「富岳」利用セミナー：入門編

～「富岳」の利用方法～

2025年4月版

登録施設利用促進機関 / 文科省委託事業「HPCIの運営」代表機関
一般財団法人　高度情報科学技術研究機構（RIST）

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

セミナーの位置づけ

本セミナーは、スーパーコンピュータ「富岳」（特定高速電子計算機施設）の利用促進のため、登録施設利用促進機関としてRISTが主催するものである。

入門編では「富岳」を初めて利用する方を対象とし、

- 「富岳」へのログイン手順
- コンパイラ及びライブラリの使用方法
- ジョブ実行方法

など、基本的な利用方法を説明する。

一般的なHPC利用に関する知識、プログラミング言語（Fortran, C/C++）については詳細な解説をしていない。

本資料各ページの左上隅に表示されている★印の意味は以下の通りである。
★★★：多くの利用者にとって重要であると考えられる情報
★★☆：一部の利用者にとって重要であると考えられる情報
★☆☆：利用者によっては重要な情報、発展的な内容を含む情報、参考情報など

参考資料等

このテキストは、以下の資料から基本的な利用方法を抜粋してまとめたものとなっている。詳細については各資料を参照のこと。

- スーパーコンピュータ「富岳」スタートアップガイド 1.11版
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/startup_guides/StartupGuide-ja.pdf
  - クライアント証明書を送付するメールに記載されたURLからも取得できる
- 利用手引書 利用およびジョブ実行編 1.48版
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/user_guides/use_latest/ （※1）
- 利用手引書 言語開発環境編 1.32版
  - https://www.Fugaku.r-ccs.riken.jp/doc_root/ja/user_guides/lang_latest/ （※1）
- 富岳 Spack利用ガイド (2024/11/20)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/user_guides/FugakuSpackGuide/ （※1）
- 富岳Open OnDemandガイド
  - https://riken-rccs.github.io/ondemand_fugaku/index_ja.html
- プリポスト環境 利用手引書 1.14版
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/user_guides/pps-slurm/ （※1）

※1 最新版へのリンクになっています。

参考資料等

- HPCIログインマニュアル OAuth対応版 (2024/06/07)
  - https://www.hpci-office.jp/download_file/view/68464801-f085-4d7c-973f-8060425f0e20/578
- HPCI共用ストレージ利用マニュアル OAuth利用編 (2024/04/23)
  - https://www.hpci-office.jp/download_file/view/d8c3560c-af25-4749-b1b9-b7d3fee547a7/578
- ジョブ運用ソフトウェア エンドユーザ向けガイド
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/manuals/jos/j2ul-2534-01z0.pdf
- ジョブ運用ソフトウェア エンドユーザ向けガイド HPC拡張機能編
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/manuals/jos/j2ul-2535-01z0.pdf
- 利用者支援ツール 使用手引書 (利用者編)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/user_guides/support_tool/
- プログラミングガイド (IO編)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/programming_guides/IO_part_Programming_Guide_JA.pdf
- プログラミングガイド (プログラミング共通編)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/programming_guides/Programming_common_part_Programming_Guide_JA.pdf

参考資料等

- プログラミングガイド (プロセッサ編)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/programming_guides/Processors_Programming_Guide_JA.pdf
- MPI使用手引書(tcsds-1.2.41)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/manuals/tcsds-1.2.41/lang/MPI/j2ul-2565-01z0.pdf
- プロファイラ使用手引書 (tcsds-1.2.41)
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/manuals/tcsds-1.2.41/lang/Tool/j2ul-2568-01z0.pdf
- HPCIポータルサイト・高度化支援
  - https://www.hpci-office.jp/user_support/tuning_support
- ユーザブリーフィング資料（2025/04/10開催）

  ※ システムの最新情報は毎月ユーザブリーフィングで情報提供される。環境の更新に注意
  - 富岳の運用状況報告
    - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/guidances/SystemReport_202504.pdf
  - 登録機関ヘルプデスクからの報告
    - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/guidances/HelpdeskReport_202504.pdf
- 富岳サポートサイト利用ガイド（第4版）
  - https://www.fugaku.r-ccs.riken.jp/doc_root/ja/other/FugakuSupportSite-guide.pdf

利用方法概要

「富岳」を利用して計算を完了するまでの流れを以下に示す。

1. 「富岳」へのファイル転送

```
[terminal]$ scp -i private_key local_file user_name@login.fugaku.r-ccs.riken.jp:remote_file
```

2. ログインノードへのログイン

```
[terminal]$ ssh -i private_key user_name@login.fugaku.r-ccs.riken.jp
```

3. コンパイル

```
[_LNlogin]$ frtpx -Kfast,openmp -o sample sample.f08
```

4. ジョブ実行

```
[_LNlogin]$ pjsub sample.sh
```

5. ジョブ状態表示

```
[_LNlogin]$ pjstat
```

6. 結果確認

```
[_LNlogin]$ less ./sample.sh.jobid.out
```

■ 本テキストで、コマンドを実行する端末を次のように記す

[terminal]$ 利用者の端末でコマンドを実行
[_LNlogin]$ ログインノード(intel)でコマンドを実行
[_CNlogin]$ 計算ノードでコマンドを実行

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

「富岳」への接続方法

「富岳」に接続するには主に以下の方法がある。

- 「富岳」ローカルアカウントを用いた接続
  - 全ての「富岳」ユーザが利用可能
  - SSH接続（公開鍵認証）
    - 参考資料
      - スーパーコンピュータ「富岳」スタートアップガイド
  - Open OnDemandを用いた接続（クライアント証明書による認証）
    - Open OnDemand: Webブラウザから並列計算機システムを利用するためのWebポータル
    - 参考資料
      - 富岳Open OnDemandガイド
- HPCIアカウントを用いた接続（シングルサインオン）
  - HPCI課題で利用可能
  - hpcissh接続（OAuth認証）
  - HPCI共用ストレージや「富岳」以外のHPCI計算資源から接続可能
  - 参考資料
    - HPCIログインマニュアル OAuth対応版
    - HPCI共用ストレージ利用マニュアル OAuth利用編

利用開始までの手順

「富岳」ローカルアカウントによる利用開始までの手順を説明する。
詳細は「スーパーコンピュータ「富岳」スタートアップガイド」参照。

1. クライアント証明書・パスフレーズの受け取り
   - クライアント証明書：”ユーザアカウント名.p12” ファイル
     - アカウント発行後に電子メールで送付される
     - 富岳ウェブサイトへのアクセスに使用する
     - 秘密鍵、クライアント証明書（公開鍵）、発行局のルート証明書が含まれる
   - クライアント証明書のパスフレーズ
     - 利用者登録が完了すると、ハガキまたはPDFファイルで送付される
     - クライアント証明書をインストールする際に使用する
     - 安全な場所に保管すること

利用開始までの手順

2. クライアント証明書のインストール

クライアント証明書は富岳ウェブサイトにアクセスする際に使用する。富岳ウェブサイトにアクセスするパソコンのブラウザにインストールすること。

- 推奨ブラウザ
  - Mozilla Firefox（Windows, Mac）
  - Google Chrome（Windows, Mac）

※ 推奨ブラウザと異なるものを利用する場合は、ブラウザの証明書管理方法を確認した後にインストールをおこなうこと。

- インストール時の注意事項
  - クライアント証明書のパスフレーズを入力する際、「パスワードを入力してください」と表示される場合がある
  - 別途、クライアント証明書利用時に入力するためのパスワード設定を求められる場合がある

利用開始までの手順

3. 富岳ウェブサイトへの接続

URI： https://www.fugaku.r-ccs.riken.jp

■ 注意事項

脆弱性対応のため、富岳ウェブサイトではTLS 1.2またはTLS1.3接続のみ受け付ける。

■ ログアウト

富岳ウェブサイトはログアウトの機能を持たない。

完全に終了させたい場合はブラウザを終了すること。

■ アカウントの切り替え

クライアント証明書（アカウント）を複数所持している場合、アカウントを切り替えるには別のブラウザを使うか、ブラウザを一旦終了してから再度アクセスすること。

■ ホーム画面

運用状況
運用スケジュール

各種申請
※ データ領域拡大

お問い合わせ
富岳サポートサイト
にアクセス

ドキュメント

サイト内検索

お知らせ

■ カテゴリ抽出

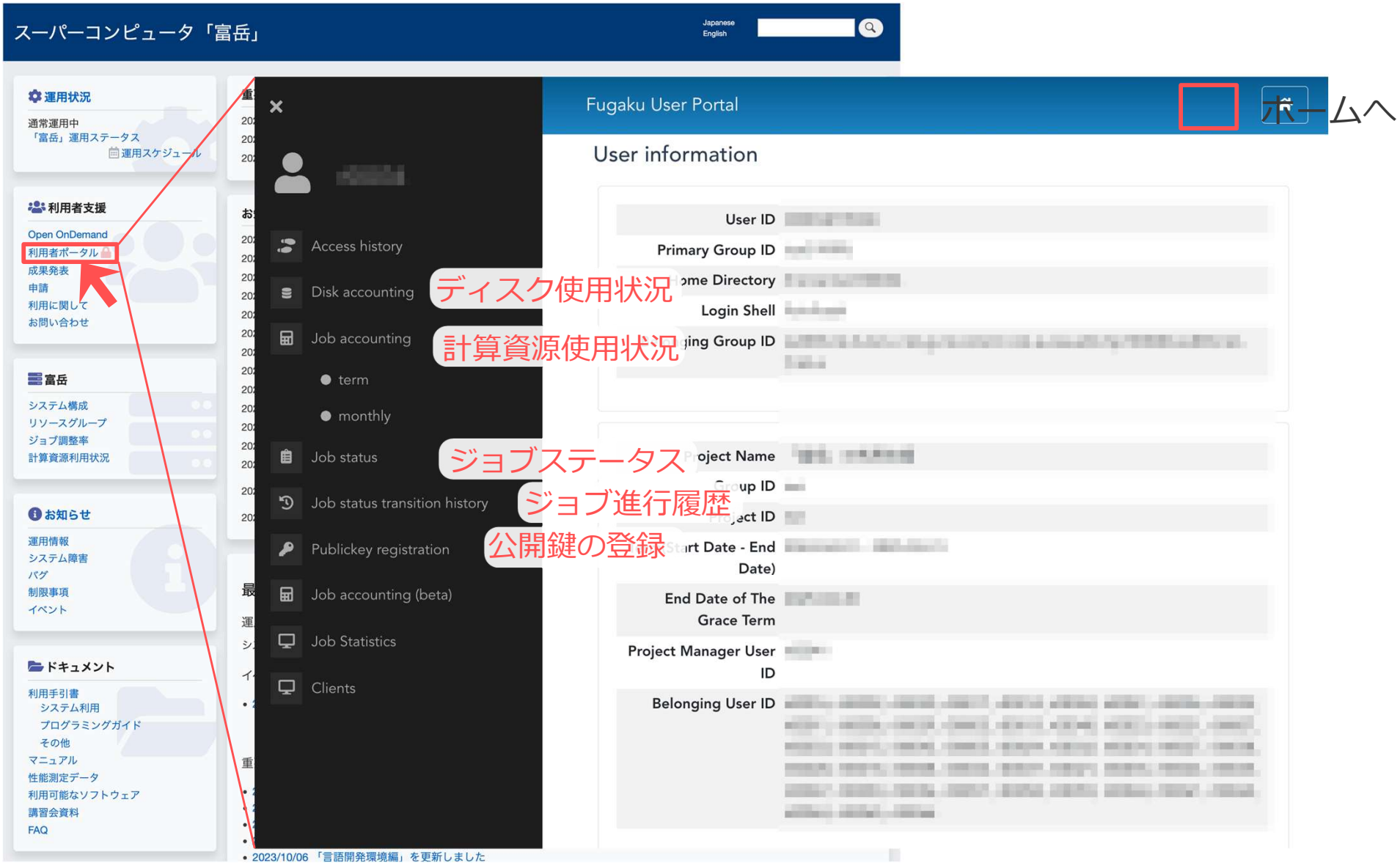
運用情報

■ アイコン

更新された記事

対応済の障害

■ 「利用者ポータル」画面


スーパーコンピュータ「富岳」
Japanese
English
運用状況
通常運用中
「富岳」運用ステータス
運用スケジュール
利用者支援
Open OnDemand
利用者ポータル
成果発表
申請
利用に関して
お問い合わせ
富岳
システム構成
リソースグループ
ジョブ調整率
計算資源利用状況
お知らせ
運用情報
システム障害
バグ
制限事項
イベント
ドキュメント
利用手引書
システム利用
プログラミングガイド
その他
マニュアル
性能測定データ
利用可能なソフトウェア
講習会資料
FAQ
Access history
Disk accounting
ディスク使用状況
Job accounting
計算資源使用状況
term
monthly
Job status
ジョブステータス
Job status transition history
ジョブ進行履歴
Publickey registration
公開鍵の登録
Job accounting (beta)
Job Statistics
Clients
Fugaku User Portal
ポータルへ
User information
User ID
Primary Group ID
Login Shell
End Date of The Grace Term
Project Manager User ID
Belonging User ID
2023/10/06 「言語開発環境編」を更新しました

