.6.36　Gamma 点版	/work/app/VASP4/vasp.4.6_gamma/vasp
VASP5.4.4　Standard 版	/work/app/VASP5/current/bin/vasp_std
VASP5.4.4　Gamma 点版	/work/app/VASP5/current/bin/vasp_gam
VASP5.4.4　non-collinear 版	/work/app/VASP5/current/bin/vasp_ncl
VASP6.1.0　Standard 版	/work/app/VASP6/vasp.6.1.0/bin/vasp_std
VASP6.1.0　Gamma 点版	/work/app/VASP6/vasp.6.1.0/bin/vasp_gam
VASP6.1.0　non-collinear 版	/work/app/VASP6/vasp.6.1.0/bin/vasp_ncl
VASP6.1.1　Standard 版	/work/app/VASP6/current/bin/vasp_std
VASP6.1.1　Gamma 点版	/work/app/VASP6/current/bin/vasp_gam
VASP6.1.1　non-collinear 版	/work/app/VASP6/current/bin/vasp_ncl
VASP6.1.2　Standard 版	/work/app/VASP6/vasp.6.1.2/bin/vasp_std
VASP6.1.2　Gamma 点版	/work/app/VASP6/vasp.6.1.2/bin/vasp_gam
VASP6.1.2　non-collinear 版	/work/app/VASP6/vasp.6.1.2/bin/vasp_ncl
VASP6.2.0　Standard 版	/work/app/VASP6/vasp.6.2.0/bin/vasp_std
VASP6.2.0　Gamma 点版	/work/app/VASP6/vasp.6.2.0/bin/vasp_gam
VASP6.2.0　non-collinear 版	/work/app/VASP6/vasp.6.2.0/bin/vasp_ncl

Wannier90 をリンクした VASP 5.4.4、VASP 6.1.1 および VASP 6.1.2 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
VASP5.4.4 (Wannier90) Standard 版	/work/app/VASP5/vasp.5.4.4_wannier90/bin/vasp_std /work/app/VASP5/vasp.5.4.4_wannier90v2.1/bin/vasp_std
VASP5.4.4 (Wannier90) Gamma 点版	/work/app/VASP5/vasp.5.4.4_wannier90/bin/vasp_gam /work/app/VASP5/vasp.5.4.4_wannier90v2.1/bin/vasp_gam
VASP5.4.4 (Wannier90) non-collinear 版	/work/app/VASP5/vasp.5.4.4_wannier90/bin/vasp_ncl /work/app/VASP5/vasp.5.4.4_wannier90v2.1/bin/vasp_ncl
VASP6.1.1 (Wannier90)	/work/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_std

Standard 版	/work/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_std
VASP6.1.1 (Wannier90) Gamma 点版	/work/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_gam /work/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_gam
VASP6.1.1 (Wannier90) non-collinear 版	/work/app/VASP6/vasp.6.1.1-wannier90v1.2/bin/vasp_ncl /work/app/VASP6/vasp.6.1.1-wannier90v2.1.0/bin/vasp_ncl
VASP6.1.2 (Wannier90) Standard 版	/work/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_std /work/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_std
VASP6.1.2 (Wannier90) Gamma 点版	/work/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_gam /work/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_gam
VASP6.1.2 (Wannier90) non-collinear 版	/work/app/VASP6/vasp.6.1.2-wannier90v1.2/bin/vasp_ncl /work/app/VASP6/vasp.6.1.2-wannier90v2.1.0/bin/vasp_ncl

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/VASP5/current/bin/vasp_std > 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun -n 36 -N 36 -j 1 /work/app/VASP5/current/bin/vasp_std > vasp.out 2> vasp.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
17.1	/work/app/WIEN2k/current
18.2	/work/app/WIEN2k/WIEN2k_18.2
19.1	/work/app/WIEN2k/WIEN2k_19.1
19.2	/work/app/WIEN2k/WIEN2k_19.2
21.1	/work/app/WIEN2k/WIEN2k_21.1

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=/work/app/WIEN2k/current
export PATH=$WIENROOT:$PATH
module load intel

aprun -b -d 並列数 -j 1 --cc depth 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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=$WORKDIR/$DIRNAME
export TMPDIR=$WORKDIR/$DIRNAME
export WIENROOT=/work/app/WIEN2k/current
export PATH=$WIENROOT:$PATH
module load intel

aprun -b -d 36 -j 1 --cc depth 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。

