計算機利用の手引き

■[最終更新日時] 2021年4月5日

English versionもあります。

目次

接続

RCCSのコンピューターへのログイン

- フロントエンド(ccfep.ims.ac.jp)へはsshの公開鍵認証を使って接続します。
- GPU搭載フロントエンド('ccgpup', 'ccgpuv')へは'ccfep'からのみ接続できます。
- メンテナンス中はログイン出来ません。→定期メンテナンス情報（メンテナンス日は原則毎月第一月曜日）
- フロントエンドへのログインは、日本に割り当てられたIPv4アドレスを持つホストまたは許可されたホストからでなければなりません。→日本国外からの接続について

ssh公開鍵とウェブページ用パスワードの登録

sshの公開鍵と秘密鍵のペアを最初に作成しておきます。作成方法が不明な方はインターネットなどで調べてください。
クイックスタートガイドのページにも情報があります。

初めて登録する場合もしくはウェブページ用パスワードを忘れた場合

1. 専用ウェブページの「登録案内メールの発行」をウェブブラウザーで開きます。
2. 利用申請書で申請したメールアドレスを入力後、「登録案内メール送信」ボタンを押します。
3. 自動送信されたメールの中に記載されているURLをウェブブラウザーで開きます。
4. ユーザー限定ページにアクセスするために、新たに設定したいパスワードを2ヶ所に入力します。
5. あらかじめ用意したsshの公開鍵をペーストなどして入力します。
6. 「保存」ボタンを押します。

ウェブページ用パスワードを使う場合

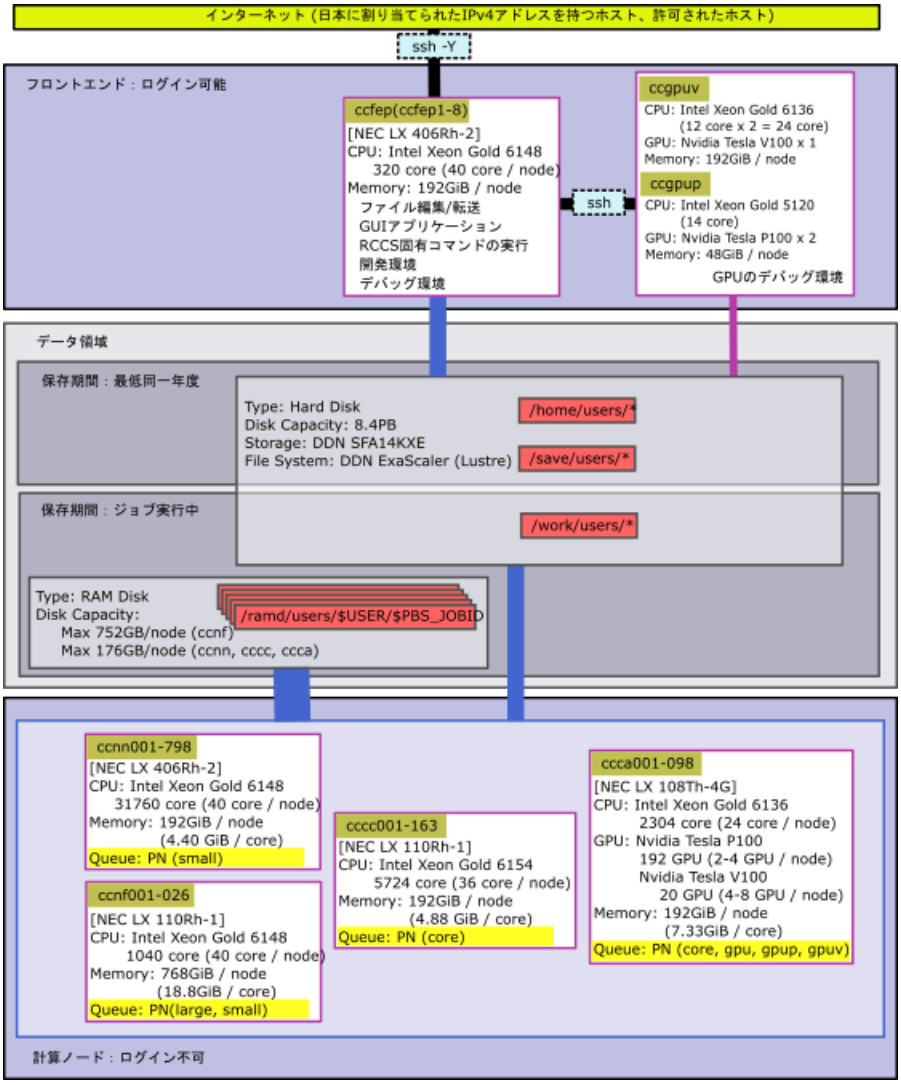
1. 専用ウェブページ(https://ccportal.ims.ac.jp/account/)をウェブブラウザーで開き、ユーザー名とパスワードを入力し、「ログイン」ボタンを押します。
2. 画面右上の「アカウント情報」を開きます。
3. 「編集」タブを押します。
4. パスワードを変更するは、現在のパスワードと新たに設定したいパスワードを入力します。
5. あらかじめ用意したsshの公開鍵をペーストなどして入力します。
6. 「保存」ボタンを押します。