「富岳」へのログイン

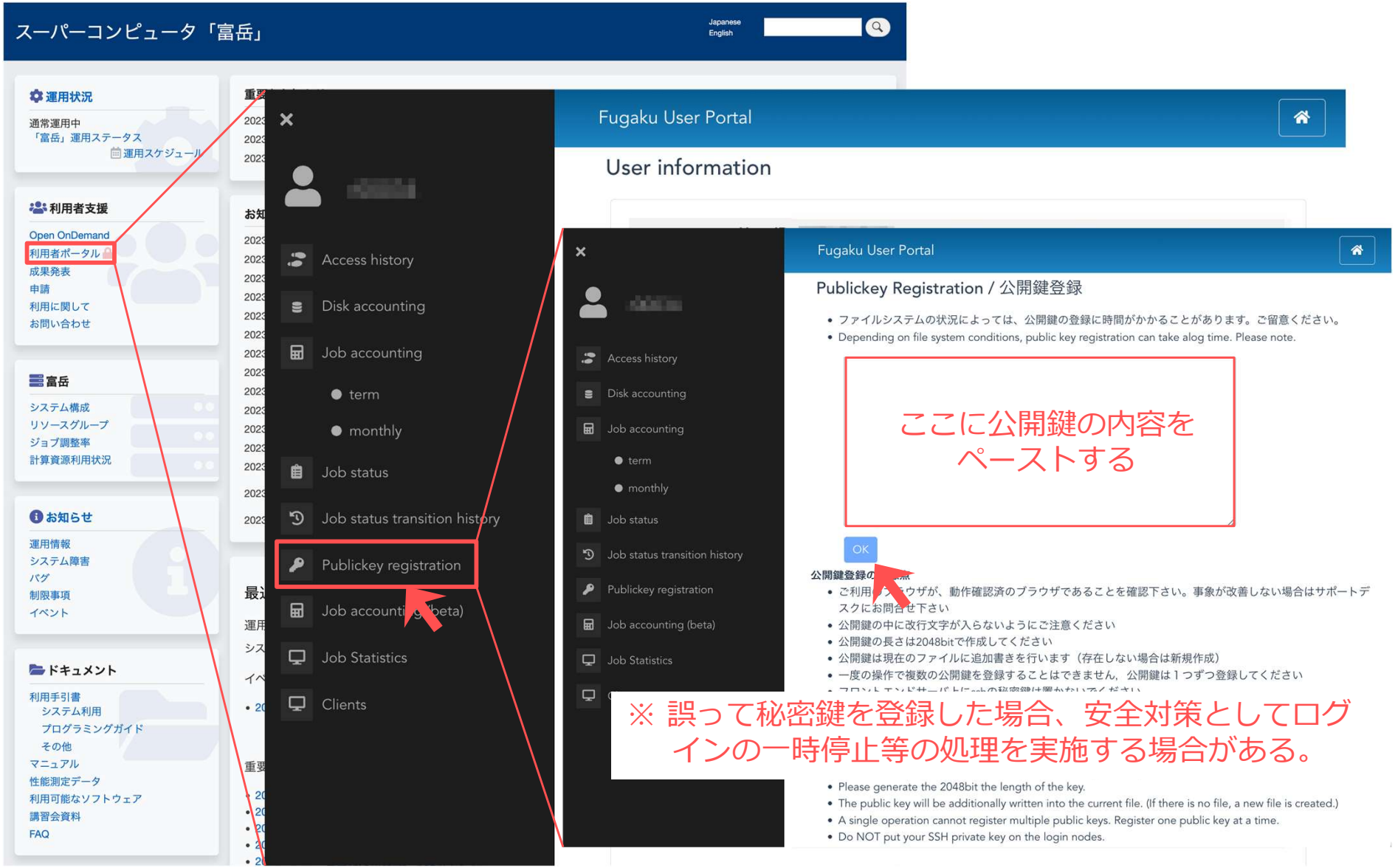
ローカルアカウントを利用して「富岳」へログインする場合、SSH Version2（公開鍵認証）を用いる。
SSHでの接続には鍵ペアの作成・公開鍵の登録が必要となる。

1. 鍵ペア（秘密鍵/公開鍵）の作成
   次のいずれかの鍵を利用者端末にて作成する。
   作成手順は「利用手引書 利用およびジョブ実行編 4.4.1」を参照。

   - Ed25519
   - ECDSA（NIST P 521）

   ※ 鍵の作成時には必ずパスフレーズを設定すること。他人が推測しにくい文字列とし、15文字以上を推奨。
   ※ RSA鍵の利用については、「ログインノードのsshアクセスに関する運用変更」を参考にすること
   （https://www.fugaku.r-ccs.riken.jp/operation/20230210_01）
   ※ Open OnDemand を使えば、公開鍵認証を経なくても様々な「富岳」上のアプリケーションを利用可能

2. 公開鍵の登録



3. アクセス方法

利用者端末から以下のホスト名でアクセスする。

ホスト名：login.fugaku.r-ccs.riken.jp

```
[terminal]$ ssh -i private_key user_name@login.fugaku.r-ccs.riken.jp
The authenticity of host 'XXXXXX (nnn.nnn.nnn.nnn)' can't be established.
XXXXX key fingerprint is XX: XX: XX: XX: XX: XX: XX: XX:XX:XX:XX:XX:XX:XX:XX:XX.
Are you sure you want to continue connecting (yes/no)? yes  #yesを入力（初回）
Enter passphrase for key 'private_key': #パスフレーズを入力
[_LNlogin]$
```

- 補足事項
  - X11 Forwarding 機能を使う場合は ssh オプション -X を指定する。
  - SSH Agent-forwarding 機能を使う場合はsshオプション -A を指定する。
  - 複数台のログインノードを運用しているが、ホーム領域（/home）、データ領域（/data）、シェア領域（/share）、2ndfs領域（/2ndfs）は各ログインノードで共用している。言語ソフトウェア環境も同じ。
  - 各ユーザーのホーム領域のサイズ上限は20GiB、200,000 i-node。
    - 1 週間以内であれば 750,000 i-node まで利用可能
      - その期間を過ぎるとファイルの作成ができなくなる。不要になったファイルは速やかに削除すること。
  - ログインシェルは/bin/bash 。

※ PuTTYなどによるアクセス方法は「利用手引書　利用およびジョブ実行編　4.4.3」参照。

4. ファイル転送方法

scp/sftp/rsync を利用してファイル転送が可能。
セキュリティに脆弱性のあるプロトコル（ftp/rcp）は禁止。

■ 端末からログインノードへの転送例

```
[terminal]$ scp -i private_key local_file user_name@login.fugaku.r-ccs.riken.jp:remote_file
Enter passphrase for key ‘private_key': #パスフレーズを入力
[terminal]$
```

■ ログインノードから端末への転送例

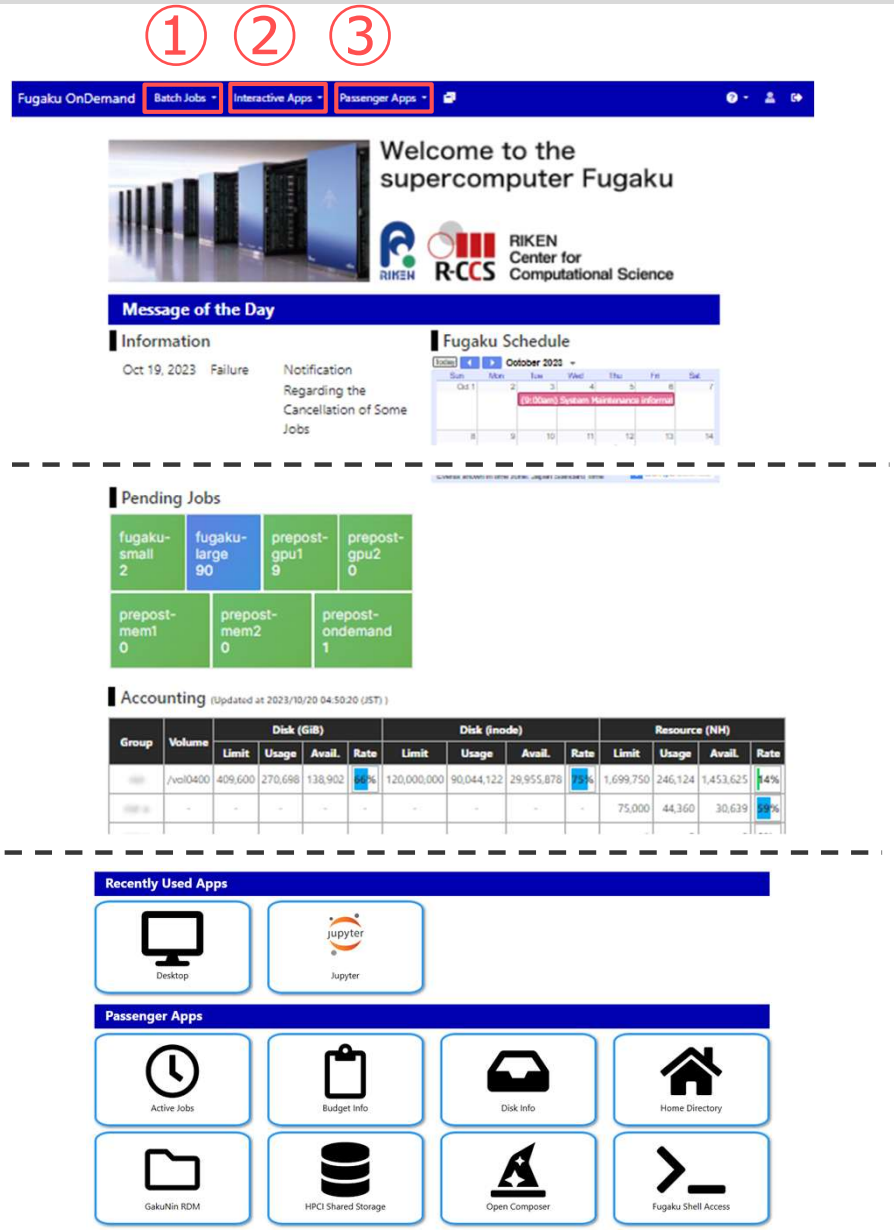
```
[terminal]$ scp -i private_key user_name@login.fugaku.r-ccs.riken.jp:remote_file local_file
Enter passphrase for key ‘private_key': #パスフレーズを入力
[terminal]$
```

※ WinSCPによる転送方法は「利用手引書　利用およびジョブ実行編　4.4.4」参照。

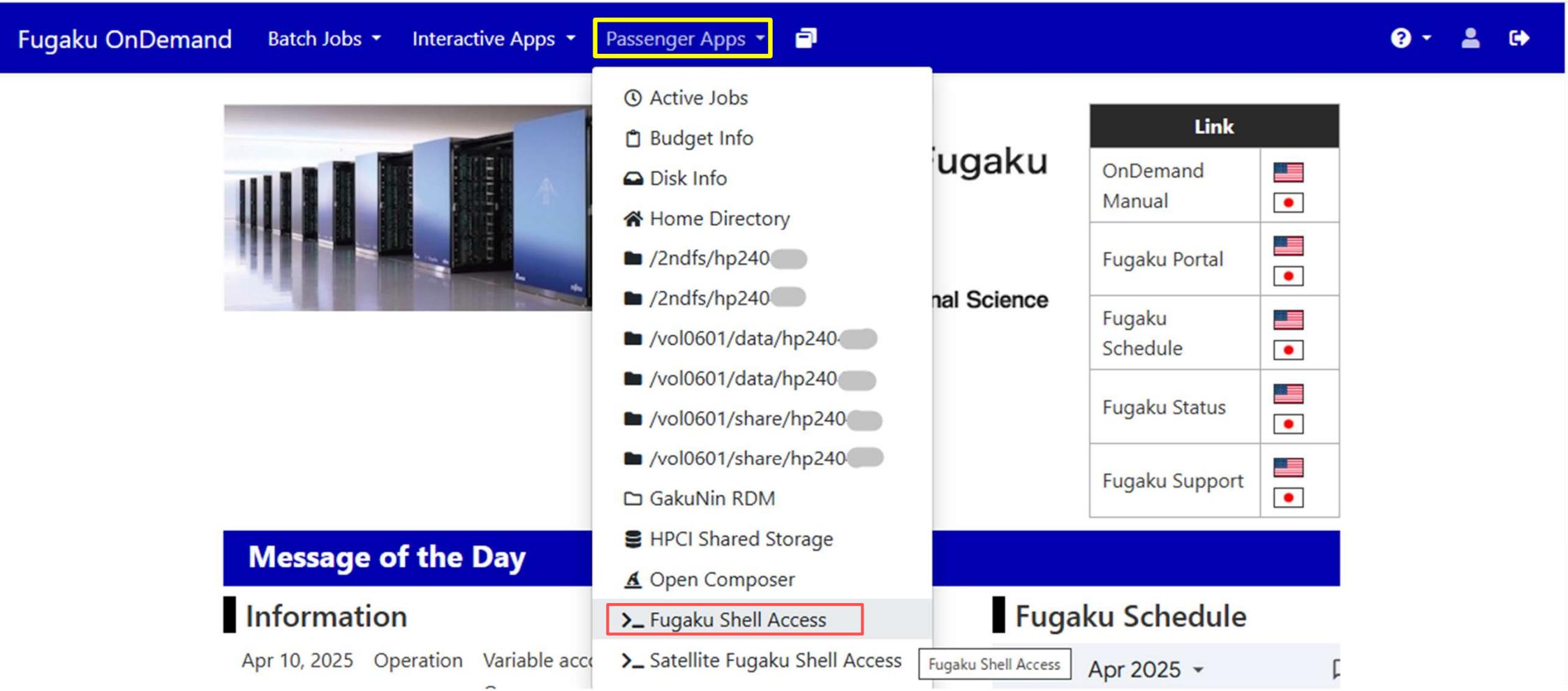
Webブラウザベースで、富岳の計算資源を利用し様々なプログラムを実行することができる Fugaku Open OnDemand が提供されている。

- Open OnDemand とは？
  - Webブラウザから並列計算機を利用するためのポータル、またそれをユーザに提供するためのシステム
  - 公式サイト https://openondemand.org/
- URL
  - https://ondemand.fugaku.r-ccs.riken.jp/pun/sys/dashboard
    - 富岳ウェブサイトへと共通のクライアント証明書を利用してアクセスする
    - 富岳ウェブサイトトップページ左メニュー「利用者支援 > Open OnDemand」からも飛ぶことができる
- 利点
  - HTTPSが利用できる環境で富岳ウェブサイト用のクライアント証明書があれば利用可能であり、公開鍵認証の必要はない
  - ジョブスクリプトを書かなくても、入力ファイルの用意と計算資源の設定を行うだけで様々な科学技術計算アプリケーションが使える
  - Web ブラウザから富岳上のファイルの閲覧・操作が可能

- 提供されている機能
  - ① Batch Jobs
    - 富岳の計算ノード上で動作する非対話的アプリケーション
    - 科学技術計算用のプログラムなど
    - Open Composerについては、p120を参照
  - ② Interactive Apps
    - 富岳のプリポストノードおよび計算ノードで動作する対話的アプリケーション
    - デスクトップ、可視化プログラム、プロファイラなど
  - ③ Passenger Apps
    - Open OnDemandがインストールされたサーバで動作するユーティリティ
    - シェル操作、ファイルのアップロード・ダウンロード、画像ファイル閲覧、テキスト編集などが行える



