プログラミング講習会

演習問題

京都大学化学研究所スーパーコンピュータシステム

2026年6月5日

Hewlett Packard Enterprise

1. moduleコマンド

- ログインノードにてどのようなアプリケーションが利用できるか確認してみましょう。(先頭の $ はプロンプトと呼ばれるもので、入力しません。ログインノード(ホスト名 : fe1, FQDN: fe1.scl.kyoto-u.ac.jp)にログインした場合、$ の部分は fe1{xxx}####: といった文字列に置換されます。)

```
$ ssh fe1
$ module avail
```

- gcc コンパイラとして利用できるバージョンにはどのようなものがあるか確認してください。

```
$ module avail gcc
```

- Intel oneAPI を利用するための環境設定を行ってください。(モジュール名は compiler です。)
  正しく環境設定ができたか、リスト表示で確認します。その後、その環境設定を破棄してください。

```
$ module purge
$ module avail compiler
$ module load compiler/latest
$ module list
$ module unload compiler    (もしくは $ module purge)
$ module list
No Modulefiles Currently Loaded.
```

1. moduleコマンド

- Intel oneAPI , Intel MPI, Intel MKL を利用するための環境設定を行ってください。それぞれのmodule名は、compiler, mpi, mkl です。それぞれの利用するバージョンは特に問いません。
- 正しく環境設定ができたか、リスト表示で確認します。
- Intel MPIの代わりに、OpenMPIの環境設定を行ってください。利用するバージョンは特に問いません。

```
$ module avail compiler
$ module load compiler/2024.1.0
$ module avail mpi
$ module load mpi/2021.12
$ module avail mkl
$ module load mkl/2024.1
$ module list
$ module unload mpi
$ module avail openmpi
$ module load openmpi/4.1.6/gcc/12.3.0/cpu
$ module list
$ module purge
```

2. コンパイルと実行

- ハイブリッドプログラム　hybrid_parallel.f90 をIntel oneAPIを使ってコンパイルしてみましょう。
- このプログラムでは、関数 $f(x) = \frac{4}{1+x^2}$ の積分（積分範囲：[0, 1])を数値的に計算します。この時 $\int_0^1 f(x)dx = \pi$ となります。

```
$ module purge
$ module load compiler/2022.1.0
$ module load mpi/2021.12
$ module list
$ mpiifort -qopenmp -o hybrid_parallel hybrid_parallel.f90

$ export OMP_NUM_THREADS=1      <---  1スレッド実行
$ /usr/bin/time ./hybrid_parallel     （実行時間：94秒）
 Calculated Pi =    3.14159265358936
 Error =   4.334310688136611E-013
 Execution Time =    93.7862889766693       seconds
92.45user 0.20system 1:33.91elapsed 98%CPU (0avgtext+0avgdata 118616maxresident)k

$ export OMP_NUM_THREADS=4    <---  4スレッド実行
$ /usr/bin/time ./hybrid_parallel         （実行時間：24秒）

$ export OMP_NUM_THREADS=1    <---  1スレッド実行
$ /usr/bin/time mpirun –np 4 ./hybrid_parallel      <--  4MPI実行（実行時間：25秒）
```

※MPI: Message Passing Interface

hybrid_parallel の経過時間（秒）の測定

MPI（mpirun –np # ）	OpenMP（OMP_NUM_THREADS）		
	1	2	4
./hybrid_parallel	93.7		
mpirun –np 2 ./hybrid_parallel	47.3		
mpirun –np 4 ./hybrid_parallel	24.1		

