名古屋大学
教育研究用高性能コンピュータシステム
利用者マニュアル
（FX100,CX400）

2019年6月14日

本マニュアルは、名古屋大学情報基盤センター スーパーコンピュータシステムの利用手引です。

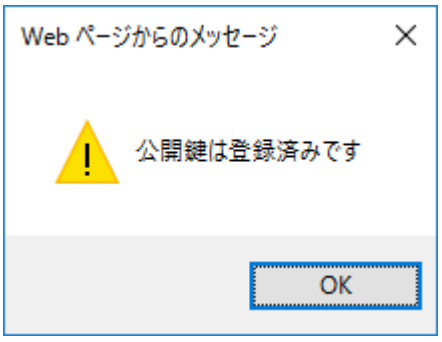
新スーパーコンピュータでは、利用方法が大きく変わっています。
別冊「FX10システムからFX100システムへの変更点について」に変更点等があります。
ご参考お願いいたします。なお、ご不明な点やご質問がございましたら、次の連絡先にお問い合わせください。

【問い合わせ先】
名古屋大学 情報連携統括本部情報推進部
情報基盤課共同利用担当（情報基盤センター）

メール： kyodo@itc.nagoya-u.ac.jp

よくあるご質問：

Q1：HPCPortal (https://portal.cc.nagoya-u.ac.jp/) にて公開鍵を登録しようとすると、以下のメッセージが出て登録できない。



A1：HPCPortal での公開鍵の登録は、一度のみ（一度に複数登録は可能）となっています。
すでに登録されていると再登録はできません。公開鍵登録済みの端末(a)で SSH にて、ログインノード(cx.cc.nagoya-u.ac.jp)にログインできる場合、以下の方法で追加登録できます。
１：公開鍵登録済みの端末(a)で SSH にて、ログインノード(cx.cc.nagoya-u.ac.jp)にログインします。
２：ホームディレクトリ /.ssh 配下の、ファイル名 authorized_keys に、追加したい端末(b)の公開鍵を追加します。
ー＞vi エディタなどで追加します。
３：公開鍵登録済みの端末(a)はそのままで、公開鍵を追加した端末(b)でログインします。

Q2：2015 年 7 月まで fx のログインノードに ssh できていたが、2015 年 9 月以降、できなくなった。
A2：fx のログインノードの OS がバージョンアップされました。
fx のログインノードの OS がバージョンアップされた為、お手元の LINUX 等のクライアント端末から、fx のログインノードに ssh しようとすると、次のメッセージが表示されログインができなくなる場合があります。

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

この時お使いの端末の .ssh ディレクトリ配下にある known_hosts ファイルから fx.cc.nagoya-u.ac.jp に関する行を削除して再度ログインを行って下さい。

次のコマンドでも削除できます。
コマンド：ssh-keygen –R fx.cc.nagoya-u.ac.jp

この操作を行っても接続ができない場合は次のコマンドを実行し再度ログインして下さい。
コマンド：ssh-add ~/.ssh/id_rsa

Q3：fx や cx のログインノードでコンパイル時に通常と比べて処理が遅い、応答が返ってこない。
A3：ログインノードのメモリ不足が原因です。対応策として資源の追加がありますが、すぐにはできないため、回避策をご案内します。
※ログインノードは計算ノードと違い、他者との「資源の共有」が発生します。その為、他のログインノードにログインのし直しをすることにより、回避できる可能性があります。
ログインノードは複数あり、メンテナンス等により入れ替わります。現在稼働中のログインノードの IP アドレスはログインノードにログイン頂き、以下の例のように、nslookup コマンドで ip アドレスを確認します。

```
[user@fx01 ~]$ nslookup    ＜ー入力
> fx.cc.nagoya-u.ac.jp       ＜ー入力　以下の６台が稼働中と分かる。
Server:              133.6.*.*
Address:             133.6.*.*#53

Name:     fx.cc.nagoya-u.ac.jp   ＜ー１台目
Address: 133.6.?.??1
Name:     fx.cc.nagoya-u.ac.jp   ＜ー２台目
Address: 133.6.?.??2
Name:     fx.cc.nagoya-u.ac.jp   ＜ー３台目
Address: 133.6.?.??3
Name:     fx.cc.nagoya-u.ac.jp   ＜ー４台目
Address: 133.6.?.??4
Name:     fx.cc.nagoya-u.ac.jp   ＜ー５台目
Address: 133.6.?.??5
Name:     fx.cc.nagoya-u.ac.jp   ＜ー６台目
Address: 133.6.?.??6
> exit                                ＜ー入力
[user@fx01 ~]$
```

確認した ip アドレスをひとつ指定して ssh で、ログインします。

目次

はじめに

本利用者マニュアルは、国立大学法人名古屋大学に導入の教育研究用高性能コンピュータシステム利用方法について説明した資料です。システムを利用する方は、必ずお読みください。

本利用者マニュアルの内容は、不定期に更新いたします。

本利用者マニュアルに記載しているシェルスクリプトやサンプルプログラムなどは、教育研究用高性能コンピュータシステム　ログインノードの以下のディレクトリに格納されていますので、併せてご利用ください。

/center/local/sample 配下

【サンプルの一覧を表示する方法】

```
[USERID@cx03 ~]$ sample
  OpenFOAM       adf              fftw       gaussian          intel      lsdyna    pyn
  _hdf5.20140618 amber            fx_script gromacs           lammps     namd      starccm+
  abaqus          cx_script       gamess    hdf5       lang_sample    phi_offload    vnode
[USEERID@cx03 ~]$
```

【サンプルのダウンロード例】

amber の内容をディレクトリ「new」を作成してダウンロード

```
[USERID@cx03 ~]$ sample    amber    new
```

1. 教育研究用高性能コンピュータシステムの概要

1.1 システム構成

教育研究用高性能コンピュータシステムは、FX100 用計算ノード群、CX 用計算ノード群、FX100 用ログインノード、CX 用ログインノード、ストレージシステム、管理ノード群から構成されるシステムです。

■ システム名称 :教育研究用高性能コンピュータシステム（フェーズⅡ）

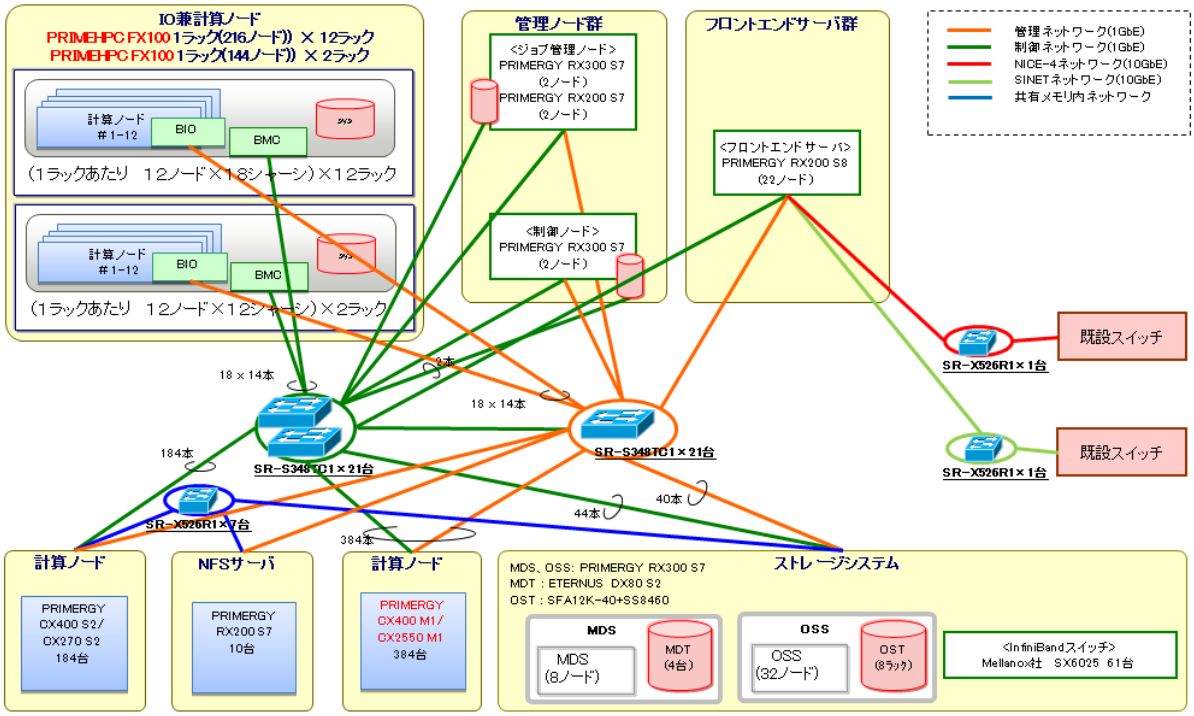


図 1-1 システム構成図

FX100 用計算ノード群は、富士通 PRIMEHPC FX100 14 ラックで構成され、総理論演算性能 2918TFLOPS、総主記憶容量 92.16TByte を有します。Tofu インターコネクト 2*1は SPARC64™ Xifx に統合され、ノード間通信バンド幅を低遅延でリンクあたり 12.5GB/s と高速化しています。

ストレージ環境は、共有ファイルシステム(FEFS)から構成されます。

CX 用計算ノード群は、富士通 PRIMERGY CX2550M1 及び PRIMERGY CX270 S2 で構成され、総理論演算性能 727.1TFLOPS、総主記憶容量 77.6TByte を有しています。

ストレージ環境は、FX100 と同様、共有ファイルシステム(FEFS)から構成されます。

---

*1 Tofu (Torus fusion) は、富士通の高速インターコネクトの呼称です。

共有ファイルシステムは、/home、/center、/large、/large2 から構成されており、各ユーザーのホームディレクトリやデータを格納するファイルシステムであり、全計算ノードおよびログインノードから参照可能です。利用可能容量は合計約 6PByte です。

システムへのアクセスは、ssh によるアクセスと HTTPS アクセス(プログラミング支援ツール)が可能です。ユーザーはログインノード上にて、プログラムの編集、コンパイル・リンクによる実行モジュールの作成、バッチジョブの操作、ジョブ実行結果の検証、デバッグ等の作業を行うことが可能です。

1.2 ハードウェア概要(FX100)

計算ノードを構成する富士通 PRIMEHPC FX100 は、HPC 分野に特化した、以下の特徴を持った計算システムであり、様々なテクニカル分野での利用が可能です。

1.2.1 SPARC64™ XIfx

SPARC64™XIfx は 2 つのコアメモリグループ(CMG)、Tofu2 コントローラ、PCI-Express コントローラなどから構成されています。1 つの CMG は 16 個のコア、1 個のアシスタントコア、17 コア間で共有される 12MB のレベル 2 キャッシュ、メモリコントローラーで構成され、2 つの CMG 間ではキャッシュー貫性が保たれます。半導体には 20nm テクノロジーを採用しています。各コアは IU（Instruction control Unit)、EU（Execution Unit)、SU（Storage Unit）の 3 つのユニットにわかれます。IU は命令のフェッチ、発行および完了を制御します。EU は 2 つの整数演算ユニット、2 つの整数演算兼アドレス計算ユニット、および 8 つの浮動小数点積和演算ユニット(FMA: Floating-point Multiply and Add)から構成され、整数演算、および浮動小数点演算命令を実行します。1 つの FMA は 1 サイクルあた 2 つの倍精度浮動小数点演算(加算と乗算)を実行可能です。各コアは 1 サイクルあたり 2 つの SIMD 演算命令を実行します。したがって各コアで 1 サイクルあたり 16 個、32 個の計算コア合計で 512 個の倍精度浮動小数点演算が実行可能となります。また、単精度浮動小数点の場合は 1 サイクルあたり 2 倍の演算が可能です。SU はロード・ストア命令を実行します。各コアは 64KB のレベル 1(L1)命令キャッシュとデータキャッシュをそれぞれ内蔵しています。

コア数	32 + 2 アシスタントコア
コアあたりスレッド数	1
L2 キャッシュ容量	24MiB
ピーク性能	> 1 Tflops
メモリ理論帯域	240GB/s x2(in/out)
インターコネクト理論帯域	125GB/s x2(in/out)
プロセステクノロジー	20nm CMOS
トランジスタ数	約 37 億 5000 万個

表 1-1 SPARC64™XIfx 諸元

1.2.2 Tofu インターコネクト 2

- インターコネクト・コントローラーICC

PRIMEHPC FX100 では前世代の Tofu インターコネクト(Tofu1)をベースに性能、機能を向上させた Tofu インターコネクト 2 を開発し、SPARC64™XIfx プロセッサに統合しました。ノード間通信バンド幅を低遅延でリンクあたり 12.5 GB/s と高速化しています。

ICC は PCI Express ルート・コンプレックスと Tofu インターコネクトを統合した LSI です。

- RDMA 通信

Tofu2 の RDMA 通信機能は、Tofu1 の Put および Get に加えて Atomic RMW(Atomic ReadModify Write)をサポートします。Tofu2 の Atomic RMW は、CPU の Atomic 演算に対し、相互に Atomicty を保障します。これによりプロセス並列とスレッド並列で資源を共有する処理において、排他制御オーバヘッドを削減します。

- 通信インターフェース

Tofu1 は送信時の遅延削減のため、通信コマンドを CPU レジスタから直接 RDMA エンジンに送る、ダイレクトディスクリプタ機能を備えていました。Tofu2 ではさらに、受信時の遅延を削減するため、受信データを L2 キャッシュメモリに直接書き込むキャッシュインジェクション機能を追加しました。

1.2.3 スケーラブル・高可用性 3 次元メッシュ/トーラス

6 次元メッシュ/トーラス・ネットワークでは、各次元の軸を X, Y, Z, A, B, C と呼び、X 軸・Y 軸は筐体間を、Z 軸・B 軸はシステムボード間を、A 軸・C 軸はシステムボード上のノード間を接続します。Z 軸は座標 0 に I/O ノード、座標 1 から 8 に計算ノードが配置されます。B 軸は、3 つのシステムボードをリング接続して冗長性を確保します。A 軸、B 軸、C 軸はそれぞれの長さが 2、3、2 の固定長であり、A 軸はメッシュ、B 軸はトーラス、C 軸はメッシュで接続されます。

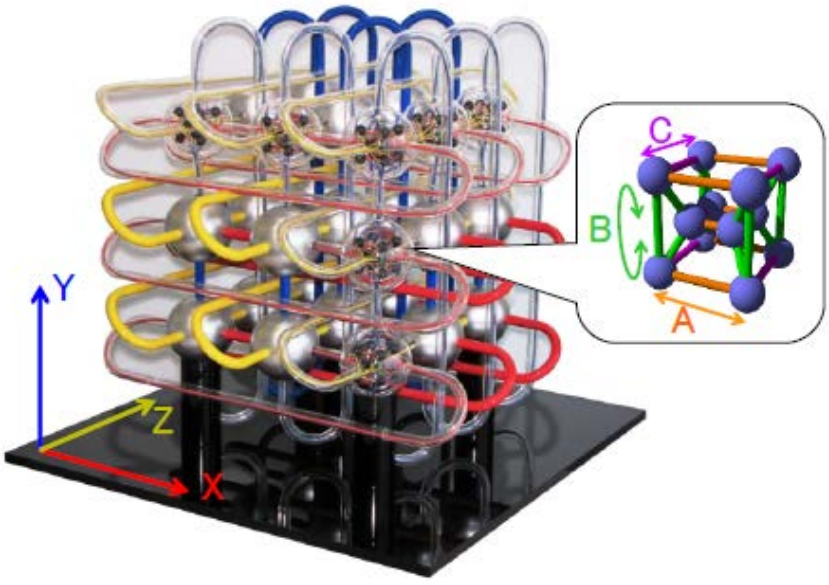


図 1-2 インターコネクトのトポロジーイメージ

Tofu インターコネクトは隣接通信を用いた通信パターンの最適化を容易にするため、ユーザーが指定する大きさの 1 次元／2 次元／3 次元トーラス空間をユーザービューとして提供します。ユーザー指定トーラス空間上の位置はランク番号で識別されます。3 次元トーラスが指定された場合、システムは XYZ の 1 軸と ABC の 1 軸の組合せによる 3 つの空間を形成します。そして、各空間で一筆書きの隣接関係を保証するようにランク番号を与えます。

1.3 ハードウェア概要(CX)

計算ノードを構成する富士通 PRIMERGY CX2550M1 及び PRIMERGY CX270 S2 は、HPC 分野に特化した、以下の特徴を持った計算システムであり、様々なテクニカル分野での利用が可能です。

システムの OS が Red Hat Enterprise Linux であるため、ISV アプリケーションが豊富にサポートされています。

表 1-2 ハードウェア概要

機種名	Fujitsu PRIMERGY CX400 S2/270 S2	Fujitsu PRIMERGY CX400 M1/2550 M1
OS	Red Hat Enterprise Linux6.4	Red Hat Enterprise Linux6.5
プロセッサ コア数	Intel IvyBridge(2.7GHz) E5-2697V2 12 コア	Intel Haswell(2.6GHz) Intel Xeon E5-2600 v3 processor family 14 コア
CPU/ノード L3 キャッシュ メモリバンド幅 ノード当りの理論演算性能(コア数) ノード当りのメモリ容量 総ノード数(総コア数) 総演算性能 総メモリ容量 コプロセッサ	2 30MB/CPU 119GB/s 518.4GFLOPS (24 コア) 128GiB 184(4,416 コア) 279.9TFLOPS 23TiB Xeon Phi3100 family(MIC)	2 35MB/CPU 136GB/s 1164.8GFLOPS (28 コア) 128GiB 384(10,752 コア) 447.2TFLOPS 48TiB

1.4 ソフトウェア構成

システムのソフトウェア環境を以下に示します。

FX100 は、計算ノード群とログインノードは異なるアーキテクチャであるため、ログインノードでは計算ノード群向けのクロスコンパイラ環境が利用可能です。

CX は、計算ノード群とログインノードは同等のアーキテクチャであるため、オウンコンパイラ環境が利用可能です。

表 1-3 システムソフトウェアー覧[FX100]

項目	計算ノード	ログインノード

OS	専用 OS(FX100 向け OS)	Red Hat Enterprise Linux
コンパイラ	富士通製コンパイラ Fortran コンパイラ C/C++ コンパイラ XPFortran コンパイラ	富士通製コンパイラ(クロスコンパイラ) Fortran コンパイラ C/C++ コンパイラ XPFortran コンパイラ
ライブラリ	富士通ライブラリ BLAS, LAPACK, ScaLAPACK, MPI, SSLII(Scientific Subroutine Library II), C-SSL II, SSL II/MPI	
ジョブ管理システム	富士通 Technical Computing Suite	

表 1-4 システムソフトウェアー覧[CX]

項目	計算ノード	ログインノード
OS	Red Hat Enterprise Linux	Red Hat Enterprise Linux
コンパイラ	富士通製コンパイラ Fortran コンパイラ C/C++ コンパイラ XPFortran コンパイラ Intel コンパイラ Fortran コンパイラ C/C++ コンパイラ	富士通製コンパイラ Fortran コンパイラ C/C++ コンパイラ XPFortran コンパイラ Intel コンパイラ Fortran コンパイラ C/C++ コンパイラ
ライブラリ	富士通ライブラリ BLAS, LAPACK, ScaLAPACK, MPI, SSLII(Scientific Subroutine Library II), C-SSL II, SSL II/MPI Intel ライブラリ MKL	
ジョブ管理システム	富士通 Technical Computing Suite	

ログインノードは、SSH によるログイン後、コマンドの対話的実行が可能であり、主にプログラムの作成・編集、実行モジュールのコンパイル/リンク、ジョブ投入を行います。ログインノードの資源は多くのユーザーで共有しますので重い処理は行わないようにしてください。ユーザーが行うジョブ操作は、ジョブ管理システムを通じて、計算ノードで行います。

バッチジョブは、投入してから実行されるまでに待ち時間がありますが、自分の順番が回ってきた際には計算ノードの資源を占有できます。

1.5 アカウントと認証方式

システムへのアクセスに使用するユーザー名は、申込み時に通知される利用者番号(ユーザー名)です。ログインノードへのアクセスは SSH(version2)をご利用ください。認証は公開鍵認証方式です。

1.6 ネットワークアクセス

ユーザーがシステムに対してアクセス可能なサーバは、ログインノードです。ログインノード(**fx.cc.nagoya-u.ac.jp, cx.cc.nagoya-u.ac.jp**)には、ssh にてアクセスします。

アクセス先は以下のとおりです。

表 1-5 アクセス先一覧

ホスト名(FQDN)	サービス	アクセス用途
fx.cc.nagoya-u.ac.jp	ssh https	• スーパーコンピューターシステム利用(仮想端末) • プログラミング支援ツール
cx.cc.nagoya-u.ac.jp	ssh https	• スーパーコンピューターシステム利用(仮想端末) • プログラミング支援ツール

1.7 システムへのログイン(Windows 環境)

Windows で使用できるターミナルソフトには PuTTY や Tera Term などがあります。PuTTY がもっとも鍵の扱いが容易なので、 PuTTY を推奨ターミナルソフトとし接続方法を説明します。

また、Cygwin を使用される方は UNIX 向けの解説をご覧ください。

PuTTY、TeraTerm は以下のサイトからダウンロードすることができます。

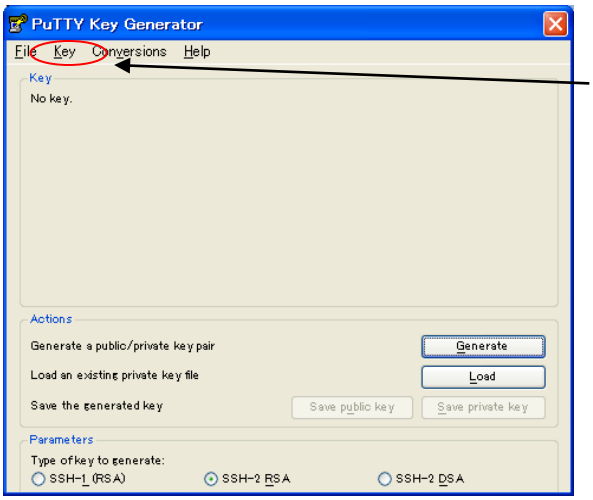
PuTTY: http://www.chiark.greenend.org.uk/~sgtatham/putty/

Tera Term: http://sourceforge.jp/projects/ttssh2/

1.7.1 鍵の作成

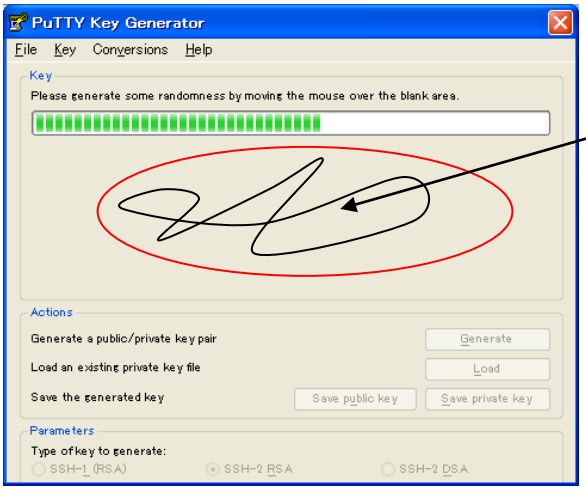
アクセス元端末(PC/WS)にて、秘密鍵/公開鍵ペアを作成します。

以下では PuTTY をインストールした際に付属する PuTTYGEN を用いた鍵の作成方法を示します。すでに鍵を作成済みの場合は、作業を行う必要はありません。



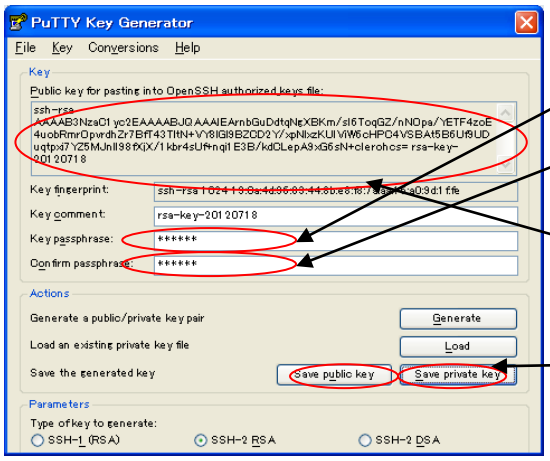
1. [Key]をクリック
2. 表示されるメニュー中の「Generate key pair」をクリック

図 1-3 仮想端末(PuTTY)での秘密鍵指定画面 1



3. 鍵を作成のための乱数を生成するため、マウスを動かす

図 1-4 仮想端末(PuTTY)での秘密鍵指定画面 2



4. パスフレーズを入力
5. 再度パスフレーズを入力
6. 表示された公開鍵情報を複写し、保存
7. [秘密鍵の保存]をクリック、ファイルを保存

図 1-5 仮想端末(PuTTY)での秘密/公開鍵作成

1.7.2 公開鍵登録

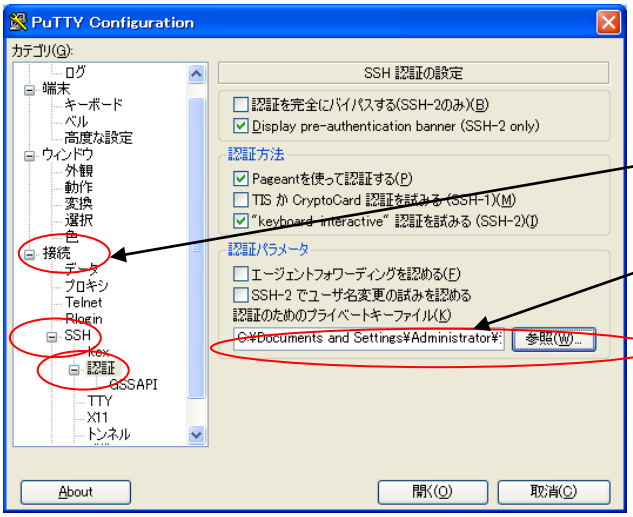
公開鍵の登録は、HPC ポータル(https://portal.cc.nagoya-u.ac.jp/)を利用してください。

HPC ポータルでの公開鍵の登録は、一度のみ（一度に複数登録は可能）可能となっています。

（すでに登録されていると再登録はできません。ご注意ください。）

1.7.3 ログイン

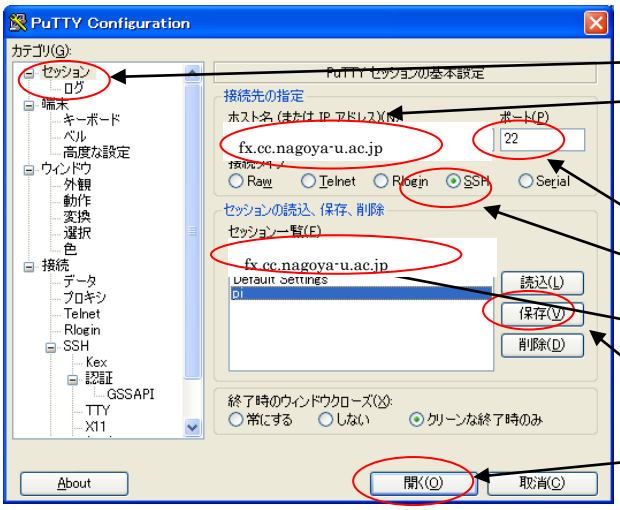
1. 仮想端末を起動して、秘密鍵ファイルを指定します。



PuTTY の場合

1. [接続] - [SSH] - [認証] メニューを選択
2. [参照] をクリックし、1.7.2 公開鍵登録で登録した公開鍵と対となる秘密鍵ファイルを指定

2. 仮想端末から FX100 スーパーコンピューターシステムにアクセスします。



1. セッション画面を開く
2. 下記情報を入力

[ホスト名]

(例)fx.cc.nagoya-u.ac.jp

[Port] 22

[接続タイプ] SSH

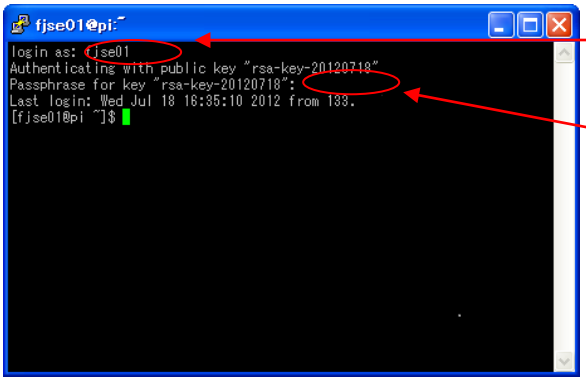
3. [セッション一覧] に session 名

(例.fx.nagoya-u.ac.jp)を入力

4. [保存] をクリック
5. [開く] をクリック

3. 初めてログインするとき、警告メッセージが表示されます。[はい]をクリックします。
次回以降のログインでは、このメッセージは表示されません。

4. ユーザーアカウントと公開鍵作成時のパスフレーズを入力します。



1. login as に
ユーザーアカウントを入力
2. Password に
公開鍵パスフレーズを入力

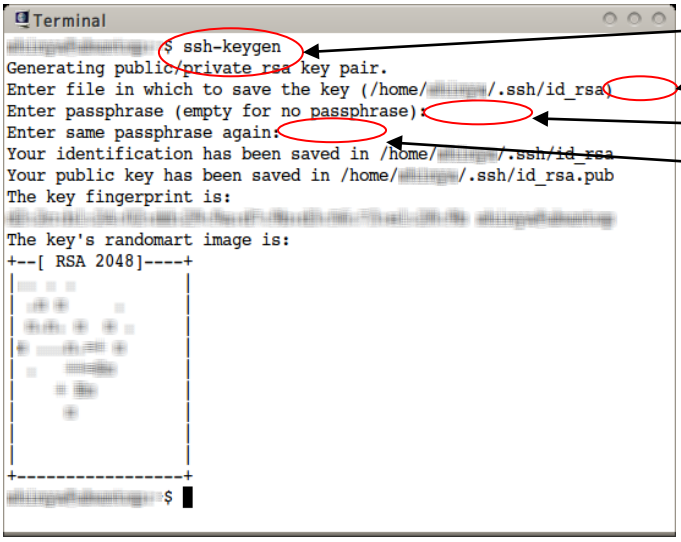
ログアウトは、ターミナルソフト上で "exit" もしくは "logout" と入力します。

1.8 システムへのログイン(UNIX 環境)

1.8.1 鍵の作成

アクセス元端末(PC/WS)にて ssh-keygen コマンドを実行し、秘密鍵/公開鍵ペアを作成します。すでに鍵を作成済みの場合は、作業を行う必要はありません。

■ UNIX/Linux: 端末エミュレータを起動して、ssh-keygen コマンドを実行します。



1. ssh-keygen コマンドを入力
2. リターンを入力**(注)**
3. パスフレーズを入力
4. 再度パスフレーズを入力

(注) ~/.ssh/id_rsa 以外のファイルとして保存する場合は、ssh コマンドで FX100 スーパーコンピューターシステムにアクセスする際、以下のように秘密鍵ファイルを指定してください。

例) `$ ssh -i 秘密鍵 -l<username> fx.cc.nagoya-u.ac.jp`

図 1-6 公開鍵ペアの作成

1.8.2 ログイン

UNIX 系 PC、WS や Windows 環境で Cygwin を使ってシステムへログインする場合は、ssh サービスを利用します。

```
% ssh -l username fx.cc.nagoya-u.ac.jp
The authenticity of host '
fx.cc.nagoya-u.ac.jp    (133.6.1.xxx)' can't be established.      初回ログイン時
   RSA key fingerprint is xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx    .のみ表示される。
   Are you sure you want to continue connecting (yes/no)? yes          ← yes を入力
   Warning: Permanently added '133.6.1.xxx' (RSA) to the
   list of known hosts.
Enter passphrase for key '/home/username/.ssh/id_rsa': +++++++++    ←公開鍵パスフレーズを入力
[username@fx01 :~]
```