Fugaku Open OnDemandから「富岳」ログインノードにアクセス可能。



その他、機能の詳細については「富岳Open OnDemandガイド」を参照

HPCIアカウントを用いた接続の流れ

FreeOTP もしくは google authenticatorによる二要素認証の設定
https://metis.hpci.nii.ac.jp/auth/realms/HPCI/account/ **3.3節**

↓

jwt-server (https://elpis.hpci.nii.ac.jp/) へアクセス **3.5節**

↓

生成されたaccess informationを使ってjwt-agentを実行 **3.5節**

↓

OAuth-SSHクライアント(hpcisshコマンド)によるログイン **3.6節**

※詳細は『HPCIログインマニュアル OAuth対応版』の各章節を参照

「富岳」で利用可能なサーバ

「富岳」には以下のサーバが用意されている。

- Intelログインノード
  - 外部から接続可能
  - 接続先：`login.fugaku.r-ccs.riken.jp`
- Armログインノード
  - Intelログインノードから接続可能
  - 接続先：`arm1`
- プリポスト環境
  - Intelログインノードからジョブ投入して利用
  - ジョブ管理システムSlurmで管理
  - 詳細はプリポスト環境利用手引書を参照
- クラウドストレージゲートウェイノード
  - 外部から接続可能
  - データ転送用ノード。HPCI共用ストレージをマウントすることも可能
  - 接続先：`csgw.fugaku.r-ccs.riken.jp`
  - 詳細は次頁

HPCI共用ストレージの利用

HPCI共用ストレージの利用資格がある場合、「富岳」にマウントして利用できる。

■ HPCI共用ストレージ概要

- ■ 広域分散ファイルシステムGfarmを利用したHPCI大規模ストレージ
- ■ 45 ペタバイト（論理）を提供
- ■ 大容量メモリ・GPGPU搭載クライアント環境も利用可能

■ 「富岳」からの利用方法

1. hpcisshコマンドを使って
   クラウドストレージゲートウェイノードにログインする
2. HPCI共用ストレージを `mount.hpci` コマンドにてマウントする
3. 並列ファイル転送コマンド `gfpcopy` 等でデータを転送する

※ 利用方法の詳細は以下を参照。

- ■ HPCI共用ストレージ利用マニュアル 共有マニュアル OAuth利用編
  - ■ https://www.hpci-office.jp/download_file/view/d8c3560c-af25-4749-b1b9-b7d3fee547a7/578

「富岳」運用情報のメール配信

「富岳」運用情報として、以下の件名でメールが配信されている。

- [Fugaku] Reports the jobs affected by the failure
  - システム異常により影響を受けたジョブの通知
- [Fugaku] LLIO Usage Limit Exceeded Notification
  - LLIO利用制限超過の通知
- [Fugaku] Information
  - ポータルお知らせ情報

- 設定方法
  - ホームディレクトリに、以下のように .forward ファイルを作成する
  - 受信を希望するメールアドレスをカンマ区切り、または改行で記載する

```
[_LNlogin]$ vi ~/.forward

*****@*****.com

*****@******.jp
```

※より詳細な使い方は https://www.fugaku.r-ccs.riken.jp/faq/20220324_01 を参照

ログインノードの使用に関する注意

ログインノード (login 1～6) は不特定多数の利用者で共有する重要なシステムであり、 安定稼働に向けて下記の事項を守ること。

- ログインノードにて、以下の目安を超過するメモリを使用するプロセスや大量プロセスの生成は控えること
  - 使用メモリ量の大きいプロセスを生成する必要がある場合は**プリポスト環境**を利用すること
  - ログインノード 1台で、以下を超過しないよう留意すること

	1利用者の最大資源量の目安
最大スレッド数	8スレッド
最大メモリ容量	12 GB

- 制限超過時は、システムにて使用メモリ量の多いプロセスを強制終了し、当該プロセスを起動した利用者へ通知される

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

提供中のコンパイラ種別

「富岳」では以下の種類のコンパイラを提供。

種別	ログインノード（Intel）	ログインノード（Arm）	計算ノード
富士通コンパイラ	クロスコンパイラ	-	ネイティブコンパイラ
GCC	クロスコンパイラ	CentOS標準添付	ネイティブコンパイラ
Clang/LLVM	クロスコンパイラ	（OSS）	ネイティブコンパイラ
Intel	Intel oneAPI Base	-	-
Arm	-	Arm Compiler for Linux	-
Java	（OSS）	-	（OSS）

- クロスコンパイラ
  - ログインノード上で動作し、計算ノード用のバイナリを出力するコンパイラ
- ネイティブコンパイラ
  - 計算ノード上で動作し、計算ノード用のバイナリを出力するコンパイラ

※OSS及びArmコンパイラは利用環境提供のみで動作保証・サポートはなし

富士通コンパイラ

富士通コンパイラでは、Fortran／C／C++のコンパイルが可能。

■ コンパイルコマンド

種別	コンパイルコマンド		コンパイルモード	自動並列化オプション	OpenMP化オプション
	クロス	ネイティブ			
Fortran	frtpx	frt	-	-Kparallel	-Kopenmp
C	fccpx	fcc	tradモード	-Kparallel	-Kopenmp
			clangモード	-	-fopenmp
C++	FCCpx	FCC	tradモード	-Kparallel	-Kopenmp
			clangモード	-	-fopenmp

- 自動並列有効化オプション、OpenMP化オプションは、デフォルト設定では無効
- 分割コンパイルで、コンパイル時に自動並列化オプション・OpenMP化オプションを指定した場合、リンク時にも同じオプションが必要
- 自動並列化オプション、OpenMP化オプションは同時指定可能

■ MPIプログラムのコンパイルコマンド

種別	コンパイルコマンド		MPIライブラリのヘッダーファイルまたはモジュール情報ファイル
	クロス	ネイティブ	
Fortran	mpifrtpx	mpifrt	mpif.h / mpi.mod mpif-ext.h / mpi_ext.mod mpi_f08.mod mpi_f08_ext.mod
C	mpifccpx	mpifcc	mpi.h mpi-ext.h
C++	mpiFCCpx	mpiFCC	

- 利用者は、MPIのヘッダファイル／ライブラリの位置を意識することなく、コンパイル／リンクすることが可能
- 分割コンパイルで、コンパイル時にMPI用のコンパイルコマンドを用いた場合、リンク時にもMPI用コンパイルコマンドが必要

- Fortranのコンパイル方法
  - MPIライブラリを使用しない場合

```
[_LNlogin]$ frtpx [コンパイルオプション] ソースファイル名
```

    - OpenMP並列のコンパイル例

```
[_LNlogin]$ frtpx -Kfast,openmp sample.f08
```

  - MPIライブラリを使用する場合

```
[_LNlogin]$ mpifrtpx [コンパイルオプション] ソースファイル名
```

    - ハイブリッド並列（OpenMP利用）のコンパイル例

```
[_LNlogin]$ mpifrtpx -Kfast,openmp sample.f08
```

■ Fortranの基本的な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の作成までを実行
`-o` *exe_file*	実行可能ファイル名／オブジェクトファイル名を*exe_file*に変更
`-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` ■ デフォルト設定は`-O2`
`-Kfast`	高速化のための最適化オプションを誘導
`-Ksimd[=1|2|auto]`	SIMD拡張命令を利用したオブジェクトを生成 ■ `-O2`オプション以上が有効で`-Ksimd`省略時は、`-Ksimd=auto`オプションが適用される ■ `-O2`オプション以上が有効な場合にのみ有効
`-Kparallel`	自動並列を行う ■ デフォルト設定は`-Knoparallel` ■ `-O0`, `-O1`オプションとの併用は不可
`-Kopenmp`	OpenMP Fortran仕様のディレクティブを有効化 ■ デフォルト設定は`-Knoopenmp`

※Fortranコンパイラのコンパイルオプションについては、Fortran使用手引書「2.2 翻訳時オプション」を参照。

- Fortranのコンパイルの推奨オプション
  - 性能重視の場合

```
-Kfast,openmp[,parallel]
```

    - -Kfast
      - SIMD化によるSVE活用、ソフトウェア・パイプライニングによる命令レベルの並列性向上、最適化による演算順序変更、逆数近似演算の利用など、A64FXの性能を最大限に引き出すためのオプション

      ※A64FXは「富岳」のCPU

      ※詳細は「Fortran使用手引書」を参照
    - -Kparallel
      - 自動並列化を行うかどうかを指示
  - 精度重視かつある程度の最適化を求める場合

```
-Kfast,openmp[,parallel],fp_precision
```

    - -Kfp_precision
      - 精度に影響のあるすべての最適化を抑止

※各オプションで誘導されるオプションについては「Fortran 使用手引書」、及び、「富岳」利用セミナー 中級編 のテキストを参照

富士通コンパイラ

■ C言語コンパイラのモード

モード名	翻訳オプション	説明
tradモード	-Nnoclangを指定 または省略	「京」およびPRIMEHPC FX100以前のシステム向け富士通コンパイラがベース
clangモード	-Nclangを指定	オープンソースソフトウェア（OSS）であるClang/LLVMコンパイラがベース

■ 各モードの特徴

項目	trad	clang
「京」、FX100で作成した資源の利用	○	×
HPC向けのチューニング	○	○ （要オプション指定）
OSSアプリ翻訳の利便性（GCC互換性）	×	○
C++17の新しい言語規格のサポート	○ （一部サポート）	○
ACLE、FP16のサポート	×	○

※C++も同様。以下、C++言語は説明を省略

- C言語tradモードのコンパイル方法
  - MPIライブラリを使用しない場合

```
[_LNlogin]$ fccpx [コンパイルオプション] ソースファイル名
```

    - OpenMP並列のコンパイル例

```
[_LNlogin]$ fccpx -Kfast,openmp sample.c
```

  - MPIライブラリを使用する場合

```
[_LNlogin]$ mpifccpx [コンパイルオプション] ソースファイル名
```

    - ハイブリッド並列（OpenMP並列を利用）のコンパイル例

```
[_LNlogin]$ mpifccpx -Kfast,openmp sample.c
```

■ C言語tradモードの基本的なコンパイルオプション

※赤字はFortranとの違い

コンパイルオプション	説明
`-c`	オブジェクトファイルの作成までを実行
`-o exe_file`	実行可能ファイル名／オブジェクトファイル名を*exe_file*に変更
`-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 ■ デフォルト設定は-O2
`-Kfast`	高速化のための最適化オプションを誘導
`-Ksimd[=1|2|auto]`	SIMD拡張命令を利用したオブジェクトを生成 ■ -O2オプション以上が有効かつ-Ksimd省略時は、-Ksimd=autoオプションが適用される ■ -O2オプション以上が有効な場合にのみ有効
`-Kparallel`	自動並列を行う ■ デフォルト設定は-Knoparallel ■ -O0, -O1オプションとの共存は不可
`-Kopenmp`	OpenMP C仕様のディレクティブを有効化 ■ デフォルト設定は-Knoopenmp。

※ Cコンパイラのコンパイルオプションについては、C言語使用手引書「2.2 翻訳時オプション」を参照。

C言語tradモードのコンパイルの推奨オプション

性能重視の場合

```
-Kfast,openmp[,parallel]
```

- -Kfast
  - SIMD化によるSVE活用、ソフトウェア・パイプライニングによる命令レベルの並列性向上、最適化による演算順序変更、逆数近似演算の利用など、A64FXの性能を最大限に引き出すためのオプション

    ※A64FXは「富岳」のCPU

    ※詳細は「C言語使用手引書」を参照
- -Kparallel
  - 自動並列化を行うかどうかを指示

精度重視かつある程度の最適化を求める場合

```
-Kfast,openmp[,parallel],fp_precision
```

- -Kfp_precision
  - 精度に影響のあるすべての最適化を抑止

※各オプションで誘導されるオプションについては「C言語使用手引書」、及び、「富岳」利用セミナー 中級編 のテキストを参照

- C言語clangモードのコンパイル方法
  - MPIライブラリを使用しない場合

```
[_LNlogin]$ fccpx [-Nclangを含むコンパイルオプション] ソースファイル名
```

  - OpenMP並列のコンパイル例

```
[_LNlogin]$ fccpx –Nclang –Ofast -fopenmp sample.c
```

  - MPIライブラリを使用する場合

```
[_LNlogin]$ mpifccpx [-Nclangを含むコンパイルオプション] ソースファイル名
```

  - ハイブリッド並列のコンパイル例

```
[_LNlogin]$ mpifccpx –Nclang –Ofast -fopenmp sample.c
```

■ C言語clangモードの基本的なコンパイルオプション

※赤字はtradモードとの違い

コンパイルオプション	説明
-c	オブジェクトファイルの作成までを実行
-o *exe_file*	実行可能ファイル名／オブジェクトファイル名を*exe_file*に変更
-O[0|1|2|3|fast]	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 ■ デフォルト設定は-O2 ■ -Ofastはtradモードの-Kfastに対応
-fvectorize	SIMD拡張命令を利用したオブジェクトを生成 ■ tradモードの-Ksimdと同じ指示
-fopenmp	OpenMP C仕様のディレクティブを有効化 ■ デフォルト設定は-fnoopenmp

※ clangモードには自動並列化オプションなし
※ Cコンパイラの clangモードのコンパイルオプションについては、C言語使用手引書「9.1.2 翻訳時オプション」を参照
※ Clang/LLVMのオプションはほぼすべて利用可能
※ 上記の節に掲載していないオプションを指定した場合の動作保証はなし

- C言語clangモードのコンパイル推奨オプション
  - 以下のオプションの指定を推奨

```
-Ofast
```

  - -Ofastオプション
    - 演算の評価方法の変更、逆数近似演算、関数のインライン展開など、A64FX上で高速に動くオブジェクトプログラムの作成を指示するオプション

※A64FXは「富岳」のCPU

※詳細は「C言語使用手引書」 を参照

- 最適化は演算結果に影響を与える場合あり

※詳細は、「C言語使用手引書」-「第9章 clangモード」-「浮動小数点演算に対する最適化とその副作用」を参照

富士通コンパイラでは組込みデバッグ機能と並列実行デバッグ機能が利用可能。

- 組込みデバッグ機能

  デバッグ用オプションをつけてコンパイルすることで実行時に各種検査が可能。実行時間が通常より長くなる。

  - 検査項目
    - 未定義データの引用検査
      - Fortran：`-Hu`
    - 配列範囲の検査
      - Fortran：`-Hs`
      - C/C++（tradモード）　：`-Nquickdbg=subchk`
      - C/C++（clangモード）　：`-fsanitize=undefined`

