計算機利用の手引き

■ [最終更新日時] 2019年7月18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('ccgpu', '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

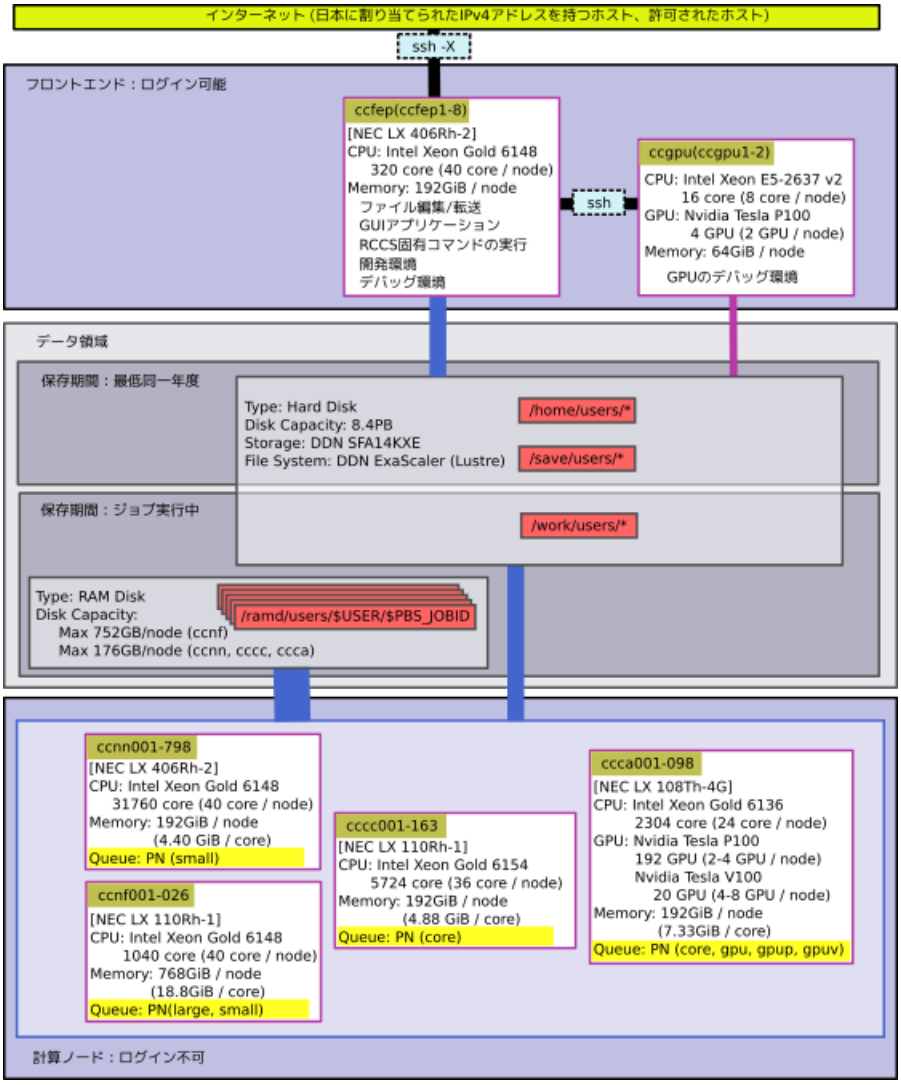
1. 専用ウェブページ(https://ccportal.ims.ac.jp/account/)をウェブブラウザーで開き、ユーザー名とパスワードを入力し、「ログイン」ボタンを押します。
2. 画面右上の「アカウント情報」を開きます。
3. 「編集」タブを押します。
4. パスワードを変更するは、現在のパスワードと新たに設定したいパスワードを入力します。
5. あらかじめ用意したsshの公開鍵をペーストなどして入力します。
6. 「保存」ボタンを押します。

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), ccgpu（2ノード）でビルドやデバッグができます。
- 'ccgpu'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。プログラムで使うメモリー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の違いは、センター側でデータのバックアップを取るか取らないかの違いです。
- /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))

- 会話処理のccfepはCPU時間でCPU点数が消費されます。
- 会話処理のccgpu1, ccgpu2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。

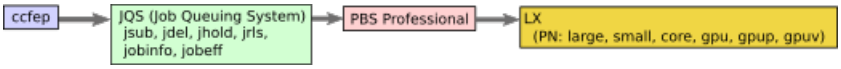
ユーザ別 リソース制限の設定／表示

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	4000/48 2560/32 1600/20 960/12 320/8	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3日以下で要求コア数が6-12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

システムで何本のジョブがどれだけのCPUを使っているかの一覧表示するには、

```
ccfep% jobinfo [-s] -h cclx
```

とします。キューで何本のジョブがどれだけのCPUを使っているかの一覧表示するには、

```
ccfep% jobinfo [-s] -q (PN|PNR[0-9])
```

とします。-sオプションは、省略できます。

システムのバッチジョブの詳細を知りたい場合は、

```
ccfep% jobinfo -l [-g|-a] -h cclx
```

とします。-gオプションをつけると同一グループのジョブも表示されます。-aオプションをつけると全ユーザーのジョブが表示されます。自分以外の情報は、暗号化されています。
キューのバッチジョブの詳細を知りたい場合は、

```
ccfep% jobinfo -l [-g|-a] -q (PN|PNR[0-9])
```

とします。ジョブの作業ディレクトリ(PBS_O_WORKDIR)を表示するには、

```
ccfep% jobinfo -w -q (PN|PNR[0-9])
```

とします。

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 be	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。

ジョブの取消

jobinfoコマンドで、取り消したいジョブのRequest IDを調べます。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

ジョブがキュー待ち状態のときに一時的に実行しないようにするためには

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

とします。
ジョブのホールド状態を解除するためには、

```
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

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	

MPIの実装

システム	MPIの種類
cclx	Intel MPI （MPI 3.0に準拠）

設定を読み込めば OpenMPI も利用可能です。

インストールされているライブラリー関数

システム	ライブラリー関数
cclx	intel MKL, Intel IPP, Intel TBB

並列プログラムの実行方法

システム	自動並列・OpenMP	MPI	ハイブリッド
cclx	setenv OMP_NUM_THREADS *4* *./a.out*	mpirun -np *4 ./a.out*	setenv OMP_NUM_THREADS *4* mpirun -np *8 ./a.out*

ハイブリッドとは、自動並列またはOpenMPとMPIを組み合わせた方法です。

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、ccadm[at]draco.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

RCCS固有コマンド

バッチジョブ関連

ジョブ状態表示

```
ccfep% jobinfo [-c] [-s] [-l|-m|-w [-g|-a]] [-n] -h cclx
```

もしくは

```
ccfep% jobinfo [-c] [-s] [-l|-m|-w [-g|-a]] [-n] -q (PN|PNR[0-9])
```

- -c　最新データを使って表示
- -s　サマリー表示
- -m　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
- -l　リスト表示
- -g　グループ内の全ユーザの情報を表示
- -a　全てのユーザの情報を表示
- -n　ノード別空きコア数を表示
- -h　対象システムを指定
- -q　対象キューを指定

ジョブ投入

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

ジョブ削除

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

ジョブのホールド

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

ジョブの解放

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

完了したジョブ情報取得

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

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。

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

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。