1.9 ログイン環境

システムは、ログインシェルとして bash が登録されています。ログインシェルの変更はできません。

なお、ログイン時にシステムを利用するための環境設定が自動で設定されます。環境変数 PATH にパスを追加する際には、~/.bashrc.local を作成し PATH の最後に追加してください。PATH の先頭に追加した場合、システムを正常に使用できなくなる恐れがあります。

1.9.1 メール転送設定

ジョブ終了時などメールにて通知を受けることができます。通知を受けるメールアドレスは、ユーザー名@ジョブ投入ホスト名に設定されています。希望するメールアドレスで受信するためには、メール転送の設定(.forward)が必要です。メール転送の設定は、以下の通りです。

例) foo@foo.com に転送する場合

```
[username@fx01:~]$ vi .forward
foo@foo.com
```

メールサーバ(nucc)はログインすることができませんので、ユーザー登録申請受付窓口へご連絡ください。

2. システム環境

2.1 FEFS (Fujitsu Exabyte File System)の概要

FEFS(Fujitsu Exabyte File System) はLustreファイルシステムをベースに開発したファイルシステムで、数万規模のクライアントによるファイル利用を想定した大規模分散ファイルシステムです。Lustre の優れた技術を受け継ぐと共に、Lustre との互換性を維持しつつ、大規模システム向けに最大ファイルサイズ、最大ファイル数等の拡張を大規模システム向けに実施しています。

2.2 利用ファイルシステム

システムが提供するファイルシステム領域は以下のとおりです。

表 2-1 利用可能ファイル領域一覧

領域	領域名	実効容量	備考
共有ファイルシステム注1	/home	約 0.5PB	ホーム領域
	/center	約 1.0PB	ISV,OSS(ソフトウェア)領域
	/large	約 1.5PB	データ領域
	/large2	約 3.0PB	データ領域(2015.9.1 新規利用開始)

注 1：ホーム領域は Quota にて使用量を各ユーザー500GB に制限されています。

注 2：データ領域 /large と /large2 の作成方法は次のとおりです。

(1) /large の場合

コマンド：largedir （/large/利用者番号 のディレクトリが作成されます。）

(2) /large2 の場合

コマンド：largedir2 （/large2/利用者番号 のディレクトリが作成されます。）

2016.4.1 追記 /large と /large2 ともにディスク容量が不足しています。
/large は、10TB(10,000,000MB)以内
/large2 は、50TB(50,000,000MB)以内
でのご利用をお願いいたします。

各ノードからのファイルシステム領域利用状況は以下のとおりです。

表 2-2 ファイルシステム利用状況

領域	ログインノード	計算ノード
共有ファイルシステム	○	○

2.2.1 共有ファイルシステム

共有ファイルシステムは富士通製 FEFS で構成され、ユーザーのホーム領域やデータ領域として提供されます。ホーム領域の使用量は Quota にて 1 ユーザーあたり 500GB に制限されています。

共有ファイルシステムはログインノード、計算ノードから参照可能であり、主な使用目的は以下のとおりです。

- /home
  - ホーム領域
  - ソースプログラム/オブジェクトファイル/実行モジュールファイルの格納
  - 小容量データの格納
  - I/O 要求が少ないジョブ実行
- /center
  - ISV,OSS の格納
- /large、/large2
  - プログラム入出力データの格納
  - 大容量データの格納

2.3 コンパイラの種類

FX100 システムでは、ログインノードと計算ノードは異なるアーキテクチャです。そのため、ログインノード上でプログラムの実行モジュールを作成するためにクロスコンパイラ環境が整備されています。

CX2550 システムでは、ログインノードと計算ノードで異なるアーキテクチャですが、同じコンパイラが利用可能です。ただし、計算ノードの性能を最大限利用するため、かつ、ログインノードでコンパイルする場合、「-KCORE_AVX2」の指定が必要（効果はプログラムに依存）です。

CX270 システムでは、ログインノードと計算ノードは同等のアーキテクチャであるため、同じコンパイラが利用できます。

表 2-3 コンパイラ環境[FX100]

コンパイラ	ログインノード	計算ノード
クロスコンパイラ	○	×
オウンコンパイラ (ジョブ実行にて利用可)	×	○

表 2-4 コンパイラ環境[CX2550]

コンパイラ	ログインノード	計算ノード
オウンコンパイラ	○※1	○

※1　計算ノードの性能を最大限利用するには「-KCORE_AVX2」の指定が必要

表 2-5 コンパイラ環境[CX270]

コンパイラ	ログインノード	計算ノード
オウンコンパイラ	○	○

2.4 コンパイル／リンクの概要

コンパイル／リンクの書式とコマンド一覧は以下のとおりです。

```
コマンド [option] sourcefile [...]
```

表 2-6 コンパイル／リンクコマンド一覧(FX100)

	言語処理系	クロスコンパイラ注1	自動並列 注2	OpenMP 注2
非並列 (非 MPI)	Fortran90	frtpx	-Kparallel	-Kopenmp
	C	fccpx		
	C++	FCCpx		
並列 (MPI)	Fortran90	mpifrtpx		
	C	mpifccpx		
	C++	mpiFCCpx		
並列	XPFortran	xpfrtpx		

注 1: クロスコンパイラはログインノード上で利用可能です。

注 2: 自動並列、OpenMP オプションはデフォルトでは無効です。

表 2-7 コンパイル／リンクコマンド一覧(CX)

	言語処理系	オウンコンパイラ注1	自動並列 注2	OpenMP 注2	AVX2 命令注3
非並列 (非 MPI)	Fortran90	frt	-Kparallel	-Kopenmp	-KCORE_AVX2
	C	fcc			
	C++	FCC			
並列 (MPI)	Fortran90	mpifrt			
	C	mpifcc			
	C++	mpiFCC			
並列	XPFortran	xpfrt			

注 3: AVX2 命令はデフォルトでは無効です。
ログインノードでコンパイルし、CX2550 に対してジョブ投入する場合計算ノードの性能を最大限利用するには指定が必要です。
ただし、-KCORE_AVX2 を指定する場合、他のオプションよりも後ろで指定してください。他のオプションよりも前に指定した場合、-KCORE_AVX2 が無効になることがあります。

2.5 Fortran

Fortran コンパイラの利用方法を示します。

Fortran コンパイラは、以下の規格に準拠しています。

JIS X 3001-1:2009 プログラム言語 Fortran

ISO/IEC 1539-1:2004 Information technology - Programming languages - Fortran

OpenMP Application Program Interface Version 3.1 July 2011

2.5.1 Fortran コンパイル/リンク方法

FX100 用 Fortran コンパイラは frtpx コマンドを利用します。MPI ライブラリを使用する場合は、mpifrtpx コマンドを利用します。

[FX100]

例1) 逐次プログラムをコンパイル/リンクする。

```
$ frtpx sample.f90
```

例2) ノード内スレッド並列(自動並列)プログラムをコンパイル/リンクする。

```
$ frtpx -Kparallel sample.f90
```

例3) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

```
$ frtpx -Kopenmp sample.f90
```

例4) ノード内スレッド並列(自動並列+OpenMP)プログラムをコンパイル/リンクする。

```
$ frtpx -Kparallel,openmp sample.f90
```

例5) MPI 並列プログラムをコンパイル/リンクする。

```
$ mpifrtpx sample.f90
```

例6) ハイブリッド並列(スレッド(自動並列 or OpenMP)+MPI)プログラムをコンパイル/リンクする。

```
$ mpifrtpx -Kparallel,openmp sample.f90
```

CX 用 Fortran コンパイラは frt コマンドを利用します。MPI ライブラリを使用する場合は、mpifrt コマンドを利用します。（Intel コンパイラについては、9 章で説明しています）

[CX]

例1) 逐次プログラムをコンパイル/リンクする。

```
$ frt sample.f90
```

例2) ノード内スレッド並列(自動並列)プログラムをコンパイル/リンクする。

```
$ frt -Kparallel sample.f90
```

例3) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

```
$ frt -Kopenmp sample.f90
```

例4) ノード内スレッド並列(自動並列+OpenMP)プログラムをコンパイル/リンクする。

```
$ frt -Kparallel,openmp sample.f90
```

例5) MPI 並列プログラムをコンパイル/リンクする。

```
$ mpifrt sample.f90
```

例6) ハイブリッド並列(スレッド(自動並列 or OpenMP)+MPI)プログラムをコンパイル/リンクする。

```
$ mpifrt -Kparallel,openmp sample.f90
```

2.5.2 コンパイルオプション

Fortran の主なコンパイルオプションは以下のとおりです。詳細は man コマンドを参照してください。

表 2-8 Fortran コンパイルオプション

コンパイルオプション	説明
-c	オブジェクトファイルまで作成
-o *exe_file*	実行ファイル名/オブジェクトファイル名を exe_file に変更 実行ファイル名を省略した場合は a.out
-I *directory*	INCLUDE ファイルまたはモジュール情報ファイルを検索するディレクトリを指定
-Fixed	ソースプログラムが固定形式で記述されていることを指示 (デフォルトはファイル拡張子を参照して判断)
-Free	ソースプログラムが自由形式で記述されていることを指示 (デフォルトはファイル拡張子を参照して判断)
-X6	言語仕様で解釈の異なる部分を FORTRAN66 仕様と解釈
-X7	言語仕様で解釈の異なる部分を FORTRAN77 仕様と解釈
-X9	言語仕様で解釈の異なる部分を Fortran95 仕様と解釈
-X03	言語仕様で解釈の異なる部分を Fortran2003 仕様と解釈
-fw	w レベル(低度のエラー)および s レベル(重度のエラー)の診断メッセージのみを出力
-fs	s レベル(重度のエラー)の診断メッセージのみを出力
-f *msg_num*	msg_num にメッセージ番号を指定することにより、特定の診断メッセージの出力を抑止
-Nmaxserious=*maxnum*	コンパイル時に検出された重度のエラーメッセージの数が maxnum に達した場合にコンパイルを中止
-Haefosux	コンパイル時および実行時に引数の整合性、添字式、部分列式の値、未定義な変数の参照または配列式の形状適合などを検査
-NRtrap	実行時の組込み演算の診断メッセージの出力の指示と、浮動小数点演算の割込み事象の検出を指示
-Qt	詳細な最適化情報および統計情報を出力
-V	コンパイラのバージョン情報を出力

2.5.3 最適化オプション

Fortran のオプションとして FX100 は「-Kfast -g -Ntl_trt -X9 -NRnotrap」、CX は「-Kfast」を設定しています。設定オプション以外の最適化機能は、プログラムデータの特性によって効果がある場合とそうでない場合があり、実際に動作して検証する必要があります。推奨オプションを指定すると、関連して複数の最適化オプションが誘導して実行されます。FX の主な最適化オプションは「表 2-8-2 最適化オプション(Fortran) [FX100]」のとおりです。CX の主な最適化オプションは「表 2-8-2 最適化オプション(Fortran) [CX]」のとおりです。

最適化は演算結果に影響を与える場合があります。詳細は man コマンドを参照してください。

表 2-8-1 最適化オプション(Fortran) [FX100]

コンパイルオプション	説明
-O [0,1,2,3]	最適化のレベルを指定。-O の後の数字を省略した場合は -O3(デフォルト: -O2)
-Kdalign	オブジェクトが 8 バイト境界にあるものとして命令生成
-Kns	FPU を non-standard floating-point mode で初期化 (デフォルト: -Knons)
-Kmfunc	マルチ演算関数を使用する最適化を行うことを指示 (デフォルト: -Knomfunc)
-Keval	演算の評価方法を変更する最適化を行うことを指示 (デフォルト: -Knoeval)
-Kprefetch_conditional	if 構文や case 構文に含まれる配列データに対して、prefetch 命令を使用したオブジェクトを生成
-Kilfunc	一部の単精度及び倍精度の組込関数のインライン展開を指示 (デフォルト: -Knoilfunc)
-Kfp_contract	Floating-Point Multiply-Add/Subtract 演算命令を使用した最適化を行うかどうかを指示 (デフォルト: -Knofp_contract)
-Kfp_relaxed	浮動小数点除算または SQRT 関数について、逆数近似演算命令と Floating-Point Multiply-Add/Subtract 演算を指示 (デフォルト: -Knofp_relaxed)
-Kfast	ターゲットマシン上で高速に実行するオブジェクトプログラムを作成。 オプション-O3 -Kdalign, eval,fp_contract,fp_relaxed,ilfunc,mfunc,ns,omitfp,prefetch_conditional と等価
-Kregion_extension	パラレルリージョンの拡大を実施。-Kparallel オプションが有効な場合に指定可能
-Kparallel	自動並列を指定(デフォルト: -Knoparallel) -Kparallel オプションが有効な場合、-O2,-Kregion_extension オプションが誘導される
-Kvisimpact	-Kfast,parallel オプションを指定した場合と等価
-Kocl	最適化制御行を有効化 (デフォルト: -Knoocl)
-Kpreex	不変式の先行評価を実施 (デフォルト: -Knopreex)
-Kswp	ソフトウェアパイプライニングの最適化を行うことを指示 （デフォルト: -Knoswp）
-Kshortloop=N	回転数の小さいループ向けの最適化を適用(N は 2 から 10)
-Kstriping[=N]	ループストライピングの最適化を行うことを指示（デフォルト: -Knostriping）
-Karray_private	ループ内のプライベート化可能な配列に対して、プライベート化を実施。-Kparallel オプションが有効な場合に意味あり。(デフォルト: -Knoarray_private)
-Kauto	SAVE 属性を持つ変数および初期値をもつ変数を除く局所変数を、automatic 変数として扱い、スタックに割り付けつけるよう指示
-Ksimd[=1|2|auto]	SIMD 拡張命令を利用したオブジェクトを生成(デフォルト:-Ksimd=auto) -Ksimd=1 :SIMD 拡張命令を利用したオブジェクトを生成 -Ksimd=2 :-Ksimd=1 に加え、if 文などを含むループに対して、SIMD 拡張命令を利用したオブジェクトを生成 -Ksimd=auto: SIMD 化するかどうかをコンパイラが自動的に判断
-Kopenmp	OpenMP 仕様の指示文を有効化 (デフォルト: -Knoopenmp)
-Koptmsg[=1|2]	最適化状況をメッセージ出力(デフォルト: -Koptmsg=1) -Koptmsg=1 :実行結果に副作用を生じる可能性がある最適化をした事をメッセージ出力 -Koptmsg=2 :-Koptmsg=1 に加えて、自動並列化、SIMD 化、ループアンローリングなどの最適化機能が動作したことをメッセージ出力
-KXFILL[=N]	ループ内で書き込みのみ行う配列データについて、データメモリからロードすることなく、キャッシュ上に書き込み用のキャッシュラインを確保する命令(XFILL 命令)を生成することを指示(デフォルト: -KNOXFILL) -O2 オプション以上が有効な場合に指定可能

表 2-8-2 最適化オプション(Fortran) [CX]

コンパイルオプション	説明
-O [0,1,2,3]	最適化のレベルを指定。-O の後の数字を省略した場合は -O3(デフォルト: -O2)
-Kns	FPU を non-standard floating-point mode で初期化(デフォルト:-Knons)
-Kmfunc	マルチ演算関数を使用する最適化を行うことを指示(デフォルト:-Knomfunc)
-Keval	演算の評価方法を変更する最適化を行うことを指示(デフォルト:-Knoeval)
-Kprefetch_conditional	if 構文や case 構文に含まれる配列データに対して、prefetch 命令を使用したオブジェクトを生成
-Kfp_relaxed	浮動小数点除算または SQRT 関数について、逆数近似演算命令と Floating-Point Multiply-Add/Subtract 演算ををを指示(デフォルト:-Knofp_relaxed)
-Kfast	ターゲットマシン上で高速に実行するオブジェクトプログラムを作成。 -O3 -Keval,fp_relaxed,mfunc,ns,omitfp オプションの指定に加え、-KSSE2,SSE3, SSE4,AVX オプションを自動的に選択。
-Kregion_extension	パラレルリージョンの拡大を実施。-Kparallel オプションが有効な場合に指定可能
-Kparallel	自動並列を指定(デフォルト: -Knoparallel) -Kparallel オプションが有効な場合、-O2,-Kregion_extension オプションが指定可能
-Kocl	最適化制御行を指定(デフォルト:-Knoocl)
-Kpreex	不変式の先行評価を実施
-Karray_private	自動並列化を促進させるために、ループ内のプライベート化可能な配列に対して、プライベート化を実施。-Kparallel オプションが有効な場合に意味あり(デフォルト: -Knoarray_private)
-Kauto	SAVE 属性を持つ変数および初期値をもつ変数を除く局所変数を、automatic 変数として扱い、スタックに割り付けつけるよう指示スタックに割り付けるよう指示

コンパイルオプションを追加することにより、推奨オプションの最適化機能を制御することができます。オプションの指定例を示します。

(1) 結果の精度が異なる場合

frtpx/frt では-Kfast を指定すると演算評価方法を変更する最適化(-Keval)が誘導されるため、精度に敏感な計算に影響を及ぼす場合があります。その場合は-Knoeval 指定により変更を抑止することができます。コンパイルオプションは後に指定されたものが優先されるため、-Kfast の後に noeval を指定します。

(2) [FX100]

```
$ frtpx -Kfast,parallel,noeval sample.f90
```

[CX]

```
$ frt -Kfast,parallel,noeval sample.f90
```

(3) コンパイルが長時間になる場合

最適化オプションのレベルを下げます。

[FX100]

```
$ frtpx -Kfast,parallel -O2 sample.f90
```

[CX]

```
$ frt -Kfast,parallel -O2 sample.f90
```

2.5.4 環境変数

Fortran コンパイラは、環境変数 FORT90CPX(CX の場合は FORT90C)をコンパイルオプションに設定することができます。

FORT90CPX(CX の場合は FORT90C)に定義されたコンパイルオプションは、自動でコンパイラに渡されます。

環境変数やシステムで定義されたコンパイルオプションには、次の優先順位があります。

① 翻訳指示行(-Koptions 指定時のみ)
② 翻訳コマンドのオペランド
③ 環境変数 FORT90CPX(CX の場合は FORT90C)
④ プロフィルファイル(システムで設定された値)
※-Kfast -g -Ntl_trt -X9 -NRnotrap が設定されています。
⑤ 標準値

ログインノード上で、推奨オプションを環境変数 FORT90CPX(CX の場合は FORT90C)に設定する例を示します。

[FX100]

```
$ export FORT90CPX=-Kfast,parallel
```

[CX]

```
$ export FORT90C=-Kfast,parallel
```

有効になったコンパイルオプションは、-Q オプションにより確認することができます。

※sample.f90 をコンパイルした場合には「sample.lst」というファイルが生成されます。

[-Q オプション指定時の出力例 :(sample.lst)]

```
 Fujitsu Fortran Version 2.0.0  Thu Aug  6 12:35:02 2015

Compilation information
   Current directory : /center/w49942a
   Source file       : sample.f90

Option information
   Environment variable : -Kfast,parallel
   Command line options : -Q
   Effective options    : -fi -g0 -AE -Free -O3 -Q -X9
                          -x0
                          -Kadr44 -Knoauto -Knoautoobjstack -Knocalleralloc
                          -Kdalign -Keval -Knofed -Knofenv_access
                          -Kfp_contract -Kfp_relaxed -Kfsimple -Kilfunc
                          -Klargepage -Kloop_blocking -Kloop_fission
                          -Kloop_nofission_if -Kloop_fusion
                          -Kloop_interchange -Kloop_nopart_parallel
                          -Kloop_nopart_simd -Kloop_noversioning -Knonf -Kns
                          -Kmfunc=1 -Knoocl -Komitfp -Koptmsg=1 -Knopreex
                          -Kprefetch_conditional -Kprefetch_noindirect
                          -Kprefetch_sequential=auto -Kprefetch_nostride
                          -Kprefetch_cache_level=all-Kprefetch_noinfer
```

2.6 C/C++

C/C++コンパイラの利用方法を示します。

C/C++コンパイラは、以下の規格に準拠しています。

C JIS X 3010-1993(ISO/IEC 9899:1990) 、C JIS X 3010-2003(ISO/IEC 9899:1999)

C++(ISO/IEC 14882:2003)、C++(ISO/IEC 14882:2011)

OpenMP Application Program Interface Version 3.1 July 2011

2.6.1 C コンパイル/リンク方法

FX100 用 C コンパイラは fccpx コマンドを利用します。MPI ライブラリを使用する場合は、mpifccpx コマンドを利用します。

[FX100]

例1) 逐次プログラムをコンパイル/リンクする。

```
$ fccpx sample.c
```

例2) ノード内スレッド並列(自動並列)プログラムをコンパイル/リンクする。

```
$ fccpx -Kparallel sample.c
```

※ 2015 年 9 月 1 日以降、-Kparallel 設定を行った際の最適化レベルのデフォルト値が、-O0 から -O2 に変更になりました。その為、コマンドを実行すると、その旨を知らせるメッセージが表示されるようになりました。

例3) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

```
$ fccpx -Kopenmp sample.c
```

例4) ノード内スレッド並列(自動並列+OpenMP)プログラムをコンパイル/リンクする。

```
$ fccpx -Kparallel,openmp sample.c
```

例5) MPI 並列プログラムをコンパイル/リンクする。

```
$ mpifccpx sample.c
```

例6) ハイブリッド並列(スレッド(自動並列 or OpenMP)+MPI)プログラムをコンパイル/リンクする。

```
$ mpifccpx -Kparallel,openmp sample.c
```

CX 用 C コンパイラは fcc コマンドを利用します。MPI ライブラリを使用する場合は、mpifcc コマンドを利用します。

[CX]

例1) 逐次プログラムをコンパイル/リンクする。

```
$ fcc sample.c
```

例2) ノード内スレッド並列(自動並列)プログラムをコンパイル/リンクする。

```
$ fcc -Kparallel sample.c
```

※ 2015 年 4 月 3 日以降、-Kparallel 設定を行った際の最適化レベルのデフォルト値が、-O0 から -O2 に変更になりました。その為、コマンドを実行すると、その旨を知らせるメッセージが表示されるようになりました。

例3) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

```
$ fcc -Kopenmp sample.c
```

例4) ノード内スレッド並列(自動並列+OpenMP)プログラムをコンパイル/リンクする。

```
$ fcc -Kparallel,openmp sample.c
```

例5) MPI 並列プログラムをコンパイル/リンクする。

```
$ mpifcc sample.c
```

例6) ハイブリッド並列(スレッド(自動並列 or OpenMP)+MPI)プログラムをコンパイル/リンクする。

```
$ mpifcc -Kparallel,openmp sample.c
```

2.6.2 C++コンパイル/リンク方法

FX100 用 C++コンパイラは FCCpx コマンドを利用します。MPI ライブラリを使用する場合は、mpiFCCpx コマンドを利用します。

[FX100]

例1) 逐次プログラムをコンパイル/リンクする。

```
$ FCCpx sample.cc
```

例2) ノード内スレッド並列(自動並列)プログラムをコンパイル/リンクする。

```
$ FCCpx -Kparallel sample.cc
```

※ 2015 年 9 月 1 日以降、-Kparallel 設定を行った際の最適化レベルのデフォルト値が、-O0 から -O2 に変更になりました。その為、コマンドを実行すると、その旨を知らせるメッセージが表示されるようになりました。

例3) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

```
$ FCCpx -Kopenmp sample.cc
```

例4) ノード内スレッド並列(スレッド+OpenMP)プログラムをコンパイル/リンクする。

```
$ FCCpx -Kparallel,openmp sample.cc
```

例5) MPI 並列プログラムをコンパイル/リンクする。

```
$ mpiFCCpx sample.cc
```

例6) ハイブリッド並列(スレッド(自動並列 or OpenMP)+MPI)プログラムをコンパイル/リンクする。

```
$ mpiFCCpx -Kparallel,openmp sample.cc
```

CX 用 C++コンパイラは FCC コマンドを利用します。MPI ライブラリを使用する場合は、mpiFCC コマンドを利用します。

[CX]

例1) 逐次プログラムをコンパイル/リンクする。

```
$ FCC sample.cc
```

例2) ノード内スレッド並列(自動並列)プログラムをコンパイル/リンクする。

```
$ FCC -Kparallel sample.cc
```

※ 2015 年 4 月 3 日以降、-Kparallel 設定を行った際の最適化レベルのデフォルト値が、-O0 から -O2 に変更になりました。その為、コマンドを実行すると、その旨を知らせるメッセージが表示されるようになりました。

例3) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

```
$ FCC -Kopenmp sample.cc
```

例4) ノード内スレッド並列(スレッド+OpenMP)プログラムをコンパイル/リンクする。

```
$ FCC -Kparallel,openmp sample.cc
```

例5) MPI 並列プログラムをコンパイル/リンクする。

```
$ mpiFCC sample.cc
```

例6) ハイブリッド並列(スレッド(自動並列 orOpenMP)+MPI)プログラムをコンパイル/リンクする。

```
$ mpiFCC -Kparallel,openmp sample.cc
```

2.6.3 コンパイルオプション

C/C++コンパイラの主なコンパイルオプションは以下のとおりです。詳細は man コマンドを参照してください。

表 2-9 コンパイルオプション(C/C++)

コンパイルオプション	説明
-c	オブジェクトファイルを作成
-o *exe_file*	実行ファイル名/オブジェクトファイル名を exe_file に変更 実行ファイル名を省略した場合は a.out
-I *directory*	名前が / 以外で始まるヘッダの検索を、*directory* で指定されたディレクトリを先に検索し、その後、通常のディレクトリを検索するように変更指定
-V	コンパイラのバージョン情報を標準エラーに出力
-Xg	GNU C コンパイラ仕様の言語仕様に基づいてコンパイル。GNU の拡張仕様と C99 規格を同時に指定する場合は、-noansi オプションを同時に指定する必要がある。
-NRtrap	実行時の割込み事象を検出するか否かを指示
-Nsrc	ソースリストを出力
-Nsta	統計情報を出力

2.6.4 最適化オプション

C/C++の標準的なオプションとして FX100 は「-Kfast -g -Ntl_trt -Xa -NRnotrap」、CX は「-Kfast」を設定しています。設定オプション以外の最適化機能は、プログラムデータの特性によって効果がある場合とそうでない場合があり、実際に動作して検証する必要があります。推奨オプションを指定すると、関連して複数の最適化オプションが誘導して実行されます。FX の主な最適化オプションは「表

2-9-1 最適化オプション(C/C++) [FX100]」のとおりです。CX の主な最適化オプションは「表 2-9-1 最適化オプション(C/C++) [CX]」のとおりです。

最適化は演算結果に影響を与える場合があります。詳細は man コマンドを参照してください。

表 2-9-1 最適化オプション(C/C++) [FX100]

コンパイルオプション	説明
-O [0,1,2,3]	最適化のレベルを指定。-O の後の数字を省略した場合は –O2(デフォルト: -O0)
-Kdalign	オブジェクトが 8 バイト境界にあるものとして命令を生成
-Kns	FPUをnon-standard floating-point modeで初期化(デフォルト:-Knons)
-Kmfunc	マルチ演算関数を使用する最適化を行うことを指示(デフォルト:-Knomfunc)
-Klib	標準ライブラリ関数の動作を認識して、最適化を促進の是非を指示(デフォルト:-Knolib)
-Keval	演算の評価方法を変更する最適化を行うことを指示(デフォルト:-Knoeval)
-Krdconv	4 バイト符号付き整数のループ変数がオーバフローしないと仮定した最適化を促進させるか否かを指示
-Kprefetch_conditional	if 構文や case 構文に含まれる配列データに対して、prefetch 命令を使用したオブジェクトを生成
-Kilfunc	一部の単精度及び倍精度の組込関数のインライン展開を指示(デフォルト:-Knoilfunc)
-Kfp_contract	Floating-Point Multiply-Add/Subtract 演算命令を使用した最適化を行うかどうかを指示(デフォルト:-Knofp_contract)
-Kfp_relaxed	浮動小数点除算または SQRT 関数について、逆数近似演算命令と Floating-Point Multiply-Add/Subtract 演算ををを指示(デフォルト:-Knofp_relaxed)
-x	ソースプログラムで定義された全関数に対して、インライン展開を実施
-Kfast	ターゲットマシン上で高速に実行するオブジェクトプログラムを作成。オプション-O3 -Kdalign, eval,fast_matmul,fp_contract,fp_relaxed,ilfunc,lib,mfunc,ns,omitfp, prefetch_conditional,lrdconv -x と等価
-Kregion_extension	パラレルリージョンの拡大を実施。-Kparallel オプションが有効な場合に指定可能
-Kparallel	自動並列を指定(デフォルト: -Knoparallel) -Kfast オプションが有効な場合、-O2,-Kregion_extension,-mt オプションが誘導される
-Kvisimpact	-Kfast,parallel オプションを指定した場合と等価
-Kocl	最適化制御行を有効化(デフォルト: -Knoocl)
-Kpreex	不変式の先行評価を実施(デフォルト:-Knopreex)
-Karray_private	自動並列化を促進させるために、ループ内のプライベート化可能な配列に対して、プライベート化を実施。-Kparallel オプションが有効な場合に意味あり。(デフォルト: -Knoarray_private)
-Kopenmp	OpenMP C 仕様のディレクティブを有効化(デフォルト:-Knoopenmp)
-Ksimd[=1|2|auto|nosimd]	SIMD 拡張命令を利用したオブジェクトを生成(デフォルト: -Ksimd=auto) -Ksimd=1 :SIMD 拡張命令を利用したオブジェクトを生成。 -Ksimd=2 :-Ksimd=1 に加え、if 文などを含むループに対して、SIMD 拡張命令を利用したオブジェクトを生成 -Ksimd=auto: SIMD 化するかどうかをコンパイラが自動的に判断
-Koptmsg[=1|2|nooptmsg]	最適化状況をメッセージ出力(デフォルト: -Knooptmsg) -Koptmsg=1 :実行結果に副作用を生じる可能性がある最適化をした事をメッセージ出力 -Koptmsg=2 :-Koptmsg=1 に加えて、自動並列化、SIMD 化、ループアンローリングなどの最適化機能が動作したことをメッセージ出力
-Kswp	ソフトウェアパイプライニングの最適化を指示(デフォルト: -Knoswp)
-Kshortloop=N	回転数の小さいループ向けの最適化を適用(N は 2 から 10)
-Kstriping[=N]	ループストライピングの最適化を行うことを指示(デフォルト:-Knostriping)

-KXFILL[=N]	ループ内で書き込みのみ行う配列データについて、データメモリからロードすることなく、キャッシュ上に書き込み用のキャッシュラインを確保する命令(XFILL 命令)を生成することを指示(デフォルト: -KNOXFILL) -O2 オプション以上が有効な場合に指定可能

表 2-9-2 最適化オプション(C/C++) [CX]