(例)

```
$ 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
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
4.0	/work/app/SIESTA/current
4.1.5	/work/app/SIESTA/siesta-4.1.5

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1 /work/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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun -n 36 -N 36 -j 1 /work/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
8.8.2	/work/app/ABINIT/current/src/98_main/abinit
8.10.2	/work/app/ABINIT/abinit-8.10.2/src/98_main/abinit
8.10.3	/work/app/ABINIT/abinit-8.10.3/src/98_main/abinit
9.2.2	/work/app/ABINIT/abinit-9.2.2/src/98_main/abinit

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun -n 36 -N 36 -j 1 /work/app/ABINIT/current/src/98_main/abinit < input.file
> 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している必要があります。
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
4.1	/work/app/CPMD/current
4.3	/work/app/CPMD/CPMD4.3

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1 /work/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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun -n 36 -N 36 -j 1/work/app/CPMD/current/bin/cpmd.x inp-1 > cpmd.out 2>
cpmd.err

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

6.11 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
6.2.1	/work/app/QuantumESPRESSO/current
6.3	/work/app/QuantumESPRESSO/qe-6.3
6.4.1	/work/app/QuantumESPRESSO/qe-6.4.1
6.5	/work/app/QuantumESPRESSO/qe-6.5
6.6	/work/app/QuantumESPRESSO/qe-6.6
6.7	/work/app/QuantumESPRESSO/qe-6.7

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun -n 36 -N 36 -j 1 /work/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
```

6.12 LAMMPS

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
31 Mar 17	/work/app/LAMMPS/current
22 Aug 18	/work/app/LAMMPS/lammps-22Aug18
12 Dec 18	/work/app/LAMMPS/lammps-12Dec18
5 Jun 19	/work/app/LAMMPS/lammps-5Jun19
7 Aug 19	/work/app/LAMMPS/lammps-7Aug19
3 Mar 20	/work/app/LAMMPS/lammps-3Mar20
29 Oct 20	/work/app/LAMMPS/lammps-29Oct20

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/LAMMPS/current/src/lmp_intel_omp  < 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun -n 36 -N 36 -j 1 /work/app/LAMMPS/current/src/lmp_intel_omp < in.ij >
lammps.out 2> lammps.err

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

6.13 OpenMX

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
3.8	/work/app/OpenMX/current
3.9.1	/work/app/OpenMX/openmx3.9.1
3.9.2	/work/app/OpenMX/openmx3.9.2

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/OpenMX/current/source/openmx  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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N openmx

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

aprun -n 36 -N 36 -j 1 /work/app/OpenMX/current/source/openmx C60.dat > C60.out
2> C60.err

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

6.14 SMASH

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1 /work/app/SMASH/current/bin/smash
< 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N smash

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

aprun -n 36 -N 36 -j 1 /work/app/SMASH/current/bin/smash < mp2-energy.inp >
mp2-energy.out 2> mp2-energy.err

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

6.15 TOMBO

TOMBO を使用するためには、利用者自身が TOMBO のライセンスを取得している必要があります。TOMBO の利用を希望される場合は、TOMBO 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。

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

mkdir tmp
aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1 /work/app/TOMBO/current/main > 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N tombo

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

mkdir tmp
aprun -n 36 -N 36 -j 1 /work/app/TOMBO/current/main > tombo.out 2> tombo.err

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

6.16 RSDFT

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

export MPICH_NO_BUFFER_ALIAS_CHECK=1
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1 /work/app/RSDFT/current/src/rsdft.x
> 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N rsdft

export MPICH_NO_BUFFER_ALIAS_CHECK=1
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

aprun -n 16 -N 16 -j 1 /work/app/RSDFT/current/src/rsdft.x > rsdft.out 2> rsdft.err

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

並列数は fort.1 の PROCS の値の積を指定します。

例)

```
$ grep PROCS fort.1
PROCS  2 2 4 1 1 1              / process partioning
```