詳細は「利用手引書 言語開発環境編 3. 富士通コンパイラ」内のコンパイラ・モードごとの「組込みデバッグ機能」参照。(Fortran, C(trad), C(clang), C++(trad), C++(clang))

- 並列実行デバッガ機能
  - mpiexec 実行時にオプションで指定することでGDB（GNUデバッガ）を使用した調査・制御が可能。
  - 各種変数の情報を取得するため、翻訳時に-gオプションの指定推奨。
  - 異常終了調査機能
    - 異常終了時にバックトレースなどの実行情報を取得

```
$ mpiexec -fjdbg-sig signal
```

  - デッドロック調査機能
    - 終了しない場合や応答しない場合にバックトレースなどの実行情報を取得

```
$ mpiexec -fjdbg-dlock
```

  - 重複除去機能
    - 調査機能の結果ファイルを加工し可読性を向上

```
$ fjdbg_summary
```

詳細は「利用手引書 言語開発環境編 3.10. 並列実行デバッガ機能」参照。

- 並列実行デバッガ機能 (つづき)
  - mpiexec 実行時にオプションで指定することでGDB（GNUデバッガ）を使用した調査・制御が可能。
  - 各種変数の情報を取得するため、翻訳時に-gオプションの指定推奨。
  - コマンドファイルによるデバッガ制御機能
    - コマンドファイルを用いてプロセスごとに異なるデバッガ制御を実施
    - ほかの調査機能と同時に使用できない

```
$ mpiexec -gdbx "[ rank-no: ] command-file [ ;... ]"
```

詳細は「利用手引書 言語開発環境編 3.10. 並列実行デバッガ機能」参照。

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

スーパーコンピュータ「富岳」では、以下の数値計算ライブラリが用意されている。
本セミナーでは特に、BLAS, LAPACK, ScaLAPACKについて紹介する。

- SSL II
- C-SSL II
- BLAS
  - ベクトル演算や行列演算を行なうライブラリ
  - 公式サイト http://www.netlib.org/blas/
- LAPACK
  - 線形代数の問題を解くライブラリ
  - 公式サイト http://www.netlib.org/lapack/
- ScaLAPACK
  - 線形代数分野の計算をメッセージパッシングで並列化したライブラリ
  - 公式サイト http://www.netlib.org/scalapack/
- 高速4倍精度基本演算ライブラリ

概要

- BLAS, LAPACK, ScaLAPACKについて
  - Fortran, C, C++で利用可能
  - コンパイル時に、オプションによって数学ライブラリの使用を指定する
  - スタティックリンク版と共有ライブラリ版があり、アーカイブの構成および結合方法が互いに異なる
- スタティックリンク版の特徴
  - コンパイルの際、ライブラリ自体を実行モジュールに含める
  - 実行モジュールのファイルサイズが大きくなるためにメモリに負荷が掛かる可能性がある
- 共有ライブラリ版の特徴
  - コンパイルの際、ライブラリを実行モジュールに含めない
  - 実行モジュールのファイルサイズを小さくできるが、共有ライブラリを最初に読み込む際、そのメモリアドレスを解決するための時間が発生する

スタティックリンク版の利用方法

スタティックリンク版の BLAS, LAPACK, ScaLAPACK について、各言語のコンパイル例を以下に示す。

- スタティックリンク版の Fortran コンパイル例
  - 逐次／BLAS,LAPACK（逐次版）
    - `-SSL2`: 逐次版の BLAS, LAPACK に必要なオプション

```
[_LNlogin]$ frtpx -Kfast sample.f90 -SSL2
```

  - OpenMP／BLAS,LAPACK（スレッド並列版）
    - `-SSL2BLAMP`: スレッド並列版の BLAS, LAPACK に必要なオプション
    - スレッド並列化オプション `-Kopenmp` または `-Kparallel` が必要

```
[_LNlogin]$ frtpx -Kfast,openmp sample.f90 -SSL2BLAMP
```

  - MPI／ScaLAPACK／BLAS,LAPACK（逐次版）
    - `-SCALAPACK`: ScaLAPACK に必要なオプション
    - MPIプログラム用のコンパイルコマンド `mpifrtpx` を使う必要あり

```
[_LNlogin]$ mpifrtpx -Kfast sample.f90 -SCALAPACK -SSL2
```

スタティックリンク版の利用方法

- スタティックリンク版の C コンパイル例
  - 逐次／BLAS,LAPACK（逐次版）
    - `-SSL2`: 逐次版の BLAS, LAPACK に必要なオプション

```
[_LNlogin]$ fccpx -Kfast sample.c -SSL2
```

  - OpenMP／BLAS,LAPACK（スレッド並列版）
    - `-SSL2BLAMP`: スレッド並列版の BLAS, LAPACK に必要なオプション
    - スレッド並列化オプション `-Kopenmp` または `-Kparallel` が必要

```
[_LNlogin]$ fccpx -Kfast,openmp sample.c -SSL2BLAMP
```

  - MPI／ScaLAPACK／BLAS,LAPACK（逐次版）
    - `-SCALAPACK`: ScaLAPACK に必要なオプション
    - MPIプログラム用のコンパイルコマンド `mpifccpx` を使う必要あり

```
[_LNlogin]$ mpifccpx -Kfast sample.c -SCALAPACK -SSL2
```

スタティックリンク版の利用方法

- スタティックリンク版の C++ コンパイル例
  - 逐次／BLAS,LAPACK（逐次版）
    - `-SSL2`: 逐次版の BLAS, LAPACK に必要なオプション

```
[_LNlogin]$ FCCpx -Kfast sample.cc -SSL2
```

  - OpenMP／BLAS,LAPACK（スレッド並列版）
    - `-SSL2BLAMP`: スレッド並列版の BLAS, LAPACK に必要なオプション
    - スレッド並列化オプション `-Kopenmp` または `-Kparallel` が必要

```
[_LNlogin]$ FCCpx -Kfast,openmp sample.cc -SSL2BLAMP
```

  - MPI／ScaLAPACK／BLAS,LAPACK（逐次版）
    - `-SCALAPACK`: ScaLAPACK に必要なオプション
    - MPIプログラム用のコンパイルコマンド `mpiFCCpx` を使う必要あり

```
[_LNlogin]$ mpiFCCpx -Kfast sample.cc -SCALAPACK -SSL2
```

スタティックリンク版の利用方法

- 注意点
  - コンパイル時、`-SSL2` などのライブラリ関連オプションは、ソースファイルの後に記述する
  - SSL II と BLAS, LAPACK の関係について
    - ライブラリの実体はSSL IIとBLAS, LAPACKを両方含んでいるため、どちらも `-SSL2` あるいは `-SSL2BLAMP` で呼び出す
    - サブルーチン名称が異なるため、それぞれのライブラリは干渉しない
  - SVE (scalable vector extension) に関して
    - SVEとは: 可変長のベクトルレジスタに対応したベクトル演算のための拡張命令セット
    - SVEを使用するか否かで、結合されるライブラリの実体が変わる
    - デフォルト設定（省略時）は、`-KSVE`
    - `-KSVE` が有効であれば、SVEを使用したライブラリを結合
    - `-KNOSVE` が有効であれば、汎用のライブラリを結合

共有ライブラリ版の利用方法

共有ライブラリ版の BLAS, LAPACK, ScaLAPACK についてコンパイルオプションの一覧及びコンパイル例を示す。

- Fortran のコンパイルオプション
  - 以下の表に示すオプションを指定
  - スレッド並列版の場合 `-Kopenmp` または `-Kparallel` の指定も必要

種類	逐次版BLAS, LAPACK	スレッド並列版BLAS,LAPACK	ScaLAPACK
LP64, 汎用版	`-lfjlapack`	`-lfjlapackex`	`-lfjscalapack`
LP64、SVE版	`-lfjlapacksve`	`-lfjlapackexsve`	`-lfjscalapacksve`
ILP64, 汎用版	`-lfjlapack_ilp64`	`-lfjlapackex_ilp64`	-
ILP64, SVE版	`-lfjlapacksve_ilp64`	`-lfjlapackexsve_ilp64`	-

共有ライブラリ版の利用方法

- C, C++ のコンパイルオプション
  - 以下の表に示すオプションを指定
  - スレッド並列版の場合 `-Kopenmp` または `-Kparallel` を指定
  - Fortran との違い
    - `-SSL2` または `-SSL2BLAMP` を追加指定
    - ScaLAPACK 使用の場合は、 `-SCALAPACK` を追加指定
    - ILP64 使用の場合は `-I${FJSVXTCLANGA}/include/lapack_ilp64` を指定

種類	逐次版BLAS, LAPACK	スレッド並列版BLAS,LAPACK	ScaLAPACK
LP64, 汎用版	`-lfjlapack`	`-lfjlapackex`	`-lfjscalapack`
LP64、SVE版	`-lfjlapacksve`	`-lfjlapackexsve`	`-lfjscalapacksve`
ILP64, 汎用版	`-lfjlapack_ilp64`	`-lfjlapackex_ilp64`	-
ILP64, SVE版	`-lfjlapacksve_ilp64`	`-lfjlapackexsve_ilp64`	-

共有ライブラリ版の利用方法

- Fortran コンパイル例
  - 汎用版BLAS, LAPACK逐次版

```
[_LNlogin]$ frtpx -Kfast sample.f -lfjlapack
```

  - SVE版BLAS、LAPACK スレッド並列版

```
[_LNlogin]$ frtpx -Kfast,openmp sample.f -lfjlapackexsve
```

  - 逐次プログラム翻訳→SVE版BLAS、LAPACK スレッド並列版結合

```
[_LNlogin]$ frtpx -Kfast -c sample.f
[_LNlogin]$ frtpx -Kfast,openmp sample.o -lfjlapackexsve
```

  - SVE版ScaLAPACK / SVE版BLAS、LAPACKスレッド並列版

```
[_LNlogin]$ mpifrtpx -Kfast,openmp sample.f -lfjscalapacksve -lfjlapackexsve
```

共有ライブラリ版の利用方法

- C,C++コンパイル例
  - C / 汎用版BLAS, LAPACK逐次版

```
[_LNlogin]$ fccpx -Kfast sample.c -lfjlapack -SSL2
```

  - C++ / SVE版BLAS、LAPACK スレッド並列版

```
[_LNlogin]$ FCCpx -Kfast,openmp sample.cpp -lfjlapackexsve -SSL2
```

  - C / 逐次プログラム翻訳→SVE版BLAS、LAPACK スレッド並列版結合

```
[_LNlogin]$ fccpx -c -Kfast sample.c
[_LNlogin]$ fccpx -Kfast,openmp sample.o -lfjlapackexsve -SSL2
```

  - C++ / SVE版ScaLAPACK / SVE版BLAS、LAPACKスレッド並列版

```
[_LNlogin]$ mpiFCCpx -Kfast,openmp sample.cpp -lfjscalapacksve -lfjlapackexsve -SSL2 -SCALAPACK
```

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

スーパーコンピュータ「富岳」において、ユーザー環境は Environment Modules package を通じて管理されており、module コマンドを使うことでアプリケーションの利用に必要な環境変数を簡単に設定可能である。

- 公式サイト
  - http://modules.sourceforge.net
- Environment Modules で設定できる環境変数の一例
  - PATH
    - 実行ファイルを参照する場所を指定
  - MANPATH
    - man コマンド使用時、リファレンスマニュアルページを検索する場所を指定
  - LD_LIBRARY_PATH
    - コンパイラオプション -l 使用時、リンカーが検索する場所を指定
    - 例えばコンパイルオプションに -l*hoge* が指定されていた時、lib*hoge*.a または lib*hoge*.so というファイルが参照されるが、検索する場所のデフォルト値が環境変数 LD_LIBRARY_PATH に設定されている

- modulefile とは
  - アプリケーションの利用に必要となる各種環境変数の設定が書かれたファイル、module コマンドによって解釈される

- 主な module コマンド一覧 (次頁以降で詳述)

コマンド	説明
`module avail`	利用可能な modulefile の一覧表示
`module list`	ロード済みの modulefile の一覧表示
`module load` *`modulefile`*	modulefile をロード
`module unload` *`modulefile`*	modulefile をアンロード
`module purge`	全ての modulefile をアンロード
`module switch [`*`modulefile1`*`] `*`modulefile2`*	modulefile を切り換える
`module show` *`modulefile`*	modulefile の設定内容を表示

コマンドの使用例

以下に主要な module コマンドの説明と、使用例を示す。

- module avail
  - 提供されている modulefile の一覧を表示する

```
[_LNlogin]$ module avail
---------------------------------------- /opt/intel/oneapi/modulefiles -----------------------------------------
advisor/2025.1                  compiler/2025.1.0       dpct/latest                  ishmem/1.3.0   umf/latest
advisor/latest                  compiler/latest         dpl/2022.6                   ishmem/latest  vtune/2025.1
ccl/2021.13.1                   compiler32/2024.2.1     dpl/2022.8                   mkl/2024.2     vtune/latest
ccl/2021.15.0                   debugger/2024.2.1       dpl/latest                   mkl/2025.1
ccl/latest                      debugger/2025.1.0       ifort/2024.2.1               mkl/latest
compiler-intel-llvm/2024.2.1    debugger/latest         ifort32/2024.2.1             mkl32/2024.2
compiler-intel-llvm/2025.1.0    dev-utilities/2024.2.0  intel_ipp_ia32/2021.12       mpi/2021.13
compiler-intel-llvm/latest      dev-utilities/2025.1.0  intel_ipp_intel64/2021.12    mpi/2021.15
compiler-intel-llvm32/2024.2.1  dev-utilities/latest    intel_ipp_intel64/2022.1     mpi/latest
compiler-rt/2024.2.1            dnnl/3.5.0              intel_ipp_intel64/latest     tbb/2021.13
compiler-rt/2025.1.0            dnnl/3.7.1              intel_ippcp_ia32/2021.12     tbb/2022.1
compiler-rt/latest              dnnl/latest             intel_ippcp_intel64/2021.12  tbb/latest
compiler-rt32/2024.2.1          dpct/2024.2.0           intel_ippcp_intel64/2025.1   tbb32/2021.13
compiler/2024.2.1               dpct/2025.1.0           intel_ippcp_intel64/latest   umf/0.10.0

------------------------------------ /work/Fugaku-environment/modulefiles -------------------------------------
lang/tcsds-1.2.39  lang/tcsds-1.2.40  lang/tcsds-1.2.41(default)
```