コンパイルオプション	説明
-O [0,1,2,3]	最適化のレベルを指定。-O の後の数字を省略した場合は –O2(デフォルト: -O0)
-Kns	FPU を non-standard floating-point mode で初期化(デフォルト:-Knons)
-Kmfunc	マルチ演算関数を使用する最適化を行うことを指示(デフォルト:-Knomfunc)
-Klib	標準ライブラリ関数の動作を認識して、最適化を促進の是非を指示(デフォルト:-Knolib)
-Keval	演算の評価方法を変更する最適化を行うことを指示(デフォルト:-Knoeval)
-Krdconv	4 バイト符号付き整数のループ変数がオーバフローしないと仮定した最適化を促進させるか否かを指示
-Kprefetch_conditional	if 構文や case 構文に含まれる配列データに対して、prefetch 命令を使用したオブジェクトを生成
-Kfp_relaxed	浮動小数点除算または SQRT 関数について、逆数近似演算命令と Floating-Point Multiply-Add/Subtract 演算をを指示(デフォルト:-Knofp_relaxed)
-x	ソースプログラムで定義された全関数に対して、インライン展開を実施
-Kfast	ターゲットマシン上で高速に実行するオブジェクトプログラムを作成 -O3 -Keval,fast_mat-mul,fp_relaxed,lib,mfunc,ns,omitfp,rdconv,*sse* -x-と等価。(sse はマシンに合わせた最適化オプションを自動選択)
-Kparallel	自動並列を指定(デフォルト: -Knoparallel) -Kfast オプションが有効な場合、-O2,-Kregion_extension,-mt オプションが誘導される
-Kocl	最適化制御行を有効化(デフォルト: -Knoocl)
-Kpreex	不変式の先行評価を実施(デフォルト:-Knopreex)
-Karray_private	自動並列化を促進させるために、ループ内のプライベート化可能な配列に対して、プライベート化を実施。-Kparallel オプションが有効な場合に意味あり。(デフォルト: -Knoarray_private)
-Kopenmp	OpenMP C 仕様のディレクティブを有効化(デフォルト:-Knoopenmp)
-Ksimd[=1|2|nosimd]	SIMD 拡張命令を利用したオブジェクトを生成(デフォルト: -Ksimd=1) -Ksimd=1 :SIMD 拡張命令を利用したオブジェクトを生成。 -Ksimd=2 :-Ksimd=1 に加え、if 文などを含むループに対して、SIMD 拡張命令を利用したオブジェクトを生成
-Koptmsg[=1|2|nooptmsg]	最適化状況をメッセージ出力(デフォルト: -Knooptmsg) -Koptmsg=1 :実行結果に副作用を生じる可能性がある最適化をした事をメッセージ出力 -Koptmsg=2 :-Koptmsg=1 に加えて、自動並列化、SIMD 化、ループアンローリングなどの最適化機能が動作したことをメッセージ出力
-Kswp	ソフトウェアパイプライニングの最適化を指示(デフォルト: -Knoswp)
-Kstriping[=N]	ループストライピングの最適化を行うことを指示(デフォルト:-Knostriping)

2.6.5 環境変数(C コンパイラ)

C コンパイラが利用する環境変数を示します。

(1) 環境変数: fccpx_ENV(CX の場合は fcc_ENV)

環境変数 fccpx_ENV(CX の場合は fcc_ENV)にコンパイルオプションを設定することができます。

fccpx_ENV に定義されたコンパイルオプションは、自動でコンパイラに渡されます。

環境変数やシステムで定義されたコンパイルオプションには、次の優先順位があります。

① 翻訳コマンドのオペランド
② 環境変数 fccpx_ENV(CX の場合は fcc_ENV)
③ プロフィルファイル(システムで設定された値)
※-Kfast -g -Ntl_trt -Xa -NRnotrap が設定されています。
④ 標準値

ログインノード上で推奨オプションを環境変数 fccpx_ENV(CX の場合は fcc_ENV)に設定する例を示します。

[FX100]

```
$ export fccpx_ENV=-Kfast,parallel
```

[CX]

```
$ export fcc_ENV=-Kfast,parallel
```

有効になったコンパイルオプションは、-Nsta オプションにより確認することができます。
※sample.c をコンパイルした場合には、統計情報が標準出力に出力されます。
[-Nsta オプション指定時の出力例]

```
Fujitsu C/C++ Version 2.0.0     Thu Aug    6 13:57:02 2015
Statistics information
   Option information
      Environment variable : -Kfast,parallel
      Command line options : -Nsta
      Effective options       : -noansi -g0 -mt -Qy -Xa -x- -O3 -Ka1 -Kadr44 -Knoalias_const
                                -Knoarray_private -Kconst -Kdalign -Knodynamic_iteration -Keval
                                -Kfast_matmul -Knofconst -Knofed -Knofenv_access -Kfp_contract
                                -Kfp_relaxed -Kfsimple -KGREG_APPLI -Kilfunc -Knoipo -Klargepage
                                -Klib -Kloop_blocking -Kloop_fission -Kloop_nofission_if
                                -Kloop_fusion -Kloop_interchange -Kloop_nopart_parallel
                                -Kloop_nopart_simd -Kloop_noversioning -Klooptype=f -Knomemalias
                                -Kmfunc=1 -Knonf -Kns -Knoocl -Komitfp -Knoopenmp -Knooptmsg
                                -Kparallel -Kparallel_nofp_precision -Knopreex
                                -Kprefetch_cache_level=all -Kprefetch_conditional
                                -Kprefetch_noindirect -Kprefetch_noinfer
                                -Kprefetch_sequential=auto -Kprefetch_nostride
                                -Kprefetch_nostrong -Kprefetch_strong_L2 -Krdconv -Kreduction
                                -Kregion_extension -Krestp=restrict -Knoshortloop -Ksimd=auto
                                -Knostriping -Kswp -Kunroll -Knouxsimd -KNOXFILL
                                -Ncancel_overtime_compilation -Nnoexceptions -Nnofjcex
                                -Nnohook_func -Nnohook_time -Nline -Nquickdbg=noheapchk
                                -Nquickdbg=nosubchk -NRnotrap -Nrt_notune -Nsetvalue=noheap
                                -Nsetvalue=nostack -Nsetvalue=noscalar -Nsetvalue=noarray
                                -Nsetvalue=nostruct -Nsta -Nuse_rodata
```

(2) 環境変数: TMPDIR
fccpx コマンド(CX の場合は fcc コマンド)が使用するテンポラリディレクトリを変更することができます。

2.6.6 環境変数(C++コンパイラ)

C++コンパイラが利用する環境変数を示します。

(1) 環境変数: FCCpx_ENV(CX の場合は FCC_ENV)
環境変数 FCCpx_ENV(CX の場合は FCC_ENV)にコンパイルオプションを設定することができ

ます。

FCCpx_ENV に定義されたコンパイルオプションは、自動でコンパイラに渡されます。

環境変数やシステムで定義されたコンパイルオプションには、次の優先順位があります。

① 翻訳コマンドのオペランド

② 環境変数 FCCpx_ENV(CX の場合は FCC_ENV)

③ プロフィルファイル(システムで設定された値)

※-Kfast -g -Ntl_trt -Xa -NRnotrap が設定されています。

④ 標準値

ログインノード上で推奨オプションを環境変数 FCCpx_ENV(CX の場合は FCC_ENV)に設定する例を示します。

```
$ export FCCpx_ENV=-Kfast,parallel
```

有効になったコンパイルオプションは、-Nsta オプションにより確認することができます。

※sample.cc をコンパイルした場合には、統計情報が標準出力に出力されます。

[-Nsta オプション指定時の出力例]

```
Fujitsu C/C++ Version 2.0.0     Thu Aug    6 14:59:32 2015
Statistics information
   Option information
      Environment variable : -Kfast,parallel
      Command line options : -Nsta
      Effective options       : -g0 -mt -Qy -Xa -std=c++03 -x- -O3 -Ka1 -Kadr44 -Knoalias_const
                                -Knoarray_private -Kdalign -Knodynamic_iteration -Keval
                                -Kfast_matmul -Knofed -Knofenv_access -Kfp_contract -Kfp_relaxed
                                -Kfsimple -KGREG_APPLI -Kilfunc -Klargepage -Klib -Kloop_blocking
                                -Kloop_fission -Kloop_nofission_if -Kloop_fusion
                                -Kloop_interchange -Kloop_nopart_parallel -Kloop_nopart_simd
                                -Kloop_noversioning -Klooptype=f -Knomemalias -Kmfunc=1 -Knonf
                                -Kns -Knoocl -Komitfp -Knoopenmp -Knooptmsg -Kparallel
                                -Kparallel_nofp_precision -Knopreex -Kprefetch_cache_level=all
                                -Kprefetch_conditional -Kprefetch_noindirect -Kprefetch_noinfer
                                -Kprefetch_sequential=auto -Kprefetch_nostride
                                -Kprefetch_nostrong -Kprefetch_strong_L2 -Krdconv -Kreduction
                                -Kregion_extension -Kremove_inlinefunction -Knorestp
                                -Knoshortloop -Ksimd=auto -Knostriping -Knostl_fast_new -Kswp
                                -Kunroll -Knouxsimd -KNOXFILL -Ncancel_overtime_compilation
                                -Nexceptions -Nnofjcex -Nnohook_func -Nnohook_time -Nline
                                -Nquickdbg=noheapchk -Nquickdbg=nosubchk -NRnotrap -Nrt_notune
```

⑵ 環境変数: TMPDIR

FCCpx コマンド(CX の場合は FCC コマンド)が使用するテンポラリディレクトリを変更することができます。

2.7 XPFortran

XPFortran コンパイラの利用方法を示します。

2.7.1 コンパイル/リンク方法

FX100 用 XPFortran コンパイラは xpfrtpx コマンドを利用します。

[FX100]

```
$ xpfrtpx sample.f
```

CX 用 XPFortran コンパイラは xpfrt コマンドを利用します。

[CX]

```
$ xpfrt sample.f
```

2.7.2 特長

一つの配列データを各ノードの主記憶上に分散して配置でき、各ノードの主記憶上から一つの配列データとしてアクセス可能です。詳細については、「XPFortran 使用手引書」1.2.2 グローバル空間をご参照ください。

2.7.3 留意事項

xpfrtpx コマンドを使用することにより、XPFortran プログラムのトランスレートが行われます。
その際、以下の形式の出力ファイルがカレントディレクトリに生成されます。
(出力ファイルのサフィックス：.mpi.f90、.mpi.f95、.mpi.f03)

(例) % xpfrtpx sampp.f90　　→　sampp.mpi.f90 が作成される。

2.8 数値計算ライブラリ

FX100 向け数値計算ライブラリとして BLAS/LAPACK/ScaLAPACK ならびに SSLII/C-SSLII が利用可能です。これらのライブラリは、SPARC64™Xlfx 向けチューニングを実施しています。

また、上記のライブラリについては、CX 用にも提供されています。

富士通 C/C++コンパイラにて数学ライブラリを使用する場合、数学ライブラリの製品マニュアルに記載されている注意事項も合わせてご参照ください。

2.8.1 BLAS/LAPACK/ScaLAPACK

Fortran/C/C++コンパイラから BLAS/LAPACK/ScaLAPACK を利用可能です。

表 2-10 BLAS/LAPACK/ScaLAPACK 概要

ライブラリ名	説明
BLAS	ベクトル演算や行列演算ライブラリ - Level3 全ルーチン、Level2 重要ルーチンでスレッド並列ルーチンを提供
LAPACK	線形代数ライブラリ - 重要ルーチンでスレッド並列ルーチンを提供
ScaLAPACK	線形代数メッセージパッシング並列ライブラリ - ScaLAPACK2.0.2 の追加機能を提供

コンパイル時に指定するオプションは以下のとおりです。

表 2-11 BLAS/LAPACK/ScaLAPACK オプション一覧

利用ライブラリ	並列性	指定オプション	備考
BLAS	逐次	-SSL2	
	スレッド並列	-SSL2BLAMP	
LAPACK	逐次	-SSL2	
	スレッド並列	-SSL2BLAMP	
ScaLAPACK	MPI 並列	-SCALAPACK	逐次版 BLAS/LAPACK をリンクする場合は -SSL2 を、スレッド並列版 BLAS, LAPACK をリンクする場合には-SSL2BLAMP を指定する

[FX100]

例1) 逐次版 BLAS/LAPACK を利用する。

```
$ frtpx -SSL2 sample.f
```

例2) スレッド並列版 BLAS/LAPACK を利用する。

```
$ frtpx -Kopenmp -SSL2BLAMP sample.f
```

例3) ScaLAPACK を利用する(逐次版 BLAS/LAPACK をリンク)。

```
$ mpifrtpx -SCALAPACK -SSL2 sample.f
```

例4) ScaLAPACK を利用する(スレッド並列版 BLAS/LAPACK をリンク)。

```
$ mpifrtpx -Kopenmp -SCALAPACK -SSL2BLAMP sample.f
```

[CX]

例1) 逐次版 BLAS/LAPACK を利用する。

```
$ frt -SSL2 sample.f
```

例2) スレッド並列版 BLAS/LAPACK を利用する。

```
$ frt -Kopenmp -SSL2BLAMP sample.f
```

例3) ScaLAPACK を利用する(逐次版 BLAS/LAPACK をリンク)。

```
$ mpifrt -SCALAPACK -SSL2 sample.f
```

例4) ScaLAPACK を利用する(スレッド並列版 BLAS/LAPACK をリンク)。

```
$ mpifrt -Kopenmp -SCALAPACK -SSL2BLAMP sample.f
```

2.8.2 SSL II(Scientific Subroutine LibraryII)系 数学ライブラリ

Fortran/C/C++コンパイラから SSL II ライブラリを利用可能です。また C/C++コンパイラ向けに C-SSLII ライブラリが利用可能です。

表 2-12 SSL II 系 数学ライブラリ概要

ライブラリ名	説明
SSL-II	スレッドセーフな逐次計算向けの数値計算ライブラリ 10 分野(線形計算、固有値固有ベクトル、非線形計算、極値問題、補間・近似、変換、数値微積分、微分方程式、特殊関数、疑似乱数)のサブルーチン等
SSL-II スレッド並列機能	並列効果の見込める重要機能にSMP向け並列処理に適合したインターフェースで並列数値計算アルゴリズム 線形計算(連立1次方程式の直接解法および反復解法、逆行列、固有値問題等)、フーリエ変換、擬似乱数など
C-SSLII	Fortran 版 SSL II の逐次機能サブセットを C 言語インターフェースで利用可能 スレッドセーフな逐次機能
C-SSLII スレッド並列機能	Fortran 版 SSL II スレッド並列機能のサブセットを C 言語インターフェースで利用可能
SSL II/MPI	富士通独自仕様で、MPI で並列化された 3 次元フーリエ変換ルーチン
高速 4 倍精度基本演算ライブラリ	4 倍精度の値を double-double 形式で表現し、高速に演算を行うライブラリ

コンパイル時に指定するオプションは以下のとおりです。SSL II(C-SSLII)ライブラリは、逐次機能とスレッド並列機能を持ちますが、サブルーチン名が異なるため、どちらも混在して利用可能です。

表 2-13 SSL II 系オプション一覧

利用ライブラリ	並列性	指定オプション	備考
SSL II C-SSL II	逐次	-SSL2	逐次版 BLAS/LAPACK をリンクする場合は-SSL2 を、スレッド並列版 BLAS, LAPACK をリンクする場合には-SSL2BLAMP を指定する
	スレッド並列	-SSL2BLAMP	
SSL II/MPI	MPI 並列	-SSL2MPI	同時に-SSL2 または -SSL2BLAMP を指定する

[FX100]

例1) 逐次版 SSL II を利用する。

```
$ frtpx -SSL2 sample.f
```

例2) スレッド並列版 SSL II を利用する。

```
$ frtpx -Kopenmp -SSL2BLAMP sample.f
```

例3) 逐次版 C-SSL II を利用する。

```
$ fccpx -Kopenmp -SSL2BLAMP sample.c
```

例4) SSL II/MPI を利用する。

```
$ mpifrtpx -Kopenmp -SSL2MPI -SSL2 sample.f
```

[CX]

例1) 逐次版 SSL II を利用する。

```
$ frt -SSL2 sample.f
```

例2) スレッド並列版 SSL II を利用する。

```
$ frt -Kopenmp -SSL2BLAMP sample.f
```

例3) 逐次版 C-SSL II を利用する。

```
$ fcc -Kopenmp -SSL2BLAMP sample.c
```

例4) SSL II/MPI を利用する。

```
$ mpifrt -Kopenmp -SSL2MPI -SSL2 sample.f
```

2.9 実行時環境変数

Fortran/C/C++プログラムにおいて、実行時に指定可能な主な環境変数について説明します。

表 2-14 実行時環境変数

環境変数	説明
PARALLEL	自動並列によりスレッド並列化されたプログラムを実行する場合は、環境変数 PARALLEL にスレッド数を指定します。省略時は、ジョブが利用可能なコア数(1 ノード 1 プロセスの場合 16)が使用されます。
OMP_NUM_THREADS	OpenMP によりスレッド並列化されたプログラムを実行する場合は、環境変数 OMP_NUM_THREADS にスレッド数を指定します。 省略時は、ジョブが利用可能なコア数(1 ノード 1 プロセスの場合 16)が使用されます。
THREAD_STACK_SIZE	スレッド毎のスタック領域の大きさを K バイト単位で指定します。 省略時は、ulimit -s の値 (unlimited) が使用されます。環境変数 OMP_STACKSIZE が指定されている場合、大きい方の指定値がスタック領域の大きさの値になります。

2.10 エンディアン変換

エンディアンとは、多バイトの数値をメモリに格納する際の方式のことをいいます。例えば 1234 という数値を 1 バイト目に 12、2 バイト目に 34 を格納する方法をビッグエンディアンといいます。逆に 1 バイト目に 34、2 バイト目に 12 を格納する方法をリトルエンディアンといいます。

FX100 システムの計算ノードは、ビッグエンディアンを採用しています。

実行時オプション（-Wl,Tu_no）(※u_no:装置番号)を指定することで、書式なし入出力でリトルエンディアンデータファイルを入出力できます。

※装置番号：入出力文に特定の番号を指定することで、存在しているファイルまたは新たに存在するファイルを
を結びつけ入出力することができます。

(1) 実行時オプションは、環境変数(FORT90L)で指定するか、または、実行可能モジュールの引数として指定します。

(2) -Wl,-T のみ指定すると、書式なし入出力とする装置番号の全てがリトルエンディアンの入出力となります。-Wl,-T で装置番号を指定した場合、指定した装置番号に対して有効となります。

エンディアン変換の指定例を示します。次の例では、装置番号 10 について、書式なし入出力をリトルエンディアンデータとしています。

```
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                    リソースグループ指定
#PJM -L "node=1:mesh"                        ノード数の指定(1 次元形状)
#PJM -L "elapse=10:00"                       経過時間指定
#PJM -j
#------ program execution ------#
export FORT90L=-Wl,-T10                      環境変数の指定
./a.out
```

図 2-1 環境変数(FORT90L)による指定例

```
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                    リソースグループ指定
#PJM -L "node=1"                             ノード数の指定(1 次元形状)
#PJM -L "elapse=10:00"                       経過時間指定
#PJM -j
#------ program execution ------#
./a.out -Wl,-T10
```

図 2-2 引数による指定例

なお、fcvendianpx コマンドで、エンディアンの変換を行うことも可能です。
詳細は「Fortran 使用手引書　付録 C エンディアン変換コマンド」をご参照ください。

```
$ fcvendianpx 入力ファイル 出力ファイル データ型
(例)
$ fcvendianpx infile outfile 8
```

2.11 ２ＧＢを超えるファイル出力時の留意点

Fortran プログラムにおいて、実行時に 2GB を超える出力を行う場合は、以下の実行時オプションを指定してください。

```
$ export FORT90L=-Wl,-Lu
```

3. ジョブ実行

3.1 ジョブシステム概要

システムの全ジョブは、ジョブ管理システムにより実行が制御されます。ユーザーはジョブ開始時に必要なリソース名と CPU 数、経過時間などを指定し、ジョブ管理システムに対してジョブ実行を指示します。
システムで利用可能なジョブはバッチジョブです。(「表 3-1 ジョブの種類」参照)
バッチジョブは、CPU やメモリなどの計算に必要なリソースが排他的に割り当てられます。

表 3-1 ジョブの種類

システム	ジョブ形式	計算ノード数	用途
FX100	バッチジョブ/ 会話型ジョブ	2880	バッチ形式でジョブを実行する。 会話型形式でジョブを実行する。 ノードダウンなどの異常発生時、ジョブの再実行が可能。
CX	バッチジョブ	384 184	バッチ形式でジョブを実行する。 ノードダウンなどの異常発生時、ジョブの再実行が可能。 vSMP 環境では実行不可。

※バッチジョブは投入形式によって、2 種類に分類されます。
※システムダウンなどでジョブが異常終了した場合に再実行を行わないようにするには、pjsub --norestart オプションを付加します。デフォルトは、--restart です。

表 3-2 バッチジョブの種類

バッチジョブ種別	用途	投入形式
通常ジョブ	スクリプト単位でジョブを実行する。	「3.4.1 バッチジョブ投入」参照
ステップジョブ	投入した複数のジョブを 1 つのまとまりとして扱い、その中で実行順序、依存関係をもつジョブ	「3.4.2 ステップジョブ投入」参照
会話型ジョブ	コマンドラインでジョブを実行する。	「3.4.1 バッチジョブ投入」参照

ユーザーがジョブ操作に用いるコマンドは、以下のとおりです。

表 3-3 バッチジョブ操作コマンド一覧

機能	コマンド名
ジョブ投入	pjsub
会話型ジョブ投入	pjsub --interact
ジョブ参照	pjstat
ジョブ削除	pjdel
ジョブ保留	pjhold
ジョブ解除	pjrls

3.2 ジョブ実行リソース

3.2.1 リソースグループ

ジョブ管理システムは、リソースグループという単位で計算ノードを管理します。バッチジョブを投入する場合、ユーザーはジョブを実行するためのリソースグループを指定します。指定可能なリソースグループは以下のとおりです。

表 3-4-1 FX100 システム リソースグループ（2015.9.1 更新）

TOFU2 によるノード間通信に 2 レーンを用いたジョブクラスです。

リソースグループ名（キュー名）	最大ノード数	最大CPU コア数	最大経過時間		最大メモリ容量	割当方法※2		備考
			標準値	制限値		Tofu	離散	
fx-interactive	4	128	1 時間	24 時間	28GiB ×4	不可	可	会話型ジョブ
fx-debug	32	1,024	1 時間	1 時間	28GiB ×32			デバッグ用
fx-small	16	512	24 時間	168 時間	28GiB ×16			
fx-middle	96	3,072	24 時間	72 時間	28GiB ×96	可		
fx-large	192	6,144	24 時間	72 時間	28GiB ×192			
fx-xlarge	864	27,648	24 時間	24 時間	28GiB ×864			
fx-special※1	2592	82,944	unlimited		28GiB × 2592			事前予約制

備考）ユーザープログラムが使用可能なメモリ容量はノードあたり 28GiB です。

※1） 大規模ジョブ「fx-special」クラスをご利用したい場合は、下記の連絡先にご相談ください。

【連絡先】電話：052-789-4372 (内線：4372) Web（メール）：https://qa.icts.nagoya-u.ac.jp/

※2） 割り当てのデフォルトは離散となっています。（2016.1.13 更新）

表 3-5-1-1 FX100 システム（ノード間通信強化型）リソースグループ（2017.4.11 更新）

TOFU2 によるノード間通信に 4 レーンを用いてリンクバンド幅性能を強化したジョブクラス（試行）です。

リソースグループ名（キュー名）	最大ノード数	最大CPUコア数	最大経過時間		最大メモリ容量※2	割当方法※3		備考
			標準値	制限値		Tofu	離散	
fx4-small※1	12	384	24 時間	48 時間	28GiB × 12	不可	可	ノード間通信 4 レーン

※1）全ノード（2880 ノード）の内、72 ノードがこのリソースグループに割当てられています。

※2）ユーザープログラムが使用可能な最大メモリ容量はノードあたり 28GiB です。

※3）割り当てのデフォルトは離散となっています。

表 3-6-1-2 FX100 システム（実行優先度強化型）リソースグループ（2019.4.2 更新）

実行優先度強化型リソースグループ（試行）で、TOFU2 によるノード間通信に 2 レーンを用いたジョブクラスです。

リソースグループ名（キュー名）	最大ノード数	最大CPUコア数	最大経過時間		最大メモリ容量※2	割当方法※3		備考
			標準値	制限値		Tofu	離散	
fx-middle2※1	96	3,072	24 時間	72 時間	28GiB × 96	可	可	ノード間通信 2レーン

※1）他のジョブクラスよりも優先して実行されますが、実行には経過時間 1 秒につき 通常の倍（0.004 ポイントに使用ノード数を乗じて得たポイント数）のポイントが必要です。

※2）ユーザープログラムが使用可能な最大メモリ容量はノードあたり 28GiB です。

※3）割り当てのデフォルトは離散となっています。

表 3-4-2 CX400/2550 リソースグループ(2016.4.1 更新)

リソースグループ名（キュー名）	最大ノード数	最大CPU コア数	最大経過時間		最大メモリ容量	備考
			標準値	制限値		
cx-debug	4	112	1 時間	1 時間	112GiB × 4	デバッグ用
cx-share※1	1/2	14	24 時間	168 時間	56GiB×1	ノード共有
cx-small	8	224	24 時間	168 時間	112GiB × 8	
cx-middle	32	896	24 時間	72 時間	112GiB × 32	
cx-large	128	3,584	24 時間	72 時間	112GiB × 128	
cx-special※2	384	10,752	unlimited		112GiB × 384	事前予約制

備考）ユーザープログラムが使用可能なメモリ容量はノードあたり 112GiB です。

※1）1 ノードを 2 件のジョブで共有します。1 CPU（14 コア）、64GB のメモリを使ってジョブが実行されます。課金は、1 ノード占有した場合と同様です。このサービスを利用する場合は、ジョブ投入時に次の指定を行ってください。

```
#!/bin/sh
#PJM -L "rscgrp=cx-share"
#PJM -L "vnode=1"
#PJM -L "vnode-core=14"
```

※2） 大規模ジョブ「cx-special」クラスをご利用したい場合は、下記の連絡先にご相談ください。
【連絡先】電話：052-789-4372 (内線：4372)　Web（メール）：https://qa.icts.nagoya-u.ac.jp/

表 3-7-2-1 CX400/2550 システム（実行優先度強化型）リソースグループ（2019.4.2 更新）

実行優先度強化型リソースグループ（試行）です。

リソースグループ名（キュー名）	最大ノード数	最大 CPU コア数	最大経過時間		最大メモリ容量	備考
			標準値	制限値		
cx-middle2※1	32	896	24 時間	72 時間	112GiB×32	

※1）他のジョブクラスよりも優先して実行されますが、実行には経過時間 1 秒につき 通常の倍（0.004 ポイントに使用ノード数を乗じて得たポイント数）のポイントが必要です。

備考）ユーザープログラムが使用可能なメモリ容量はノードあたり 112GiB です。

表 3-4-3 CX400/270 リソースグループ

リソースグループ名（キュー名）	最大ノード数	最大 CPU コア数	最大 Phi 数	最大経過時間		最大メモリ容量	備考
				標準値	制限値		
cx2-debug	4	96	4	1 時間	1 時間	112GiB×4	デバック用
cx2-single	1	24	1	24 時間	336 時間	112GiB×1	
cx2-small	8	192	8	24 時間	72 時間	112GiB×8	
cx2-middle	32	768	32	24 時間	72 時間	112GiB×32	
cx2-special※1	150	3,600	150	unlimited		112GiB×150	事前予約制

備考）ユーザープログラムが使用可能なメモリ容量はノードあたり 112GiB です。

※1） 大規模ジョブ「cx2-special」クラスをご利用したい場合は、下記の連絡先にご相談ください。
【連絡先】電話：052-789-4372 (内線：4372)　Web（メール）：https://qa.icts.nagoya-u.ac.jp/

3.3 ジョブ投入オプション

ジョブ投入時は、ジョブの実行に応じて、3 つのオプションを指定します。CX はノードの配置オプションが指定可能です。

3.3.1 基本オプション

ジョブに指定する基本オプションは以下のとおりです。

表 3-5 ジョブ投入基本オプション

オプション名	説明
--fs <*filesystem*>[,<*filesystem*>]	ジョブ実行時に利用するファイルシステムを指定
-g <*groupname*>	ジョブ実行時にジョブプロセスが所属するグループを指定
-j	ジョブの標準エラー出力を標準出力へ出力
-L	ジョブ資源に関するオプションを指定
--mail-list	メールの送信先を指定
-m	メール通知を指定
b	ジョブ開始時にメール通知を指定
e	ジョブ終了時にメール通知を指定
r	ジョブ再実行時にメール通知を指定
--mpi	MPI プログラムの動作について指定 詳細は「3.3.6 MPI オプション」を参照
-N <*JOBNAME*>	ジョブ名を指定
-o <*filename*>	標準出力を指定されたファイルに出力
--restart	2016.4.1 追記 障害発生時ジョブを再実行する（デフォルトです） 再実行しないようにするには、 --norestart　としてください。
-step	ステップジョブを投入します。
jid=<*jobid*>	関連付けるジョブ ID を指定
sn=<*stepno*>	ステップ番号を指定
sd=<*form*>	依存関係式を指定
-X	ジョブ投入時の環境変数を計算ノードに引き継ぐ

3.3.2 ジョブ資源オプション [FX]

FX についてジョブが利用する資源に関する主要オプションは以下のとおりです。-L オプションに続けて利用資源を指定します。

表 3-6 FX ジョブ資源オプション

オプション名	説明
-L	ジョブ実行に必要な資源の上限値を指定
elapse=<*limit*>	経過時間を指定([[time:]minute:]second で指定)
node-mem=<*limit*>	ノード単位の使用メモリ制限を指定
rscgrp=<*rsrgrp*>	投入するリソースグループ名を指定
proc-core=<*limit*>	プロセス単位の最大コアファイルサイズリミットを指定
proc-data=<*limit*>	プロセス単位の最大データセグメントサイズリミットを指定
proc-stack=<*limit*>	プロセス単位の最大スタックセグメントサイズリミットを指定

3.3.3 ノード形状の指定 [FX]

ノード形状、割り当て方法は-L オプションの node パラメータで指定します。

表 3-7 FX100 ノード形状オプション

オプション名	説明
-L node	ノード数およびノード形状の指定

	<*shape*>	1 次元指定の場合は node=N1 2 次元指定の場合は node=N1xN2 3 次元指定の場合は node=N1xN2xN3
	:torus または :mesh または :noncont	ジョブがノード専有ジョブの場合、ノードの割り当て方法(トーラスモード、メッシュモード、離散割り当て)を指定できます。 torus は、Tofu 単位(12 ノード)で計算機資源をジョブに割り当てるトーラスモードを意味します。 mesh は、ノード単位で計算機資源をジョブに割り当てるメッシュモードを意味します。 noncont は、ノード単位で計算機資源をジョブに割り当てる離散割り当てを意味します。 省略時は、ジョブ ACL 機能で定義されているデフォルト値に従います。

3.3.4 ノード単位または Tofu 単位でのノード割り当て[FX]

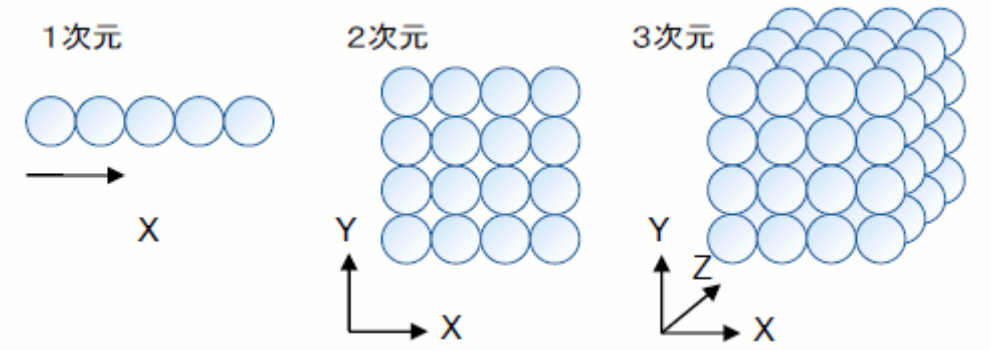
ノード専有ジョブに対する、ノード単位または Tofu 単位でのノード割り当てに関して以下を指定できます。