実行中のジョブの状況は top コマンドで確認できます

OpenMPによる4並列実行の場合

```
$ setenv OMP_NUM_THREADS 4; ./hybrid_parallel
$ top -u fukumoto

top - 11:02:08 up 222 days, 22:02, 21 users,  load average: 0.38, 0.52, 0.54
Tasks: 1040 total,   2 running, 1034 sleeping,   4 stopped,   0 zombie
%Cpu(s):  3.3 us,  0.1 sy,  0.0 ni, 96.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 515254.9 total, 234544.3 free,  45796.2 used, 234914.4 buff/cache
MiB Swap:  10240.0 total,   7530.0 free,   2710.0 used. 454215.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3034453 fukumoto  20   0  629812 120144  18288 R 399.3   0.0   0:50.39 hybrid_parallel
 455660 fukumoto  20   0   41764   3472   2876 S   0.0   0.0   0:00.02 qsub
 455686 fukumoto  20   0   41764    612      0 S   0.0   0.0   0:00.16 qsub
```

Hキーで表示切り替え

```
（Threads表示）

top - 10:57:08 up 222 days, 21:57, 21 users,  load average: 0.94, 0.64, 0.56
Threads: 1762 total,   2 running, 1752 sleeping,   8 stopped,   0 zombie
%Cpu(s):  3.3 us,  0.1 sy,  0.0 ni, 96.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 515254.9 total, 230127.4 free,  46419.6 used, 238707.9 buff/cache
MiB Swap:  10240.0 total,   7530.0 free,   2710.0 used. 453592.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3034753 fukumoto  20   0  631860 118128  18364 R  99.9   0.0   0:06.57 hybrid_parallel
3034754 fukumoto  20   0  631860 118128  18364 R  99.9   0.0   0:06.57 hybrid_parallel
3034747 fukumoto  20   0  631860 118128  18364 R  99.7   0.0   0:06.71 hybrid_parallel
3034752 fukumoto  20   0  631860 118128  18364 R  99.7   0.0   0:06.56 hybrid_parallel
```

自分のプロセスだけを確認したい場合、u キーでユーザ名を指定します。
終了するには、q キーを押下します。
Threads表示の場合はPID欄にTID(Thread ID)が表示されます。
psコマンドでPID, TIDを表示するには ps –L –p <PID> -o pid,tid,comm

3. PBS

- どのようなキューが利用できるか確認してみましょう。

```
$ qstat -Q
```

- sleep 60 を実行するバッチスクリプトを作成し、投入してみてください。必要なリソースは、4core, 1GBメモリとし、SMALLキューを指定してください。sleep 60 の前後にはdateコマンドを実行し、確かに60秒であるか確認します。バッチジョブ名は自由につけてください。
- バッチスクリプト名は、sleep.qsub とします。ファイルを作成する場合、例えば $ nano sleep.qsub とします。
- ジョブ実行中にqstatコマンドでジョブの状況を確認してみてください。

```
#!/bin/bash
#PBS -N test
#PBS -q SMALL
#PBS -l select=1:ncpus=4:mem=1gb

date
sleep 60
date
```

← バッチスクリプト

```
$ qsub sleep.qsub
$ qstat
```

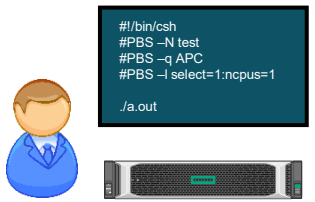
バッチスクリプトの作成方法

（1） 端末上で作成しファイル転送ソフトで計算サーバへコピーする

- 修正のたびに計算サーバへ転送が必要
- Windowsの改行コードに注意。このことが原因で実行時にエラーとなった場合dos2unix コマンドでファイルフォーマットを変換してください

(実行例)
% dos2unix sleep.qsub
% dos2unix: converting file sleep.qsub to Unix format...

（2） 直接計算サーバにログインして作成する



- 計算サーバで使用できるEditerを利用
（vi, emacs, nano etc.）

3. PBS

- 2. コマンドと実行　で使用した hybrid_parallel プログラムをバッチジョブとして実行してみてください。
- 作成するバッチスクリプト名は、hybrid_parallel.qsub とします。
- バッチキューはSMALLとします。MPIプロセス数は4で実行します。使用可能メモリは1GBとします。
- スレッドによる並列は行わないため、OMP_NUM_THREADS は 1 とします。
- $PBS_O_WORKDIRはqsubコマンドを実行した絶対パスになります。

```
#!/bin/bash
#PBS -N hybrid
#PBS -q SMALL
#PBS -l select=1:ncpus=4:mpiprocs=4:mem=1gb

source /etc/profile.d/modules.sh
module load compiler/2022.1.0
module load mpi/2021.12

export OMP_NUM_THREADS=1
cd $PBS_O_WORKDIR
mpirun ./hybrid_parallel
```