→ 2*2*4*1*1*1=16 を指定

6.17 HPhi

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 OMP_NUM_THREADS=ノードあたりの並列数
aprun [ -n ノード数 ][ -d ノードあたりの並列数 ] -j 1 --cc depth
/work/app/HPhi/current/build/src/HPhi -s 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N hphi

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

export OMP_NUM_THREADS=36
aprun -n 1 -d 36 -j 1 --cc depth /work/app/HPhi/current/build/src/HPhi -s  stan.in
> hphi.out 2> hphi.err

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

6.18 mVMC

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/mVMC/current/build/src/mVMC/vmc.out -s 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N mvmc

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

aprun -n 36 -N 36 -j 1 /work/app/mVMC/current/build/src/mVMC/vmc.out -s
StdFace.def > mvmc.out 2> mvmc.err

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

6.19 CP2K

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
7.0	/work/app/CP2K/current
8.1.0	/work/app/CP2K/cp2k-8.1.0

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 CP2K_DATA_DIR=/work/app/CP2K/current/data

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/CP2K/current/exe/CRAY-XC50-cce/cp2k.popt 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N cp2k

DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME
export CP2K_DATA_DIR=/work/app/CP2K/current/data

aprun -n 36 -N 36 -j 1 /work/app/CP2K/current/exe/CRAY-XC50-cce/cp2k.popt
H2O-32.inp > cp2k.out 2> cp2k.err

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

6.20 Elk

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
6.3.2	/work/app/Elk/current/src/elk
6.8.4	/work/app/Elk/elk-6.8.4/src/elk

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

module load intel
export OMP_NUM_THREADS=ノードあたりの並列数
aprun [ -n ノード数 ][ -d ノードあたりの並列数 ] -j 1 --cc depth
/work/app/Elk/current/src/elk > 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N elk

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

module load intel
export OMP_NUM_THREADS=36
aprun -n 1 -d 36 -j 1 --cc depth /work/app/Elk/current/src/elk > elk.out 2> elk.err

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

6.21 ALAMODE

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
#PBS -q キュー名
#PBS -N ジョブ名

module load intel
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/work/app/ALAMODE/current/spglib/¥
install_dir/lib
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

export OMP_NUM_THREADS=並列数
aprun -d 並列数 -j 1 --cc depth /work/app/ALAMODE/current/anphon/anphon 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N alamode

module load intel
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/work/app/ALAMODE/current/spglib/¥
install_dir/lib
DIRNAME=`basename $PBS_O_WORKDIR`
DIRNAME=`basename $PBS_O_WORKDIR`
WORKDIR=/work/$USER/$PBS_JOBID
mkdir -p $WORKDIR
cp -raf  $PBS_O_WORKDIR $WORKDIR
cd $WORKDIR/$DIRNAME

export OMP_NUM_THREADS=36
aprun -d 36 -j 1 --cc depth /work/app/ALAMODE/current/anphon/anphon test.in >
anphon.out 2> anphon.err

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

6.22 SALMON

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
1.2.1	/work/app/SALMON/current
2.0.0	/work/app/SALMON/SALMON2-v.2.0.0

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/SALMON/current/bin/salmon.cpu < 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N salmon

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

aprun -n 36 -N 36 -j 1 /work/app/SALMON/current/bin/salmon.cpu < test.inp > salmon.out 2> salmon.err

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

6.23 OCTOPUS

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/app/OCTOPUS/current/bin/octopus > 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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N octopus

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

aprun -n 36 -N 36 -j 1 /work/app/OCTOPUS/current/bin/octopus > octopus.out 2>
octopus.err

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

6.24 Wannier90

以下のバージョンが利用可能です。

バージョン	パス
1.2	/work/app/Wannier90/wannier90-1.2
2.1.0	/work/app/Wannier90/wannier90-2.1.0
3.1.0	/work/app/Wannier90/current

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

aprun [ -n 並列数 ][ -N ノードあたりの並列数 ] -j 1
/work/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)

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#PBS -N wannier90

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

aprun -n 36 -N 36 -j 1 /work/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 Python 使用方法

7.1 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。

7.2 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
```

7.3 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
```

7.4 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。
```

7.5 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/user01/.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

7.6 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。

・ジョブスクリプト例

```
#!/bin/sh
#PBS -l select=1
#PBS -q P_016
#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

aprun python program.py

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