・割り当てるノードの形状とノード数

・MPI プログラムを実行する場合、ノード割り当てのルール

割り当てるノードは、仮想的な 1 次元、2 次元、または 3 次元の空間に配置される形状として指定します。

図 3-1 ノードの形状（イメージ）



ノードの割り当て方法には、トーラスモード、メッシュモード、および離散割り当ての 3 種類があります。

表 3-8 ノード割り当て方法

機能	コマンド名
トーラスモード	ノードの最小割り当て単位は Tofu 単位(12 ノード) です。 割り当てられるノードは、Tofu 座標上で隣接するノードが選択されます。
メッシュモード	ノードの最小割り当て単位は 1 ノードです。 割り当てられるノードは、Tofu 座標上で隣接するノードが選択されます。

離散割り当て	ノードの最小割り当て単位は 1 ノードです。 割り当てられるノードは、できるだけ Tofu 座標上で隣接するように選択されます。 以下の場合は隣接しないノードが選択されます。 ・隣接する空きノードがない場合 ・隣接しないノードを選択することでジョブの実行開始を早められる場合

3.3.5 ジョブ資源オプション [CX]

CX についてジョブが利用する資源に関する主要オプションは以下のとおりです。-L オプションに続けて利用資源を指定します。また、-P オプションでノード配置を指定できます。

表 3-9 CX ジョブ資源オプション

オプション名		説明
-L		ジョブ実行に必要な資源の上限値を指定
	elapse=<*limit*>	経過時間を指定([[time:]minute:]second で指定)
	vnode=<*share*>	vnode 数の指定
	vnode-core=<*share*>	vnode-core 数の指定※1
	rscgrp=<*rsrgrp*>	投入するリソースグループ名を指定
-P		ノード配置の各種パラメタを指定
	“vn-policy=abs-unpack”	各ノードに強制的に 1 プロセスずつ配置
	“vn-policy=unpack”	可能な限り各ノードに分散してプロセスを配置
	“vn-policy=abs-pack”	各ノードにプロセスを強制的に配置
	“vn-policy=pack”	可能な限りプロセスを少ないノードに配置（デフォルト）

※1）cx2550 の場合、1 ノードあたりのコア数が 28 のため、1,14,28 のコア数指定を推奨します。
cx270 の場合、1 ノードあたりのコア数が 24 のため、1,12,24 のコア数指定を推奨します。

3.3.6 MPI オプション

MPI ジョブを実行する際に指定するオプションは以下のとおりです。--mpi オプションに続けて MPI 実行時の動作を指定します。

表 3-10 MPI オプション

オプション名		説明
--mpi		MPI ジョブの各種パラメタを指定
	proc=*num*	静的に起動する最大プロセス数を指定(フラット MPI の場合は指定必須)
	rank-map-bynode[=*rankmap*]	ノードに 1 プロセス生成すると、次のノードへ移動し、ラウンドロビンでランクを割り付ける(rank-map-bychip と排他)
	rank-map-bychip[:*rankmap*]	ノードに [proc÷shape の node 数](小数点以下切り上げ)のプロセスを生成すると、次のノードへ移動し、ランクを割り付ける(rank-map-bynode と排他)
	rank-map-hostfile=<*filename*>	filename に従って生成するプロセスのランクを割り付ける

3.4 バッチジョブ投入(pjsub コマンド)

バッチジョブを実行するためには、実行するプログラムとは別に「ジョブスクリプト」を作成し、利用するジョブクラス、実行時間、CPU 数などの資源や実行形式を記載したオプションを記述した上で、実行するプログラムを記載します。ユーザーはジョブスクリプトを pjsub コマンドで投入し、実行を待ちます。投入されたジョブはスケジューラにより自動で実行開始、完了が制御されます。

3.4.1 バッチジョブ投入

バッチジョブを投入する場合、pjsub コマンドの引数にバッチジョブとして実行するスクリプトファイルを指定します。

```
pjsub [option] [script-file]
```

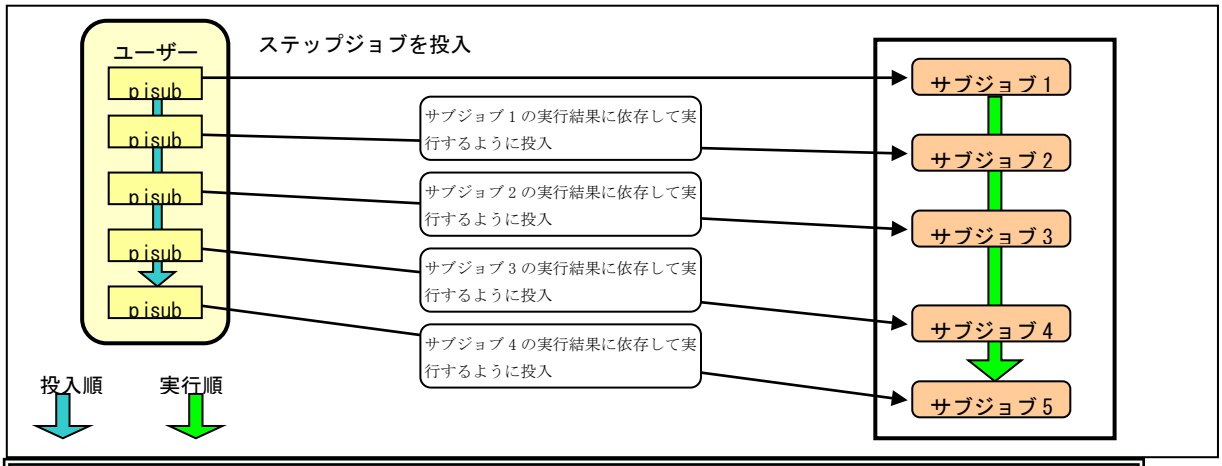
- スクリプトファイルを指定しない場合、標準入力から実行命令を読み込みます。
- ジョブ投入オプションは、スクリプトファイル内にディレクティブを用いて記載可能です。
- ジョブ投入が完了後、ジョブに対して識別用 ID(ジョブ ID)が割り当てられます。

例)バッチジョブ投入例

```
[username@fx01:~] pjsub go.sh                    バッチジョブの投入
[INFO]PJM 0000 pjsub Job 12345 submitted.
```

3.4.2 ステップジョブ投入

ステップジョブは、複数のバッチジョブを 1 つのまとまりとして扱い、その中で実行の順序関係や依存関係を指定することで、ジョブチェイン機能を実現するジョブモデルです。ステップジョブは複数サブジョブから構成され、各サブジョブは同時に実行されることはありません。ステップジョブの動作イメージを以下に示します。




```
pjsub --step [--sparam “sn=stepno [,依存関係式]
  [,send={yes|no}]”] script-file script-file [Script..]
```

表 3-11 ステップジョブ依存関係式

条件	説明
NONE	依存関係がないことを示す
終了ステータス == value[,value,value..] 終了ステータス != value[,value,value..] 終了ステータス > value 終了ステータス >= value 終了ステータス < value 終了ステータス <= value	value には任意の値を指定可能 「==」「!=」の場合は ”,”(カンマ)を用いて、value を複数指定可能 例: ec==1,3,5 → 終了ステータスが 1,3,5 のいずれかであれば真 ec!=1,3,5 → 終了ステータスが 1,3,5 のいずれでもない場合真

表 3-12 ステップジョブ依存関係式で指定可能な削除タイプ

削除タイプ	説明
one	当該ジョブのみを削除します。
after	当該ジョブおよび当該ジョブに依存するジョブを再帰的に削除します。
all	当該ジョブ及び後続のすべてのジョブを削除します。

例)ステップジョブ投入例(ステップ番号を 10 に設定して投入)

```
[username@fx01:~] pjsub --step --sparam "sn=10" stepjob1.sh
[INFO]PJM 0000 pjsub Job 12345 submitted.
```

3.4.3 バッチジョブの終了確認

バッチジョブの実行が終了すると、標準出力ファイルと標準エラー出力ファイルがジョブスケジューラの終了処理としてジョブ投入ディレクトリに出力されます。

標準出力ファイルにはジョブ実行中の標準出力、標準エラー出力ファイルにはジョブ実行中のエラーメッセージが出力されます。

ジョブ名.oXXXXX ･･･ 標準出力ファイル

ジョブ名.eXXXXX ･･･ 標準エラー出力ファイル

ジョブ名.iXXXXX ･･･ ジョブ統計情報出力ファイル（※pjsub コマンドの-S オプション指定時）

(XXXXX はジョブ投入時に表示されるジョブのジョブ ID)

3.4.4 バッチジョブスクリプト記述

バッチジョブを投入するためには、vi コマンドや emacs コマンドにてスクリプトを作成します。

(1) 先頭行は ”#!” に続けて、ジョブで利用するシェル名を指定してください。

[記述例]

```
#!/bin/bash                    bash を利用
```

(2) ジョブ投入オプションは pjsub コマンドのオプションまたはスクリプト中に"#PJM"を用いて指定します。

[記述例]

```
#PJM -L "node=1:mesh"        ノード数[FX100] （メッシュモード）
#PJM -L "node=1:torus"       ノード数[FX100] （トーラスモード）
#PJM -L "node=1:noncont"     ノード数[FX100] （離散割り当て）
#PJM -L "vnode=28"           仮想ノード数[CX2550]
#PJM -L "vnode=24"           仮想ノード数[CX270]
#PJM -L "vnode-core=1"       仮想ノード数あたりのコア数[CX]
#PJM -L "elapse=1:00:00"     経過時間
```

(3) ジョブ投入オプションに続けて、実行時の環境変数設定と、プログラムを指定します。

[記述例]

```
export PARALLEL=8            環境変数を設定
./a.out                      プログラムを実行
```

3.4.4.1 逐次ジョブ用スクリプト

以下、FX100 へのジョブ実行を想定した記述方法を説明します。

- ノード数 : 1 ノード
- プロセス数(スレッド数) : 1 プロセス(1 スレッド)
- 経過時間 : 10 分

[FX]

```
[username@fx01:Fortran] vi sample1.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                リソースグループ
#PJM -L "node=1:mesh"                    ノード数[FX100]
#PJM -L "elapse=10:00"                   経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
./a.out                                  ジョブの実行
```

- ノード数 : 1 ノード(1 仮想ノード)
- プロセス数(スレッド数) : 1 プロセス(1 スレッド)
- 経過時間 : 10 分

[CX]

```
[username@cx01:Fortran] vi sample1.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=cx-small"                リソースグループ
#PJM -L "vnode=1"                        仮想ノード数[CX]
#PJM -L "elapse=10:00"                   経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
./a.out                                  ジョブの実行
```

3.4.4.2 スレッド並列(自動並列)スクリプト

以下、FX100 へのジョブ実行を想定した記述方法を説明します。

- ノード数、スレッド数 : 1 ノード
- プロセス数(スレッド数) : 1 プロセス(32 スレッド:自動並列)

・経過時間　　：10 分

[FX100]

```
[username@fx01:Fortran] vi sample2.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                リソースグループ
#PJM -L "node=1:mesh"                    ノード数
#PJM -L "elapse=10:00"                   経過時間
#PJM -j
#PJM -S                                  統計情報を出力
#------- Program execution -------#
export PARALLEL=32                       自動並列用環境変数設定
./a.out                                  ジョブの実行
```

・ノード数、スレッド数　　：1 ノード
・プロセス数(スレッド数)　　：1 プロセス(28 スレッド:自動並列)
・経過時間　　：10 分

[CX]

```
[username@cx01:Fortran] vi sample2.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=cx-small"                リソースグループ
#PJM -L "vnode=1"                        仮想ノード数 [CX]
#PJM -L "vnode-core=28"                  仮想ノードあたりのコア数[CX]
#PJM -L "elapse=10:00"                   経過時間
 #PJM -j
#PJM -S
#------- Program execution -------#
export PARALLEL=28                       自動並列用環境変数設定
./a.out                                  ジョブの実行
```

3.4.4.3 スレッド並列(OpenMP)スクリプト

以下、FX100 へのジョブ実行を想定した記述方法を説明します。

・ノード数、スレッド数(コア数)　　：1 ノード
・プロセス数(スレッド数)　　：1 プロセス(32 スレッド:OpenMP)
・経過時間　　：10 分

[FX100]

```
[username@fx01:Fortran] vi sample3.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                  リソースグループ指定
#PJM -L "node=1:mesh"                      ノード数指定
#PJM -L "elapse=10:00"                     経過時間指定
#PJM -j
#PJM -S                                    統計情報を出力
#------- Program execution -------#
export OMP_NUM_THREADS=32                  スレッド並列用環境変数設定
./a.out                                    ジョブの実行
```

以下、CX へのジョブ実行を想定した記述方法を説明します。

- ノード数、スレッド数(コア数)　：1 ノード
- プロセス数(スレッド数)　：1 プロセス(28 スレッド:OpenMP)
- 経過時間　：10 分

[CX]

```
[username@cx01:Fortran] vi sample3.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=cx-small"                  リソースグループ
#PJM -L "vnode=1"                          仮想ノード数[CX]
#PJM -L "vnode-core=28"                    仮想ノードあたりのコア数[CX]
#PJM -L "elapse=10:00"                     経過時間指定
#PJM -j
#PJM -S
#------- Program execution -------#
export OMP_NUM_THREADS=28                  スレッド並列用環境変数設定
./a.out                                    ジョブの実行
```

3.4.4.4 MPI(1 次元形状)並列ジョブスクリプト[FX100]

以下、FX100 へのジョブ実行を想定した記述方法を説明します。

- ノード数　：12 ノード(1 次元)
- プロセス数(スレッド数)　：12 プロセス(1 スレッド)
- 経過時間　：10 分

[FX100]

```
[username@fx01:MPI] vi sample4.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                  リソースグループ
#PJM -L "node=12:mesh"                     ノード数(1 次元)
#PJM -L "elapse=10:00"                     経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                            ジョブの実行
```

3.4.4.5 MPI(2 次元形状)並列ジョブスクリプト[FX100]

以下、FX100 へのジョブ実行を想定した記述方法を説明します。

- ノード数 : 12 ノード(2 次元、メッシュ)
- プロセス数(スレッド数) : 12 プロセス(1 スレッド)
- 経過時間 : 10 分

[FX100]

```
[username@fx01:MPI] vi sample5.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                  リソースグループ
#PJM -L "node=6x2:mesh"                    ノード数(2 次元)
#PJM -L "elapse=10:00"                     経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                            ジョブの実行
```

- ノード数 : 12 ノード(2 次元、トーラス)
- プロセス数(スレッド数) : 12 プロセス(1 スレッド)
- 経過時間 : 10 分

[FX100]

```
[username@fx01:MPI] vi sample6.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-middle"                リソースグループ
#PJM -L "node=6x2:torus"                  ノード数(2 次元)
#PJM -L "elapse=10:00"                    経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                           ジョブの実行
```

- ノード数　：12 ノード(2 次元、離散割り当て)
- プロセス数(スレッド数)　：12 プロセス(1 スレッド)
- 経過時間　：10 分

[FX100]

```
[username@fx01:MPI] vi sample7.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                 リソースグループ
#PJM -L "node=6x2:noncont"                ノード数(2 次元)
#PJM -L "elapse=10:00"                    経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                           ジョブの実行
```

3.4.4.6 MPI(3 次元形状)並列ジョブスクリプト[FX100]

以下、FX100 へのジョブ実行を想定した記述方法を説明します。

- ノード数　：96 ノード(3 次元、メッシュ)
- プロセス数(スレッド数)　：96 プロセス(1 スレッド)
- 経過時間　：10 分

```
[username@fx01:MPI] vi sample8.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-middle"                リソースグループ
#PJM -L "node=4x3x8:mesh"                 ノード数(3 次元)、ノード割り当て指定
#PJM -L "elapse=10:00"                    経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                           ジョブの実行
```

- ノード数　　：96 ノード(3 次元、トーラス)
- プロセス数(スレッド数)　　：96 プロセス(1 スレッド)
- 経過時間　　：10 分

```
[username@fx01:MPI] vi sample9.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-middle"                リソースグループ
#PJM -L "node=4x3x8:torus"                ノード数(3 次元)、ノード割り当て指定
#PJM -L "elapse=10:00"                    経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                           ジョブの実行
```

- ノード数　　：12 ノード(3 次元、離散割り当て)
- プロセス数(スレッド数)　　：12 プロセス(1 スレッド)
- 経過時間　　：10 分

```
[username@fx01:MPI] vi sample10.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                 リソースグループ
#PJM -L "node=3x2x2:noncont"              ノード数(3 次元)、ノード割り当て指定
#PJM -L "elapse=10:00"                    経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                           ジョブの実行
```

3.4.4.7 フラット MPI 並列ジョブスクリプト(1 ノード内複数プロセス)

以下のジョブ実行を想定した記述方法を説明します。

- ノード数 ：12 ノード(1 次元)
- プロセス数(スレッド数) ：192 プロセス(1 スレッド)
- 経過時間 ：10 分

[FX100]

```
[username@fx01:MPI] vi sample11.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                  リソースグループ
#PJM -L "node=12:mesh"                     ノード数(1 次元)
#PJM --mpi "proc=192"                      プロセス数
#PJM -L "elapse=10:00"                     経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                            ジョブの実行
```

・ノード数　　：1 ノード(1 次元)
・プロセス数(スレッド数)　　：28 プロセス(1 スレッド)
・経過時間　　：10 分

[CX]

```
[username@cx01:MPI] vi sample4.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=cx-large"                  リソースグループ
#PJM -L "vnode=28"                         仮想ノード数
#PJM -L "vnode-core=1"                     仮想ノードあたりのコア数
#PJM -L "elapse=10:00"                     経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
mpiexec ./a.out                            ジョブの実行
```

3.4.4.8 ハイブリッド MPI 並列ジョブスクリプト

以下のジョブ実行を想定した記述方法を説明します。

- ノード数　：12 ノード(1 次元)
- プロセス数(スレッド数)　：12 プロセス(32 スレッド)
- 経過時間　：10 分

[FX100]

```
[username@fx01:MPI] vi sample12.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                  リソースグループ
#PJM -L "node=12:mesh"                     ノード数(１次元)
#PJM --mpi "proc=12"                       プロセス数
#PJM -L "elapse=10:00"                     経過時間
#PJM -j
#PJM -S
#------- Program execution -------#
export OMP_NUM_THREADS=32                  ハイブリッド並列用環境変数
mpiexec ./a.out                            ジョブの実行
```

• ノード数 : 8 ノード
• プロセス数(スレッド数) : 8 プロセス(28 スレッド)
• 経過時間 : 10 分

[CX]

```
[username@cx01:MPI] vi sample5.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=cx-large"                リソースグループ
#PJM -L "vnode=8"                        仮想ノード数
#PJM -L "vnode-core=28"                  仮想ノードあたりのコア数
#PJM -L "elapse=10:00"                   経過時間
#PJM -S
#PJM -j
#------- Program execution -------#
export OMP_NUM_THREADS=28                ハイブリッド並列用環境変数
mpiexec ./a.out                          ジョブの実行
```

3.5 ジョブ状態表示(pjstat コマンド)

投入したジョブ状態やリソース情報を確認する場合、pjstat コマンドを使用します。
各クラスタのジョブ状況は、当該クラスタに属するログインノードでしか確認できません。

```
pjstat [option] [JOBID[JOBID…]]
```

表 3-13 pjstat コマンドオプション一覧

オプション説明	内容
なし	自分の実行待ち、実行中のジョブ情報を表示
-A	全ユーザーのジョブ情報を表示(他人のユーザー名、ジョブ名は表示されない)
-H	処理が終了したジョブ情報を表示
-E	ステップジョブ情報を表示
-s	-v オプションで出力される情報のほか、資源使用状況、資源制限値などの詳細情報を追加で表示
-S	-s オプションで出力される情報に加えて、そのジョブに設定されているノード単位の情報を表示
-v	標準形式で出力されない、ジョブ情報を追加して表示
--rsc	リソースグループ情報を表示
--limit	システム制限値を表示

3.5.1 ジョブ状態表示

pjstat コマンドを実行すると、現在実行中もしくは実行待ちのジョブ状態を表示します。

[FX100] ※表示例

```
[username@fx01:~] pjstat
  ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
       0       0       0       0       2       0       0       0       0       2
s      0       0       0       0       2       0       0       0       0       2

JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM NODE_REQUIRE  CORE V_MEM
696        run_4.sh   NM RUN user01   07/20 15:26:35  0000:10:00 4   -   -
697        run_4.sh   NM RUN user01   07/20 15:26:36  0000:10:00 4   -   -
```

表 3-14 FX ジョブ情報の表示項目

項目	説明
JOB_ID	ジョブ ID
JOB_NAME	ジョブ名
MD	ジョブモード(normal,step)
ST	ジョブの現在の状態
USER	ユーザー名
RSCGRP	リソースグループ名(-v --pattern=1 指定時のみ)
START_DATE	ジョブが実行前の場合は開始予測時刻("()"で表示)、実行中および実行後の場合は実際に実行を開始した時刻。 実行開始時刻を指定して投入したジョブが実行を開始するまでの間、時刻の後ろに「@」が出力される。 バックフィルが適用されたジョブは、時刻の後ろに「<」が出力される。
ELAPSE_LIM	ジョブの経過時間(実行中でないジョブは"--:--:--)
NODE_REQUIRE	ジョブのノード数とノード形状(nnnn:XXxYxZ)

[CX] ※表示例

```
[username@cx01:~] pjstat
  ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
       0       0       0       1       0       0       0       0       0       1
s      0       0       0       1       0       0       0       0       0       1

JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM NODE_REQUIRE    VNODE  CORE V_MEM
888        sample1.sh NM RUN user01   02/20 17:36:40  0000:10:00 -               1      1    unlimited
```

表 3-15 CX ジョブ情報の表示項目

項目	説明
JOB_ID	ジョブ ID
JOB_NAME	ジョブ名

MD	ジョブモード(normal,step)
ST	ジョブの現在の状態
USER	ユーザー名
RSCGRP	リソースグループ名(-v --pattern=1 指定時のみ)
START_DATE	ジョブが実行前の場合は開始予測時刻("()"で表示)、実行中および実行後の場合は実際に実行を開始した時刻。 実行開始時刻を指定して投入したジョブが実行を開始するまでの間、時刻の後ろに「@」が出力される。 バックフィルが適用されたジョブは、時刻の後ろに「<」が出力される。
ELAPSE_LIM	ジョブの経過時間(実行中でないジョブは"--:--:--)
NODE_REQUIRE	ジョブ投入時のノード数"nnnnnn"(指定がない場合は"-"を出力)
VNODE	仮想ノード数"nnnnnn"
CORE	仮想ノードあたりの CPU コア数"nnn"
V_MEM	仮想ノードあたりのメモリ量(vnode-mem)"nnnnnnnnnnMiB" cpu-memが指定されている場合はvnode-memに変換して(CPUコア数で乗算する)出力

表 3-16 ジョブの状態一覧

状態	説明
ACCEPT	ジョブ受け付け待ち状態
QUEUED	ジョブ実行待ち
STGIN	ステージイン中(FX100)
READY	ジョブ実行開始待ち状態
RUNNING	ジョブ実行中
RUNOUT	ジョブ終了待ち状態
STGOUT	ステージアウト中のジョブ数(FX100)
HOLD	ユーザによる固定状態のジョブ数
ERROR	エラーによる固定状態のジョブ数

3.5.2 詳細ジョブ情報の表示(-v オプション)

-v オプションを指定すると、詳細なジョブ情報を表示します。

[FX100]※表示例

```
[username@fx01:~] pjstat -v
  ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
       0      0      0      0      1      0      0      0      0       1
s      0      0      0      0      1      0      0      0      0       1
JOB_ID     JOB_NAME    MD ST  USER     GROUP    START_DATE      ELAPSE_TIM ELAPSE_LIM NODE_REQUIRE  VNODE  CORE
V_MEM  LST EC  PC  SN PRI ACCEPT         RSC_UNIT REASON
577        go.sh       NM RUN user01   grp1     06/15 16:13:33  0000:00:15 0048:00:00 2              -      -
-      RNA 0   0   0  30  09/15 16:12:32 fx    -
```

表 3-17 詳細ジョブ情報の表示項目(追加項目のみ)

オプション名	説明
GROUP	実行ユーザのグループ名 ステップジョブのサマリ情報の場合は、実行中のサブジョブの情報を出力します。 実行中のサブジョブがない場合、次に実行される予定のサブジョブの情報を出力します。
ELAPSE_LIM	ジョブの経過時間制限
LST	ジョブの以前の処理状態
EC	ジョブスクリプトの終了コード ステップジョブのサマリ情報の場合は、"-" を出力します。
PC	ジョブ終了コード (PJM コード) ジョブ実行における、ジョブマネージャーの処理結果を示すコードです。 ステップジョブのサマリ情報の場合は、"-" を出力します。 コードの意味は以下のとおりです。 0:ジョブの正常終了 1:pjdel コマンドによる CANCEL 2:ジョブの受け付け拒否判定による REJECT 3:改札制御による実行拒否 4:pjhold コマンドによる HOLD 6:ステップジョブ依存関係式による CANCEL 7:デッドライン強制指定により CANCEL 8:改札制御により CANCEL 9:再実行不可指定のため、ジョブ再構築時に EXIT 10:CPU 時間制限違反によるジョブ実行タイムアウト 11:経過時間制限違反によるジョブ実行タイムアウト 12:メモリ使用量超過による強制終了 13:ディスク使用量超過による強制終了 16:カレントディレクトリまたは標準入力/標準出力/標準エラー出力ファイルへのアクセス不可による終了 20:ノードダウン

SN	シグナル番号 ステップジョブのサマリ情報の場合は、"-" を出力します。
PRI	ジョブの優先度(0: 低 <-> 255:高)
ACCEPT	ジョブの投入日時 "MM/DD hh:mm:ss"
RSC_UNIT	ジョブ投入時のリソースユニット ステップジョブのサマリ情報の場合は、"-" を出力します。
REASON	エラーメッセージ

[CX]※表示例

```
[username@cx01:~] pjstat -v
  ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
       0      0      0      0      1      0      0      0      0       1
s      0      0      0      0      1      0      0      0      0       1

JOB_ID      JOB_NAME    MD ST  USER      GROUP     START_DATE       ELAPSE_TIM ELAPSE_LIM NODE_REQUIRE     VNODE
CORE V_MEM         V_POL E_POL RANK      LST EC  PC  SN PRI ACCEPT          RSC_UNIT REASON
895         sample1.sh NM RUN w49942a  center   02/20 18:03:22  0000:00:01 0000:10:00 4                1      1
unlimited    PACK  SMPLX -         RNP 0   0   0  127 02/20 18:03:21 cx       -
```

表 3-18 詳細ジョブ情報の表示項目(追加項目のみ)

オプション名	説明
GROUP	実行ユーザのグループ名
ELAPSE_LIM	実行経過時間"hhhh:mm:ss"
V_POL	仮想ノード配置ポリシー A_PCK : Absolutely PACK PACK : PACK A_UPK : Absolutely UNPACK UPCK : UNPACK
E_POL	実行モードポリシー SHARE : SHARE SMPLX : SIMPLEX
RANK	ランクマップの指定方法 bynode : rank-map-bynode bychip : rank-map-bychip
LST	ジョブの以前（「ジョブの現在の処理状態」に遷移する前）の処理状態
EC	ジョブスクリプトの終了コード
PC	ジョブ終了コード(PJM コード) 0:ジョブの正常終了 1:pjdelコマンドによる CANCEL 2:ジョブの受け付け拒否判定による REJECT
SN	シグナル番号
PRI	ジョブの優先度 緊急ジョブの優先度は 256、非緊急ジョブの優先度は 0 から 255
ACCEPT	ジョブの投入日時 "MM/DD hh:mm:ss"

RSC_UNIT	ジョブ投入時のリソースユニット
REASON	エラーメッセージ ジョブを実行する、しないに関わらず、そのジョブの何らかの処理に対する結果コードに対応するメッセージ

3.5.3 終了ジョブ情報の表示(-H オプション)

-H オプションを指定すると、過去に投入したジョブで、既に実行が終了したジョブの一覧(終了ジョブ一覧)を表示します。

[FX100]

```
[username@fx01:~] pjstat -H
  ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
       0      0      0      0      0      0      0      0      0       0
s      0      0      0      0      0      0      0      0      0       0
  REJECT   EXIT CANCEL   TOTAL
       0     57      7      64
s      0      0      0       0
JOB_ID      JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM NODE_REQUIRE VNODE  CORE V_MEM
657         go.sh      NM EXT user01   06/21 15:30:54  0000:10:00 12   -    -    -
660         go.sh      NM CCL user01   06/21 15:40:41  0000:10:00 12   -    -    -
```

表 3-19 FX ジョブ情報の表示項目

項目	説明
JOB_ID	ジョブ ID
JOB_NAME	ジョブ名
MD	ジョブモード(normal,step)
ST	ジョブの現在の状態
USER	ユーザー名
RSCGRP	リソースグループ名(-v --pattern=1 指定時のみ)
START_DATE	ジョブが実行前の場合は開始予測時刻("()"で表示)、実行中および実行後の場合は実際に実行を開始した時刻。 実行開始時刻を指定して投入したジョブが実行を開始するまでの間、時刻の後ろに「@」が出力される。 バックフィルが適用されたジョブは、時刻の後ろに「<」が出力される。
ELAPSE_LIM	ジョブの経過時間(実行中でないジョブは"--:--:--)
NODE_REQUIRE	ジョブのノード数とノード形状(nnnn:XXxYxZ)

[CX]

```
[username@cx01:~] pjstat -H
  ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
       0      0      0      0      0      0      0      0      0       0
s      0      0      0      0      0      0      0      0      0       0
  REJECT   EXIT CANCEL   TOTAL
       0     57      7      64
s      0      0      0       0
JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM NODE_REQUIRE    VNODE  CORE V_MEM
479        go.sh      NM EXT w49942a  02/18 10:43:28  0024:00:00 -               24     1    unlimited
480        go.sh      NM EXT w49942a  02/18 11:06:14  0024:00:00 -               24     1    unlimited
```

表 3-20 CX ジョブ情報の表示項目

項目	説明
JOB_ID	ジョブ ID
JOB_NAME	ジョブ名
MD	ジョブモード(normal,step)
ST	ジョブの現在の状態
USER	ユーザー名
RSCGRP	リソースグループ名(-v --pattern=1 指定時のみ)
START_DATE	ジョブが実行前の場合は開始予測時刻("()"で表示)、実行中および実行後の場合は実際に実行を開始した時刻。 実行開始時刻を指定して投入したジョブが実行を開始するまでの間、時刻の後ろに「@」が出力される。 バックフィルが適用されたジョブは、時刻の後ろに「<」が出力される。
ELAPSE_LIM	ジョブの経過時間(実行中でないジョブは"--:--:--)
NODE_REQUIRE	ジョブ投入時のノード数"nnnnnn"(指定がない場合は"-"を出力)
VNODE	仮想ノード数"nnnnnn"
CORE	仮想ノードあたりの CPU コア数"nnn"
V_MEM	仮想ノードあたりのメモリ量(vnode-mem)"nnnnnnnnnnMiB" cpu-memが指定されている場合はvnode-memに変換して(CPUコア数で乗算する)出力

