ジョブ投入に関するTips

最終更新: 2025/1/30

基本

- 各アプリ用のサンプルが /apl/(アプリ名)/(バージョン)/samples にあります。テンプレートとしてもご利用いただけます。

sh/bash での基本形:

```
#!/bin/sh
#PBS -l select=...
#PBS -l walltime=[時間]

# PBS_O_WORKDIR はジョブを投入した時の作業ディレクトリ(CWD)に当たります
# ccfep でのジョブの投入時と計算ノードでの作業ディレクトリをそろえるために
# スクリプト内でディレクトリ移動を行います
if [ ! -z ${PBS_O_WORKDIR} ]; then
  cd ${PBS_O_WORKDIR} # PBS_O_WORKDIR 変数が存在している場合、そのディレクトリへ移動
fi

(実際の処理)
```

csh/tcsh での基本形:

```
#!/bin/csh -f
#PBS -l select=...
#PBS -l walltime=[時間]

if ( $?PBS_O_WORKDIR ) then
  cd $PBS_O_WORKDIR # sh/bash の場合に同じ
endif

(実際の処理)
```

ジョブヘッダのサンプル(select文のサンプル)

- select= の直後にくる数字はノードの数を表します(省略すると 1)。
- 他の数字(ncpus,mpiprocs,ompthreads,ngpus)はノードあたりの数字です。
- OMP_NUM_THREADS 環境変数は自動的に ompthreads の設定値になります。
- /apl に導入されている MPI 環境(OpenMPI,IntelMPI, MVAPICH)の場合、実行ホストのリストを指定する必要はありません。
  - mpirun -np (mpiprocsで指定した数字) command options の形で大丈夫です。
  - (ご自分で OpenMPI をビルドしていて、configure 時に --with-tm=/apl/pbs/22.05.11 を指定していない場合はホストリストの指定が必要になる場合があります。こちらのページに関連情報があります。)
- GPU 利用時に CUDA_VISIBLE_DEVICES 環境変数に特別の注意を払う必要はありません。(リソースはジョブサーバできちんと管理されています)
  - (ソフトによっては GPU 複数利用時に手動で割当が必要な場合があります。)

1 vnode, vnode あたり: 64 CPU コア, MPI*64, OpenMP 無し (Flat MPI) (合計 1 vnode, 64 CPU コア), 72 時間(3日) (jobtype=vnode が自動指定)

```
#PBS -l select=1:ncpus=64:mpiprocs=64:ompthreads=1
#PBS -l walltime=72:00:00
```

4 vnode, vnode あたり: 64 CPU コア, MPI*64, OpenMP 無し (合計 4 vnode, 256 CPU コア), 168 時間(一週間) (jobtype=vnode が自動指定)

```
#PBS -l select=4:ncpus=64:mpiprocs=64:ompthreads=1
#PBS -l walltime=168:00:00
```

jobtype=largemem, 1 ノード, ノードあたり: 128 CPU コア, MPI*64, OpenMP*2 (合計 1 ノード, 128 CPU コア), 30 分

```
#PBS -l select=1:ncpus=128:mpiprocs=64:ompthreads=2:jobtype=largemem
#PBS -l walltime=00:30:00
```

jobtype=largemem, 2 vnode, vnode あたり: 64 CPU コア, MPI*64, OpenMP 無し (合計 2 vnode, 128 CPU コア)、1 時間

```
#PBS -l select=2:ncpus=64:mpiprocs=64:ompthreads=1:jobtype=largemem
#PBS -l walltime=01:00:00
```

1 ノードで実行される場合と 2 ノードに分かれて実行される場合があります。一つ上の例の場合には確実に 1 ノードで実行されます。

1 CPU コア、168 時間 (jobtype=core が自動指定)

```
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=168:00:00
```

12 CPU コア, MPI*4, OpenMP 無し, 12 時間 (jobtype=core が自動指定)

```
#PBS -l select=1:ncpus=12:mpiprocs=4:ompthreads=1
#PBS -l walltime=12:00:00
```

利用可能なメモリ量はncpusの値に比例します。そのため、jobtype=core で MPI のプロセス数を増やさずにメモリ量を増やしたい場合はこの記述が有用です。

18 CPU コア, MPI*9, OpenMP*2, 3時間 (jobtype=core が自動指定)

```
#PBS -l select=1:ncpus=18:mpiprocs=9:ompthreads=2
#PBS -l walltime=03:00:00
```

32 CPU コア, MPI*8, OpenMP*4, 168 時間 (jobtype=core が自動指定)

```
#PBS -l select=1:ncpus=32:mpiprocs=8:ompthreads=4
#PBS -l walltime=168:00:00
```

60 CPU コア, OpenMP*60, 168 時間 (jobtype=core が自動指定)

```
#PBS -l select=1:ncpus=60:mpiprocs=1:ompthreads=60
#PBS -l walltime=168:00:00
```

64 CPU コア(確保), OpenMP*60, 168 時間 (jobtype=vnode が自動指定)

```
#PBS -l select=1:ncpus=64:mpiprocs=1:ompthreads=60
#PBS -l walltime=168:00:00
```

一つ上の例と実質的に同じ計算を行いますが、64 コアを確保して jobtype=vnode となっている点だけ異なります。こちらの場合は 1 時間あたりの CPU 点数が 45 点となり、一つ上の例の場合(60 点)よりも CPU 点数的に効率的です。

1 CPU コア, GPU*1, 24時間 (jobtype=gpu が自動指定)

```
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1:ngpus=1
#PBS -l walltime=24:00:00
```

16 CPU コア, MPI*16, OpenMP 無し, 1 GPU, 12 時間 (jobtype=gpu が自動指定)

```
#PBS -l select=1:ncpus=16:mpiprocs=16:ompthreads=1:ngpus=1
#PBS -l walltime=12:00:00
```

ngpus=1 の時は ncpus=16 が最大です。

4 ノード(vnode), ノードあたり: 8 CPU コア, MPI*2, OpenMP*4, 2 GPU (合計 4 ノード, 32 CPU コア, 8 GPU), 12 時間 (jobtype=gpu が自動指定)

```
#PBS -l select=4:ncpus=8:mpiprocs=2:ompthreads=4:ngpus=2
#PBS -l walltime=12:00:00
```

mpiprocs の値は ngpus の値の倍数である必要があります。実際には GPU だけを使い、CPU を使わない場合でも select 文では CPU を使うという指定が必要です。

2 ノード、ノードあたり: 24 CPU コア, MPI*24, OpenMP 無し、2 GPU (合計 2 ノード, 48 CPU コア, 4 GPU), 24 時間 (jobtype=gpu が自動指定)

```
#PBS -l select=2:ncpus=24:mpiprocs=24:ompthreads=1:ngpus=2
#PBS -l walltime=24:00:00
```

最大で指定数の倍のノード(この場合ならば 4 ノード)にプロセスが散らばる可能性があります。