(1) ssh コマンドを利用する場合

```
$ qstat -n1      （実行ノードを確認）
$ ssh node top -u usrname -b -n 1

ユーザlect-1, 実行ノードcs01-admの場合
$ ssh cs01-adm top -u lect-1 -b -n 1
```

ノード名：
クラスターの場合は –adm を付加
SDFの場合は sdf1 または sdf2 を指定

(2) インタラクティブバッチジョブを利用する場合

```
$ qstat -n1      （実行ノードを確認）
$ qsub -I -q SMALL -l select=1:ncpus=1:host=node
$ top -u username
$ exit
```

- 計算ノードにログインして直接実行ジョブを確認することができません。そのため、sshコマンドもしくはインタラクティブバッチジョブを利用して、ジョブが実行されている計算ノードにログインし、topコマンドでこのジョブのプロセスを確認してみましょう。
- 余裕があれば、MPIプロセスは2、OMP_NUM_THREADS を 2 として実行してみてください。その場合、select文も修正が必要になります。

5. 計算化学アプリケーション (Gaussianジョブ)

- 量子化学計算プログラム(Gaussian)を実行してみましょう。
- サンプルプログラム（test0397.com）を利用します。
- 使用するメモリは8gbとします。
- 利用するキューはSMALL/APCキューとします。
- バッチスクリプト名は g16.qsub とします。
- 以下を参考に入力ファイルを修正します。

入力ファイル：test0397.com

```
%nproc=16
%mem=1gb
%chk=test0397.chk

#p rb3lyp/3-21g force test scf=novaracc

Gaussian Test Job 397:
Valinomycin force

0,1
0,-1.3754834437,-2.5956821046,3.7664927822
…
…
```

バッチスクリプト：g16.qsub

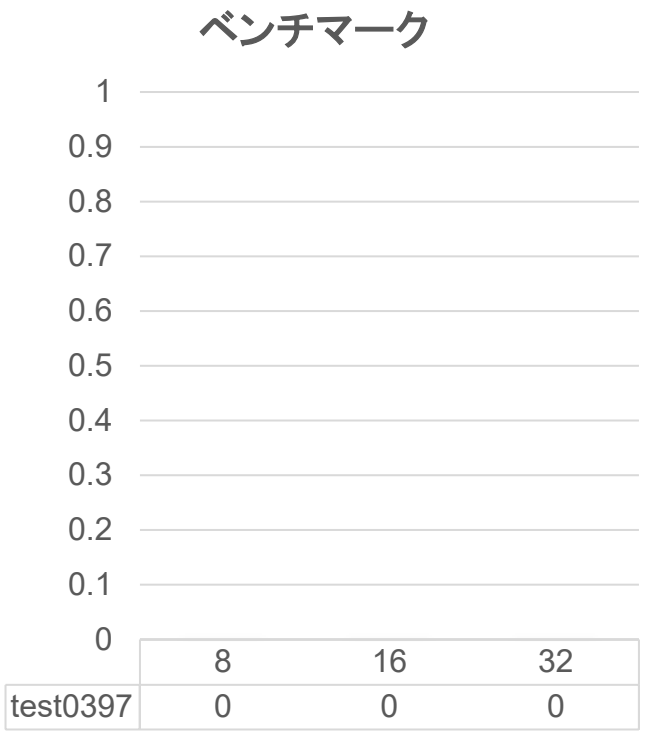
```
#!/bin/bash
#PBS -N g16
#PBS -q APC
#PBS -l select=1:ncpus=16:mem=2gb

source /etc/profile.d/modules.sh
module load g16/c01
cd $PBS_O_WORKDIR

/usr/bin/time -p rung16 test0397.com test0397.log

#
```