3.5.4 リソースグループの表示(--rsc オプション)

--rsc オプションを指定すると、ユーザーが利用可能なリソースグループを表示します。

[FX100]

※表示例）

```
[username@fx01:~] pjstat --rsc
RSCUNIT                        RSCUNIT_SIZE  RSCGRP                          RSCGRP_SIZE
fx[ENABLE,START]               4x7x9         fx-interactive[ENABLE,START]    8x3x18
fx[ENABLE,START]               4x7x9         fx-debug[ENABLE,START]          8x3x18
fx[ENABLE,START]               4x7x9         fx-small[ENABLE,START]          8x3x18
fx[ENABLE,START]               4x7x9         fx-middle[ENABLE,START]         8x18x18
fx[ENABLE,START]               4x7x9         fx-large[ENABLE,START]          8x18x18
fx[ENABLE,START]               4x7x9         fx-xlarge[ENABLE,START]         8x18x18
fx[ENABLE,START]               4x7x9         fx-special[ENABLE,STOP]         8x18x18
```

[CX]

※表示例）

```
[username@cx01:~] pjstat --rsc
RSCUNIT              RSCUNIT_SIZE  RSCGRP                       RSCGRP_SIZE
cx[ENABLE,START]     584           cx-debug[ENABLE,START]       400
cx[ENABLE,START]     584           cx-single[ENABLE,START]      400
cx[ENABLE,START]     584           cx-small[ENABLE,START]       400
cx[ENABLE,START]     584           cx-middle[ENABLE,START]      400
cx[ENABLE,START]     584           cx-large[ENABLE,START]       400
cx[ENABLE,START]     584           cx-special[ENABLE,START]     400
cx[ENABLE,START]     584           cx2-debug[ENABLE,START]      184
cx[ENABLE,START]     584           cx2-small[ENABLE,START]      184
cx[ENABLE,START]     584           cx2-middle[ENABLE,START]     184
cx[ENABLE,START]     584           cx2-large[ENABLE,START]      184
cx[ENABLE,START]     584           cx2-special[ENABLE,START]    184
```

表 3-21 リソースグループの表示項目

項目	説明
RSCUNIT	リソースユニット名とその状態 「状態」には以下があります。 ENABLE: ジョブの投入は可能 DISABLE: ジョブの投入は不可 START : ジョブは実行可能 STOP : ジョブは実行不可

RSCUNIT SIZE	リソースユニットのサイズ 【FX100】 Tofu 単位の数を 1 辺の長さとする X、Y、Z 軸方向の直方体として表現されます。 書式： XxYxZ 【CX】 リソースユニットを構成するノード数 N が表示されます。
RSCGRP	リソースグループ名
RSCGRP_SIZE	リソースグループのサイズ（投入、実行可否）

3.6 ジョブキャンセル(pjdel コマンド)

投入済みのジョブをキャンセルする場合、pjdel コマンドを実行します。

```
pjdel [JOBID [JOBID…]]
```

ジョブのジョブ ID を pjdel の引数に指定します。

```
[username@fx01:~] pjdel 670
[INFO] PJM 0100 pjdel Job 670 canceled.
```

3.7 ジョブ保留(pjhold コマンド)

投入済みのジョブ実行を保留する場合、pjhold コマンドを指定します。

```
pjhold [-R <reasonmessage>] [JOBID [JOBID…]]
```

ジョブのジョブ ID を pjhold の引数に指定します。

```
[username@fx01:~] pjhold 671
[INFO] PJM 0300 pjhold Accepted job 671.
```

表 3-22 ジョブ保留コマンドオプション一覧

オプション名	説明
-R *reasonmessage*	ジョブを保留した理由を指定。指定した文字列は、pjstat -v の出力結果の REASON に出力。

3.8 ジョブ開放(pjrls コマンド)

保留されたジョブを解除する場合、pjrls コマンドを指定します。

```
pjrls [JOBID [JOBID…]]
```

ジョブのジョブ ID を pjrls の引数に指定します。

```
[username@fx01:~] pjrls 671
[INFO] PJM 0400 pjrls jobid 671 released.
```

4. MPI 実行

並列ジョブを実行する場合、ジョブスクリプトの投入オプションに--mpi オプションを付与します。

4.1 MPI プログラム実行

MPI ライブラリを付与した実行モジュールを実行するために、mpiexec コマンドを利用します。

```
mpiexec [option] 実行モジュール
```

表 4-1 オプション一覧

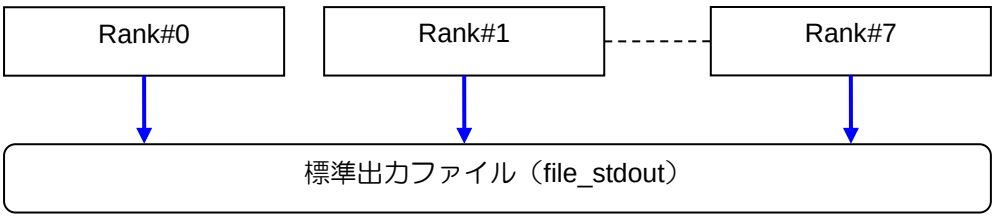
オプション名	説明
-n <proc_num>	MPI プログラムのプロセス数を指定 設定しない場合、--mpi オプションで指定したプロセス数が設定される
-of <*fname*>	並列プロセスの標準出力および標準エラー出力をファイル名 *fname* に出力
-of-proc <*fname*>	並列プロセスの標準出力および標準エラー出力をプロセス毎にファイル名"*fname*.ランク番号"に出力
-oferr <*fname*>	並列プロセスの標準エラー出力をファイル名 *fname* に出力
-oferr-proc <*fname*>	並列プロセスの標準エラー出力をファイル名"*fname*.ランク番号"に出力
-ofout <*fname*>	並列プロセスの標準出力をファイル名 *fname* に出力
-ofout-proc <*fname*>	並列プロセスの標準出力をファイル名"*fname*.ランク番号"に出力
-stdin <*fname*>	全並列プロセスの標準入力を、ファイル名 *fname* から読み込む

4.1.1 標準出力/標準エラー出力/標準入力

MPI プログラムの標準出力／標準エラー出力／標準入力の指定方法を示します。mpiexec では、標準出力／標準エラー出力をファイルに出力するオプション、ファイルから標準入力を読み込むオプションを用意しています。

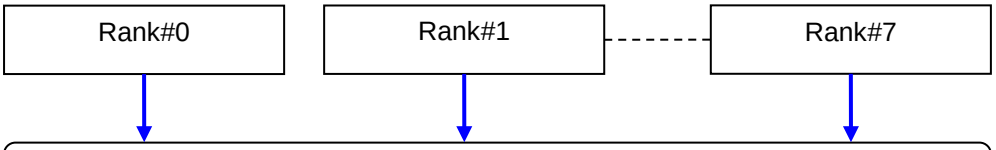
- 各並列プロセスと mpiexec コマンドの標準出力／標準エラー出力は、通常はジョブ運用ソフトウェアによって生成されるジョブ実行結果ファイル（ジョブ名.o.ジョブ ID／ジョブ名.e.ジョブ ID）に出力されます。
- mpiexec コマンドのリダイレクション指定による標準入力は、各並列プロセスの標準入力として使用することはできません。

(1) 並列プロセスの標準出力を指定ファイルに出力します。標準出力を “file_stdout” に出力する例を示します。



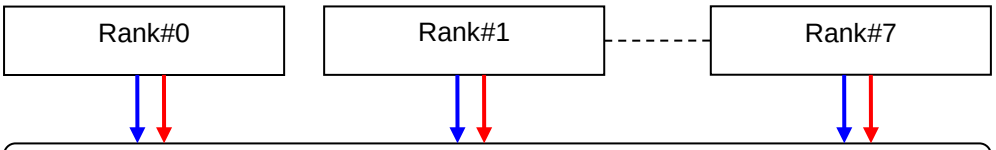
```
mpiexec -ofout file_stdout -n 8 ./a.out
```

(2) 並列プロセスの標準エラー出力を指定ファイルに出力します。標準エラー出力を“file_stderr” に出力する例を示します。



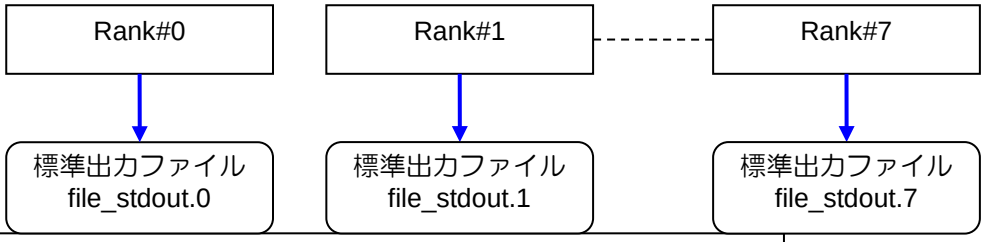

```
mpiexec -oferr file_stderr -n 8./a.out
```

(3) 並列プロセスの標準出力および標準エラー出力を指定ファイルに出力します。標準出力および標準エラー出力を “file_outfile” に出力する例を示します。



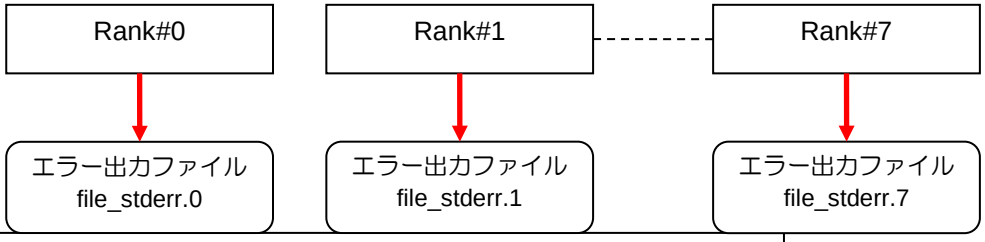

```
mpiexec -of file_outfile -n 8 ./a.out
```

(4) 各並列プロセスの標準出力を別ファイルに出力します。各並列プロセスからの標準出力は、-ofout-proc に指定したファイル名にランク番号が付加された名前で出力されます。



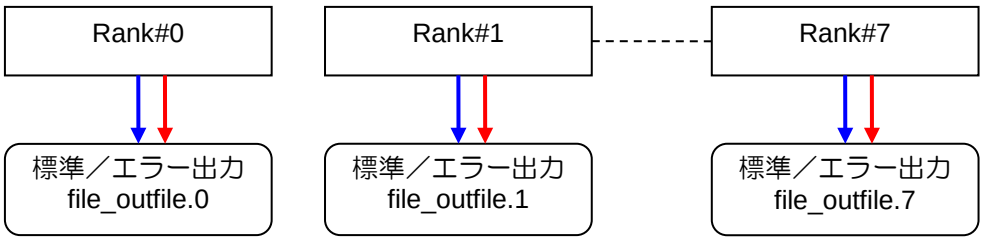

```
mpiexec -ofout-proc file_stdout -n 8 ./a.out
```

(5) 各並列プロセスの標準エラー出力を別ファイルに出力します。各並列プロセスからの標準エラー出力は、-oferr-proc に指定したファイル名にランク番号が付加された名前で出力されます。



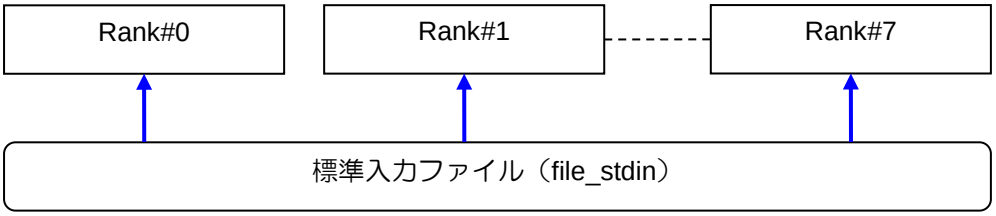

```
mpiexec -oferr-proc file_stderr -n 8 ./a.out
```

(6) 各並列プロセスの標準出力および標準エラー出力を別ファイルに出力します。各並列プロセスからの標準出力／標準エラー出力は、-of-proc に指定したファイル名にランク番号が付加された名前で出力されます。




```
mpiexec -of-proc file_outfile -n 8 ./a.out
```

(7) 並列プロセスの標準入力を指定ファイルから入力します。標準入力を “file_stdin” から入力する例を示します。




```
mpiexec -stdin file_stdin -n 8 ./a.out
```

4.2 MPI ジョブ投入時の指定

MPI ジョブ投入時には、pjsub コマンドの --mpi オプションを利用して、起動プロセス形状の指定、ランク割付ルールの指定、起動プロセスの最大値の指定が可能です。

表 4-2 起動プロセスの割付け方法

指定方法	指定オプション
起動プロセス形状を指定する	`pjsub --mpi shape`（FX100 のみ）
起動プロセスの最大数を指定する	`pjsub --mpi proc`

生成するプロセスのランク割付けルールを指定する	`pjsub --mpi rank-map-bynode`
	`pjsub --mpi rank-map-hostfile`
	`pjsub --mpi rank-map-bychip`

4.2.1 静的プロセスの形状指定

pjsub コマンド(--mpi shape)を使用することで、静的に起動するプロセスの形状を指定できます。プロセスの形状は、1 次元、2 次元、3 次元の形状で --rsc-list（-L）の node パラメタで指定するノード形状と同じ次元数を指定する必要があります。--mpi オプションの shape パラメタが省略された場合は、-L オプションで指定された node パラメタの値が使用されます。

shape パラメタ指定例

```
[1 次元形状] --mpi  "shape=X"
[2 次元形状] --mpi  "shape=XxY"
[3 次元形状] --mpi  "shape=XxYxZ"
```

例) 3 次元のプロセス形状(X 軸 2、Y 軸 3、Z 軸 2)を指定

```
[username@fx01:MPI] vi sample1.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-large"                    リソースグループ
#PJM --rsc-list  "node=2x3x2:torus"          ノード数(3 次元形状)
#PJM -L "elapse=10:00"                       経過時間
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

4.2.2 静的プロセスの最大数指定

生成するプロセス数を指定するには、pjsub --mpi proc で指定します。

(1) --mpi proc で指定可能なプロセス数は、(--mpi proc 指定値)×32 以下となります。

(2) --mpi proc を省略した場合は、1 ノードに 1 プロセスを生成します。

(3) フラット MPI で実行する場合は、本オプションを使用してプロセス数を指定します。
FX100 にて 4 ノードでフラット MPI を実行する場合は、--mpi proc=128 (4 ノード×32 プロセス)となります。

例) 割当て最大プロセス数(128)を指定

```
[username@fx01:MPI] vi sample2.sh
#!/bin/sh
#------ pjsub option --------#
#PJM  -L "rscgrp=fx-large"                  リソースグループ
#PJM  -L  "node=4:mesh"                     ノード数(1 次元形状):メッシュモード
#PJM  --mpi  "proc=128"                     静的プロセスの最大数 128
#PJM -L "elapse=10:00"                      経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out                            128 プロセスで実行
```

4.2.3 MPI ランク割当

MPI では、プロセス識別のために、プロセス番号に相当する「ランク」番号を割り当てます。

ランクの割り当てルールは、pjsub コマンドの 以下の--mpi オプションでユーザーが直接指定することも可能です。MPI ランクの割当て指定は 3 つ種類です。

表 4-3 MPI ランク割り当て方法

オプション名	説明
rank-map-bynode	計算ノードに 1 プロセスを生成すると、次の計算ノードに移動し、ラウンドロビンで自動的に割り付けます。座標の原点をランク 0 とし、rank-map-bynode の先頭文字の軸方向にランクを並べ、上限まで達した時点で、次の文字に移動します。
rank-map-bychip (デフォルト指定)	計算ノードに n プロセスを生成すると、次の計算ノードに移動し、ラウンドロビンで自動的に割り付けます。座標の原点をランク 0 とし、rank-map-bynode の先頭文字の軸方向にランクを並べ、上限まで達した時点で、次の文字に移動します。
rank-map-hostfile	ランクマップファイルに指定された座標を基に、ランクを割り当てます。

4.2.4 rank-map-bynode

rank-map-bynode は、ジョブ割り当て時に先頭の計算ノードからラウンドロビンで自動的にランク割り当てが行われます。

例1）1 次元形状の rank-map-bynode を指定する。

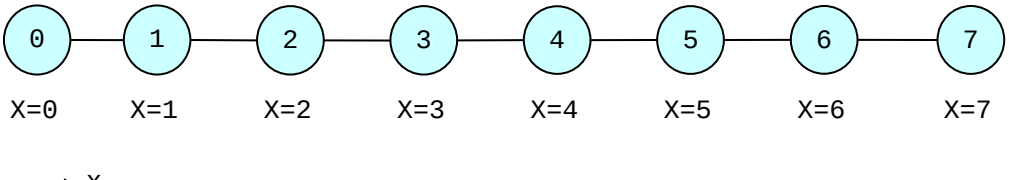


図 4-1 ランク割当例(1 次元形状)

```
[username@fx01:MPI] vi sample3.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L  "rscgrp=fx-large"                  リソースグループ指定
#PJM -L  "node=8:mesh"                      ノード数の指定(1 次元形状)
#PJM --mpi "rank-map-bynode"                動的 MPI ランク割当て
#PJM -L "elapse=10:00"                      経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

例2）2 次元形状の rank-map-bynode を指定する。

--mpi "rank-map-bynode=XY"　　--mpi "rank-map-bynode=YX"

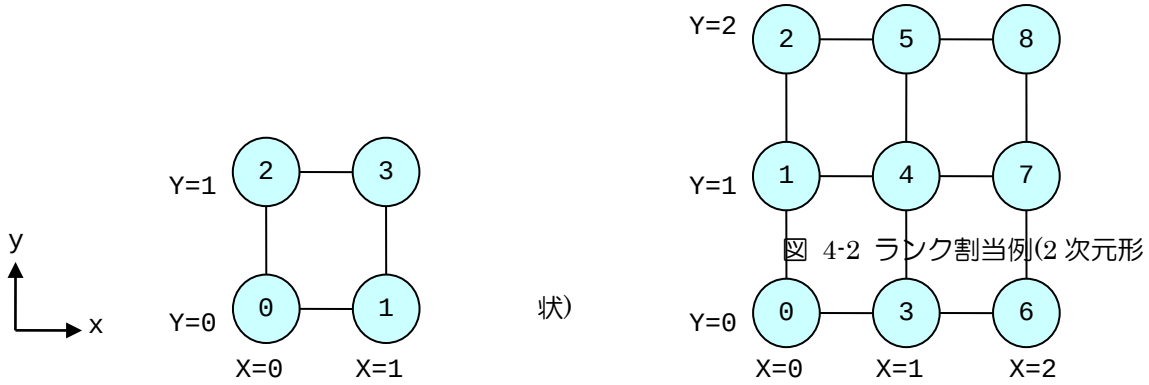


図 4-2 ランク割当例(2 次元形状)

```
[username@fx01:MPI] vi sample4.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                   リソースグループ指定
#PJM -L "node=2x2:mesh"                     ノード数の指定(2 次元形状)
#PJM --mpi "rank-map-bynode=XY"             動的 MPI ランク割当て
#PJM -L "elapse=10:00"                      経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

例3) 3 次元形状の rank-map-bynode を指定する。

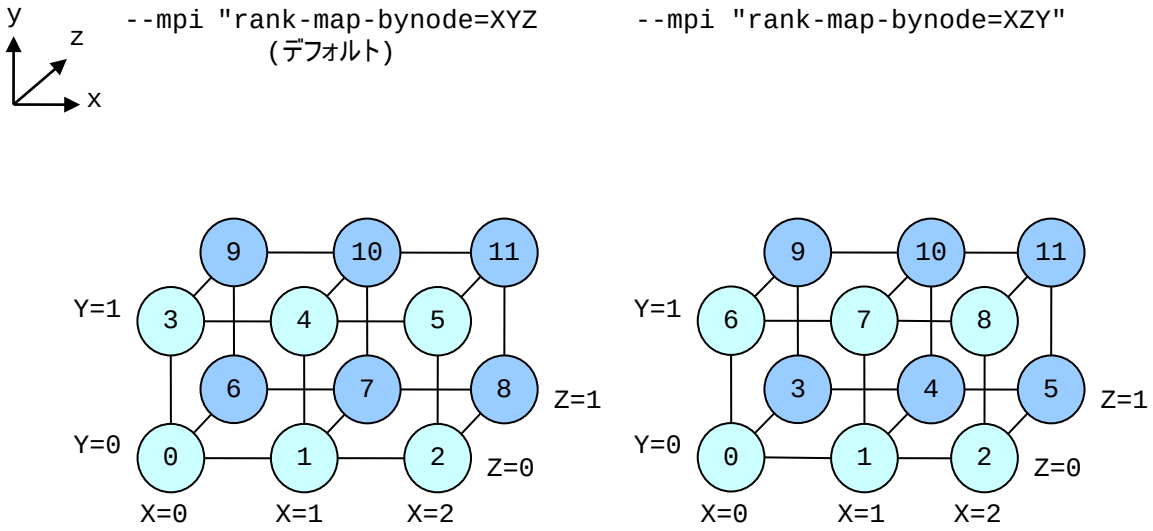


図 4-3 ランク割当例(3 次元形状)

```
[username@fx01:MPI] vi sample5.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-large"                  リソースグループ指定
#PJM -L "node=2x3x2:mesh"                  ノード数の指定(3 次元形状)
#PJM --mpi  "rank-map-bynode=XYZ"          動的 MPI ランク割当て
#PJM -L "elapse=10:00"                     経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

例4) 1 次元形状の rank-map-bynode を指定し、ノード内に複数プロセスを起動する。

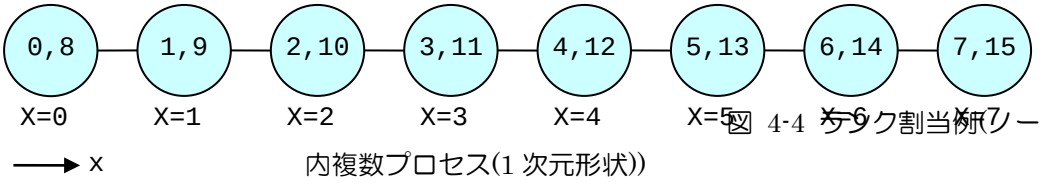


図 4-4 ランク割当例(ノード内複数プロセス(1 次元形状))

```
[username@fx01:MPI] vi sample6.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                  リソースグループ指定
#PJM -L  "node=8:mesh"                     ノード数の指定(1 次元形状)
#PJM --mpi  "proc=16"                      16 プロセスを起動
#PJM --mpi  "rank-map-bynode"              静的プロセス形状
#PJM -L "elapse=10:00"                     経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

4.2.5 rank-map-bychip

rank-map-chip は、計算ノードに指定した n プロセスを生成すると、次の計算ノードに移動しラウンドロビンで自動的にランクを割り付けます。座標の原点をランク 0 とし、rank-map-bychip の先頭文字の軸方向にランクを並べ、上限まで達した時点で、次の文字に移動します。

以下に rank-map-bychip の指定例を示します。

```
#PJM --mpi  "rank-map-bychip[:{XY|YX}]"                    (2 次元)
#PJM --mpi  "rank-map-bychip[:{XYZ|XZY|YXZ|YZX|ZXY|ZYX}]"  (3 次元)
```

例1) 2 次元形状の rank-map-bychip を指定し、ノード内に複数プロセスを起動する。

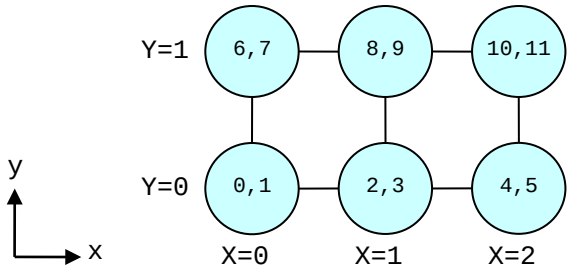


図 4-5 ランク割当例(ノード内複数プロセス(2 次元形状))

```
[username@fx01:MPI] vi sample7.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                    リソースグループ指定
#PJM -L "node=3x2:mesh"                      ノード数の指定(2 次元形状)
#PJM --mpi  "proc=12"                        12 プロセスを起動
#PJM --mpi  "rank-map-bychip:XY"             動的 MPI ランク割当て
#PJM -L "elapse=10:00"                       経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

4.2.6 rank-map-hostfile

rank-map-hostfile は、ユーザーが指示するホストマップファイルの座標をもとに、ランクを割り当てます。

以下に rank-map-hostfile の指定例を示します。

```
[username@fx01:MPI] vi sample8.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                    リソースグループ指定
#PJM -L  "node=2x2x2:mesh"                   ノード数の指定(3 次元形状)
#PJM --mpi "rank-map-hostfile=rankmap"       動的 MPI ランク割当て
#PJM -L "elapse=10:00"                       経過時間指定
#PJM -j
#------ program execution ------#
mpiexec  ./a.out
```

(1) ファイル rankmap は、pjsub コマンドを実行するカレントディレクトリに配置します。

(2) ファイル rankmap 内のランク指定は、ノード形状に合わせて、1 次元、2 次元または 3 次元座標で指定します。

(3) ファイル rankmap には、1 行に 1 座標を記述し、括弧で囲んで指定します。

例1) 1 次元形状の rank-map-hostfile を指定、ファイル rankmap に配置を記載。

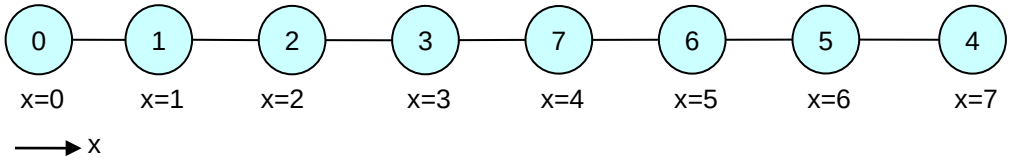


図 4-6 ランク割当例(ホストマップファイル／1 次元形状)

```
[username@fx01:MPI] vi sample9.sh
#!/bin/sh
#------ pjsub option --------#
#PJM  -L  "rscgrp=fx-small"                  リソースグループ指定
#PJM  -L  "node=8:mesh"                      ノード数の指定(1 次元形状)
#PJM  --mpi "rank-map-hostfile=rankmap1"     動的 MPI ランク割当て
#PJM -L "elapse=10:00"                       経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

1 次元座標を指定する場合、ホストマップファイルは(X)を指定します

```
[username@fx01:MPI] vi rankmap1
# ホストマップファイル(1 次元指定例)
(0)                                     #rank0
(1)                                     #rank1
(2)                                     #rank2
(3)                                     #rank3
(7)                                     #rank4
(6)                                     #rank5
(5)                                     #rank6
(4)                                     #rank7
```

例2) 2 次元形状の rank-map-hostfile を指定、ファイル rankmap に配置を記載。

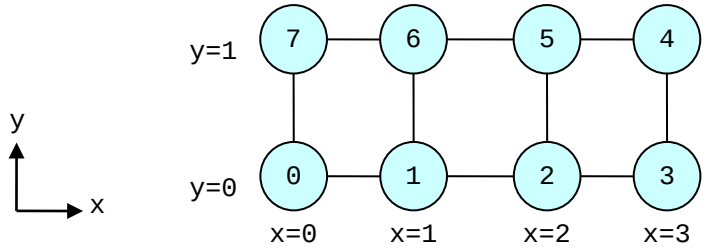


図 4-7 ランク割当例(ホストマップファイル／2 次元形状)

```
[username@fx01:MPI] vi sample10.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-small"                リソースグループ指定
#PJM -L "node=4x2:mesh"                  ノード数の指定(2 次元形状)
#PJM --mpi "rank-map-hostfile=rankmap2"  動的 MPI ランク割当て
#PJM -L "elapse=10:00"                   経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

2 次元座標を指定する場合、ホストマップファイルは(X,Y)を指定します。

```
[username@fx01:MPI] vi rankmap2
# ホストマップファイル(2 次元指定例)
(0,0)                                   #rank0
(1,0)                                   #rank1
(2,0)                                   #rank2
(3,0)                                   #rank3
(3,1)                                   #rank4
(2,1)                                   #rank5
(1,1)                                   #rank6
(0,1)                                   #rank7
```

例3) 3 次元形状の rank-map-hostfile を指定、ファイル rankmap に配置を記載。

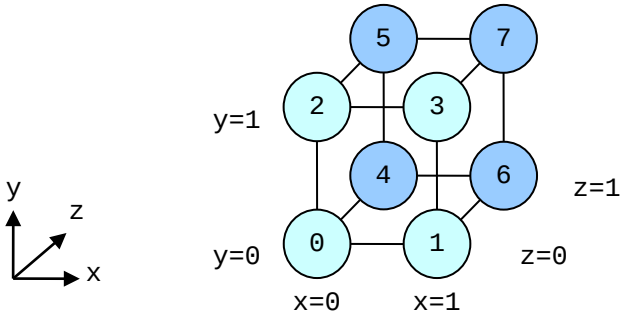


図 4-8 ランク割当例(ホストマップファイル／3 次元形状)

```
[username@fx01:MPI] vi sample11.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L  "rscgrp=fx-small"                    リソースグループ指定
#PJM -L  "node=2x2x2:mesh"                    ノード数の指定(3 次元形状)
#PJM --mpi "rank-map-hostfile=rankmap3"       動的 MPI ランク割当て
#PJM -L "elapse=10:00"                        経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

3 次元座標を指定する場合、ホストマップファイルは(X,Y,Z)を指定します。

```
[username@fx01:MPI] vi rankmap3
# ホストマップファイル(3 次元指定例)
(0,0,0)                                    #rank0
(1,0,0)                                    #rank1
(0,1,0)                                    #rank2
(1,1,0)                                    #rank3
(0,0,1)                                    #rank4
(0,1,1)                                    #rank5
(1,0,1)                                    #rank6
(1,1,1)                                    #rank7
```

ランクマップファイル割当て(rank-map-hostfile)と起動プロセス数の指定を組み合わせることで、ノード内に複数プロセスを割り当てる事も可能です。

例4) 2 次元形状の rank-map-hostfile を指定し、ノード内に複数ランクを指定する。

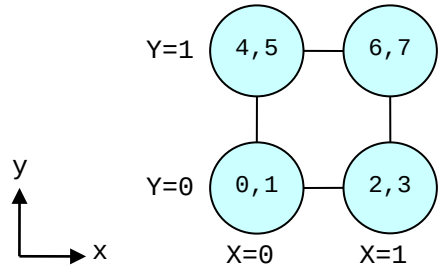


図 4-9 ランク割当例(ホストマップファイル／2 次元形状)

```
[username@fx01:MPI] vi sample12.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L  "rscgrp=fx-small"                リソースグループ指定
#PJM -L  "node=2x2:mesh"                  ノード数の指定(2 次元形状)
#PJM --mpi "proc=8"                       プロセス数指定
#PJM --mpi "rank-map-hostfile=rankmap4"   動的 MPI ランク割当て
#PJM -L "elapse=10:00"                    経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

```
[username@fx01:MPI] vi rankmap4
# ホストマップファイル(2 次元指定例)
(0,0)
(1,0)
(0,1)
(1,1)
```

rank-map-hostfile と rank-map-bychip を同時に指定することで、ホストマップファイルで指定される座標(1 行)に、rank-map-bychip で指定された数のプロセスを割り当てることも可能です。