ログインシェル

- /bin/csh(tcsh)、/bin/bash、/bin/zshを利用できます。
- 変更はssh公開鍵と同じウェブページで行ないます。変更が反映されるまで時間がかかる場合があります。
- .loginや.cshrcはカスタマイズしても構いませんが、十分な注意が必要です。

RCCSシステムの全体像

- いわゆる会話処理ができるのは、ccfep (8ノード) と ccgpup, ccgpuv です。ビルドやデバッグにご利用下さい。
- 'ccgpup', 'ccgpuv' 以外の会話処理ノードはインターネットから直接ログインできます。
- 保存期間が異なるディスクが、/work、/ramd、/home、/saveの4種類あります。
- アクセス速度は、下図の太い線の方が高速です。
- /workは計算途中の一時ファイルを置くのに使います。ジョブ終了後に削除されます。
- /ramdは容量が176GBもしくは752GB程度のラムディスクです。プログラムで使う容量とラムディスクで使う容量の総計はキューイングシステムによって管理されています。
- /homeと/saveは現在全く同様に扱われています。容量制限についても合算値で判断されます。(歴史的な経緯で名前だけが異なっています。)
- /tmp、/var/tmp、/dev/shmの一時ディレクトリーの使用を禁止します。一時ディレクトリーを使用しているジョブは見つけ次第削除しますので予めご了承ください。



リソース関係

CPU点数とキュー係数

CPU点数は、CPUやGPUを使うことによって減ります。
減る点数はシステム毎に設定されているCPUキュー係数とGPUキュー係数により求められます。

システム	CPUキュー係数	GPUキュー係数
cclx (jobtype=large)	42 / (点/(1ノード * 1時間))	-
cclx (jobtype=small)	28 / (点/(1ノード * 1時間))	-
cclx (jobtype=core)	1.0 / (点/(1コア * 1時間))	-
cclx (jobtype=gpu, gpup)	1.0 / (点/(1コア * 1時間))	10 / (点/(1GPU * 1時間))
cclx (jobtype=gpuv)	1.0 / (点/(1コア * 1時間))	15 / (点/(1GPU * 1時間))

- 会話処理ノードの内、ccfep, ccgpuv はCPU時間でCPU点数が消費されます。
- 会話処理ノードの ccgpup ではCPU点数が消費されません。
- 他のシステムは、経過時間でCPU点数が消費されます。
- 実際の費用は無料です。

現在の使用CPU点数、残りのCPU点数を知るためには、showlim -cコマンドを使います。

リソース集計

- キューイングシステムで実行されたジョブの集計とディスク使用量の集計は10分毎に行います。
- 会話処理の集計は、毎日5:15に行ないます。
- CPU点数を使いきると、グループ内利用者全員の全ての実行中ジョブは削除され、新たなジョブ投入が抑止されます。
- ディスク使用量が制限値を越えると、新たなジョブ投入が抑止されます。

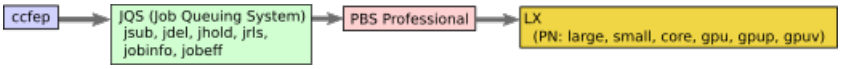
ユーザ別 リソース制限の設定／表示

ウェブブラウザーでリソース制限設定ページにアクセスします。

- 代表利用者のみがリグループ内のメンバー個々に対してグループに割り当てられたリソース量を限度に制限値を設定できます。
- 一般利用者はリソース制限の値を確認することができます。
- リソース制限の種類は、CPU数、点数、ディスク容量です。