バッチスクリプト内で指定する値は入力ファイル内で指定するメモリサイズよりも**大きく設定**

```
$ qsub g16.qsub
$ qstat
```

5. 計算化学アプリケーション（Gaussianジョブ）

- コア数を8, 16, 32 と変えて、経過時間を測定してください。



5. 計算化学アプリケーション (LAMMPSジョブ)

- 分子動力学プログラム(LAMMPS)を実行してみましょう。
- サンプルプログラム(rhodo)を利用します。
- 利用するキューはSMALLとします。
- バッチスクリプト名は lammps.qsub とします。
- INTELパッケージを使用します。
- 入力ファイル in.rhodo および data.rhodo はあらかじめ用意されたファイルを使用します。
- 以下を参考にバッチファイルを作成します。
- スレッド数およびMPI数は適宜変更してTotal wall timeを比較します。
- なお計算ステップ数は時間的な制約のため3,000ステップに設定しています。
- 12分ほどでジョブが終了します。

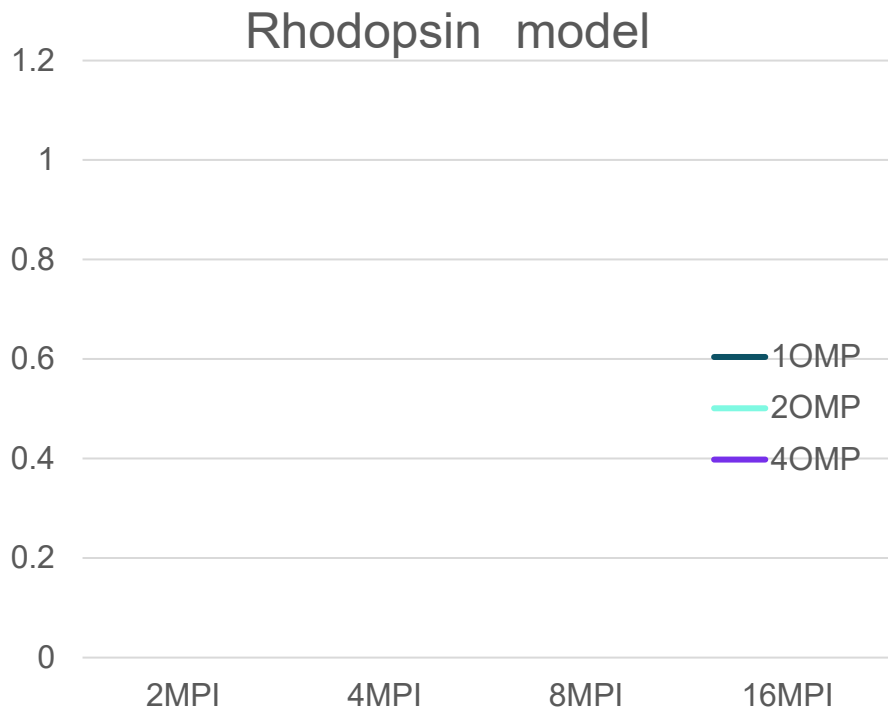
バッチスクリプト:lammps.qsub

```
#!/bin/bash
#PBS -N lammps
#PBS -q SMALL
#PBS -l select=1:ncpus=8:mpiprocs=4

source /etc/profile.d/modules.csh
module load lammps/2408.1
export OMP_NUM_THREADS=2

cd $PBS_O_WORKDIR

mpirun -np 4 lmp_intel_cpu -sf intel -in in.rhodo > lammps.log
```

```
$ qsub lammps.qsub
$ qstat
```