例5) 2 次元形状の rank-map-hostfile を指定し、rakn-map-bychip を同時に指定する

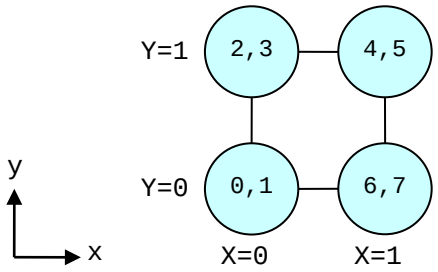


図 4-10 ランク割当例 (ホストマップファイル／2 次元形状)

```
[username@fx01:MPI] vi sample13.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L  "rscgrp=fx-small"                  リソースグループ指定
#PJM -L  "node=2x2:mesh"                    ノード数の指定(2 次元形状)
#PJM --mpi "proc=8"                         プロセス数指定
#PJM --mpi "rank-map-hostfile=rankmap5"     動的 MPI ランク割当て
#PJM --mpi "rank-map-bychip:XY"             動的 MPI ランク割当て
#PJM -L "elapse=10:00"                      経過時間指定
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec  ./a.out
```

起動するランク配置(2 次元)をホストマップファイルに記載します。

```
[username@fx01:MPI] vi rankmap5
# ホストマップファイル(2 次元指定例)
(0,0)                                       #rank 0,1
(0,1)                                       #rank 2,3
(1,1)                                       #rank 4,5
(1,0)                                       #rank 6,7
```

4.2.7 ホストマップファイル利用時の注意事項

ホストマップファイルについて注意事項を示します。

(1) ファイル中の空行は、無視されます。

(2) ファイル中の有効な座標の行数が、--mpi proc（rank-map-bychip の場合は、proc÷n）で指定した値よりも多い場合、残りの行は無視されます。

(3) ファイル中の有効な座標の行数が、--mpi proc（rank-map-bychip の場合は、proc÷n）で指定した値よりも少ない場合、最後の行まで割り当てたら、先頭行に戻って割り当てます。

(4) ホストマップファイル中にノード座標を記述する場合は、各ノードに割り当てるプロセス数を均等にする必要があります。例えば、4 ノードで 7 プロセスを実行する場合、4-1-1-1 のプロセス数配置を指定してもジョブの実行はできません。2-2-2-1 のように、プロセス数を均等にするように記述してください。

4.2.8 ジョブ形状(トーラス)

pjsub -L node で指定された形状(1 次元／2 次元／3 次元)、計算ノード数に応じて、ジョブに資源が割り当てられます。割り当てられた資源は、各軸単位にジョブ内でトーラスが構成されます。

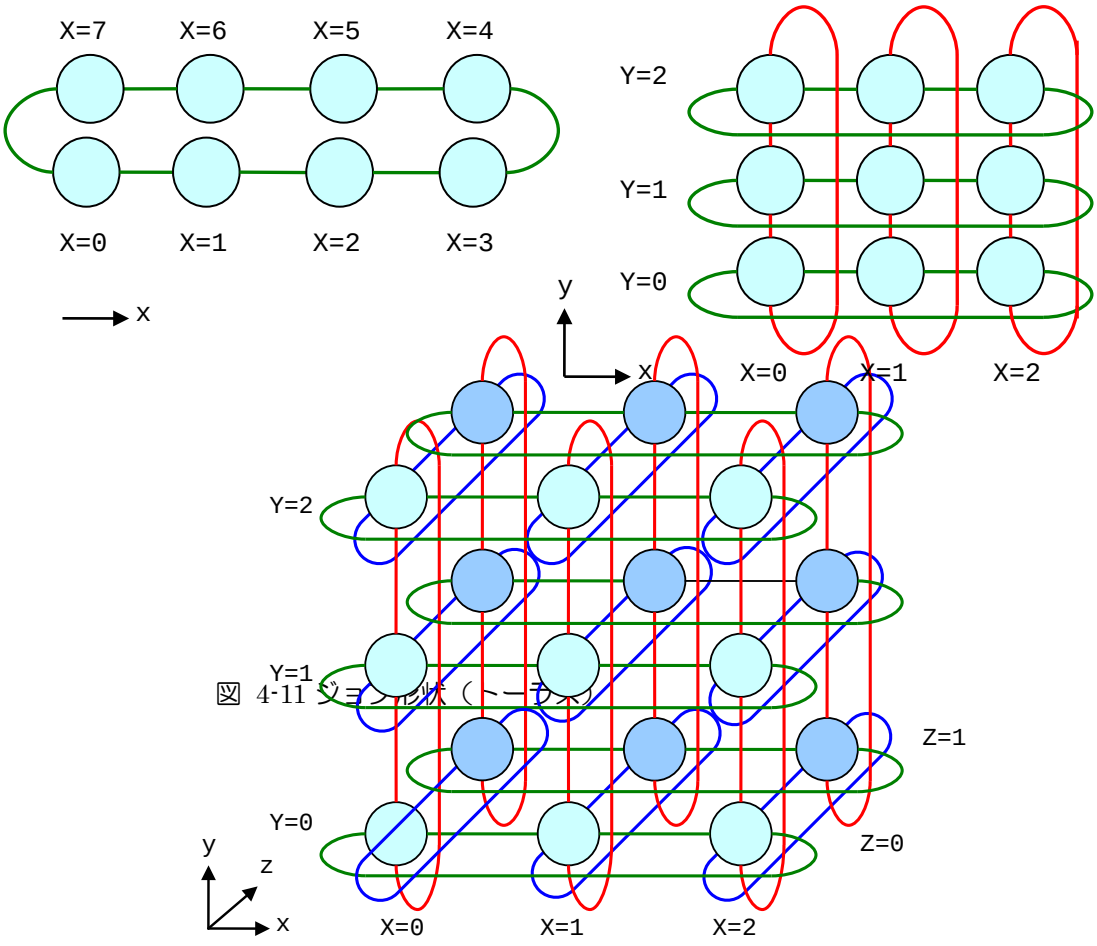


図 4-11 ジョブ形状（トーラス）

4.2.9 複合 MPI オプション指定(MPI 関連)

MPI プログラムを実行するには、pjsub のオプションを適切に指定する必要があります。

1 次元の指定例を示します。

【ジョブスクリプト】

```
[username@fx01:MPI]$ vi sample14.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L "rscgrp=fx-large"              リソースグループ指定
#PJM -L "node=24:torus"                ノード数の指定(１次元形状)
#PJM --mpi "proc=12"                   最大プロセス形状指定
#PJM -L "elapse=10:00"                 経過時間指定
#PJM -j
#------ program execution ------#
mpiexec -n 12 ./a.out                  並列プロセス数 12 で a.out を実行
```

上記の-L node で要求した形状/計算ノード数が確保されます。1 次元形状(24)が指定されているため、利用するプロセスが 12 であっても、1 次元で 24 計算ノードが確保されます。

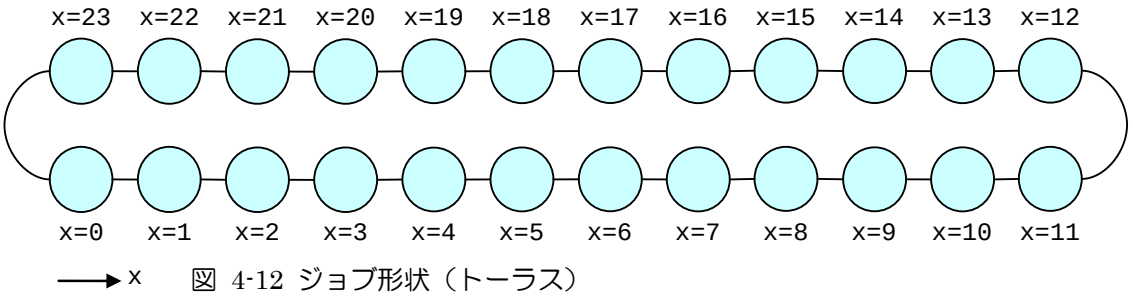


図 4-12 ジョブ形状（トーラス）

5. プログラミング支援ツール

本ツールは、アプリケーションプログラム開発の各種作業フェーズを支援する GUI 開発環境です。独自のファイルエクスプローラやエディタをはじめ、デバッガ、プロファイラなど高機能の開発ツールを実装しています。プログラミング支援ツールの主要な特徴を「表 5-1 プログラミング支援ツールの機能概要」に示します。利用方法の詳細は、「プログラミング支援ツールマニュアル」「デバッガ使用手引書」「プロファイラ使用手引書」を参照してください。

表 5-1 プログラミング支援ツールの機能概要

機能	内容
マネージャ機能	各機能の起動、各種メッセージの表示、サーバへのコマンドの投入機能などを行うメイン画面
プログラム作成支援機能	ファイルの作成/操作を行うファイルエクスプローラ、ファイル内容表示/編集のエディタ
アプリケーションビルド支援機能	Makefile ファイルの作成/実行を行うビルダ
アプリケーション実行支援機能	実行スクリプトの作成/実行を行うエグゼキュータ
デバッグ機能	Fortran77/90、C、C++コンパイラで作成された逐次アプリケーション、並列アプリケーション(スレッド並列、MPI)に対して使用可能な GUI デバッギングツール ・ アプリケーションの実行制御(実行中断、再開、ステップ・ネクスト実行) ・ アプリケーションのブレークポイントの設定、解除 ・ アプリケーションのウォッチポイントの設定、解除 ・ アプリケーションのバリアポイントの設定、解除(スレッド並列) ・ 変数値の表示、変更、変数値の自動表示設定などの変数操作 ・ スタックフレーム(呼び出し経緯)の表示とフレームの変更
プロファイリング機能	アプリケーションの実行性能情報を収集/解析するプロファイラ ・基本プロファイラ サンプリングによりプログラム全体のチューニング情報(コスト)を収集 - 経過時間、ユーザーCPU 時間、システム CPU 時間の内訳など - サンプリングに基づくコスト、同期待ちコスト、MPI ライブラリ通信コスト - アプリケーション実行時のプロセッサ動作状況 - 手続きの呼出し経路とコスト - ソースコードの各行にコスト情報を付加して出力 ・詳細プロファイラ カウンタにより、プログラムの測定区間のチューニング情報を収集 - 測定区間の呼び出し回数、経過時間、CPU 時間の内訳など - 測定区間の MPI ライブラリの実行情報 - 測定区間のハードウェアモニタ情報
トレーサ機能	MPI 関数トレース情報を収集/解析する機能

5.1 プログラミング支援ツールインストール

(1) 下記 URL にアクセスします。FX100 と CX とでツールが異なりますので、それぞれダウンロードをお願いします。

※過去バージョンをインストール済みの場合でも新規インストールが必要です。

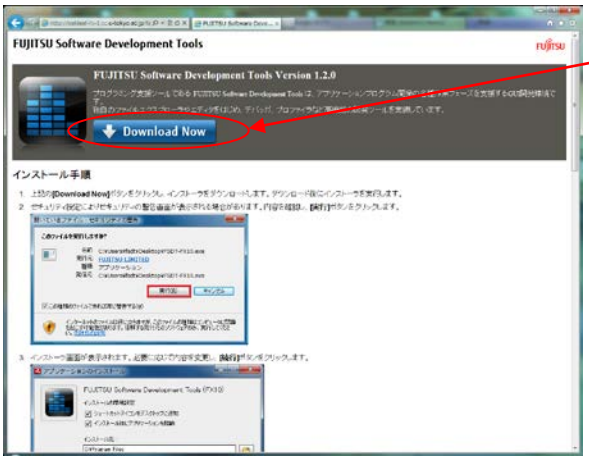
[FX100]

https://fx.cc.nagoya-u.ac.jp/fsdtfx100/install/index.html

[CX]

https://cx.cc.nagoya-u.ac.jp/fsdtpcc/install/index.html

(2) プログラミング支援ツールをインストールします。



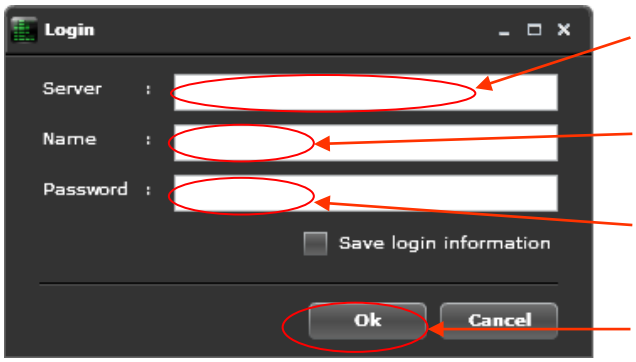
1. [Download Now] をクリックして、ダウンロードを開始する。

図 5-1 プログラミング支援ツールダウンロードサイト画面

ダウンロードサイトにインストール手順が記載されています。手順に従い、インストールを行ってください。

5.2 ツール起動方法

(1) FUJITSU Software Development Tools を起動します。(以下は FX100 の例です)



1. [server] に fx.cc.nagoya-u.ac.jp を入力します。
2. [Name] にユーザーアカウントを入力します。
3. [Password]にパスワードを入力します。
4. OK をクリックします。

図 5-2 プログラミング支援ツールログイン画面

(2) ツールのメニュー画面が起動します。

アイコンをクリックすると
各サービスが起動します。
(1)File explorer: ファイル操作
(2)Editor: ファイル編集
(3)Bulider: コンパイル/リンク
(4)Executer: ジョブ実行
(5)Debugger: デバッガ
(6)Profiler: プロファイラ

図 5-3 プログラミング支援ツールメニュー画面

5.3 ツール終了

メニュー画面上部の「x」ボタンをクリックすると、終了確認画面が表示されます。

図 5-4 プログラミング支援ツールメニュー画面

5.4 デバッガの利用

デバッガの制御下でアプリケーションを実行し、処理論理の検証などを行うことができます。

富士通コンパイラで作成した Fortran, C/C++ 言語の逐次アプリケーション、MPI アプリケーションおよび XPFortran アプリケーションに対して、次の操作が可能です。

- アプリケーションの実行制御
- アプリケーションの実行停止位置の設定
- 式および変数についての評価と表示
- 呼出しスタックの表示とフレームの変更

5.4.1 デバッガ利用の準備

デバッグするアプリケーションを翻訳する際に、-g -Ntl_trt の 2 つの翻訳時オプションを指定し、再コンパイルしてください。

※名古屋大学の環境ではデフォルトで設定されているため、明に指定する必要はありません。

```
[username@fx01:~] frtpx -g -Ntl_trt  sample.f
```

表 5-2 デバッグオプション一覧

オプション	説明
-g デ	デバッグ情報の生成を指示するオプションです。本オプションが指定されていないと、デバッグ中に変数の値を参照することや、ソースプログラムと対応を取ることができません。
-Ntl_trt K	ツールランタイムライブラリを結合するオプションです。本オプションを指定すると、アプリケーションの実行時にデバッグ機能、プロファイリング機能および MPI トレース機能を使用できます。

5.4.2 デバッガ利用方法(GUI)

利用支援ツールに含まれる GUI デバッガは、以下の 3 種類のデバッグが可能です。

- 通常デバッグ
  デバッガからジョブ投入し、プログラムの先頭から実行してデバッグする方法です。デバッグ中に、プログラムの式や変数の表示・実行制御・実行停止位置の設定などができます。
- コアファイルデバッグ
  ジョブが異常終了した場合に出力されるコアファイルを使用し、異常終了時の状態を静的に検証するデバッグです。
- ジョブ ID アタッチデバッグ
  実行中のジョブ ID を指定してジョブの全てのプロセスを補足します。

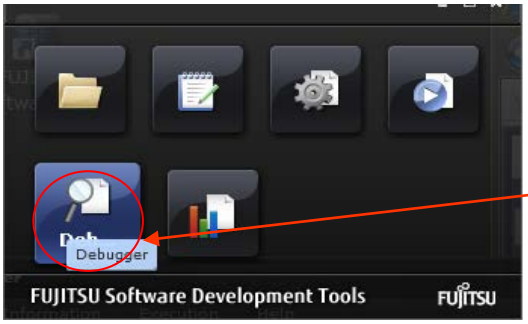
本手引書では通常デバッグの起動手順を説明します。デバッガの詳細な利用方法は「デバッガ使用手引書」を参照してください。

⑴ プログラミングコンパイル
デバッグする実行モジュールは必ずデバッグオプション “-g -Ntl_trt” を付与してコンパイル/リンクしてください。

⑵ デバッグジョブスクリプトの準備
実行モジュールを作成するデバッグ投入用ジョブスクリプトを作成します。
デバッグ投入用ジョブスクリプトは、あらかじめデバッグ用にコンパイルした実行モジュールを指定する必要があります。

```
[username@fx01:SSL2] vi sample21_dbg.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L  "rscgrp=fx-small"
#PJM -L  "node=1:mesh"
#PJM -L "elapse=10:00"
#PJM -j
#PJM -S
#------ program execution ------#
mpiexec ./dbg.out
```

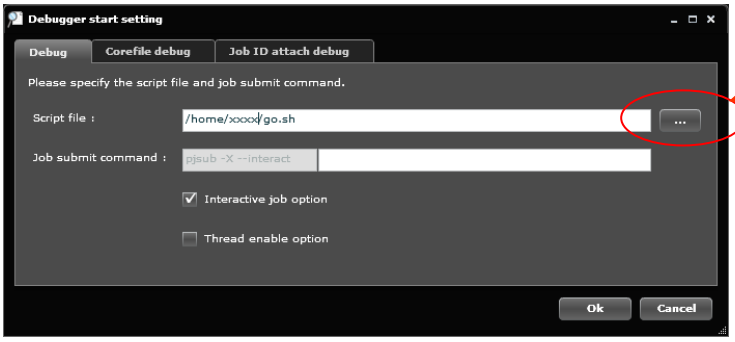
(3) デバッガツールを起動します。



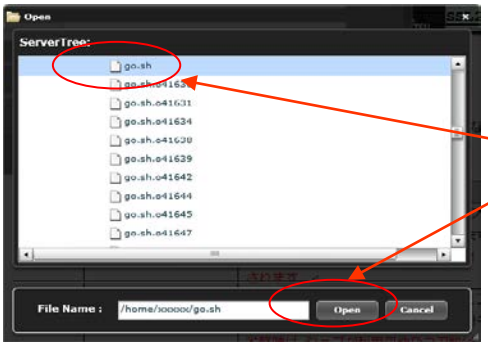
[Debugger]メニューをクリック

図 5-5 デバッガ起動

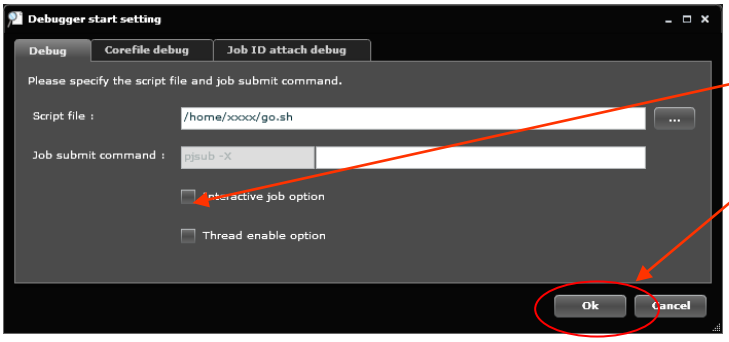
(4) デバッグ用ジョブスクリプトを選択し、投入します。



ファイル選択ボタンを
クリック



1. 投入するスクリプトファイルを選択
2. [Open]をクリック



1. [interactive job option] チェックボックスを外す
2. [OK]をクリック

図 5-6 デバッグ用スクリプトファイルの投入

(5) デバッグ開始

ジョブ開始後、デバッグ操作を行います。

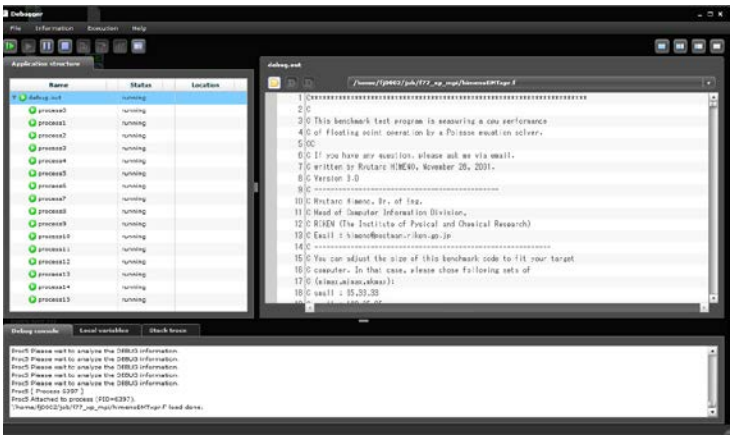


図 5-7 デバッガ画面

5.4.3 ブレークポイント、ウォッチポイント設定

デバッガではプログラムの任意の場所で実行を停止する機能を持っています。停止位置での停止後、

変数の値を表示するなど、プログラムが意図した動作をしているか確認できます。

また、停止位置は通常、デバッグ作業中は有効ですが、停止しないよう、一時的に無効にできます。

停止位置には次の種類があります。

- ブレークポイント

  プログラムのデバッグ中に意図的に一時停止させる箇所をブレークポイントといいます。

  MPIプログラムでは、プログラムの全プロセスに同じブレークポイントを設定した場合、個々のプロセスがブレークポイントに到達した時点で、プロセスの実行が一時停止します。

  一方、スレッドを含むプロセスに対するブレークポイントを設定した場合は、いずれかのスレッドがブレークポイントに達した時点で、プロセス内の各スレッドは実行を一時停止します。

  一度停止すると解除される「一時ブレークポイント」もあります。

- バリアポイント

  バリアポイントは、スレッドを含むプロセスに対してのみ有効な停止位置です。

  プロセス中のすべてのスレッドがバリアポイントに到達した時点で、実行を一時停止します。すべてのスレッドがバリアポイントに到達するまで、そのプロセスに対するデバッガの操作は行えません。

- ウォッチポイント

  特定の変数に着目し、変数にアクセス（参照、変更、参照と変更）された時点でプログラムの実行を一時停止させる設定をウォッチポイントといいます。

  なお、ウォッチポイントは変数アクセスを監視できる強力な機能ですが、利用するとメモリアクセスを監視するため、通常の実行よりも実行性能が悪くなりますので注意が必要です。

6. チューニング

6.1 チューニング概要

プログラムの実行がより短時間で終了するように、プログラムを改善することをチューニングと呼びます。プログラムをチューニングするには、チューニング情報の収集、性能評価・分析、プログラムの修正と性能測定などの一連の作業を実施します。

一般に、プログラムの中で、多くの実行時間が費やされている箇所を見つけ出して、その部分を高速化すると、大きなチューニング効果を得ることができます。

- プログラム内で実行時間計測サブルーチンの呼び出し
- バッチジョブ統計情報取得オプションの指定
- プロファイラの指定

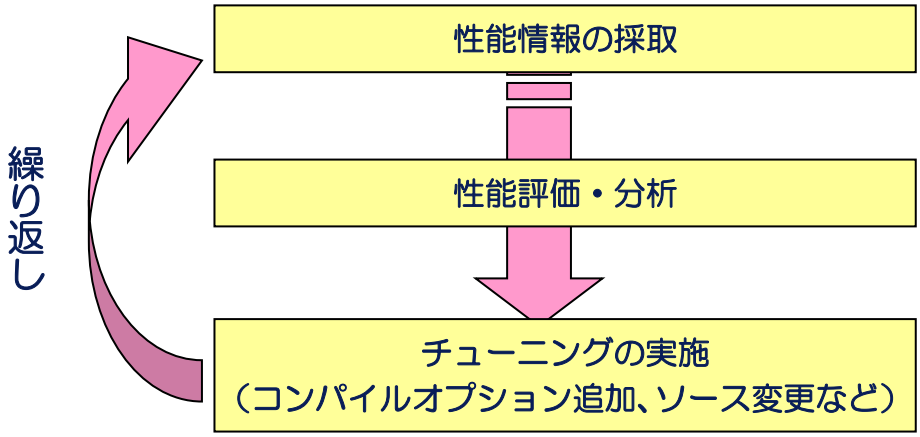


図 6-1 チューニング手順概要

6.2 プロファイラ

富士通製コンパイラでコンパイル／リンクを行った実行モジュールは、プロファイラ機能を利用するために必要なツールライブラリがデフォルトでリンクされます。これらのライブラリを用いて、実行モジュールの性能をプロファイリング可能です。プロファイラは、データ採取の方法の違いにより、基本プロファイラと詳細プロファイラの 2 種類があります。

本項では以下のプロファイラに関する利用方法を説明します。「プロファイラ使用手引書」も併せてご参照ください。

表 6-1 基本／詳細プロファイラ

種別	収集	表示	説明
基本プロファイラ	fipp -C	fipppx GUI	ユーザープログラムに対し一定間隔(デフォルト時 100 ミリ秒間隔)毎に割り込みをかけ情報を収集します。収集した情報を基に、時

			間統計情報、コスト情報、等の分析結果を表示します。
詳細プロファイラ	fapp -C	fapppx GUI	アプリケーションの指定した区間の実行性能情報の収集および出力を行うことができます。収集した情報を基に、測定区間の呼出し回数、経過時間、ユーザ CPU 時間、およびシステム CPU 時間の内訳、MPI ライブラリ実行情報等の詳細な分析結果を表示します。

6.2.1 基本プロファイラ

基本プロファイラは、サンプリングによる情報収集に基づいたコスト分析をします。

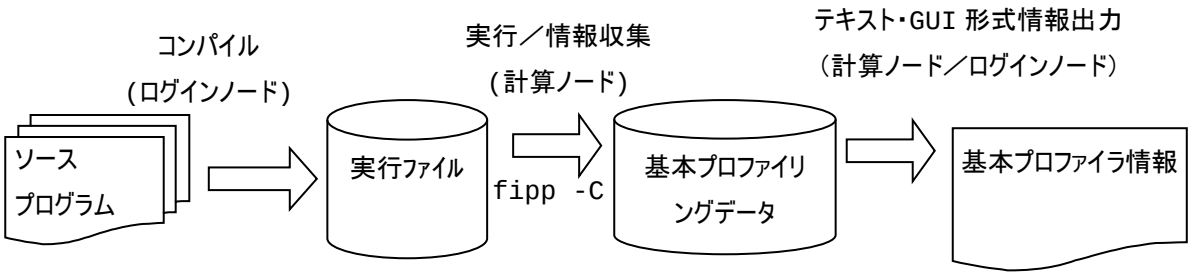


図 6-2 基本プロファイラ概要

6.2.1.1 プロファイリングデータ収集

fipp -C コマンドを使用して、プログラムのサンプリングデータを収集します。

```
fipp -C -d <dir> [option] ./a.out          (逐次/スレッドジョブ)
fipp -C -d <dir> [option] mpiexec ./aout (MPI ジョブ)
```

表 6-2 主要オプション(fipp コマンド)

オプション		意味
-C		基本プロファイリングデータの収集処理を行うことを指定
-d dir_data		サンプリングデータの保存先ディレクトリ名を指定 (実行前に指定したディレクトリは内容が空でなければならない)
-I		収集する基本プロファイラ情報の項目を指定
	call|nocall	コールグラフ情報収集を指定 [call:収集する|nocall:収集しない(default)]
	hwm|nohwm	ハードウェアモニタ情報収集を指定 [hwm:収集する|nohwm:収集しない(default)]
-i <*interval*>		サンプリング間隔(単位:ミリ秒)を指定する。デフォルト値は-i 100 interval : 整数値(ミリ秒単位) 指定可能範囲 1～3,600,000
-H		ハードウェアモニタ情報の測定を指示します。
	mode={sys|usr}	測定モードを指定 sys: カーネルモードおよびユーザーモードの情報収集を指定 usr: ユーザーモードの情報収集を指定

6.2.1.2 プロファイリングデータ参照

FX100 ログインノードで、fipppx コマンドを使用して、収集したプロファイリングデータを CUI 形式で表示します。

CX ログインノードでは、fipp コマンドになります。

```
$ fipppx -A -d <dir> [option]
```

表 6-3 主要オプション(fipppx コマンド)

オプション		意味
-A		基本プロファイラ情報の出力処理を指定
-d dir_data		基本プロファイリングデータ名 (基本プロファイリングデータファイルを格納するディレクトリ名)を、相対パス、または絶対パスで指定します。 dir_data に“-”で始まる基本プロファイリングデータ名を指定する場合は、絶対パス、またはカレントディレクトリ(“./”)を含む相対パスで指定してください。本オプションを、fipp コマンドのオプション並びの最後に指定する場合は、-d を省略することができます。
-I		出力する項目を指定
	cpu|nocpu	コスト情報出力を指定 [cpu:出力する|nocpu:出力しない](default)
	balance|nobalance	並列実行単位間のコストバランス出力を指定 [balance:出力する|nobalance:出力しない(default)]
	call|nocall	コールグラフ情報出力を指定 [call:出力する|nocall:出力しない(default)]
	hwm|hwm	ハードウェアモニタ情報の出力を指定 [hwm:出力する|nohwm:出力しない(default)]
	src[:path]…| nosrc	ソースコード情報を出力するかどうかを指定 [src[:path]...:出力する|nosrc:出力しない(default)]
-o <outfile>		出力するファイル名を指定(default: -ostdout)
-p p_no		基本プロファイラ情報の入出力対象プロセス(p_no)を指定
	N[,N]… all limit=n	N[,N]…: スレッド番号 N の情報を出力 all: 全スレッド情報を出力 limit=n: 指定したスレッド番号、高コストの上位順に N 件の情報を出力

6.2.1.3 カウンタ測定範囲の指定

基本プロファイラは、全プログラムのコスト情報を測定しますが、特定の区間を指定して測定する場合には、コストを測定する開始位置と終了位置にサブルーチンを挿入します。

表 6-4 測定開始／終了指定関数

言語種別	ヘッダファイル	関数名	機能	引数*1
Fortran	なし	fipp_start	コスト情報測定開始	なし
		fipp_stop	コスト情報測定終了	なし
C／C++	fj_tool/fipp.h	void fipp_start	コスト情報測定開始	なし
		void fipp_stop	コスト情報測定終了	なし

```
[username@fx01:~] vi test.c
 #include <fj_tool/fipp.h>                    ヘッダファイルをインクルードする
 (略)
   printf ("%d, %d¥n", N, N);
   void fipp_start();                              測定開始位置
   for (j = 0; j < N; j++) {
      for (i = 0; i < N; i++) {
         a[j][i] = 0.1;                    測定範囲
         b[j][i] = 0.3;
      }
   }
   void fipp_stop();                               測定終了位置
```

6.2.1.4 測定方法

(1) コンパイル／リンク

プロファイラに必要なライブラリを結合するため、以下の方法でコンパイル/リンクします。測定区間を指定したい場合は、「6.2.1.3 カウンタ測定範囲の指定」を参照してください。

※名古屋大学の環境ではデフォルトで設定されているため、明に指定する必要はありません。

```
[username@fx01:~] fccpx -Ntl_trt test.c
[username@fx01:~]
```

(2) 測定／収集

fipp コマンドを実行するスクリプトファイルの例を示します。ジョブ終了時、指定ディレクトリ(testdir)に結果が出力されます。

```
[username@fx01:Fortran] vi sample22_prof.sh
 #!/bin/bash -x
 #------ pjsub option --------#
 #PJM -L "rscgrp=fx-small"
 #PJM -L "node=1:mesh"
 #PJM -L "elapse=10:00"
 #------ program execution ------#
 fipp -C -d prof ./a.out

[username@fx01:~] pjsub sample22_prof.sh
[INFO] PJM 0000 pjsub Job 750 submitted.
```