キューイングシステム関連

キューイングシステムの全体像



キュー構成

■全利用者が利用可能なキュー

システム	キュー名	演算ノード	メモリー	1ジョブの制限	グループ実行制限		グループサブミット制限	
					割当点数	コア数/GPU数	割当点数	ジョブ数
cclx	PN (jobtype=large)	ccnf	18.8GB/core	1〜10ノード（40〜400コア）	300万点以上 100万点以上 30万点以上 10万点以上 10万点未満	4000/72 2560/48 1600/30 960/18 320/12	300万点以上 100万点以上 30万点以上 10万点以上 10万点未満	4000 2560 1600 960 320
cclx	PN (jobtype=small)	ccnn ccnf	4.4GB/core	1〜32ノード（40〜1280コア）				
cclx	PN (jobtype=core)	cccc ccca	4.8GB/core	1〜36コア				
cclx	PN (jobtype=gpu, gpup)	ccca	7.3GB/core	1〜48GPU 2〜24コア/ノード(2GPU/ノード) 1〜12コア/ノード(1GPU/ノード)				
cclx	PN (jobtype=gpuv)	ccca	7.3GB/core	1〜8GPU 1〜3コア/GPU(単一ノード限定)				

- jobtype=coreで19コア以上のジョブの最大時間は一週間です。
- jobtype=gpuvのジョブについても最大時間は一週間です。
- 上記以外のジョブの最大時間は、定期メンテナンスまでです。ただし、1週間を越えるジョブが実行できる演算ノードは全体の半数です。
- 526ノード並列までのジョブは、同一OmniPathグループ内に接続された演算ノードで実行されます。
- 演算ノードccnnのうち272ノードは1-4ノードのジョブ専用です。
- ジョブの最大時間が1日以下のジョブタイプsmallのジョブは、演算ノードccnfで実行される場合があります。
- ジョブの最大時間が1日以下で要求コア数が4-18のジョブタイプcoreのジョブは、演算ノードcccaで実行される場合があります。
- ジョブタイプcore, gpu, gpup, gpuvのジョブは他のジョブとノードを共有します。
- 演算ノードcccaには2-8 GPUが搭載されており、ノード内GPU間高速通信(Peer-to-peer通信)が可能です。
  - jobtype=gpup を指定した場合には NVIDIA Tesla P100 搭載ノードで実行されます。
  - jobtype=gpuv を指定した場合には NVIDIA Tesla V100 搭載ノードで実行されます。
  - jobtype=gpu を指定した場合には P100, V100 のどちらかが使われます。
- グループ制限を判断する点数には追加点数を含みません。

■別途申請が必要なキュー

キューの設定は下記の通りです。

システム	キュー名	制限時間	メモリー	1ジョブあたりのコア数	グループ制限
cclx	専有利用	7日間単位	4.4GB/core	応相談	許可されたコア数

ジョブの状態表示

```
ccfep% jobinfo [-h HOST] [-q QUEUE] [-c|-s|-m|-w] [-n] [-g GROUP|-a] [-n]
```

表示する情報の選択

以下のオプションで表示する情報を選択します。複数同時に指定することはできません。

- -c 最新のジョブの状態を表示します(利用 GPU 数やグループの情報等、一部情報が利用できません)
- -s キューのサマリーを表示
- -m ジョブのメモリ情報を表示
- -w ジョブをサブミットしたディレクトリーを表示
- -n 各演算ノードの状態を表示する

他ユーザーのジョブ表示について

- -g 同一グループ内の全ユーザの情報を表示
  - 複数のグループに属している場合、-g GROUP名 でグループ名を指定することもできます。
- -a 全てのユーザの情報を表示
  - 他ユーザーのジョブ名等の情報は隠蔽されます。