コマンドの使用例

- `module load`
  - 提供されている modulefile を読み込み、指定された環境変数を追加、アプリケーションを使える状態にする
- `module unload`
  - `module load` で追加された環境変数を削除する
- `module list`
  - load 済みの modulefile の一覧を表示する

コマンドの使用例

■ 言語環境 lang を list で表示, unload, load する例

```
[_LNlogin]$ module list
Currently Loaded Modulefiles:
 1) lang/tcsds-1.2.41(default)
[_LNlogin]$ module unload lang
[_LNlogin]$ module list
No Modulefiles Currently Loaded.
[_LNlogin]$ module load lang
[_LNlogin]$ module list
Currently Loaded Modulefiles:
 1) lang/tcsds-1.2.41(default)
```

コマンドの使用例

- module switch
  - 同じルート名(スラッシュより前が同じ)を持つ modulefile を切り換え

```
[_LNlogin]$ module list
Currently Loaded Modulefiles:
 1) lang/tcsds-1.2.41(default)
[_LNlogin]$ module switch lang/tcsds-1.2.40
[_LNlogin]$ module list
Currently Loaded Modulefiles:
  1) lang/tcsds-1.2.40
```

- module purge
  - 全ての modulefile のアンロード

```
[_LNlogin]$ module list
Currently Loaded Modulefiles:
 1) lang/tcsds-1.2.41(default)
[_LNlogin]$ module purge
[_LNlogin]$ module list
No Modulefiles Currently Loaded.
```

コマンドの使用例

- module show
  - modulefile の設定内容を表示

```
[_LNlogin]$ module show lang
-------------------------------------------------------------------
/work/Fugaku-environment/modulefiles/lang/tcsds-1.2.41:

module-whatis   {Fujitsu Compiler 4.12.0 (Fortran/C/C++/Tool)}
conflict        lang
setenv          FJSVXTCLANGA /opt/FJSVxtclanga/tcsds-1.2.41
prepend-path    PATH /opt/FJSVxtclanga/tcsds-1.2.41/bin
prepend-path    LD_LIBRARY_PATH /opt/FJSVxtclanga/tcsds-1.2.41/lib64
-------------------------------------------------------------------
```

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

スーパーコンピュータ「富岳」では、Spack を用いてオープンソースソフトウェア（OSS）を管理、提供する。

- Spackとは
  - 主にスーパーコンピュータ向けのパッケージ管理ツール
  - 公式ページ `https://spack.io/`

- Spack の主な特徴
  - Spack 用に書かれた OSS のレシピを用いて、依存関係も含めてソースからコンパイルできる
  - コンパイラや適用するライブラリの切り替えが容易
  - 「チェイニング」の機能を用いて、一つの Spack インスタンス (※) が管理するパッケージ群から、他の Spack インスタンスが管理するパッケージ群を参照・利用可能
  - 「富岳」上では、システム側で用意したパブリックインスタンスと、各自が管理するプライベートインスタンスを使い分けられる

※ Spack インスタンスとは、Spack の実行ファイル及び関連ファイル、及びそれらによって管理されるパッケージ群（がある一つのフォルダの中身全体）を指す

パブリック・インスタンス

「富岳」では、システム側でインストール済みのOSSをパブリック・インスタンスによって提供している。パブリックインスタンスは環境の読み込みのみで利用できる。

- パブリック・インスタンス利用のための環境設定
  - bash の場合：

```
$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh
```

  - csh/tcsh の場合：

```
$ setenv SPACK_ROOT /vol0004/apps/oss/spack
$ source /vol0004/apps/oss/spack/share/spack/setup-env.csh
```

  - バッチ型ジョブで利用する場合は、ジョブスクリプトにも同様の記述をする
  - ログインシェルの .bashrc 等にこの記述をしないこと。ファイルシステムにトラブルがあったとき最悪ログインすらできなくなるおそれがある

Spack の主なコマンド

Spack の主なコマンドを以下に示す。

- spack find
  - インストール済みパッケージの確認
  - オプション -x で、管理者が明示的にインストールしたアプリのみ表示
    - 「富岳」で提供されているOSSの確認に有効
    - オプション -x を付けない場合、提供アプリの依存関係により非明示的にインストールされた多数のパッケージも含めて表示される

```
$ spack find -x
-- linux-rhel8-a64fx / fj@4.8.1 ----------------------------------
ffvhc-ace@0.1

-- linux-rhel8-a64fx / fj@4.10.0 ---------------------------------
adios2@2.9.2            frontistr@5.5             modylas-new@1.1.0       paraview@5.11.2         py-seaborn@0.12.2
akaikkr@2002v010        fugaku-frontistr@master   mptensor@0.3.0          parmetis@4.0.3          py-spglib@2.0.2
akaikkr@2021v001        fujitsu-fftw@1.1.0        mvmc@1.2.0              petsc@3.19.6            py-toml@0.10.2
akaikkr@2021v002        fujitsu-mpi@head          n2p2@2.1.4              pfapack@2014-09-17      py-xarray@2023.7.0
alamode@1.3.0           fujitsu-ssl2@head         nemo@4.2.0              phase0@2021.02          python@3.11.6
alamode@1.4.2           genesis@2.1.1             netcdf-c@4.9.2          phase0@2021.02          quantum-espresso@6.5
alamode@1.5.0           genesis@2.1.1             netcdf-cxx@4.2          phase0@2023.01          quantum-espresso@6.6
batchedblas@1.0         genesis@2.1.2             netcdf-cxx4@4.3.1       phase0@2023.01          quantum-espresso@6.7
bcftools@1.12           genesis@2.1.2             netcdf-fortran@4.6.1    picard@3.0.0            quantum-espresso@6.8
bedtools2@2.31.0        gmt@6.2.0                 netlib-lapack@3.10.1    povray@3.7.0.8          quantum-espresso@7.0
biobambam2@2.0.177      grads@2.2.3               netlib-scalapack@2.2.0  py-ase@3.21.1           quantum-espresso@7.1
blitz@1.0.2             gromacs@2020.6            nwchem@master           py-dask@2022.10.2       quantum-espresso@7.2
boost@1.83.0            gromacs@2021.5            octa@8.4                py-devito@4.8.1         quantum-espresso@7.3

…… (略)
```

Spack の主なコマンド

- `spack load`
  - 環境変数 PATH などを変更し、Spack インスタンスが管理しているパッケージを利用可能にする
  - パッケージ octa に対する使用例

```
$ spack load octa
```

- `spack unload`
  - 前述の `spack load` により導入された環境変数などを元の状態に戻す
  - パッケージ octa に対する使用例

```
$ spack unload octa
```

複数パッケージの区別

Spack 上で同一名のパッケージが存在する場合、単純にパッケージ名を指定して `spack load` しただけではエラーになる。その場合、特定のパッケージを選んで使用するには、パッケージ同士を区別する記述をする。

■ 同一名のパッケージが存在し、`spack load` がエラーになる例

```
$ spack load cmake
==> Error: cmake matches multiple packages.
  Matching packages:
    fvvft23 cmake@3.17.1%fj@4.10.0 arch=linux-rhel8-a64fx
    4axaai6 cmake@3.21.4%fj@4.10.0 arch=linux-rhel8-a64fx
    ecejifv cmake@3.24.3%fj@4.10.0 arch=linux-rhel8-a64fx
    ys33lmx cmake@3.27.7%gcc@8.5.0 arch=linux-rhel8-a64fx
    u2uczoe cmake@3.27.7%gcc@8.5.0 arch=linux-rhel8-skylake_avx512
    zkni3sm cmake@3.27.7%gcc@13.2.0 arch=linux-rhel8-a64fx
    ylpx52y cmake@3.27.7%gcc@13.2.0 arch=linux-rhel8-cascadelake
    jg5g76j cmake@3.27.7%fj@4.10.0 arch=linux-rhel8-a64fx
    ussgjuq cmake@3.27.7%fj@4.10.0 arch=linux-rhel8-a64fx
    7664nus cmake@3.27.7%fj@4.10.0 arch=linux-rhel8-a64fx
  Use a more specific spec.
```

短縮ハッシュ値　バージョン番号　使用コンパイラ　アーキテクチャ

複数パッケージの区別

- バージョン番号を用いた指定
  - パッケージ名の後に@で指定

```
$ spack load lammps@20201029
$ spack load lammps@20220623.2
$ spack load lammps@20230802.3
```

- ビルドしたコンパイラでの指定
  - パッケージ名の後に % で指定

```
$ spack load tmux%fj@4.10.0
$ spack load tmux%gcc@13.2.0
```

複数パッケージの区別

- アーキテクチャでの指定
  - パッケージ名の後に arch= で指定

```
$ spack load tmux arch=linux-rhel8-cascadelake
$ spack load tmux arch=linux-rhel8-a64fx
```

  - アーキテクチャの詳細
    - linux-rhel8-cascadelake: ログインノード用にビルドされたもの
    - linux-rhel8-skylake_avx512: ログインノード用にビルドされたもの
    - linux-rhel8-a64fx: 計算ノード用にビルドされたもの

複数パッケージの区別

- ハッシュ値での指定
  - Spack ではパッケージ名に詳細なビルド条件を付記した spec に対して一意にハッシュ値が定まるようになっている
  - / 以降に短縮ハッシュ値(※)を書くことで指定

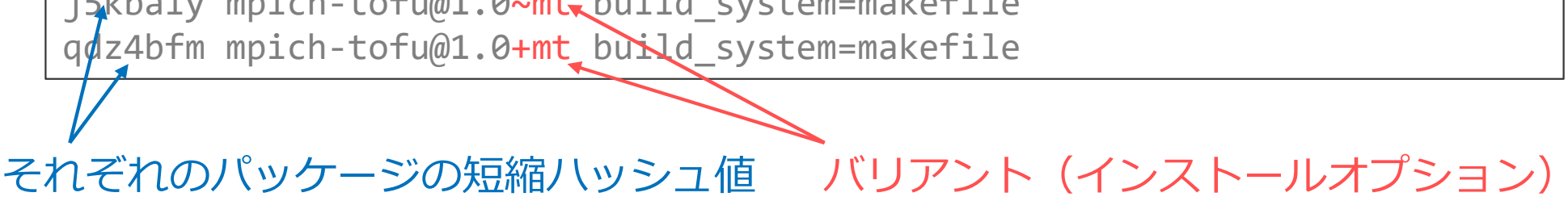
```
[_LNlogin]$ spack find -l python

……(略)……

-- linux-rhel8-cascadelake / gcc@13.2.0 --------------------------
yjlixq5 python@3.11.6  so5pyv6 python@3.11.6

……(略)……

[_LNlogin]$ spack load /yjlixq5
[_LNlogin]$ spack load /so5pyv6
```

※ 短縮ハッシュ値とは、識別用の文字列であるハッシュ値の最初の7文字であり、`spack find -l` で確認できる。短縮されていないフルハッシュ値は `spack find -L` で確認できる。

複数パッケージの区別

- バリアント（インストールオプション）の有効／無効の確認
  - 場合によっては、バージョン番号やアーキテクチャ等で区別できない複数のパッケージがインストールされていることがある
  - そのような場合、コマンド `spack find -lv` によって、バリアントで表現されるインストールの詳細を確認できる
  - 以下の例では、特に `~mt` と `+mt` となっている部分に違いがある
  - この時、`~` は該当のバリアントが無効になっていること、`+` は有効になっていることを示している

```
$ spack find -lv mpich-tofu@1.0
-- linux-rhel8-a64fx / gcc@8.5.0 ---------------------------------
j5kbaiy mpich-tofu@1.0~mt build_system=makefile
qdz4bfm mpich-tofu@1.0+mt build_system=makefile
```



利用可能なソフトウェア

「富岳」で提供されているソフトウェアの詳細は、富岳ウェブサイト上の以下のページで確認できる。

- https://www.fugaku.r-ccs.riken.jp/docs/software_r01

プライベート・インスタンス

前述のパブリック・インスタンスの他に、各ユーザがホームディレクトリ以下に Spack をインストールし（プライベート・インスタンス）、各自でインストールするアプリの管理に利用することができる。

プライベート・インスタンスを用いてアプリをインストールする際には、チェイニングによってパブリック・インスタンスで提供されているパッケージを参照することができる。

- プライベート・インスタンス利用の際の参考資料
  - 富岳 Spack 利用ガイド（富岳ウェブサイトに掲載）
  - Spack 公式ドキュメント (https://spack.readthedocs.io/en/stable/)

既知の問題と対処

現環境での Spack 利用において確認されている問題と、その対処方法について以下に記す。

- OS標準の動的ライブラリパスが上書きされる問題
  - Spackで何らかのパッケージをロードした後、プログラム実行時に以下の警告やエラーが発生することがある

```
[WARN] xos LPG 2002 - Failed to map HugeTLBfs for data/bss: /usr/bin/file The
e_type of elf header must be ET_EXEC when using libmpg. You can check it on your
load module by readelf -h command.
```

```
[WARN] xos LPG 2003 - Failed to map HugeTLBfs for data/bss: Layout problem with
segments 0 and 1:
        Segments would overlap.
```

```
libmpg BUG!! mpiexec: __mpg_resolve_libc_symbol[776]: Assertion
`__libc_calloc_fp != ((void *)0)' failed.
```

  - Spackでパッケージをロードした後に、以下のような LD_LIBRARY_PATH の追加によって抑止できる

```
export LD_LIBRARY_PATH=/lib64:$LD_LIBRARY_PATH
```

既知の問題と対処

- マルチノードジョブにおける性能劣化
  - Spackで提供するパッケージは第2階層ストレージ（※1）上に存在するため、マルチノードジョブからそれらを利用する場合、特定のストレージI/Oノードにアクセスが集中し、性能劣化を引き起こす可能性がある
  - そのような場合は、 必要なすべてのパッケージをロード後、 以下のようにLD_LIBRARY_PATH と PATH に設定されたパスに対して dir_transfer コマンドを実行（※2）することで性能劣化を避けられる

※1 LLIOの章参照

※2 これにより、参照される可能性のある共有ライブラリと実行ファイルが第1階層ストレージ上の第2階層ストレージのキャッシュ領域に配布され、アクセス集中を避けることができる

```
$ spack load xxx
$ echo $LD_LIBRARY_PATH | sed –e ‘s/:/¥n/g’ | grep ‘^/vol0004/apps/oss/spack’ |
xargs /home/system/tool/dir_transfer
$ echo $PATH | sed -e 's/:/¥n/g' | grep '^/vol0004/apps/oss/spack' | xargs
/home/system/tool/dir_transfer
```