5. 計算化学アプリケーション（LAMMPSジョブ）

- スレッド数およびMPI数を変えて、ログファイルの最終行に表示されるTotal wall time: を測定してください。



MPI	OpenMP（OMP_NUM_THREADS）		
	1	2	4
2			
4			
8			
16			

（過去の実行例）

MPI	OpenMP（OMP_NUM_THREADS）		
	1	2	4
2	171	124	73
4	87	65	54
8	45	48	56
16	24	44	

補足） 計算サーバにおける主な利用コマンド

講習会用アカウントとパスワード

- usrname: lect-1, lect-2, lect-3
- password: prog.6.5

サンプルコード格納先

- /tmp/training/samples/

Linux環境で利用できるエディタ

- vi: システム管理者向けエディタ
- emacs：Unix環境にてユーザがよく利用するエディタ
- nano: 初心者向けエディタ

nanoの簡単な使い方

- 新規作成/開く：$ nano filename
- 保存：Ctrl-O
- 終了：Ctrl-X
- 取消（Undo）：Esc U

ジョブスケジューラ関連コマンド

- qsub *script*: ジョブ投入
- qstat [-a/-n1/-Q/-f *JobID*]：ジョブ状況確認
- qdel *JobID* : ジョブ停止

- qstatmyjobs: ジョブスケジューラ状況確認

Linuxでのコマンド

- サーバにログイン：ssh *usrname@fe1*
- サーバからログアウト：exit, logout

- 階層ディレクトリの移動：cd *xxx*, cd .., pushd, popd
- ホームディレクトリへの移動：cd
- 現在の階層ディレクトリの表示：pwd
- 現在の階層ディレクトリに置かれたファイルやディレクトリの表示：ls
- ファイルの中身を表示：more, less, cat
- ファイル名の変更：mv *file1 file2*
- ファイルのコピー：cp *file1 file2*
- ファイルの削除：rm *file1*
- Windows形式のファイルをUNIX形式に変換：dos2unix *file*

- ファイルの先頭/最後に近い内容を表示：head/tail
- 2つのファイルの内容の違いを比較：diff
- ファイル内容に対するキーワード検索：grep

- プロセス情報の確認：ps
- 計算ノードのリソース使用状況の確認：top

- モジュールファイルをロード：module load *xxx*
- ロードされたモジュールファイルの確認：module list
- ロードされたモジュールファイルの設定をリセット：module purge

- ある特定の環境変数の確認：echo *$XXXX*
- 設定されているすべての環境変数の確認：printenv
- 環境変数設定：setenv / export

補足） vi エディタの基本コマンド（システム管理者向けエディタ）

◆vi の基本モード

モード	説明
ノーマルモード	コマンド入力（初期状態）
挿入モード	文字入力（iキーなどで入る）
コマンドモード	:で開始、保存や終了などを実行

◆よく使うキー操作

操作内容	キー入力	説明
挿入開始	i	カーソルの位置から挿入
	a	カーソルの次の文字の位置から挿入
	o	下の行に新しい行を追加し挿入
	O	上の行に新しい行を追加し挿入
終了・保存	:w	保存（write）
	:q	終了（quit）
	:wq または ZZ	保存して終了
	:q!	保存せず終了（強制）
カーソル移動	h / l	左 / 右
	j / k	下 / 上
	0 / ^	行頭 / 行の最初の文字
	$	行末
	w / W	次の単語へ（区切り小 / 大）
	b / B	前の単語へ（同上）

◆よく使うキー操作

操作内容	キー入力	説明
スクロール	Ctrl+d / Ctrl+u	半ページ下 / 上
	Ctrl+f / Ctrl+b	1ページ下 / 上
削除・カット	x	1文字削除
	dd	行を削除（cut）
	d$	カーソルから行末まで削除
	dw	単語削除（cut）
コピー・ペースト	yy	行をコピー（yank）
	yw	単語をコピー
	y$	カーソルから行末までコピー
	p / P	貼り付け（後 / 前）
元に戻す・やり直し	u	直前の操作を取り消す（undo）
検索・置換	/文字列	下方向に検索
	?文字列	上方向に検索
	n / N	次 / 前の一致へ移動
	:%s/旧/新/g	全文検索置換（確認なし）
	:%s/旧/新/gc	全文検索置換（確認あり）
表示補助	:set number	行番号を表示
	:set nonumber	行番号を非表示