6.2.1.5 結果確認(CUI)

FX の場合、fipppx コマンド実行時に、取得したディレクトリを指定して基本プロファイラを起動します。CX の場合、fipp -A コマンドで基本プロファイリングデータのテキスト出力を行います。

```
[username@fx01:~] fipppx -A -d prof
------------------------------------------------------------------------
Fujitsu Instant Profiler Version 1.2.0
   Measured time              : Fri Jun 22 15:51:09 2013
   CPU frequency              : Process     0  1650 (MHz)
   Type of program            : SERIAL
   Average at sampling interval  : 100.0 (ms)
   Measured range             : All ranges
------------------------------------------------------------------------
________________________________________________________________________
Time statistics

        Elapsed(s)       User(s)     System(s)
   ------------------------------------------
           0.0027       0.0000       0.0000   Application
   ------------------------------------------
           0.0027       0.0000       0.0000   Process     0
________________________________________________________________________
```

6.2.1.6 結果確認(GUI)

プログラミング支援ツールを使って、GUI で基本プロファイラを起動可能です。基本プロファイラでは測定結果の可視化が可能です。プログラミング支援ツールの起動方法は「5.2 ツール起動方法」を参照してください。

また、基本プロファイラ GUI ツールの利用詳細は「プロファイラ使用手引書 2.3 章 基本プロファイラ情報」を参照してください。

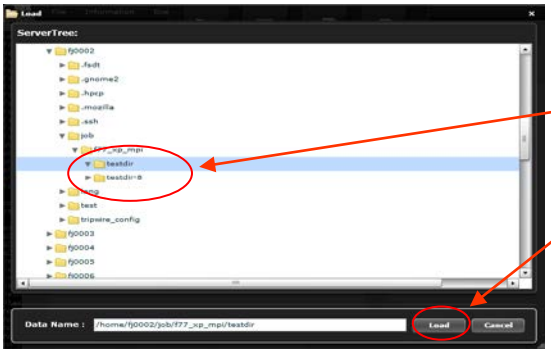
以下にプロファイラの起動方法を示します。

(1) プロファイラを起動します。

[Profiler]メニューをクリック

図 6-3 プロファイラ起動

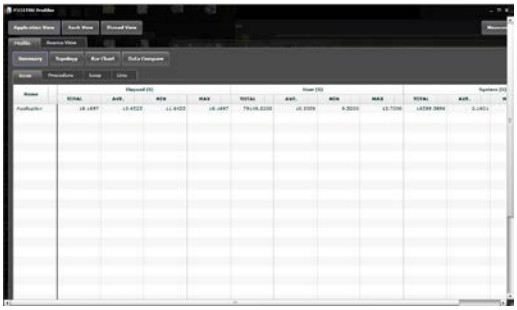
(2) プロファイラデータの格納ディレクトリを指定します。



1. プロファイラデータが格納されているディレクトリを選択
2. [load]をクリック

図 6-4 プロファイラディレクトリ選択

(3) プロファイラが起動し、操作が可能となります。



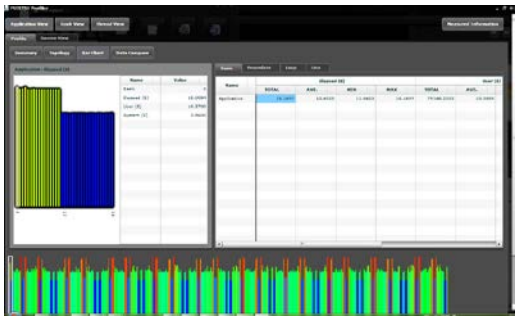


図 6-5 プロファイラ表示

6.2.2 詳細プロファイラ

詳細プロファイラは、CPU の PA カウンタの値を計測し、詳細なプログラムの動作を分析します。fapp コマンドにより収集したプロファイリングデータを、専用の GUI ツールで表示し、分析します。

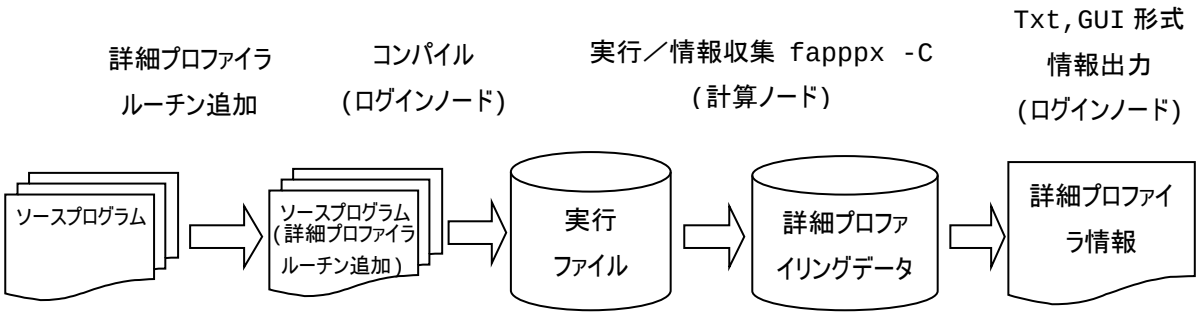


図 6-6 詳細プロファイラ概要

6.2.2.1 プロファイリングデータ収集

fapp コマンドを使用して、プログラムの詳細プロファイリングデータを収集します。

```
fapp -C -d <dir> [option] ./a.out (逐次/スレッドジョブ)
fapp -C -d <dir> [option] mpiexec ./a.out (MPI ジョブ)
```

表 6-5 主要オプション(fapp コマンド)

オプション	意味
-C	詳細プロファイリングデータの収集処理を行うことを指定
-d dir_data	サンプリングデータの保存先ディレクトリ名を指定 (ディレクトリが存在しない場合は新規に作成されます。ディレクトリが存在する場合は、内容が空でなければなりません)
-I item	サンプリングで測定する項目を指定
mpi|nompi	MPI 情報を収集するかどうかを指定 ・mpi ： MPI 情報を収集 ・nompi ： MPI 情報を収集しない 逐次プログラムの場合、mpi は指定できません。 本オプションの省略値は、次のとおりです。 ・MPI アプリケーションの場合 ： mpi ・逐次アプリケーションの場合 ： nompi

	hwm|nohwm	ハードウェアモニタ情報収集を指定 [hwm:収集する|nohwm:収集しない(default)]
-H item		サンプリングで測定する項目を指定
	event={ AVX | Cache | TLB | Statistics }	測定イベントを指定 ・AVX : CPU core 動作状況(AVX 命令) ・Cache : キャッシュミス率 ・TLB : TLB ミス率 ・Statistics: CPU core 動作状況 (default)
	mode={sys|usr}	測定モードを指定 sys: カーネルモードおよびユーザーモードの情報収集を指定(default) usr: ユーザーモードの情報収集を指定

6.2.2.2 測定範囲の指定

詳細プロファイラで測定するためには、測定する範囲を指定する必要があります。
測定範囲を指定する関数を示します。

表 6-6 測定開始／終了指定関数

言語種別	ヘッダファイル	関数名	機能	引数*1
Fortran	なし	fapp_start	情報測定開始	name,number,level
		fapp_stop	情報測定終了	name,number,level
C／C++	fj_tool/fapp.h	void fapp_start	情報測定開始	const char *name, int number, int level
		void fapp_stop	情報測定終了	const char *name, int number, int level

6.2.2.3 プロファイラ測定

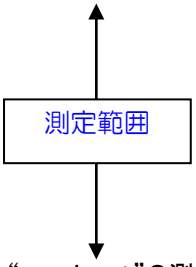
詳細プロファイラの利用手順を示します。

(1) 測定範囲指定

プログラムに測定範囲を指定します。

【サンプルプログラム】

```
[username@fx01:~] vi test.c
#include <stdio.h>
#include <fj_tool/fapp.h>                    ヘッダファイルをインクルードする
#define SIZE
int main(){
(略)
   printf ("%d, %d¥n", N, N);
   void fapp_start("region1",1,1);           “region1”の測定開始位置
   for (j = 0; j < N; j++) {
      for (i = 0; i < N; i++) {
         a[j][i] = 0.1;
         b[j][i] = 0.3;
      }
   }
   void fapp_stop("region1",1,1);            “region1”の測定終了位置
(略)
```



(2) コンパイル／リンク

詳細プロファイラ機能に必要なツールライブラリを結合するために、以下のようにコンパイル／リンクします。

※名古屋大学の環境ではデフォルトで設定されているため、明に指定する必要はありません。

```
[username@fx01:~] fccpx -Ntl_trt test.c
```

(3) 測定/収集

プログラムのカウンタデータを収集する場合は、fapp コマンドを使用します。

```
[username@fx01:Fortran] vi prof.sh
 #!/bin/bash -x
 #------ pjsub option --------#
 #PJM -L "rscgrp=fx-small"
 #PJM -L "node=1:mesh"
 #PJM -L "elapse=10:00"
 #------ program execution ------#
 fapp -C -d prof2 ./a.out
 [username@fx01:~] pjsub sample23_prof.sh
[INFO] PJM 0000 pjsub Job 753 submitted.
```

6.2.2.4 結果確認(CUI)/プロファイリングデータ参照

FX100 ログインノードで、fapppx コマンドを使用して収集したプロファイリングデータを CUI 形式で表示します。CX ログインノードでは、fapp コマンドを使用して収集しますが、CUI で表示できません。GUI での表示方法は、6.2.2.5　結果確認(GUI)をご参照ください。(2017.01.20 追加修正)

```
$ fipppx -A -d <dir> [option]
```

```
[username@fx01:Fortran] $ fapppx -A -d prof2
-------------------------------------------------------------------------
Fujitsu Advanced Performance Profiler Version 1.2.0
   Measured time               : Mon Sep 23 18:36:46 2013
   CPU frequency               : Process     0  1848 (MHz)
   Type of program             : Thread (OpenMP)  16
-------------------------------------------------------------------------
Basic profile
 *************************************************************************
       Application
 *************************************************************************

     Kind   Elapsed(s)     User(s)    System(s)        Call
    ------------------------------------------------------------
       AVG      0.1862      2.0500      0.0600      1.0000    all 0
       MAX      0.1862      2.0500      0.0600           1
       MIN      0.1862      2.0500      0.0600           1

*************************************************************************
       Process 0
*************************************************************************

            Elapsed(s)     User(s)    System(s)        Call
    ------------------------------------------------------------
               0.1862      2.0500      0.0600           1    all 0
```

6.2.2.5 結果確認(GUI)

詳細プロファイラは、プログラミング支援ツールを使って起動します。詳細プロファイラでは測定結果の可視化が可能です。プログラミング支援ツールの起動方法は「5.2 ツール起動方法」を参照してください。また、詳細プロファイラGUIツールの利用詳細は、「プロファイラ使用手引書 3.3章 詳細プロファイラ情報」を参照してください。

(1) プロファイラを起動します。

図 6-7 プロファイラ起動

(2) プロファイラデータの格納ディレクトリを指定します。

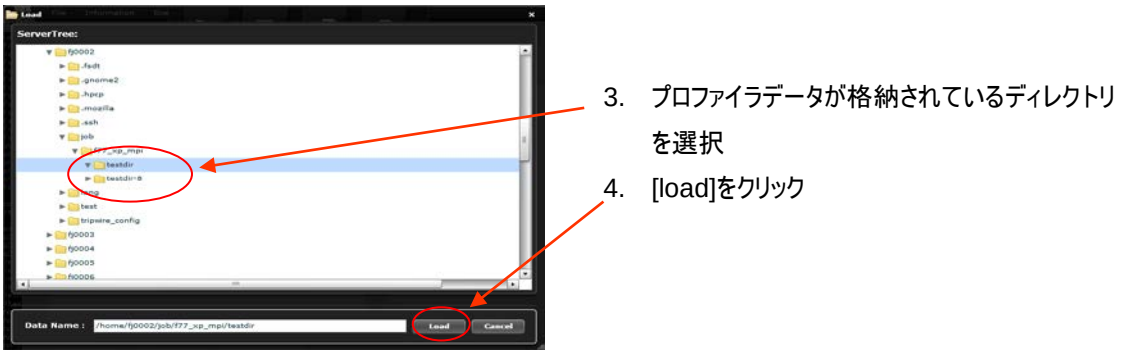


図 6-8 プロファイラディレクトリ選択

(3) プロファイラが起動し、操作が可能となります。

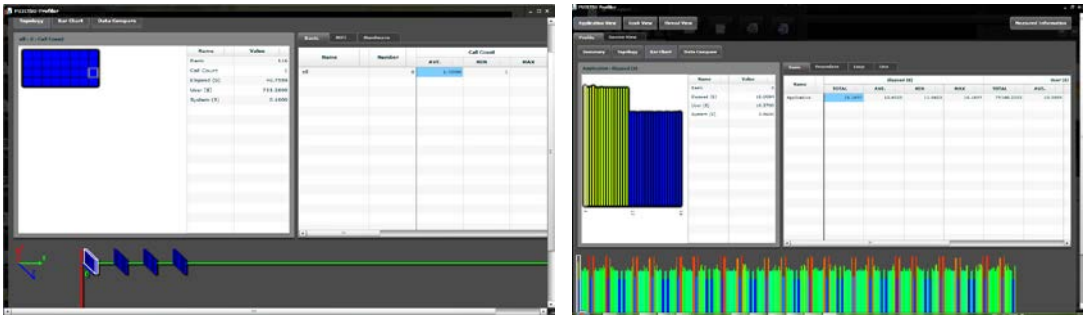


図 6-9 プロファイラ表示

7. ファイル転送

7.1 システムへのファイル転送(Windows 環境)

Windows で使用できる SCP クライアントソフトには WinSCP などがあります。WinSCP は推奨ターミナルソフトである PuTTY と同じ鍵を使用できるので、WinSCP を推奨 SCP クライアントソフトとし接続方法を説明します。

WinSCP は以下のサイトからダウンロードすることができます。

WinSCP: http://winscp.net/eng/docs/lang:jp

7.1.1 鍵の作成

アクセス元端末(PC/WS)にて、秘密鍵/公開鍵ペアを作成します。

「1.7.1　鍵の作成」を参考に秘密鍵/公開鍵ペアを作成します。すでに鍵を作成済みの場合は、作業を行う必要はありません。

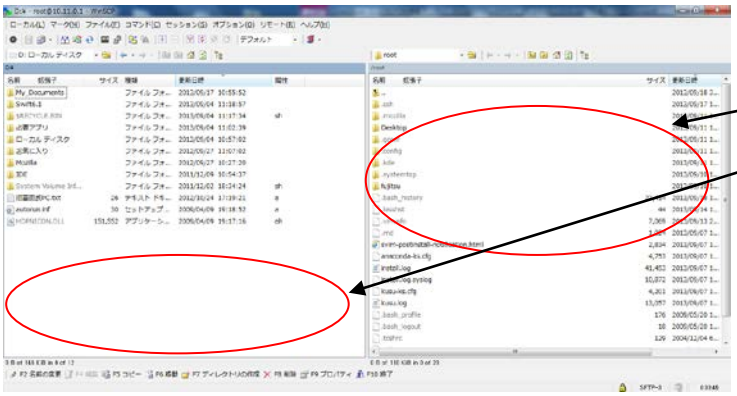
7.1.2 公開鍵登録

公開鍵の登録は、HPC ポータル(https://portal.cc.nagoya-u.ac.jp/)を利用ください。

7.1.3 ファイル転送

1. ホスト名を入力
2. ユーザー名を入力
3. パスワードを入力
4. PuTTY で作成した秘密鍵を指定
5. プロトコルを SCP に設定
6. [保存] をクリック
7. [ログイン] をクリック

図 7-1WinSCP 画面(1)



1. 必要なファイルをドロップ(ファイル配置)
2. 必要なファイルをドロップ(ファイル採取)

図 7-2WinSCP 画面(2)

7.2 システムへのファイル転送(Linux 環境)

7.2.1 鍵の作成

「1.8.1　鍵の作成」を参考に秘密鍵/公開鍵ペアを作成します。すでに鍵を作成済みの場合は、作業を行う必要はありません。

7.2.2 ログイン

「1.8.2　ログイン」を参考にシステムへログインします。

7.2.3 ファイル転送

■ファイル採取(クライアントから testfile1 を採取)
```
[username@fx01 :~]$ scp -P 22 username@[クライアント名]:/tmp/testfile1 ./
```
username@クライアント名's password: ****** (クライアントのパスワードを入力)
testfile1 100% 32 0.0KB/s 00:00

■ファイル配置(クライアントへ testfile2 を配置)
```
[username@fx01 :~]$ scp -P 22 ./testfile2 username@[クライアント名]:/tmp
```
username@クライアント名's password: ****** (クライアントのパスワードを入力)
testfile2 100% 32 0.0KB/s 00:00

8. vSMP

8.1 vSMP の利用方法

8.1.1 ログイン

vSMP システムへログインするには、ssh サービスを利用しvSMP ログインノードへログイン後、ログインノード経由で vSMP システムにログインします。

(1) 指定されたログインユーザーを使用し、vSMP ログインノードのアドレス(133.6.1.150)に ssh 接続を行います。

(2) vSMP ログインノードにログイン後、指定されたログインユーザーを使用し、vSMP システムのアドレスに ssh 接続を行います。

$ ssh [ユーザー名]@[vSMP システムのアドレス]

vSMP 構成ノード	vSMP システムのアドレス	CPU 数 (コア数)	物理メモリ容量	実効メモリ容量
12 ノード	10.30.0.4	24 (288)	1536GB	1075GB
	10.30.0.22	24 (288)	1536GB	1075GB

(3) vSMP システムにログイン後、以下のコマンドで CPU・メモリの情報が確認できます。

$ vsmpversion --full

出力例：

```
[root@cx7-001 ~]$ vsmpversion --full
vSMP Foundation: 5.1.135.49 (Aug 29 2013 00:29:33)

System configuration:
    Boards:        46                  b0:00.0#1=>09:10.11.12.13.14.15.16
                                                17.18
                                           09-19-35:01.02.03.04.05
                                                06.07.08.09.10
                                                 11.12.13.14.15
                                                16.17.18
                                           09-19-36:01.02.03.04.05
                                                06.07.08.09.10
                                                11.12.13.14.15
                                                16.17.18
    Processors:  92, Cores: 1104
        Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz Stepping 04
    Memory (MB): 5149197 (out of 6027564), Cache: 449311, Private: 429056
    Link Rate:    40Gb/s
    Boot device: [HDD0] ATA WDC WD5003ABYX-5
License server: 10.15.30.1:5053 (Serial number: 16922360) - Active
[root@cx7-115 ~]$
```

(4) vSMP システムからログアウトする場合は、exit コマンドで vSMP ログインノードへ戻ります。

$ exit

(5) vSMP ログインノードからログアウトする場合は、exit コマンドでログアウトします。

$ exit

8.1.2 Technical Computing Language

Technical Computing Language は、Fortran、C 言語、C++、または並列プログラム言語 XPFortran による、高性能な並列アプリケーションプログラムの開発から実行までを支援するソフトウェアです。

本項では、vSMP システムでの Technical Computing Language の利用方法(実行例)を示します。

8.1.2.1 Fortran

⑴ サンプルプログラムをコピーします。

```
$ cd [作業用ディレクトリ]
$ mkdir Fortran
$ cd Fortran
$ cp /opt/FJSVpclang/1.2.0/sample/Fortran/* ./
```

⑵ 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

```
$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH
$ export PATH
$  LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
```

⑶ サンプルプログラムの翻訳と結合を行います。

```
$ frt normal_end.f95
$ ls -l
```

a.out が作成されていることを確認します。

-rwxr-xr-x 1 fj-lang fj-se 19213　8月 30 10:07 2013 a.out

⑷ 実行

```
$ ./a.out
```

以下のように出力されることを確認します。

１０１０″a″ ″x″ ″a″ ″x″

″Fujitsu Fortran system OK″

8.1.2.2 C コンパイラ

⑴ サンプルプログラムをコピーします。

```
$ cd [作業用ディレクトリ]
$ mkdir C
$ cd C
$ cp /opt/FJSVpclang/1.2.0/sample/C/sample.c ./.
```

⑵ 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

```
$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH
$ export PATH
$  LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH
```

$ export LD_LIBRARY_PATH

(3) サンプルプログラムの翻訳と結合を行います。

$ fcc sample.c

$ ls -l

a.out が作成されていることを確認します。

-rwxr-xr-x 1 fj-lang fj-se 16430　8月 30 10:15 2013 a.out

(4) 実行

$ ./a.out

以下のように出力されることを確認します。

Fujitsu C Compiler:

　OK

8.1.2.3 C++コンパイラ

(1) サンプルプログラムをコピーします。

$ cd [作業用ディレクトリ]

$ mkdir C++

$ cd C++

$ cp /opt/FJSVpclang/1.2.0/sample/C++/sample.cc ./.

(2) 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH

$ export PATH

$　LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH

$ export LD_LIBRARY_PATH

(3) サンプルプログラムの翻訳と結合を行います。

$ FCC sample.cc

$ ls -l

a.out が作成されていることを確認します。

-rwxr-xr-x 1 fj-lang fj-se 16430　8月 30 10:15 2013 a.out

(4) 実行

$ ./a.out

以下のように出力されることを確認します。

Fujitsu C++ Compiler:

　OK

8.1.2.4 MPI

別紙ドキュメント
「vSMP 上で Technical Computing Language を使用する場合の注意事項について」の
以下の項を参照し vSMP 環境下での環境設定を行います。

■マニュアルについて ■使用上の注意

⑴ サンプルプログラムをコピーします。

```
$ cd［作業用ディレクトリ］
$ mkdir MPI
$ cd MPI
$ cp /opt/FJSVpclang/1.2.0/sample/MPI/* ./.
```

⑵ 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

```
$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH
$ export PATH
$  LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
```

⑶ スレッド数を指定します。

```
$ OMP_NUM_THREADS=3
$ export OMP_NUM_THREADS
$ PARALLEL=3
$ export PARALLEL
```

⑷ 環境変数 OMPI_USE_ORTED に 1 文字以上の英数字を設定します。

```
$ OMPI_USE_ORTED=1
$ export OMPI_USE_ORTED
```

⑸ サンプルプログラムの翻訳と結合を行います

```
$ taskset 5 ./compsample1.sh
$ ls -l
```

sample1.out が作成されていることを確認します。 -rwxr-xr-x 1 fj-pa fj-se　18178　9月 13 08:48 2013 sample1.out

⑹ 実行

$ numactl --physcpubind=all mpiexec -n 64 ./sample1.out

以下のように出力されることを確認します。

MPI communication start. size=64

MPI communication end

result is 0+1+...size-1.check result(2016)

8.1.2.5 XPFortran トランスレータ

⑴ サンプルプログラムをコピーします。

$ cd [作業用ディレクトリ]

$ mkdir XPFortran

$ cd XPFortran

$ cp /opt/FJSVpclang/1.2.0/sample/XPFortran/* ./.

⑵ 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH

$ export PATH

$ LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH

$ export LD_LIBRARY_PATH

⑶ サンプルプログラムの翻訳と結合を行います。

$ ./compxpf.sh

$ ls -l

sampxpf.out が作成されていることを確認します。

-rwxr-xr-x 1 fj-pa fj-se 82910 9月 13 19:24 2013 sampxpf.out

⑷ 環境変数 OMPI_USE_ORTED に 1 文字以上の英数字を設定します。

$ OMPI_USE_ORTED=1

$ export OMPI_USE_ORTED

⑸ 実行

$ mpiexec -n 4 ./sampxpf.out

以下のように出力されることを確認します。

*** XPFortran sample program ***

result = 5050 5050

OK!

8.1.2.6 SSLⅡ

(1) サンプルプログラムをコピーします。

```
$ cd [作業用ディレクトリ]
$ mkdir SSL2
$ cd SSL2
$ cp /opt/FJSVpclang/1.2.0/sample/SSL2/samps.f ./.
$ cp /opt/FJSVpclang/1.2.0/sample/SSL2/comps.sh ./.
```

(2) 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

```
$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH
$ export PATH
$  LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
```

(3) サンプルプログラムの翻訳と結合を行います。

```
$ ./comps.sh
$ ls -l
```

samps が作成されていることを確認します。

```
-rwxr-xr-x 1 fj-lang fj-se 86435  8月 30 10:26 2013 samps
```

(4) 実行

```
$ ./samps
```

以下のように出力されることを確認します。

```
**************************************************************************
*                                                                        *
* --- VPST2#DLAX ---                    DATE 13-09-10                    *
*                                                                        *
  ---------------------------------- 中略 ----------------------------------
100   0.2597E+00   1001       0   0.34E-15   0.0000E+00   1001       0   0.34E-15
OK
128   0.5495E+00   1001       0   0.30E-15   0.9990E-02   1001       0   0.30E-15
OK
150   0.8691E+00   1001       0   0.87E-15   0.1998E-01   1001       0   0.87E-15
OK
200   0.2158E+01   1001       0   0.48E-15   0.2997E-01   1001       0   0.48E-15
OK

                                    *** END OF TEST ***
```

8.1.2.7 C-SSLⅡ

(1) サンプルプログラムをコピーします。

```
$ cd [作業用ディレクトリ]
$ mkdir CSSL2
$ cd CSSL2
$ cp /opt/FJSVpclang/1.2.0/sample/CSSL2/sampc.c ./.
$ cp /opt/FJSVpclang/1.2.0/sample/CSSL2/compc.sh ./.
$ cp /opt/FJSVpclang/1.2.0/sample/CSSL2/compcpp.sh ./.
```

(2) 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

```
$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH
$ export PATH
$  LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
```

(3) サンプルプログラムの翻訳と結合を行います。

```
$ ./compc.sh
$ ./ compcpp.sh
$ ls -l
```

sampc、sampcpp が作成されていることを確認します。

```
-rwxr-xr-x 1 fj-lang fj-se 1935767  8月 30 10:29 2013 sampc
-rwxr-xr-x 1 fj-lang fj-se 1939337  8月 30 10:29 2013 sampcpp
```

(4) 実行

```
$ ./sampc
```

以下のように出力されることを確認します。

```
**********************************************************************
*                                                                    *
*  --- c_dvlax ---                                                   *
  ---------------------------------- 中略 ----------------------------------
      dimension       error          time(s)    remark
          100         4.26e-15         0.000      OK
          200         4.17e-15         0.000      OK
          300         6.25e-15         0.000      OK
          400         6.82e-15         0.010      OK
          500         8.75e-15         0.010      OK

                                  *** end of test ***
```

$ ./sampcpp

以下のように出力されることを確認します。

```
**************************************************************
*                                                            *
*  --- c_dvlax ---                                           *
  ------------------------------------ 中略 -------------------------------------
      dimension       error          time(s)    remark
          100         4.26e-15         0.000      OK
          200         4.17e-15         0.000      OK
          300         6.25e-15         0.000      OK
          400         6.82e-15         0.010      OK
          500         8.75e-15         0.010      OK

                                  *** end of test ***
```

8.1.2.8 SSLⅡ/MPI

(1) サンプルプログラムをコピーします。

$ cd［作業用ディレクトリ］
$ mkdir SSL2MPI
$ cd SSL2MPI
$ cp /opt/FJSVpclang/1.2.0/sample/SSL2MPI/samps.f ./.
$ cp /opt/FJSVpclang/1.2.0/sample/SSL2MPI/comps.sh ./.