キュー指定関連オプション

何も指定しなければ PN 及び専有利用キュー(PNR[0-9])のジョブ全てが対象となります。そのため、通常は指定する必要ありません。

- -h HOST: ホスト名で指定します(現在は cclx だけが有効です)
- -q QUEUE: キュー名で指定します

実行例: サマリーの表示

利用中/キュー待ち中/ホールド中ジョブの数やCPU数、GPU数を表示します。制限値についても表示されます。
また、各 jobtype の混雑状況も表示されます。

```
ccfep% jobinfo -s
User/Group Stat:
-------------------------------------------------------------------------------
 queue: PN                    | user                | group
-------------------------------------------------------------------------------
  NJob (Run/Queue/Hold/-)     |   0/   0/   0/-   |   0/   0/   0/-
  CPUs (Run/Queue/Hold/RunLim) |   0/   0/   0/3000 |   0/   0/   0/4000
  GPUs (Run/Queue/Hold/RunLim) |   0/   0/   0/48   |   0/   0/   0/72
  core (Run/Queue/Hold/RunLim) |   0/   0/   0/500  |   0/   0/   0/500
-------------------------------------------------------------------------------
Queue Status (PN):
-------------------------------------------------------------------
     job        | free  |     free     | # jobs  |  requested
     type       | nodes  | cores (gpus) | waiting | cores (gpus)
-------------------------------------------------------------------
week jobs
-------------------------------------------------------------------
small 1-4 nodes  |     0 |     0      |    142 |  11400
small 5+  nodes  |     0 |     0      |     17 |  10600
large            |     0 |     0      |      0 |     0
core             |     0 |   200      |     33 |   585
gpu              |     0 |   430 (22) |     98 |   252 (98)
-------------------------------------------------------------------
long jobs
-------------------------------------------------------------------
small 1-4 nodes  |     0 |     0      |      1 |    40
small 5+  nodes  |     0 |     0      |      3 |  1920
large            |     0 |     0      |      0 |     0
core             |     0 |   133      |      0 |     0
gpu              |     0 |   207 (11) |      0 |     0 (0)
-------------------------------------------------------------------
```

出力上部の core (Run/Queue/Hold/RunLim) はコア単位で利用する場合の制限値です。
上記出力の場合は最大 500 コアまで利用できます。jobtype=small や jobtype=large での利用状況には影響を受けません。

実行例: 個々のジョブの状態を見る

-c オプションを指定すると、ジョブの最新の状態を確認できます。(以前と同様に -l を指定しても問題ありません)

```
ccfep% jobinfo -c
-------------------------------------------------------------------------------
Queue   Job ID Name          Status CPUs User/Grp      Elaps Node/(Reason)
-------------------------------------------------------------------------------
PN      9999900 job0.csh      Run     16  zzz/---    24:06:10 cccc047
PN      9999901 job1.csh      Run     16  zzz/---    24:03:50 cccc003
PN      9999902 job2.sh       Run      6  zzz/---     0:00:36 cccc091
PN      9999903 job3.sh       Run      6  zzz/---     0:00:36 cccc091
PN      9999904 job4.sh       Run      6  zzz/---     0:00:36 cccc090
...
PN      9999989 job89.sh      Run      1  zzz/---     0:00:11 ccca013
PN      9999990 job90.sh      Run      1  zzz/---     0:00:12 ccca010
-------------------------------------------------------------------------------
```

-c を指定しない場合は、数分ほど古い情報になる場合がありますが、GPU 数や jobtype などの情報も確認できます。

```
ccfep% jobinfo
-------------------------------------------------------------------------------
Queue   Job ID Name          Status CPUs User/Grp      Elaps Node/(Reason)
-------------------------------------------------------------------------------
PN(c)   9999900 job0.csh      Run     16  zzz/zz9    24:06:10 cccc047
PN(c)   9999901 job1.csh      Run     16  zzz/zz9    24:03:50 cccc003
PN(c)   9999902 job2.sh       Run      6  zzz/zz9     0:00:36 cccc091
PN(c)   9999903 job3.sh       Run      6  zzz/zz9     0:00:36 cccc091
PN(c)   9999904 job4.sh       Run      6  zzz/zz9     0:00:36 cccc090
...
PN(g)   9999989 job89.sh      Run    1+1  zzz/zz9     0:00:11 ccca013
PN(g)   9999990 job90.sh      Run    1+1  zzz/zz9     0:00:12 ccca010
-------------------------------------------------------------------------------
```

例: ジョブの作業ディレクトリを表示する

どこでジョブを実行したのかがわからなくなった場合には -w オプションを追加することで、
ジョブの作業ディレクトリ(PBS_O_WORKDIR)を表示するを表示することができます。

```
ccfep% jobinfo -w
-------------------------------------------------------------------------------
Queue   Job ID Name          Status Workdir
```

```
--------------------------------------------------------------------------------
PN    9999920 PN_12345.sh    Run    /home/users/zzz/gaussian/mol23
PN    9999921 PN_23456.sh    Run    /home/users/zzz/gaussian/mol74
...
```

-c とは併用できません。

ジョブの投入

下記の２つの方法があります

- jsubコマンドにバッチファイルを指定してジョブを投入する
- g09sub / g16subコマンドにgaussianのインプットファイルを指定してジョブを簡単に投入する（gaussian専用）