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

「富岳」では、OSS のスクリプト言語は Spack にて提供されている。

■ 主なスクリプト言語とそのバージョン (2025/4/17)

名称	Spack (ログインノード)	Spack （計算ノード）
Python3	3.11.6	3.10.8, 3.10.13, 3.11.6 (*)
Ruby	-----	3.1.0
R	-----	4.3.0
Julia		1.9.3, 1.10.2

(*) NumPy, SciPy には、 「富岳」に最適化されたスレッド並列版ライブラリ fjlapackexsve（BLAS, LAPACK）を使用

- Spack 版 Python3 の利用方法
  - Python 依存パッケージをロードするとPython 本体も自動でロードされる

```
[_CNlogin]$ spack load /7hfnxfw   # py-pandasの短縮ハッシュタグ
[_CNlogin]$ which python # Python3が自動でロードされていることを確認できる
/vol0004/apps/oss/spack-v0.21/opt/spack/linux-rhel8-a64fx/fj-4.10.0/python-
3.10.13-bjhdatlk74sdqya3xxy2r6doz6y24iha/bin/python
[_CNlogin]$ spack load /pxi6xpt #py-pipの短縮ハッシュ
[_CNlogin]$ pip list
Package         Version
--------------- -------
Bottleneck      1.3.7
llvmlite        0.40.0
numba           0.57.0
numexpr         2.8.4
numpy           1.24.4   # pandasだけでなくnumpyもロードされている。
pandas          2.1.2
pip             23.1.2
python-dateutil 2.8.2
pytz            2023.3
setuptools      59.4.0
six             1.16.0
tzdata          2023.3
```

- 「富岳」で提供されている主なAIフレームワークと、利用方法
  - scikit-learn

```
[_CNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh
[_CNlogin]$ spack load py-scikit-learn
```

  - Keras

```
[_CNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh
[_CNlogin]$ spack load py-keras
```

  - PyTorch-1.7.0

```
[_CNlogin]$ export PATH=/vol0004/apps/oss/PyTorch-1.7.0/bin:$PATH
[_CNlogin]$ export LD_LIBRARY_PATH=/vol0004/apps/oss/PyTorch-1.7.0/lib:$LD_LIBRARY_PATH
```

  - TensorFlow-2.2.0

```
[_CNlogin]$ export PATH=/vol0004/apps/oss/TensorFlow-2.2.0/bin:$PATH
[_CNlogin]$ export LD_LIBRARY_PATH=/vol0004/apps/oss/TensorFlow-2.2.0/lib:$LD_LIBRARY_PATH
```

※ PyTorchとTensorFlowについては、Spackでの提供を準備中。現時点では、PATHとLD_LIBRARY_PATHを直接編集して使う必要がある。

Ruby, R, Julia

「富岳」では、計算ノード上で Ruby, R, Julia の利用が可能である。
利用方法を以下に示す。

■ Ruby の利用方法

```
[_CNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh
[_CNlogin]$ spack load ruby
[_CNlogin]$ ruby sample.rb
```

■ R の利用方法

```
[_CNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh
[_CNlogin]$ spack load r
[_CNlogin]$ Rscript sample.R
```

■ Julia の利用方法

```
[_CNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh
[_CNlogin]$ spack load julia@1.10.2
[_CNlogin]$ julia sample.jl
```

「富岳」では、Java コンパイラの利用が可能である。利用方法を以下に示す。

■ コンパイル環境設定

■ ログインノードで翻訳する場合: Spackで提供しているopenjdkを使用

```
[_LNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh # Spackの環境設定
[_LNlogin]$ spack load openjdk arch=linux-rhel8-cascadelake # OpenJDKの環境設定
```

■計算ノードで翻訳する場合: Spackで提供しているopenjdkを使用

```
[_CNlogin]$ . /vol0004/apps/oss/spack/share/spack/setup-env.sh # Spackの環境設定
[_CNlogin]$ spack load openjdk@17.0.8.1_1 arch=linux-rhel8-a64fx # OpenJDKの環境設定
```

■利用できる環境 (2025/4/17時点)

ノード	ソフトウェア名	言語	バージョン
ログインノード	OpenJDK	Java	11.0.20.1_1
計算ノード	OpenJDK	Java	11.0.20.1_1 17.0.8.1_1

- コンパイルコマンド
  - クロスコンパイラとネイティブコンパイラでコマンドは同じ
  - MPIライブラリを使用しない場合

```
[_LNlogin]$ javac ［コンパイルオプション］ソースファイル名
```

  - MPIライブラリを使用する場合

```
[_LNlogin]$ mpijavac ［コンパイルオプション］ソースファイル名
```

■ mpijavacの主なコンパイルオプション

オプション	説明
`--showme`	MPIプログラムの翻訳コマンドが javac コマンドを呼び出すときの呼び出し行を表示する、実際には翻訳処理は行わない
`--verbose`	MPIプログラムの翻訳コマンドが javac コマンドを呼び出すときの呼出し行を表示する、翻訳処理を行う
`--help、-help、-h`	ヘルプメッセージを表示する、実際には翻訳処理は行わない
`avac_arguments`	javac コマンドに渡すオプションを指定する
`-classpath`	独自に用意したjarファイルのパスを指定

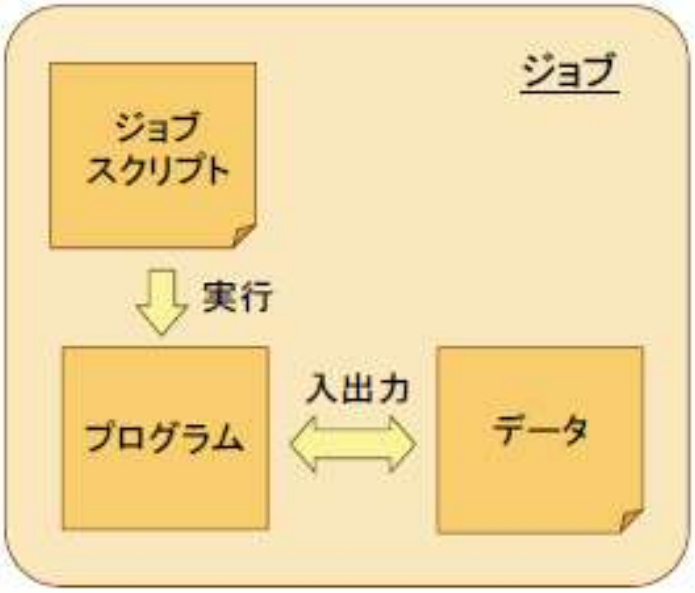
※ jar ファイルのパス指定において、`-classpath` オプションと環境変数 `CLASSPATH` の両方を設定した場合、`-classpath` オプションが優先される

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

ジョブについて

プログラム実行時に必要なジョブの概念について説明する。

- ジョブとは
  - プログラム・データ・ジョブスクリプトから構成される処理単位のこと
  - 「富岳」では、ユーザはプログラムの実行をジョブの形でジョブ運用ソフトウェアに依頼する
  - ジョブ運用ソフトウェアは、投入された複数のジョブに対して選択ポリシーに従った順位付けを行い、計算機資源を割り当て順次実行する



ジョブスクリプト

ジョブスクリプトの実体はシェルスクリプト。

■ ジョブスクリプト例：MPIを使用しない場合

```
#!/bin/bash    # 指定のない場合はユーザーのログインシェル
#PJM --gname hp250xxx        # 実行グループ名を指定
#PJM -L "node=1"             # ノード数
#PJM -L "rscgrp=small"       # リソースグループの指定
#PJM -L "elapse=60:00"       # ジョブの経過時間制限値
#PJM -s                      # 統計情報ファイル出力の指示

export OMP_NUM_THREADS=12    # 環境変数の設定

# execute job
./a.out                      # プログラムの実行
```

- --gname または --gid の指定は必須
- PJMで始まる行はジョブ実行コマンド（pjsub）のオプション
- 一度コメント行以外が現れると以下の #PJM はコメント行として無視される
- /dev/stdout, /dev/stderrへのリダイレクトはしないこと

■ ジョブスクリプト例：MPIを使用する場合

```
#!/bin/bash          # 指定のない場合はユーザーのログインシェル
#PJM --gname hp250xxx        # 実行グループ名を指定
#PJM -L "node=4"             # ノード数
#PJM -L "rscgrp=small"       # リソースグループの指定
#PJM -L "elapse=10:00"       # ジョブの経過時間制限値
#PJM --mpi "max-proc-per-node=4"
                             # 1ノードあたりのMPIプロセス数の上限値
#PJM -S

export OMP_NUM_THREADS=12
llio_transfer ./a.out  #共通ファイル配布（LLIOの章で説明）
mpiexec -n 16 ./a.out  #16プロセスで実行
```

- --gname または --gid の指定は必須
- 富岳の1ノードは4CMGで構成されているため、
  1ノードに対して4プロセスでの実行を推奨　※CMGについては後述

ジョブ実行時に利用するvolumeの指定

- ホームディレクトリがあるvolume以外を利用する場合、ジョブ投入時に環境変数PJM_LLIO_GFSCACHEに指定する必要がある。

```
#!/bin/bash          # 指定のない場合はユーザーのログインシェル
#PJM --gname hp250xxx           # 実行グループ名を指定
#PJM -L "node=1"               # ノード数
#PJM -L "rscgrp=small"         # リソースグループの指定
#PJM -L "elapse=10:00"         # ジョブの経過時間制限値
#PJM -x PJM_LLIO_GFSCACHE=/vol0004    # spack利用時はvol0004を指定
#PJM -s

. /vol0004/apps/oss/spack/share/spack/setup-env.sh
spack load dssp
export LD_LIBRARY_PATH=/lib64:$LD_LIBRARY_PATH
```

- --gname または --gid の指定は必須
- カレントディレクトリのvolumeはログインノードで pwd –P コマンドで確認可能
- /vol0001 (2ndfs領域)は指定せずとも使用可能

ログインノード上には、ジョブスクリプトの雛形を作成するためのコマンドが用意されている。

- コマンドおよびオプション
  - `make_jobscript`: ジョブスクリプトを生成し、標準出力
  - `--gname`: グループ名の指定（必須）
  - `--distribute-common-file`: 配布する共通ファイルを指定
  - `--use-directory`: 計算で使用するディレクトリを指定
  - `--use-spack`: Spack 利用の際に必須な記述を追加

```
[_LNlogin]$ make_jobscript --gname (your groupname) --distribute-common-file a.out --use-directory input_dir --use-spack
#!/bin/sh
#PJM -L node=XXXX
#PJM -L rscgrp=XXXX
#PJM -L elapse=XX:XX:XX
#PJM -N XXXX
#PJM -g (your groupname)
#PJM -x PJM_LLIO_GFSCACHE=/vol0004

. /vol0004/apps/oss/spack/share/spack/setup-env.sh
spack load XXXX
export LD_LIBRARY_PATH=/lib64:$LD_LIBRARY_PATH

/usr/bin/llio_transfer a.out
```

■ ジョブ実行コマンドの基本オプション（抜粋）

オプション名	機能
`-L "resource=value[,...]"`	ジョブのリソースを指定 (詳細は後述) ※`-L`は`--rsc-list`とも書ける（同じ意味を持つオプション）
`--mpi "parameter[,...]"`	MPIジョブの各種パラメーターを指定 (詳細は後述)
`-g gname | -g gid`	実行するジョブのプロセスが所属するグループ名またはグループIDを指定（必須） ※ それぞれ--gname, --gid というオプションでの指定もできる
`-j`	標準エラー出力を標準出力へ出力
`-m "mailoption[,...]"`	ジョブのステータス等についてメール通知を設定 ■ `--mail-list`の同時指定が必須
`--mail-list "mailaddress[,...]"`	メールの送信先を指定 ■ 複数指定時はコンマ (",") で区切る ■ 指定する文字列は255文字まで ■ 無効なメールアドレス指定時のエラー通知なし
`--name "name"`	ジョブの名前を指定 ■ ジョブ名は63バイトまでで、先頭は半角アルファベットのみ ■ 本オプションを指定しない場合、ルールに従ってスクリプトファイル名または"STDIN"がジョブ名に

■ ジョブ実行時のリソース指定

項目名	機能
-L "rscgrp=*rscgname*"	ジョブを投入するリソースグループ名を指定 ■ 最新のリソースグループ情報 https://www.fugaku.r-ccs.riken.jp/resource_group_config ■ 省略時の値は ■ small（バッチジョブ） ■ int（会話型ジョブ）
-L "node=*nodeshape*"	ジョブに割り当てるノード数および形状を指定 ■ 1次元の場合：node=N1 ■ 2次元の場合：node=N1xN2 ■ 3次元の場合：node=N1xN2xN3 ■ リソースグループsmallでは、 ノードの割り当て方式 torus/mesh の指定も可能 ■ 省略時の値は以下で確認 pjacl --rg <リソースグループ名>
-L "elapse=*elapsetimelimit*"	ジョブの経過時間制限値を設定 ■ elapsetimelimitは、"[[時間:]分:]秒" の形式 ■ -L "elapse=30"（30秒の場合） ■ -L "elapse=2:30"（2分30秒の場合） ■ -L "elapse=1:00:00"（1時間の場合） ■ 省略時の値 ■ 1分（バッチジョブ） ■ 10秒（会話型ジョブ）

■ MPIジョブの各種パラメタ指定

項目名	機能
`--mpi "shape=`*shape*`"`	静的に起動するプロセスの形状（各軸で使用するノードの数）を指定 ■ `-L`（`--rsc-list`）オプションのnodeで指定した値と同じ次元数を指定する必要あり ■ 省略時は、nodeで指定した値と同じ値
`--mpi "proc=`*num*`"`	静的に起動する最大のプロセス数を指定 ■ 省略時はshapeで指定した値の積 ■ （shapeで指定した値の積）×（ノード内のCPUコア数）より大きい値を指定した場合、ジョブの受付拒否
`--mpi "max-proc-per-node=`*mppnnum*`"`	1ノードに生成する最大のプロセス数を指定 ■ 省略時は、proc で指定した値を 1ノード内に生成するプロセス数に変換した値（A） ■ 指定された値が1ノード内のCPUコア数を超える場合、または上記Aより小さい場合は、ジョブの受付拒否