(2) 環境変数 PATH、LD_LIBRARY_PATH の設定を行います。

$ PATH=/opt/FJSVpclang/1.2.0/bin:$PATH
$ export PATH
$ LD_LIBRARY_PATH=/opt/FJSVpclang/1.2.0/lib64:/usr/lib64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH

(3) サンプルプログラムの翻訳と結合を行います。

$ ./comps.sh
$ ls -l

samps が作成されていることを確認します。

```
-rwxr-xr-x 1 fj-pa fj-se 3833061  9月 13 19:36 2013 samps
```

(4) スレッド数、サイズを指定します。

$ OMP_NUM_THREADS=1
$ export OMP_NUM_THREADS

```
$ THREAD_STACK_SIZE=64000
$ export THREAD_STACK_SIZE
```

⑸ 環境変数 OMPI_USE_ORTED に 1 文字以上の英数字を設定します。

```
$ OMPI_USE_ORTED=1
$ export OMPI_USE_ORTED
```

⑹ 実行

```
$ mpiexec -n 4 ./samps
```

以下のように出力されることを確認します。

```
**********************************************************
*                                                        *
*  --- ds_v3dcft ---                                     *
*                                                        *
*  if sign of 'ok' is found in every 'remark' entry      *
*  the above subroutine have been certified as correct   *
*                                                        *
**********************************************************


      n1        n2        n3            error         remark
     512       512       512          0.178D-14         ok


                                    *** end of test ***
```

9. Intel コンパイラ・Xeon Phi 利用について

9.1 Intel コンパイラ

CX システムでは富士通コンパイラの他、Intel コンパイラが利用できます。

CX2550M1/CX270 ともに利用可能です。CX2550 システムはログインノードと計算ノードで異なるアーキテクチャですが、同じコンパイラが利用可能です。ただし、計算ノードの性能を最大限引き出すため、かつ、ログインノードでコンパイルする場合、「-xCORE-AVX2」の指定が必要（効果はプログラムに依存）です。

表 9-1-1 コンパイラ環境(CX2550)

コンパイラ	ログインノード	計算ノード
Intel コンパイラ	○※1	○

※1　計算ノードの性能を最大限利用するには「-xCORE-AVX2」の指定が必要

表 9-1-2 コンパイラ環境(CX270)

コンパイラ	ログインノード	計算ノード
Intel コンパイラ	○	○

9.1.1 コンパイル／リンクの概要

コンパイル／リンクの書式とコマンド一覧は以下のとおりです。

```
コマンド　[option]　sourcefile [...]
```

表 9-2 コンパイル／リンクコマンド一覧

	言語処理系	コマンド名	OpenMP 注1	AVX2 命令 注2
非並列 (非 MPI)	Fortran90	ifort	-openmp	-xCORE_AVX2
	C	icc		
	C++	icpc		
並列 (MPI)	Fortran90	mpiifort		
	C	mpiicc		
	C++	mpiicpc		

注 1:　OpenMP オプションはデフォルトでは無効です。

注 2:　AVX2 命令はデフォルトでは無効です。
ログインノードでコンパイルし、CX2550 に対してジョブ投入する場合計算ノードの性能を最大限利用するには指定が必要です。
ただし、-xCORE-AVX2 を指定する場合、他のオプションよりも後ろで指定してください。他のオプションよりも前に指定した場合、-xCORE-AVX2 が無効になる

ことがあります。

また、コンパイラ環境は以下にインストールされています。

・/center/local/apl/cx/intel 配下

9.1.2 環境設定

ログイン直後は、富士通コンパイラの環境が設定されています。Intel コンパイラをご利用の前に、Intel コンパイラの環境変数の設定が必要になります。

Intel コンパイラのバージョンは４種類（Ver.2013, Ver.2015, Ver.2018, Ver.2019）あります。標準は Ver.2018 です。2019.06.14 更新

●Ver.2019 の場合　2019.06.14 更新

以下のコマンドを実行してください。実行後はログアウトされるまでは有効です。

詳細実行

```
$ source /center/local/apl/cx/intel_2019/compilers_and_libraries_2019/linux/bin/compilervars.sh intel64
```

MPI プログラムをご利用の前には、以下のコマンドを実行してください。

Intel MPI 2019 Library を利用して MPI ノード間通信を行った際に、期待した速度が出ません。CX400 システムは InfiniBand FDR(4xFDR 56Gbps) で接続されておりますが、 1Gbps で頭打ちとなります。原因、および回避方法について調査中です。

代替策として、 Intel MPI 2018 Library の ご利用をご検討ください。2019.06.14 更新

詳細実行

```
$ source /center/local/apl/cx/intel_2019/compilers_and_libraries_2019/linux/mpi/intel64/bin/mpivars.sh
```

MKL をご利用の前には、以下のコマンドを実行してください。

詳細実行

```
$ source /center/local/apl/cx/intel_2019/compilers_and_libraries_2019/linux/mkl/bin/mklvars.sh intel64
```

●Ver.2018 の場合　2019.04.02 更新

以下のコマンドを実行してください。実行後はログアウトされるまでは有効です。

詳細実行

```
$ source /center/local/apl/cx/intel_2018/compilers_and_libraries_2018/linux/bin/compilervars.sh intel64
```

MPI プログラムをご利用の前には、以下のコマンドを実行してください。

詳細実行

```
$ source /enter/local/apl/cx/intel_2018/compilers_and_libraries_2018/linux/mpi/intel64/bin/mpivars.sh
```

MKL をご利用の前には、以下のコマンドを実行してください。

詳細実行

```
$source /center/local/apl/cx/intel_2018/compilers_and_libraries_2018/linux/mkl/bin/mklvars.sh intel64
```

●Ver.2015 の場合（メーカサポート終了）

以下のコマンドを実行してください。実行後はログアウトされるまでは有効です。

詳細実行

```
$ source /center/local/apl/cx/intel/composer_xe_2015/bin/compilervars.sh intel64
```

MPI プログラムをご利用の前には、以下のコマンドを実行してください。

詳細実行

```
$ source /center/local/apl/cx/intel/impi/5.0.3.049/bin64/mpivars.sh
```

MKL をご利用の前には、以下のコマンドを実行してください。

詳細実行

```
$ source /center/local/apl/cx/intel/composer_xe_2015/mkl/bin/mklvars.sh intel64
```

●Ver.2013 の場合（メーカサポート終了）

以下のコマンドを実行してください。実行後はログアウトされるまでは有効です。

簡易実行

```
$ intelset
```

詳細実行

```
$ source /center/local/apl/cx/intel/composer_xe_2013_sp1/bin/compilervars.sh intel64
```

以下としても設定可能です。2016.04.21 更新

```
$ source /center/local/apl/cx/intel/composerxe/bin/compilervars.sh intel64
```

MPI プログラムをご利用の前には、以下のコマンドを実行してください。

簡易実行

```
$ intelmpi
```

詳細実行

```
$ source /center/local/apl/cx/intel/impi/4.1.1.036/bin64/mpivars.sh
```

MKL をご利用の前には、以下のコマンドを実行してください。

簡易実行

```
$ intelmkl
```

詳細実行

```
$ source /center/local/apl/cx/intel/mkl/bin/mklvars.sh intel64
```

なお、サンプルプログラムは以下にあります。

(Fortran/C/C++)・/center/local/apl/cx/intel/composerxe/Samples/

(MPI)　　・/center/local/apl/cx/intel/impi/4.1.1.036/test/

(MKL)　　・/center/local/apl/cx/intel/mkl/examples

9.1.3 Fortran コンパイル/リンク/実行方法

Intel Fortran コンパイラは ifort コマンドを利用します。MPI ライブラリを使用する場合は、mpiifort コマンドを利用します。

●コンパイル・リンク

例1)　逐次 Fortarn プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/composerxe/Samples/en_US/Fortran/optimize 配下のプログラムを利用。

```
$ ifort int_sin.f90
```

例 2)　ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/composerxe/Samples/en_US/Fortran/openmp_samples 配下のプログラムを利用。

```
$ ifort -openmp -fpp openmp_sample.f90
```

例 3)　MPI 並列プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/impi/4.1.1.036/test/配下のプログラムを利用。

```
$ mpiifort test.f90
```

●実行

例 1)　TSS 実行(逐次、OpenMP)

※OpenMP 実行時、環境変数 OMP_NUM_THREADS にスレッド数を指定してください。

```
$ ./a.out
```

例 2) TSS 実行(MPI)

```
$ mpirun -n 4 ./a.out
```

例 3) バッチ実行(逐次)

```
$ cat go_intel.sh
#!/bin/bash -x
#PJM -L "vnode=1"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx-small"
#PJM -j
#PJM -S

./a.out
```

例 4) バッチ実行(OpenMP)

```
$ cat go_intel_OMP.sh
#!/bin/bash -x
#PJM -L "vnode=1"
#PJM -L "vnode-core=14"
#PJM -L "rscgrp=cx-small"
#PJM -j
#PJM -S

source /center/local/apl/cx/intel/composerxe/bin/compilervars.sh intel64

OMP_NUM_THREADS=8; export OMP_NUM_THREADS
THREAD_STACK_SIZE=8192; export THREAD_STACK_SIZE

./a.out
```

例5) バッチ実行(MPI)

9.1.6 MPI 実行方法をご参照ください。

9.1.4 C コンパイル/リンク/実行方法

Intel C コンパイラは icc コマンドを利用します。MPI ライブラリを使用する場合は、mpiicc コマンドを利用します。

●コンパイル・リンク

例 1)　逐次 C プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/composerxe/Samples/en_US/C++/optimize 配下の
プログラムを利用。

```
$ icc int_sin.c
```

例2)　ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/composerxe/Samples/en_US/C++/openmp_samples 配下
のプログラムを利用。

```
$ icc -openmp openmp_sample.c
```

例 3)　MPI 並列プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/impi/4.1.1.036/test/配下のプログラムを利用。

```
$ mpiicc test.c
```

●実行

例 1)　TSS 実行(逐次、OpenMP)

※OpenMP 実行時、環境変数 OMP_NUM_THREADS にスレッド数を指定してください。

```
$ ./a.out
```

例 2)　TSS 実行(MPI)

```
$ mpirun -n 4 ./a.out
```

例 3)　バッチ実行(逐次)

```
$ cat go_intel.sh
#!/bin/bash -x
#PJM -L "vnode=1"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx-small"
#PJM -j
#PJM -S

./a.out
```

例 4)　バッチ実行(OpenMP)

```
$ cat go_intel_OMP.sh
#!/bin/bash -x
#PJM -L "vnode=1"
#PJM -L "vnode-core=14"
#PJM -L "rscgrp=cx-small"
#PJM -j
#PJM -S

source /center/local/apl/cx/intel/composerxe/bin/compilervars.sh intel64

OMP_NUM_THREADS=14; export OMP_NUM_THREADS
THREAD_STACK_SIZE=8192; export THREAD_STACK_SIZE

./a.out
```

例 5) バッチ実行(MPI)

9.1.6 MPI 実行方法をご参照ください。

9.1.5 C++コンパイル/リンク/実行方法

Intel C++コンパイラは icpc コマンドを利用します。MPI ライブラリを使用する場合は、mpiicpc コマンドを利用します。

●コンパイル・リンク

例 1) 逐次 C++プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/composerxe/Samples/en_US/C++/optimize 配下のプログラムを利用。

```
$ icpc int_sin.c
```

例 2) ノード内スレッド並列(OpenMP)プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/composerxe/Samples/en_US/C++/openmp_samples 配下のプログラムを利用。

```
$ icpc -openmp openmp_sample.c
```

例 3) MPI 並列プログラムをコンパイル/リンクする。

※/center/local/apl/cx/intel/impi/4.1.1.036/test/配下のプログラムを利用。

```
$ mpiicpc test.cpp
```

●実行

例 1)　TSS 実行(逐次、OpenMP)

※OpenMP 実行時、segmentation fault となる場合は、スタックサイズを拡張してください。

※OpenMP 実行時、環境変数 OMP_NUM_THREADS にスレッド数を指定してください。

```
$ ./a.out
```

例 1 で segmentation fault となる場合のスタックサイズ拡張方法）

```
$ ulimit -s unlimited
```

例 2)　TSS 実行(MPI)

```
$ mpirun -n 4 ./a.out
```

例 3)　バッチ実行(逐次)

```
$ cat go_intel.sh
#!/bin/bash -x
#PJM -L "vnode=1"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx-small"
#PJM -j
#PJM -S


./a.out
```

例 4)　バッチ実行(OpenMP)

```
$ cat go_intel_OMP.sh
#!/bin/bash -x
#PJM -L "vnode=1"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx-small"
#PJM -j
#PJM -S

source /center/local/apl/cx/intel/composerxe/bin/compilervars.sh intel64

OMP_NUM_THREADS=14; export OMP_NUM_THREADS
THREAD_STACK_SIZE=8192; export THREAD_STACK_SIZE

./a.out
```

例 5)　バッチ実行(MPI)

9.1.6 MPI 実行方法をご参照ください。

なお、詳細は、「第 3.4 バッチジョブ投入」をご参照ください。Fortran/C/C++のバッチジョブ実行のスクリプトについては、/center/local/sample/lang_sample/intel 配下をご参照ください。

9.1.6 MPI 実行方法

Intel MPI は、以下のように実行します。

注意事項）CX2550 システム（cx-*** リソースグループ）を使った
IntelMPI ジョブ実行時は、スクリプト内（手続き処理部）に
#PJM -L "vnode-core=28"
#PJM -P "vn-policy=abs-unpack"
を指定してください。

なお、CX270 システム（cx2-*** リソースグループ）を使った
IntelMPI ジョブ実行時は、スクリプト内（手続き処理部）に
#PJM -L "vnode-core=24"
#PJM -P "vn-policy=abs-unpack"
を指定してください。

例1)　自ノードで 4 プロセスを実行する場合

```
$ mpirun -n 4 ./test_mpi
Hello world: rank 0 of 4 running on cx01
Hello world: rank 1 of 4 running on cx01
Hello world: rank 2 of 4 running on cx01
Hello world: rank 3 of 4 running on cx01
```

例2)　バッチジョブで CX2550 で実行させる場合（フラット MPI）
※2 ノード×28 プロセス=56 プロセスの場合

```
$ cat mpi_flat.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L    "rscgrp=cx-small"
#PJM -L    "vnode=2"
#PJM -L    "vnode-core=28"
#PJM --mpi  "rank-map-bynode"
#PJM -P    "vn-policy=abs-unpack"
#PJM -L    "elapse=10:00"
#PJM -j
#PJM -S
#------ program execution ------#

source /center/local/apl/cx/intel/composerxe/bin/compilervars.sh intel64
source /center/local/apl/cx/intel/impi/4.1.1.036/bin64/mpivars.sh
source /center/local/apl/cx/intel/mkl/bin/mklvars.sh intel64

export I_MPI_HYDRA_BOOTSTRAP=rsh
export I_MPI_HYDRA_BOOTSTRAP_EXEC=/bin/pjrsh
export I_MPI_HYDRA_HOST_FILE=${PJM_O_NODEINF}

mpiexec.hydra -np 56 ./a.out
```

例3)　バッチジョブで CX2550/CX270 で実行させる場合（ハイブリッド）

```
$ cat mpi_hybrid.sh
#!/bin/sh
#------ pjsub option --------#
#PJM -L    "rscgrp=cx-small"
#PJM -L    "vnode=2"
#PJM -L    "vnode-core=28"
#PJM --mpi  "rank-map-bynode"
#PJM -P    "vn-policy=abs-unpack"
#PJM -L    "elapse=10:00"
#PJM -j
#PJM -S
#------ program execution ------#

source /center/local/apl/cx/intel/composerxe/bin/compilervars.sh intel64
source /center/local/apl/cx/intel/impi/4.1.1.036/bin64/mpivars.sh
source /center/local/apl/cx/intel/mkl/bin/mklvars.sh intel64

export I_MPI_PIN_DOMAIN=omp
export OMP_NUM_THREADS=28
export I_MPI_HYDRA_BOOTSTRAP=rsh
export I_MPI_HYDRA_BOOTSTRAP_EXEC=/bin/pjrsh
export I_MPI_HYDRA_HOST_FILE=${PJM_O_NODEINF}

mpiexec.hydra -np 2 ./a.out
```

※Intel MPI について、mpiexec.hydra での実行方法に変更になりました。
従来指定 mpdboot については intel mpi の次期バージョンアップで削除される可能性があります。mpdboot を指定して実行する場合、環境変数 I_MPI_CPUINFO に”proc”を指定してください。

参考例)バッチジョブで CX2550M1/CX270 で実行させる場合(ハイブリッド)(mpdboot)

```
#!/bin/bash
#PJM -L "rscgrp=cx-small"
#PJM -L "vnode=1"
#PJM -L "vnode-core=1"
##PJM -P "vn-policy=abs-unpack"
#PJM -L "elapse=10:00"
#PJM -j
#PJM -X
#PJM -S

source /center/local/apl/cx/intel/impi/4.1.1.036/intel64/bin/mpivars.sh

NODES=${PJM_VNODES}
CORES=${PJM_VNODE-CORES}
PROCS=1

export I_MPI_PERHOST=$CORES
export I_MPI_FABRICS=shm:ofa

export I_MPI_CPUINFO=proc
mpdboot -n $NODES -f ${PJM_O_NODEINF} -r /bin/pjrsh
mpiexec -n $PROCS ./a.out
mpdallexit
```

なお、詳細は、「第 3.4 バッチジョブ投入」をご参照ください。MPI のバッチジョブ実行のスクリプトについては、/center/local/sample/lang_sample/intel 配下をご参照ください。

9.1.7 MKL

サンプルプログラムを用いたコンパイルと実行結果は以下のとおりです。

```
(コンパイルまで)
$ cp /center/local/apl/cx/intel/mkl/examples/examples_f95.tgz .
$ tar zxf examples_f95.tgz
$ cd blas95
$ make libintel64 >make.log 2>&1
(実行(例)BLAS の dasumx)
$ cd _results/intel_lp64_parallel_intel_iomp5_intel64_lib
$ ./dasumx.out < ../../data/dasumx.d


   D A S U M  EXAMPLE PROGRAM


     INPUT DATA
      N= 7
      VECTOR X    INCX= 1
          1.630    -2.220    3.870    4.910    -5.450    6.200    -7.770


     OUTPUT DATA
        DASUM =   32.050
```

9.1.8 MKL(DFTI モジュールを利用する場合)

DFTI モジュールを Fortran からご利用される場合は、以下のようにコンパイルしてください。（一例です）

※DFTI モジュール定義を呼び出す必要があります。

```
$ ifort -O3 -openmp -I/center/local/apl/cx/intel/mkl/include ¥
                    -L/center/local/apl/cx/intel/mkl/lib/intel64/ ¥
                    -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core ¥
                    /center/local/apl/cx/intel/mkl/include/mkl_dfti.f90 test.f90
```

他のモジュールをプログラム内で USE する場合は、

/center/local/apl/cx/intel/mkl/include 配下のソースプログラムをご指定ください。

9.1.9 その他

●CX で MKL ライブラリーが提供する FFTW インタフェースをご利用される場合
以下のオプションが必要です。

```
-I/center/local/apl/cx/intel/mkl/include/fftw

-l /center/local/apl/cx/intel/mkl/lib/intel64
```

Fortran プログラムから LAPACK を利用する場合は、以下のオプションが必要です。

```
-lmkl_lapack95_lp64
```

Fortran プログラムから BLAS を利用する場合は、以下のオプションが必要です。

```
-lmkl_blas95_lp64
```

その他の基本的な MKL の関数・機能については、以下のオプションを指定してください。

```
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
```

●CX で FFTW インタフェースをご利用される場合
以下のオプションが必要です。

```
-I/center/local/apl/cx/fftw-3.3.4/include

-L/center/local/apl/cx/fftw-3.3.4/lib
```

基本的な関数・機能については、以下のオプションを指定してください。

```
-lfftw3 -lfftw3_mpi -lfftw3_omp
```

●FX で FFTW インターフェースをご利用される場合

以下のオプションが必要です。

```
-I/center/local/apl/fx/fftw-3.3.4/include
-L/center/local/apl/fx/fftw-3.3.4/lib
```

基本的な関数・機能については、以下のオプションを指定してください。

```
-lfftw3 -lfftw3_mpi -lfftw3_omp
```

9.2 Phi の利用について

CX270 システムでは Xeon Phi 3120P が利用できます。

以下に Phi 3120P の仕様を示します。

表 9-2 Xeon Phi の主な仕様

仕様	
プロセッサ・ナンバー	3120P
コア数	57
スレッド数	228(57×4)
動作周波数(1 コア)	1.1GHz
キャッシュ	28.5MB
主記憶	6GB

9.2.1 Phi の環境および留意事項について

Phi の環境および留意事項について、以下に示します。

- Xeon Phi 用のロードモジュールと Xeon 用のロードモジュールは、お互いに互換性がありません。
- Xeon Phi は MMX, SSE, SSE2, AVX 命令をサポートしていません。
- Xeon Phi には、２つの動作モード(ネイティブ,Offload)がありますが、名古屋大学の環境では、「Offload モード」での実行を許可しています。

9.2.2 コンパイル・リンク/実行方法

XeonPhi 用バイナリは、Xeon ノード上でコンパイル(=クロスコンパイル)し、作成します。

以下にサンプルプログラムを利用した、コマンド例を示します。

※サンプルプログラムは以下にあります。

(Offload モード)

- /center/local/apl/cx/intel/composerxe/Samples/ja_JP/C++/mic_samples/LEO_tutorial/tbo_sort.c

9.2.2.1 Offload モード

①コンパイル・リンク及び実行

MPSS が導入されている CX270 計算ノードで Phi 対応のコンパイルを行います。

ログインノードからバッチジョブにてコンパイル及び実行します。

バッチジョブ実行のスクリプトは、/center/local/sample/phi_offload/go_cx_phi.sh を参照してください。

```
$ cat go_cx_phi.sh
#!/bin/bash -x
#PJM -L "vnode=24"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx2-small"
#PJM -j
#PJM -S

source /center/local/apl/cx/intel/composer_xe_2013_sp1/bin/compilervars.sh intel64

####### compile #######
icc -openmp tbo_sort.c -o tbo_sort

NUM_THREADS=24; export NUM_THREADS

THREAD_STACK_SIZE=8192; export THREAD_STACK_SIZE
FLIB_FASTOMP=TRUE; export FLIB_FASTOMP

####### execute #######
export OFFLOAD_REPORT=1

./tbo_sort
```

②MKL(例 blas)

バッチジョブ実行のスクリプトについては、

/center/local/sample/phi_offload/go_cx_phi_mkl_comp.sh、go_cx_phi_mkl_run.sh を参照してください。

```
(コンパイルまで)
$ cp /center/local/apl/cx/intel/mkl/examples/examples_mic.tgz .
$ tar zxf examples_mic.tgz
$ cd mic_offload/blasc
$ cat go_cx_phi_mkl_comp.sh
#!/bin/bash -x
#PJM -L "vnode=24"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx2-small"
#PJM -j
#PJM -S
source /center/local/apl/cx/intel/composer_xe_2013_sp1/bin/compilervars.sh intel64
make libintel64 > make.log 2>&1
```

(実行)

`_result` ディレクトリが作成される。

```
$ cat go_cx_phi_mkl_run.sh
#!/bin/bash -x
#PJM -L "vnode=24"
#PJM -L "vnode-core=1"
#PJM -L "rscgrp=cx2-small"
#PJM -j
#PJM -S
source /center/local/apl/cx/intel/composer_xe_2013_sp1/bin/compilervars.sh intel64
NUM_THREADS=24; export NUM_THREADS
PARALLEL=$NUM_THREADS; export PARALLEL
OMP_NUM_THREADS=$PARALLEL; export OMP_NUM_THREADS
THREAD_STACK_SIZE=8192; export THREAD_STACK_SIZE
FLIB_FASTOMP=TRUE; export FLIB_FASTOMP
export OFFLOAD_REPORT=1
/center/meidai/intel_sample/mic_offload/blasc/_results/intel_lp64_parallel_libintel64/
sgemm_reuse.out 10
$ tail -30 go_cx_phi_mkl_run.sh.o39459
Matrix dimension is being set to 10
Resulting matrix C:
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
  0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
[Offload] [MIC 0] [File]          ./source/sgemm_reuse.c
[Offload] [MIC 0] [Line]          100
[Offload] [MIC 0] [CPU Time]       0.524114 (seconds)
[Offload] [MIC 0] [MIC Time]       0.041518 (seconds)
 :
```

10. HPC ポータル

HPC ポータルとは web ベースのログイン環境になります。
以下の URL よりアクセスしてください。

https://portal.cc.nagoya-u.ac.jp/

ログイン画面

10.1 HPC ポータル機能

各機能の詳細については HPC ポータルの「10.1.1 5) ポータル利用手引き」を参照して下さい。

10.1.1 メイン

1）About

HPC ポータルの概要や機能紹介

2）設定

HPC ポータルのユーザー設定

3）パスワード変更

パスワード変更

4）SSH 公開鍵登録

公開鍵の登録

5）ポータル利用手引き

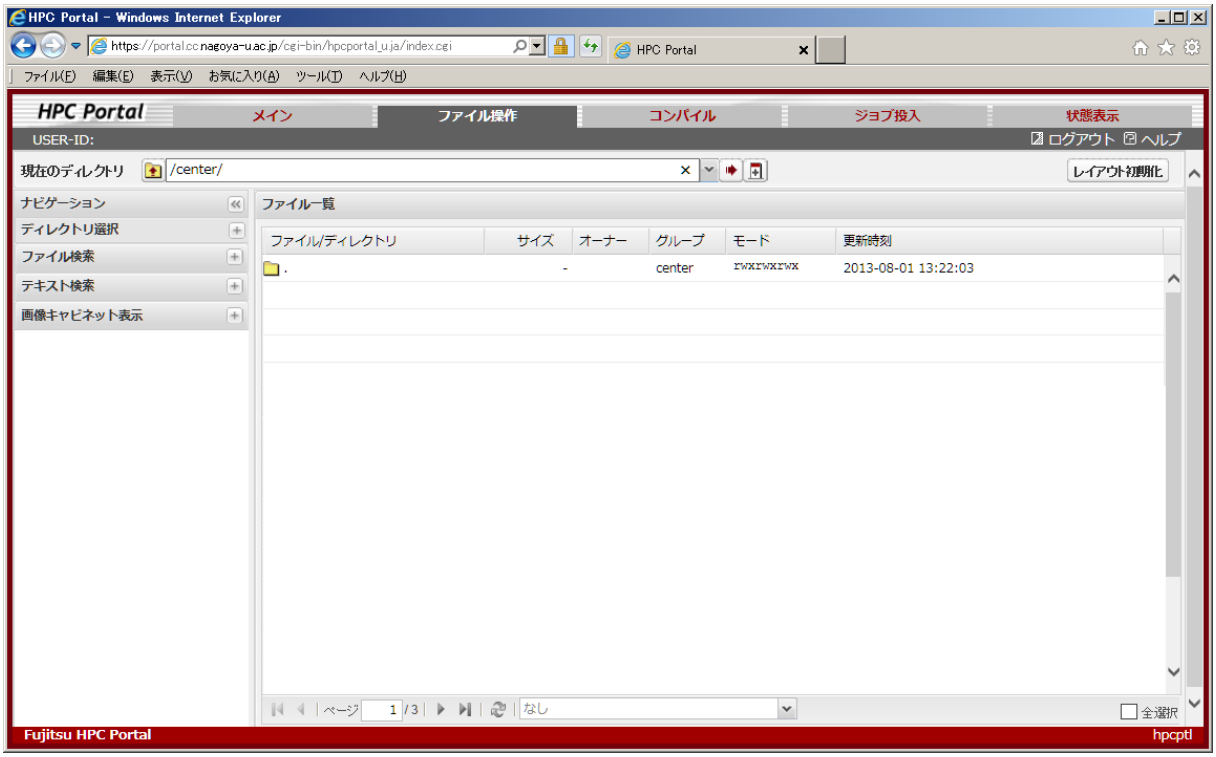
HPC ポータルの利用者マニュアル

6） 富士通マニュアル

富士通のコンパイラ、ツール、ライブラリ等のマニュアル

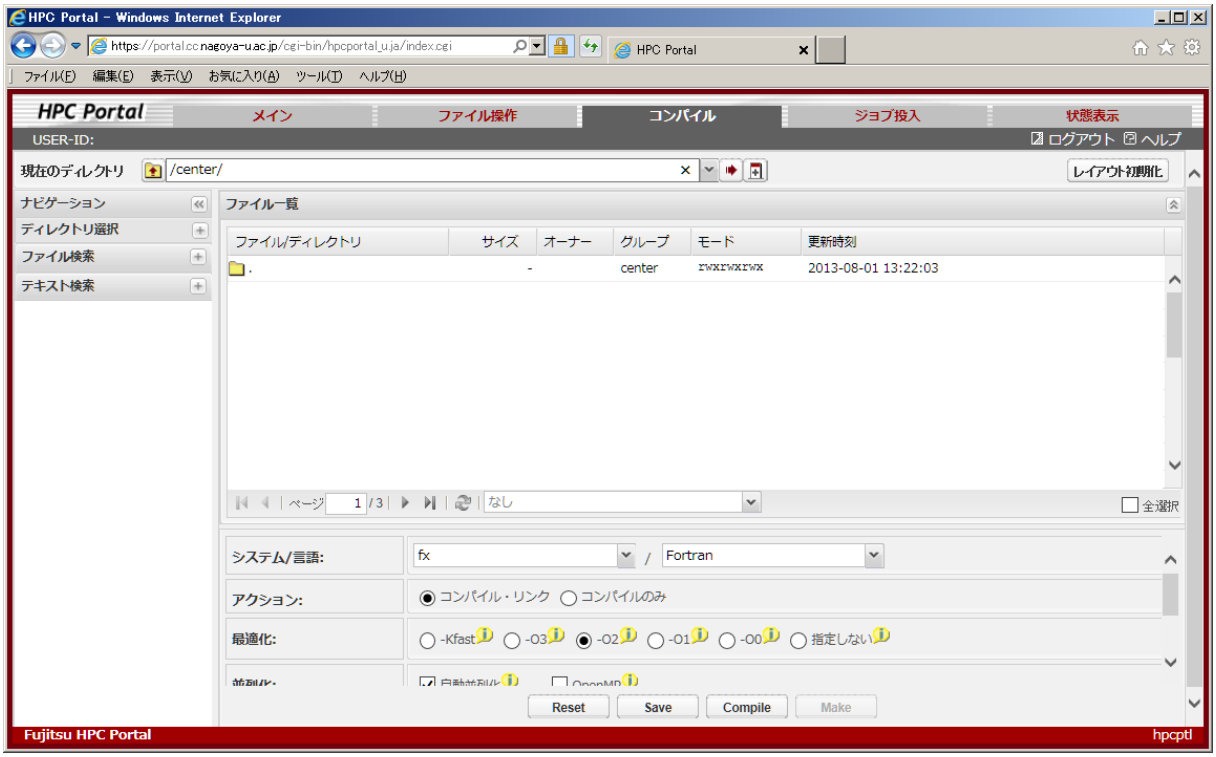
10.1.2 ファイル操作

ユーザーディレクトリのファイル操作



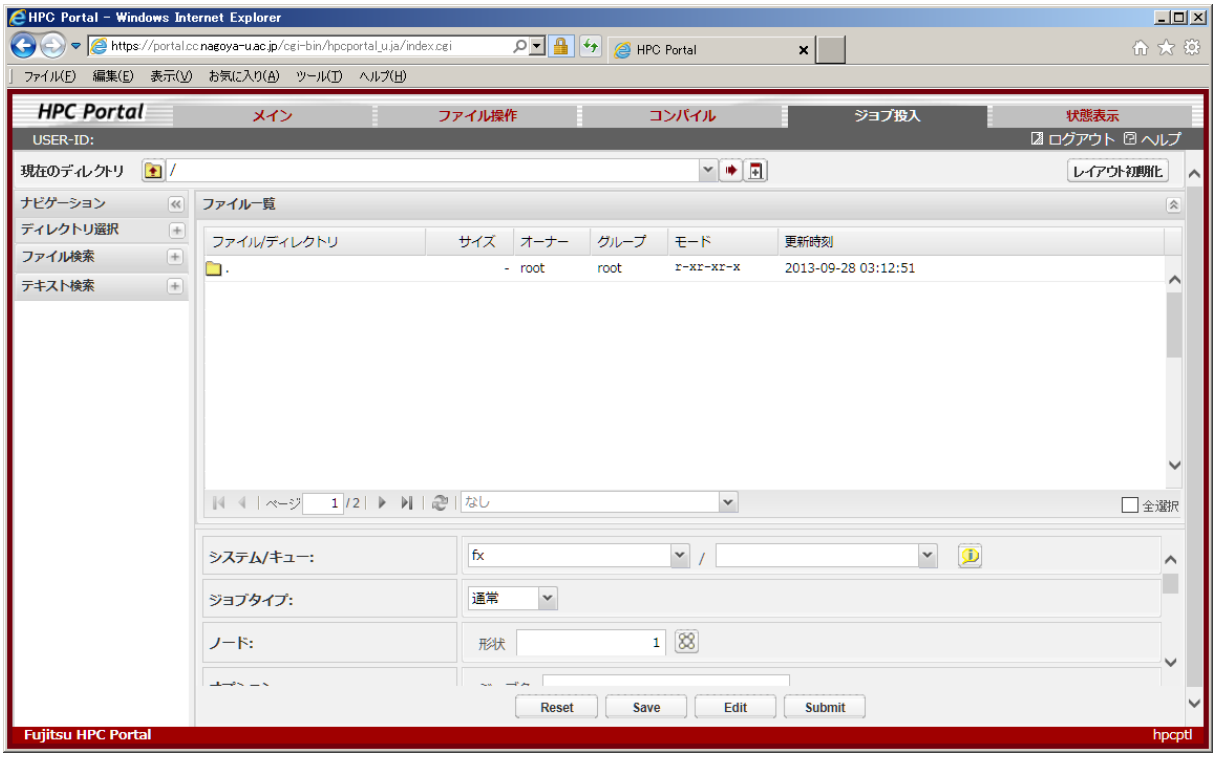
10.1.3 コンパイル

コンパイル環境



10.1.4 ジョブ投入

ジョブの実行環境



10.1.5 状態表示

ジョブの実行状況などの状態表示

11. マニュアル

システム上の以下のディレクトリ配下にマニュアルが配置されています。

- （FX100）：/opt/FJSVmxlang/manual　　（2016.06.08 更新）
- （CX）　：/opt/FJSVpclang/1.2.0/manual/

表 7-1 マニュアル一覧

分類	資料名
コンパイラ 並列ライブラリ	Fortran 文法書
	Fortran 使用手引書
	Fortran 翻訳時メッセージ
	C 言語使用手引書
	C++言語使用手引書
	C/C++最適化メッセージ
	XPFortran 使用手引書
	Fortran/C/C++実行時メッセージ
	MPI 使用手引書
プログラム開発支援ツール	実行時情報出力機能使用手引書
	プログラミング支援ツール使用手引書
	プロファイラ使用手引書
	デバッガ使用手引書
	ランク配置最適化ツール使用手引書
数学ライブラリ	数学ライブラリの利用手引
	富士通 SSL II 使用手引書
	FUJITSU SSL II 拡張機能使用手引書
	FUJITSU SSL II 拡張機能使用手引書 II
	FUJITSU SSL II スレッド並列機能 使用手引書
	FUJITSU C-SSL II 使用手引書
	FUJITSU C-SSL II スレッド並列機能 使用手引書
	FUJITSU SSL II/MPI 使用手引書
	BLAS LAPACK ScaLAPACK 使用手引書