以下はjsubを使ってジョブを投入する方法です

ヘッダー部の書き方

ジョブの投入には、バッチスクリプトが必要です。その際、バッチコマンドをスクリプトの最初に記述しなければなりません。

- csh, bash (/bin/sh), zsh でのジョブ投入が可能です。
- どのシェルでも #PBS で始まる行は共通に使えます。
- バッチスクリプトのサンプルは、ccfep:/local/apl/lx/*アプリケーション名*/samples/にあります。

意味	ヘッダー部	重要度
第一行目	(csh の場合) #!/bin/csh -f (bash の場合) #!/bin/sh (zsh の場合) #!/bin/zsh	必須 (どれか一つ)
使用CPU数	#PBS -l select=[*Nnode*:]ncpus=*Ncore*:mpiprocs=*Nproc*:ompthreads=*Nthread*:jobtype=*Jobtype*[:ngpus=*Ngpu*]	必須
時間制限	#PBS -l walltime=*72:00:00*	必須
ジョブの開始前後にメールで通知	#PBS -m abe	オプション
ジョブの再実行抑止	#PBS -r n	オプション
バッチジョブ投入ディレクトリへの移動	cd ${PBS_O_WORKDIR}	推奨

- Nnode: 実ノード数
- Ncore: ノードあたりの確保するコア数
- Nproc: ノードあたりのプロセス数
- Nthread: プロセスあたりのスレッド数
- Jobtype: large, small, core, gpu, gpup, gpuvのいずれか
  - large: 18.8GB / core
  - small: 4.4GB / core
  - core: 18コア以下のジョブ
  - gpu, gpup, gpuv: GPUを使う計算
- Ngpu: 使用するGPUの数

2ノード80プロセス並列(MPI並列)する場合の「使用CPU数」行の書き方

```
#PBS -l select=2:ncpus=40:mpiprocs=40:ompthreads=1:jobtype=small
```

GPGPUを使用する場合の「使用CPU数」行の書き方

```
#PBS -l select=1:ncpus=6:mpiprocs=1:ompthreads=1:jobtype=gpu:ngpus=1
```

こちらのページにもいくつかサンプルがあります。

ジョブの投入コマンド

バッチスクリプトが準備できたら、下記のようにジョブを投入します。

```
ccfep% jsub -q (PN|PNR[0-9]) [-g XXX] [-W depend=(afterok|afterany):JOBID1[:JOBID2...]] script.csh
```

計算物質科学スパコン共用事業利用枠としてジョブ投入する場合は、-gオプションをつけます。（XXXは計算物質科学スパコン共用事業利用枠のグループ名）
ジョブの依存関係を-Wオプションで設定できます。正常終了後に実行させる場合はafterok、異常終了後でも実行させる場合はafteranyを指定します。依存関係のあるジョブIDをコロンで区切って指定します。
バッチスクリプトのサンプルは、ccfep:/local/apl/lx/*アプリケーション名*/samples/にあります。

一連のジョブ実行(ステップジョブ)

--step もしくは --stepany オプションをつけることで、比較的簡単にジョブを順番に実行することができます。

■ステップジョブ1: 複数のジョブを順番に実行する。前のジョブが異常終了した場合には以後のジョブは破棄。

```
ccfep% jsub -q (PN|PNR[0-9]) [-g XXX] --step [-W depend=(afterok|afterany):JOBID1[:JOBID2...]] script.csh
script2.csh ...
```

■ステップジョブ2: 複数のジョブを順番に実行する。前のジョブが終了したら次のジョブを実行。

```
ccfep% jsub -q (PN|PNR[0-9]) [-g XXX] --stepany [-W depend=(afterok|afterany):JOBID1[:JOBID2...]] script.csh
script2.csh ...
```

実行例:

```
ccfep% jsub -q PN --stepany job1.csh job2.csh job3.csh
```

ジョブの取消

あらかじめjobinfoコマンドで、取り消したいジョブのRequest IDを調べておきます。その後、

```
ccfep% jdel [-h cclx]  RequestID
```

とします。

ジョブのホールドとリリース

キュー待ち状態のジョブを実行されないように留めておく(ホールドする)ことができます。
あらかじめjobinfoコマンド等でジョブIDを調べておいた上で以下のコマンドを実行することでジョブをホールドできます。

```
ccfep% jhold [-h cclx]  RequestID
```

ホールドしたジョブを解放するには、

```
ccfep% jrls [-h cclx]  RequestID
```

とします。

実行済みジョブの情報取得

ジョブの終了日時、経過時間、並列化効率の情報をjobeffコマンドで得ることができます。

```
ccfep% jobeff -h (cclx|cck|ccpg|ccuv) [-d "last_n_day"] [-a] [-o  item1[,item2[,...]]]
```