■ ジョブ統計情報の出力

項目名	機能
-s (小文字)	投入したジョブの統計情報をファイルに出力 ■ -sオプションとの併用は不可
-S (大文字)	-sオプションで出力される情報に加えて、投入したジョブのノード単位の情報を出力 ■ -sオプションとの併用は不可
--spath *pathname*	ジョブの統計情報を、*pathname*で指定されたファイルに出力 ■ -sまたは-Sオプションの指定が必要

■ ジョブの統計情報内容の詳細については以下を参照
- 「ジョブ運用ソフトウェア エンドユーザ向けガイド」
- 「ジョブ運用ソフトウェア コマンドリファレンス」－「3.3.2 pjstatsinfo」
- manマニュアル pjstatsinfo(7)

■ ジョブ実行時の制限値およびデフォルト値の確認方法

```
[_LNlogin]$ pjacl --rscgrp リソースグループ名
```

■ リソースグループ名に small を指定した場合の出力例

```
 (省略)
defines
    default rscunit             rscunit_ft01
    default rscgroup            small
 (省略)
pjsub option parameters
    (-L/--rsc-list)                         lower           upper           default
        (elapse=)                           00:01:00        72:00:00        00:01:00
        (adaptive elapsed time min)         00:01:00        72:00:00        00:01:00
        (adaptive elapsed time max)         00:01:01        144:00:00       144:00:00
 (省略)
        (node=)                             1               384             1
 (省略)
```

※リソースグループ名を指定しない場合は、デフォルトのリソースグループである small の制限値・デフォルト値が出力される

ジョブの種類

ジョブは実行形態や構造などによっていくつかの種類に分類される。

- ジョブの実行形態による分類（ジョブタイプ）
  - バッチジョブ
  - 会話型ジョブ

- ジョブの構造による分類（ジョブモデル）
  - 通常ジョブ
  - バルクジョブ
  - ステップジョブ
  - ワークフロージョブ

※全てバッチジョブ

まずは、基本となる通常ジョブについて、ジョブの投入・状態の表示・削除の方法及び結果の確認方法等を見ていく。

通常ジョブ

通常ジョブは特殊な処理形態を持たない一般的なバッチジョブ。投入順に対して実行順は保証されない。

- ジョブの投入
  - ジョブの投入はデータ領域から行うこと

```
[_LNlogin]$ pjsub ./sample.sh
[INFO] PJM 0000 pjsub Job 9714 submitted.
```

ジョブID

※ ジョブ投入時にエラーメッセージが表示された場合の対処法については、利用手引書 利用およびジョブ実行編「5.2.1. ジョブの投入」を参照

■ ジョブの状態の表示

```
[_LNlogin]$ pjstat

JOB_ID  JOB_NAME  MD  ST   USER   START_DATE      ELAPSE_LIM  NODE_REQUIRE  VNODE  CORE  V_MEM
238     job.sh    NM  RUN  user1  11/17 09:01:41  0001:00:00  12:2x3x2      -      -     -
239     bulk.sh   BU  RUN  user1  11/17 09:01:42  0001:00:00  12:2x3x2      -      -     -
240     step.sh   ST  RUN  user1  11/17 09:01:42  -           -             -      -     -
241     job2.sh   NM  RUN  user1  11/17 09:01:42  0001:00:00  2             -      -     -
```

■ 特定のジョブや範囲の指定も可能

```
[_LNlogin]$ pjstat 238 239
```

```
[_LNlogin]$ pjstat 238-240
```

■ ジョブの削除

```
[_LNlogin]$ pjdel 12345678
[INFO] PJM 0100 pjdel Accepted job 12345678.
```

■ 複数のジョブIDの指定も可能

```
[_LNlogin]$ pjdel 12345678 12345680
[INFO] PJM 0100 pjdel Accepted job 12345678.
[INFO] PJM 0100 pjdel Accepted job 12345680.
```

■ 存在しないジョブを指定するとエラーメッセージが出力される

```
[ERR.] PJM 0112 pjdel Job "存在しないジョブID" does not exist.
```

通常ジョブ

■ ジョブ実行結果の参照

ジョブが終了すると、ジョブ投入時のカレントディレクトリにジョブ実行結果ファイルが出力される。

■ 単一プロセスジョブの場合

項目名	機能
ジョブ名.*ジョブID*.out	ジョブが標準出力に書き出したデータ
ジョブ名.*ジョブID*.err	ジョブが標準エラー出力に書き出したデータ
ジョブ名.*ジョブID*.stats	ジョブの統計情報が出力されたファイル ※-s,-Sオプションが指定された場合のみ出力

■ MPIジョブの場合

■ 作業ディレクトリ直下に単一プロセス実行の場合と同名のファイルが生成される他、以下のようなディレクトリが生成される。

```
output.XXXXXXXX/              #XXXXXXXX はジョブID
└ 0/                          #ランク番号1000ごとに分けられたディレクトリ
   ├ 1/ #ジョブスクリプト内での1回目のジョブ実行（mpiexec = 1）
   └ 2/ #ジョブスクリプト内での2回目のジョブ実行（mpiexec = 2）
```

■ 各ディレクトリ内に以下の名前でファイルが生成される

項目名	機能
stdout.*mpiexec*.*rank*	mpiexecコマンドが標準出力に書き出したデータ
stderr.*mpiexec*.*rank*	mpiexecコマンドが標準エラー出力に書き出したデータ

※ *mpiexec*：ジョブスクリプト内で何回目のmpiexecコマンド実行かを示す数字
rank：ランク番号

■ 以上はデフォルト設定の場合。MPIジョブにおける標準入出力ファイル名の変更方法については p.125 を参照。

ジョブ正常終了・異常終了の確認方法

ジョブが正常終了したか確認するには以下の情報が参考になる。

- ジョブマネージャの終了コード
  - ジョブ統計情報ファイル（pjsubで-sまたは-Sオプション指定時）に出力
- ジョブ実行時に出力されるメッセージ
  - 標準エラー出力ファイルに出力

ジョブ正常終了・異常終了の確認方法

- ジョブマネージャの終了コード
  - ジョブ統計情報ファイルに出力されたPJM CODEを確認する
  - PJM CODEとその意味（抜粋）

PJM CODE	意味
0	ジョブの正常終了
1	ユーザが操作したpjdelコマンドによるCANCEL
2	ジョブの受付判定によるREJECT
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
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

ジョブ正常終了・異常終了の確認方法

- ジョブ実行時に出力されるエラーメッセージ
  - 標準エラー出力ファイルに出力されたエラーメッセージを確認する
  - メッセージの種類と参照すべきマニュアル

メッセージ	参照マニュアル
PLE *nnnn plexec*	ジョブ運用ソフトウェア ■ コマンドリファレンス ■ エンドユーザ向けガイド
PJM *nnnn xxxxxx*	ジョブ運用ソフトウェア ■ コマンドリファレンス ■ エンドユーザ向けガイド
mpi::	MPI使用手引書
jwe*nnnn*	Fortran/C/C++実行時メッセージ

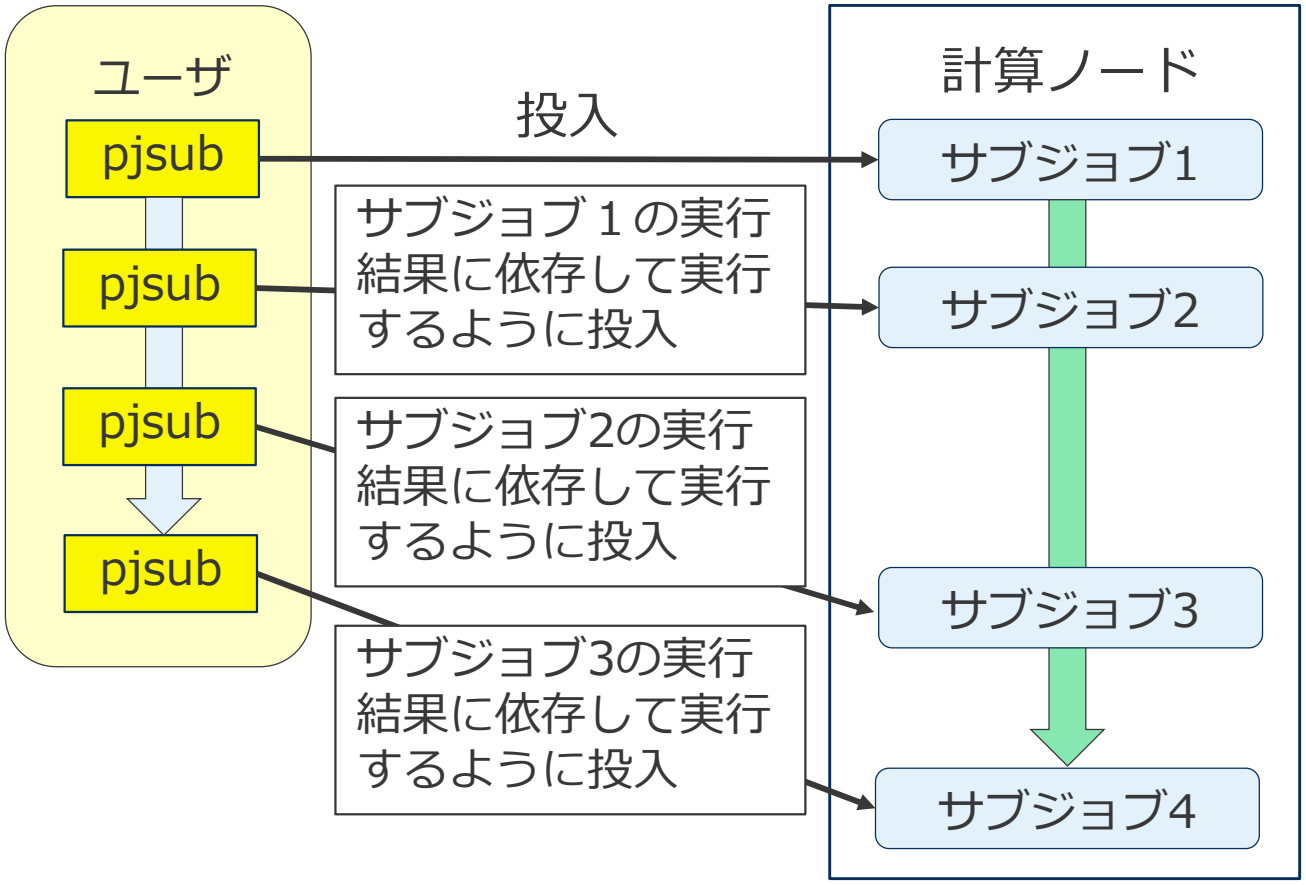
  - ハード故障でエラーが起きた場合、タイミングによってはPJM codeがハード故障にならない場合がある。以下のコマンドで、システム障害の影響などを受けたジョブを確認することができる

```
[_LNlogin]$ job_events
```

ステップジョブ

ステップジョブは、投入された複数のジョブに実行順序関係や依存関係を持たせられるジョブである。

■ ステップジョブの動作イメージ



サブジョブ：
ステップジョブを構成する複数のジョブ

■ サブジョブは投入順に1つずつ実行される

■ 先に実行されたサブジョブの終了ステータスを基に、後続のサブジョブを実行するか否かを制御可能

ステップジョブは長時間かかり、分割実行可能な計算を行う際に有用である。

ステップジョブ

ステップジョブでは、通常ジョブで使用するジョブスクリプトをそのまま使用可能。各サブジョブで異なる資源（計算ノード数、メモリー量等）を指定することが可能。

- ステップジョブの投入方法
  - ステップジョブの開始（--stepオプションが必要）

```
[_LNlogin]$ pjsub --step job_1.sh
[INFO.] PJM 0000 Job 1234_0 submitted
```

1234: ジョブID
0　:ステップ番号
1234_0: サブジョブID

  - 2つ目以降のサブジョブの投入（jidでジョブIDを指定する）

```
[_LNlogin]$ pjsub --step --sparam "jid=1234" job_2.sh
[INFO.] PJM 0000 Job 1234_1 submitted
```

  - 複数のサブジョブを一度に投入することも可能

```
[_LNlogin]$ pjsub --step job_1.sh job_2.sh
[INFO.] PJM 0000 Job 1235_0 submitted
[INFO.] PJM 0000 Job 1235_1 submitted
```

■ ジョブ名を指定したステップジョブ投入方法

■ 1番目のサブジョブ投入（--stepオプションが必要）

```
[_LNlogin]$ pjsub --step job_1.sh     # 例：ジョブスクリプト内でジョブ名を moon と指定
[INFO] PJM 0000 pjsub Job 2345_0 submitted.
```

■ 2番目以降のサブジョブ投入

jnamで1番目のサブジョブのジョブ名を指定する

```
[_LNlogin]$ pjsub --step --sparam "jnam=moon" job_2.sh
[INFO] PJM 0000 pjsub Job 2345_1 submitted.
```

※ 同じジョブ名を冠するステップジョブが複数存在する場合、--sparamでジョブ名を指定すると、最後に生成されたステップジョブのサブジョブとして投入される

※ ジョブスクリプト中で--nameオプションでジョブ名を指定しているジョブでも、
ステップジョブ投入時にjnamで異なるジョブ名を指定した場合、jnamで指定したジョブ名に上書きされる

ステップジョブの確認

通常の表示

```
[_LNlogin]$ pjstat
JOB_ID     JOB_NAME   MD ST  USER  START_DATE       ELAPSE_LIM  NODE_REQUIRE
1234       galaxy     ST RUN user  12/31 23:45:01   -           -
```

サブジョブの状態を表示させる場合には、-Eオプションを使用

```
[_LNlogin]$ pjstat -E
JOB_ID     JOB_NAME   MD ST  USER  START_DATE       ELAPSE_LIM  NODE_REQUIRE
1234       galaxy     ST RUN user  12/31 23:45:01   -           -
1234_0     galaxy     ST EXT user  12/31 23:45:01   0000:30:00  8
1234_1     galaxy     ST RUN user  01/01 00:12:34   0000:30:00  8
1234_2     galaxy     ST CCL user  -                0002:00:00  4
1234_3     galaxy     ST QUE user  -                0002:00:00  8
```

- 現行のサブジョブの実行が終了するまで、次のサブジョブのスケジューリングは実施されない

ステップジョブの取り消し

ジョブIDを指定して全てのサブジョブの実行を取りやめる場合

```
[_LNlogin]$ pjdel 1234
[INFO] PJM 0100 pjdel Accepted job 1234_0.
[INFO] PJM 0100 pjdel Accepted job 1234_1.
[INFO] PJM 0100 pjdel Accepted job 1234_2.
```