表示される項目を-oオプションを使ってカスタマイズすることができます。*item*には次のキーワードを指定することができます。

- queue: キュー名
- jobid: ジョブID
- user: ユーザー名
- group: グループ名
- node: 計算に使われた最初のノード名
- Node: 計算に使われた全ノード名
- start: ジョブの開始時刻（YYYY/MM/DD HH:MM）
- type: ジョブタイプ
- Start: ジョブの開始時刻（YYYY/MM/DD HH:MM:SS）
- finish: ジョブの終了時刻（YYYY/MM/DD HH:MM）
- Finish: ジョブの終了時刻（YYYY/MM/DD HH:MM:SS）
- elaps: 経過時間
- cputime: 全CPU時間
- used_memory: 使用したメモリー量
- ncpu: 予約したCPU数
- ngpu: 予約したGPU数
- nproc: MPIのプロセス数
- nsmp: プロセスあたりのスレッド数
- peff: 並列化効率
- attention: 非効率なジョブかどうか
- command: ジョブ名
- exit_status: ジョブの終了コード
- point: ジョブが使用したCPU点数
- all: 主要なもの全て

■例1: 最近 10 日以内に終わったジョブの ID, 開始日時、終了日時、点数を表示

```
ccfep% jobeff -h cclx -d 10 -o jobid,user,start,finish,point
```

■例2: 2020年度に実行したジョブの ID, 終了日時、点数、実行ディレクトリを表示

```
ccfep% jobeff -h cclx -y 2020 -o jobid,finish,point,Workdir
```

■例3: 最近 2 日以内に終了したジョブの主要な情報を表示

(all は点数(point)を表示しません。点数を表示したい場合は他の例を参照下さい)

```
ccfep% jobeff -h cclx -d 2 -o all
```

ビルドと実行

ビルドのコマンド

システム	言語	非並列	自動並列	OpenMP	MPI
cclx (Intel)	Fortran	ifort	ifort -parallel	ifort -qopenmp	mpiifort
	C	icc	icc -parallel	icc -qopenmp	mpiicc
	C++	icpc	icpc -parallel	icpc -qopenmp	mpiicpc
cclx (PGI)	Fortran	pgfortran	pgfortran -Mconcur	pgfortran -mp	
	C	pgcc	pgcc -Mconcur	pgcc -mp	
	C++	pgcpp	pgcpp -Mconcur	pgcpp -mp	

各種ライブラリ、MPI環境等の導入状況については[パッケージプログラム一覧のページ]をご覧下さい。

並列プログラムの実行方法

cclx

ジョブスクリプト中、select 行で MPI プロセスの数を mpiprocs に、OpenMP のスレッド数を ompthreads に正しく設定して下さい。
例: 合計 12 CPU で 4 MPI プロセス、3 OpenMP スレッドの場合は ncpus=12:mpiprocs=4:ompthreads=3 となります。

センター側で導入したアプリケーションのサンプル(/local/apl/(システム名)/(アプリ名)/samples 以下に有り)も参考にして下さい。

■ジョブスクリプトでの OpenMP スレッド指定

jsub で実行する場合、ompthreads で指定した値が自動的に指定されます。
スクリプト内で OMP_NUM_THREADS 環境変数で手動指定しても問題ありません。
当然ですが、jsub で実行しない場合(フロントエンドノードでのテスト場合等)は OMP_NUM_THREADS 環境変数を設定する必要があります。

■MPIでのホスト指定

jsub で実行する場合、MPI が使うホストリストのファイル名が PBS_NODEFILE 環境変数に入ります。
センター側で導入している MPI 環境(Intel MPI, OpenMPI)ではこの環境変数を自動的に読み込むため、
いわゆる machine file 指定を省略して実行できます。

例: 4 MPI * 3 OpenMP ハイブリッド並列の例

```
#!/bin/sh
#PBS -l select=1:ncpus=12:mpiprocs=4:ompthreads=3:jobtype=core
#PBS -l walltime=24:00:00
cd $PBS_O_WORKDIR
mpirun -np 4 /some/where/my/program options
```

- OpenMP のスレッド数は ompthreads で指定された値が使われます(OMP_NUM_THREADS=3 を指定したと同義)
- センター側で導入した MPI 環境の場合 machine file は通常省略できます。
- PBS_NODEFILE 環境変数を machine file に指定しても動作は同じです。

開発支援ツール

一部コマンドライン版も使えますが、一般的にはX Window版の方が使いやすいです。

Intel Inspector

- メモリ／スレッドエラー検証ツール
- (GUI command) inspxe-gui
- (CUI command) inspxe-cl

Intel Vtune Amplifier XE

- 性能解析ツール
- (GUI command) amplxe-gui
- (CUI command) amplxe-cl

Allinea Forge

- デバッガー
- (GUI command) ddt

Environment Modules

2018年7月よりEnvironment Modules(moduleコマンド)の利用も可能です。詳細については、[こちらのページ]をご覧ください。

パッケージプログラム

- 各システムにインストールされている最新の一覧は、パッケージプログラム状態一覧で参照できます。
- バッチスクリプトのサンプルは、ccfep:/local/apl/*システム名*/*アプリケーション名*/samples/ディレクトリーを御覧下さい。
- アプリケーションの実体は、各システムの/local/apl/*システム名*/*アプリケーション名*/にあります。
- センターでビルドしたアプリケーションの構築法はアプリケーションライブラリーの構築方法を御覧下さい。

ソフトウェア導入の要望

下記の項目を全てご記入の上、rccs-admin[at]ims.ac.jp宛(迷惑メール対策のため、@を[at]に置換しています)に送信してください。
有料ソフトウェアの場合、導入できないことがあります。

- 導入を希望するソフトウェアの名前、バージョン
- ソフトウェアの概要と特長
- 共同利用システムに導入を希望する必要性
- 開発元のURL

その他のRCCS固有コマンド

バッチジョブ関連

jobinfo, jsub, jdel, jhold, jrls, jobeff については上に説明があります。

Gaussian専用ジョブ投入ツール

■g16の場合

```
ccfep% g16sub [-q "QUE_NAME"] [-j "jobtype"] [-g "XXX"] [-walltime "hh:mm:ss"] [-noedit] \
        [-rev "g16xxx"] [-np "ncpus"] [-ngpus "n"] [-mem "size"] [-save] [-mail] input_files
```

- g09を使うg09subコマンドも有ります。
- デフォルトのwalltimeは72時間に設定しています。ジョブの実行時間より多少多めに時間を設定してください。
- "g16sub"と入力すると各オプションの意味や使用例が表示されます。
- 元となるインプット中の %nproc や %cpu、%mem の情報は g16sub や g09sub に上書きされます。それらについては -np や -mem 等のオプションで指定して下さい
  - -noedit を使えば %mem の上書きを抑制することはできますが、非推奨です。
  - メモリ量については自動的に上限に近い値が指定されます。値を減らしたい時のような特殊ケースを除けば、ユーザ側で指定する必要はありません。

■例: g16c01 で制限時間 24 時間、12 コア利用ジョブ(my_gaussian_input.gjf は Gaussian のインプットファイル)

```
ccfep% g16sub -rev g16c01 -walltime 24:00:00 -np 12 my_gaussian_input.gjf
```

ジョブの実行開始時間予測(waitest)

ccfep では waitest コマンドで実行開始時間が予測できます。各ジョブが walltime 一杯まで実行されるという前提の元で実行開始時間を予測します。
そのため、細かな誤差を無視すれば waitest の出力は最悪予測になります。
一方でジョブ種類に依存する優先度や振り分けの設定や救済措置等により後続のジョブに順番を追い抜かれる場合もあるので、確実性はあまり高くありません。
jobinfo -s で出力されるキューの空き状態なども確認した上でご利用ください。

基本形

待ち状態にあるジョブの実行開始時間予測:

```
$ waitest [jobid1] ([jobid2] ...)
```

指定したジョブスクリプトを投入した場合の開始時間予測:

```
$ waitest -s [job script1] ([jobscript2] ...)
```

実行例1: キュー待ち中のジョブ ID を指定する場合

(ターミナル上では緑や赤の色付けはありません)

```
[user@ccfep2]$ waitest 4923556
Current Date   : 2019-10-15 14:32:30
2019-10-15 14:32:30 ...
2019-10-15 16:40:44 ...
2019-10-15 22:26:07 ...
2019-10-16 00:43:43 ...
2019-10-16 03:03:11 ...
2019-10-16 05:58:00 ...
2019-10-16 11:34:12 ...
Job 4923556 will run at 2019-10-16 13:03:11 on ccnn500,ccnn496,ccnn497,ccnn494,ccnn495,ccnn489.
Estimation completed.
```

実行例2: 実行前のジョブスクリプトを指定する場合

```
[user@ccfep2]$ waitest -s run_small_4N.sh run_small_6N.sh run_small_8N.sh
Current Date   : 2019-10-15 14:34:39
Job Mapping "run_small_4N.sh" -> jobid=1000000000
Job Mapping "run_small_6N.sh" -> jobid=1000000001
Job Mapping "run_small_8N.sh" -> jobid=1000000002
2019-10-15 14:34:39 ...
2019-10-15 16:40:52 ...
2019-10-15 22:26:15 ...
2019-10-16 00:43:51 ...
2019-10-16 03:03:18 ...
2019-10-16 05:58:08 ...
2019-10-16 11:34:10 ...
Job 1000000001 will run at 2019-10-16 13:03:18 on ccnn500,ccnn496,ccnn497,ccnn494,ccnn495,ccnn489.
2019-10-16 13:28:41 ...
2019-10-16 16:00:36 ...
2019-10-16 20:52:10 ...
```

```
2019-10-17 01:08:27 ...
Job 1000000002 will run at 2019-10-17 03:03:18 on
ccnn458,ccnn329,ccnn515,ccnn520,ccnn373,ccnn437,ccnn380,ccnn352.
2019-10-17 03:33:56 ...
2019-10-17 06:43:51 ...
2019-10-17 08:50:03 ...
2019-10-17 11:34:10 ...
2019-10-17 13:03:18 ...
2019-10-17 16:08:16 ...
2019-10-17 18:35:08 ...
2019-10-17 20:36:56 ...
2019-10-17 22:38:49 ...
Job 1000000000 will run at 2019-10-18 00:18:34 on ccnn760,ccnn789,ccnn791,ccnn787.
Estimation completed.
```

(優先度や振り分けの関係で、大規模ジョブが小規模ジョブより入りやすいことがあります。)

実行例3: 一般的なサイズのジョブについての予測情報

一般的なジョブの種類については定期的に予測を行っており、その結果は以下のコマンドで確認できます。

```
[user@ccfep2]$ waitest --showref
```

資源使用状況表示

```
ccfep% showlim (-cpu|-c|-disk|-d) [-m]
```

- -cpu|-c: 使用した使用点数と割当点数の表示
- -disk|-d: 使用しているディスク容量と上限容量の表示
- -m: 所属全メンバー毎の使用状況と上限値の表示

■例1: 自身及びグループの利用/割り当て CPU 点数の表示

```
ccfep% showlim -c
```

■例2: グループ全体及びグループの各メンバの利用/割り当て CPU 点数の表示

```
ccfep% showlim -c -m
```

■例3: グループ全体及びグループの各メンバの利用/割り当てディスク容量表示

```
ccfep% showlim -d -m
```

バッチスクリプト用ユーティリティーコマンド

コマンドの実行時間の制限

```
/local/apl/lx/ps_walltime -d duration -- command [arguments...]
```

- -d *duration*: コマンドを実行させたい時間を"-d 72:00:00"のような形式で記述します。
- 指定時間を過ぎるとcommandをkillします。

ジョブの統計情報の表示

```
/local/apl/lx/jobstatistic
```

- PBSのヘッダー行でジョブ終了時にメール送信を指示したときに含まれるジョブ統計情報相当を出力します。
- コマンド実行時までのジョブ統計情報です。

■出力項目

- resources_used.cpupercent: CPU利用効率。最大の値はスレッド数x100。複数ノード使用時には異常値を示す場合があります。
- resources_used.cput: 各CPUで実際に演算した時間の総和。
- resources_used.mem: 実際に使用したメモリー量。
- resources_used.ncpus: 使用したCPU数。
- resources_used.walltime: ジョブの実行時間。

演算ノード上のファイル操作

remshコマンドを使うと、演算ノードのramdiskのようにフロントエンド(ccfep)から直接アクセスできないファイルへアクセスできます。

```
remsh hostname command options
```

- hostname cccc???, cccca???, ccnn???, ccnf??? のようなホスト名です。
- command 実行するコマンド。ls, cat, cp, find のいずれかを指定できます。
- options コマンドのオプションです。

例: ユーザzzzによる演算ノードccnnXXXでのramdisk操作

```
remsh ccnnXXX ls /ramd/users/zzz
```

```
remsh ccnnXXX cat /ramd/users/zzz/99999/fort.10 | tail
```

ジョブを実行しているノード名はjobinfoコマンドより確認できます。

問い合わせ

https://ccportal.ims.ac.jp/contact
をご参照下さい。