特定のサブジョブだけを取り消す場合

```
[_LNlogin]$ pjdel 1234_2
[INFO] PJM 0100 pjdel Accepted job 1234_2.
```

範囲を指定して複数のサブジョブを取り消したい場合

```
[_LNlogin]$ pjdel 1234_1-2
[INFO] PJM 0100 pjdel Accepted job 1234_1.
[INFO] PJM 0100 pjdel Accepted job 1234_2.
```

※サブジョブの一部をキャンセルした後でも、ステップジョブを投入可能

■ 実行結果の出力

サブジョブが終了すると、サブジョブ投入時のカレントディレクトリに
ジョブ実行結果がファイル出力される。

■ サブジョブごとに出力されるファイル

形式	説明
ジョブ名.サブジョブID.out	サブジョブが標準出力に書き出したデータ
ジョブ名.サブジョブID.err	サブジョブが標準エラー出力に書き出したデータ
ジョブ名.サブジョブID.stats	サブジョブの統計情報に関するファイル

■ ステップジョブごとに出力されるファイル

形式	説明
1つめのジョブ名.ジョブID.stats	ステップジョブの統計情報に関するファイル

※ 統計情報ファイルはジョブ投入時に-s、-Sオプションを指定した場合
に出力

ステップジョブ

ステップジョブの投入時には、サブジョブの終了ステータスに応じた後続のサブジョブの実行条件を指定することが可能である。

■ 依存関係のあるジョブの投入

```
[_LNlogin]$ pjsub --step --sparam "jid=1234,sd=pc!=0:all:2" job_4.sh job_5.sh
```

終了ステータスを参照するステップ番号

■ 終了ステータス（sdで指定する値）

ec: ジョブスクリプトの終了ステータス
pc: ジョブ終了コード

※　ジョブが正常に終了しなかった場合でも、ジョブスクリプトの終了ステータス (ec) が0になる場合が存在する（例: 経過時間制限超過）

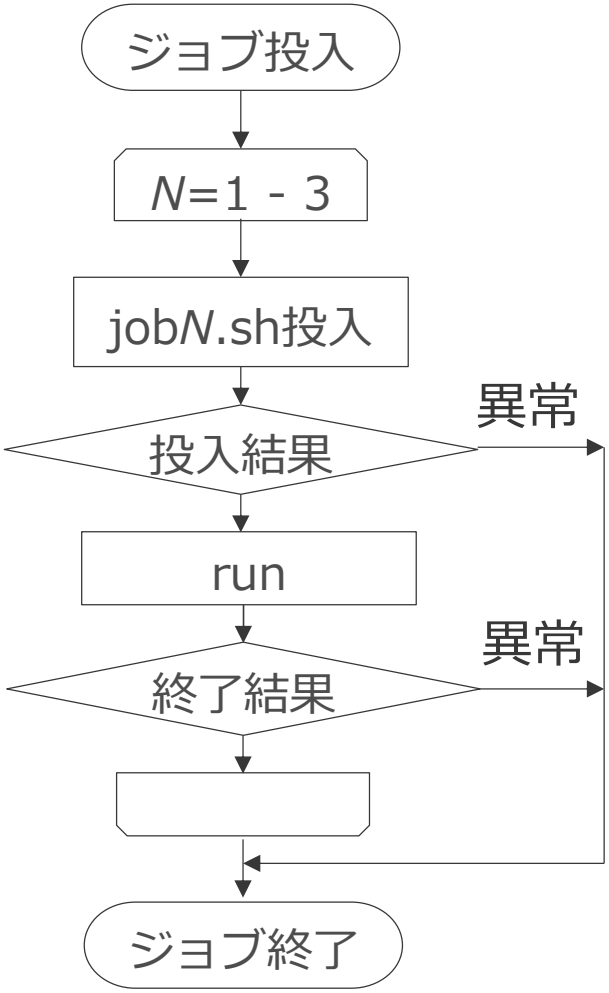
■ 削除タイプ(: の後で指定する値)

one:　投入したサブジョブだけ削除
　　このサブジョブに依存する後続のサブジョブは削除されない
after: 投入したサブジョブとそれに依存する後続のサブジョブだけを削除
all:　投入したサブジョブ、および後続のサブジョブをすべて削除

ワークフロージョブ

ワークフロージョブとは、複数のジョブの投入を、シェルスクリプトによってユーザが制御するジョブモデルである。

■ ワークフロージョブの例



※ *job1.sh*などは、通常ジョブとして投入可能なジョブスクリプト

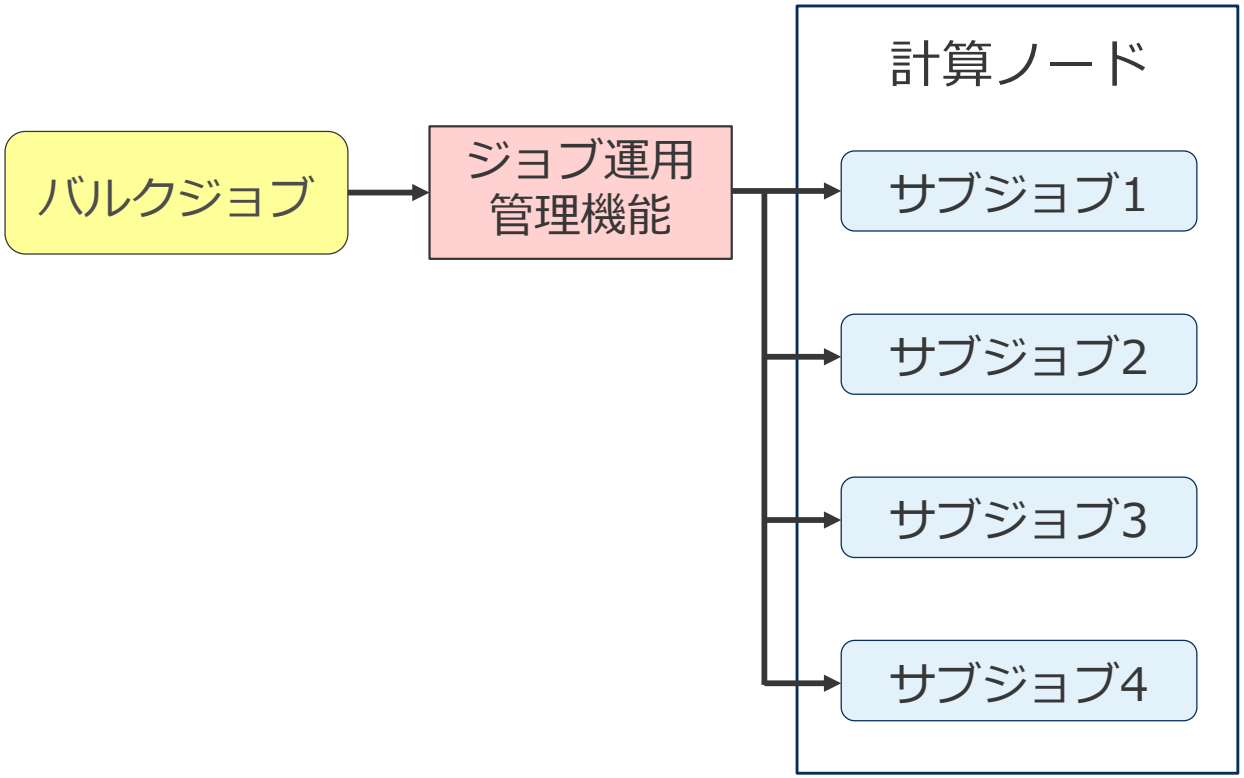
```
#!/bin/sh -x
for no in {1..3}
do
  JID=`pjsub -z jid job${no}.sh`
  if [ $? -ne 0 ];   # $?: ジョブ投入結果
    exit 1
  fi
  set -- `pjwait $JID`
         # pjwait: 指定したジョブの終了を待つコマンド
  if [ $2 != "0" -o $3 != "0" ]; then
    exit 1
  fi
done

# pjwait $JID ⇒   $2: ジョブ終了コード
                  $3: ジョブスクリプトの終了ステータス
```

※ pjwait や pjsub -z jid などの説明は、「ジョブ運用ソフトウェア　コマンドリファレンス」を参照すること

バルクジョブ

バルクジョブとは、複数の同じ通常ジョブを、入力パラメータを変えて１度に同時投入し、実行することが可能なジョブモデルである。

■ バルクジョブの動作イメージ



サブジョブ：
バルクジョブを
構成する複数のジョブ

サブジョブは、
ノードが空いていれば
同時並列に実行される

バルクジョブはパラメータを変えた多数の計算を行う際に有用である。

バルクジョブ

バルクジョブを行う際は、サブジョブごとにジョブの入力パラメーターを変えられるように、ジョブスクリプト内で指定する入出力データのファイル名にバルク番号を含めることが可能である。

■ バルクジョブ用のジョブスクリプト

```
#!/bin/bash
#PJM -g hp250xxx
#PJM -L "node=2"
#PJM -L "rscgrp=small"
#PJM -L "elapse=01:00:00"
#PJM --mpi "max-proc-per-node=4
#PJM -S

llio_transfer ${PJM_JOBDIR}/a.out   # 共通ファイル配布（LLIOの章で説明）
# ${PJM_JOBDIR}はジョブが投入されたディレクトリ

cd ${PJM_JOBDIR}/param_id${PJM_BULKNUM}
# ${PJM_BULKNUM}はバルク番号
mpiexec --stdin test.inp --std-proc test.out ${PJM_JOBDIR}/a.out
```

バルクジョブ

- バルクジョブの投入
  - --bulkオプションをつけるとバルクジョブとして投入される

```
[_LNlogin]$ pjsub --bulk --sparam "1-4" bulkjob.sh
[INFO] PJM 0000 pjsub Job 8123 submitted.
```

  - バルクジョブ投入時は--sparamオプションが必須
  - バルク番号は--sparam "*startbulkno-endbulkno*"という形で指定
    - 指定するバルク番号は連続していなければならない
      ※ --sparam "3, 5, 8" といった指定は不可
    - *startbulkno*は*endbulkno*より小さくなければならない
      ※ --sparam "4-1"という指定は不可
    - 指定するバルク番号は0以外の数字で始めることも可能
      - 複数回に分けてバルクジョブを投入することも可能
        例） 1回目のバルクジョブ投入： --sparam "1-100"
        2回目のバルクジョブ投入： --sparam "101-200"

バルクジョブ

■ バルクジョブの確認

■ 通常の表示

```
[_LNlogin]$ pjstat
JOB_ID      JOB_NAME    MD ST  USER  START_DATE        ELAPSE_LIM  NODE_REQUIRE
1234        planet      BU RUN user  -                 0001:00:00  2
```

■ サブジョブ情報を参照する場合には、-Eオプションを使用

```
[_LNlogin]$ pjstat -E
JOB_ID      JOB_NAME    MD ST  USER  START_DATE        ELAPSE_LIM  NODE_REQUIRE
1234        planet      BU RUN user  -                 0001:00:00  2
1234[1]     planet      BU RUN user  01/01 06:42:34    0001:00:00  2
1234[2]     planet      BU RUN user  01/01 06:42:34    0001:00:00  2
1234[3]     planet      BU RUN user  01/01 06:50:56    0001:00:00  2
1234[4]     planet      BU RUN user  01/01 07:06:23    0001:00:00  2
1234[5]     planet      BU QUE user  -                 0001:00:00  2
```

※ バルクジョブでは、要求した計算資源が確保されたサブジョブから順次実行されていくため、サブジョブごとに状態は異なる。そのため、ジョブの状態は、必ずしも個々のサブジョブの状態とは一致しない。

※ JOB_ID の項目の 1234[1] 等の [1] はバルク番号を表す。

バルクジョブ

- バルクジョブの取り消し
  - ジョブIDを指定して全てのサブジョブをキャンセルする場合

```
[_LNlogin]$ pjdel 1234
[INFO] PJM 0100 pjdel Accepted job 1234.
```

  - 特定のサブジョブだけを取り消す場合

```
[_LNlogin]$ pjdel 1234[2] 1234[3]
[INFO] PJM 0100 pjdel Accepted job 1234[2].
[INFO] PJM 0100 pjdel Accepted job 1234[3].
```

  - サブジョブの範囲を指定して複数のサブジョブを取り消す場合

```
[_LNlogin]$ pjdel 1234[2-3]
[INFO] PJM 0100 pjdel Accepted job 1234[2-3].
```

■ 実行結果の出力

サブジョブが終了すると、サブジョブ投入時のカレントディレクトリに
ジョブ実行結果がファイル出力される。

■ サブジョブごとに出力されるファイル

形式	説明
ジョブ名.サブジョブID.out	サブジョブが標準出力に書き出したデータ
ジョブ名.サブジョブID.err	サブジョブが標準エラー出力に書き出したデータ
ジョブ名.サブジョブID.stats	サブジョブの統計情報に関するファイル

■ バルクジョブごとに出力されるファイル

形式	説明
ジョブ名.ジョブID.stats	バルクジョブの統計情報に関するファイル

※ 統計情報ファイルはジョブ投入時に-s、-sオプションを指定した場合
に出力

会話型ジョブ

会話型ジョブは、ユーザーが計算ノード内で対話的にコマンド実行するジョブタイプである。

■ 計算ノード内で対話型で実行する方法

```
[_LNlogin]$ pjsub --interact -g hp250xxx -L "rscgrp=int" --mpi "proc=12" ¥
> -L "node=1, elapse=50:00" --sparam "wait-time=600" -x PJM_LLIO_GFSCACHE=/vol0004

[INFO] PJM 0000 pjsub Job 5678 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 5678 started.

[_CNlogin]$ python test.py
...test program running...
[_CNlogin]$ exit
[INFO] PJM 0083 pjsub Interactive job 5678 completed.
```

- elapse: ジョブの最大実行時間を指定
  デフォルト値は10秒なので必ず指定する
- sparam "wait-time": 計算資源割り当て待ち時間の指定（単位：秒）
  デフォルト値は0、少なくとも60に指定することを推奨、
  unlimitedを指定可能

※ 会話型ジョブでは、操作されなくても資源を消費することに注意

会話型ジョブ

会話型ジョブでは、スクリプトファイルを指定して計算ジョブ内で実行させることも可能である。

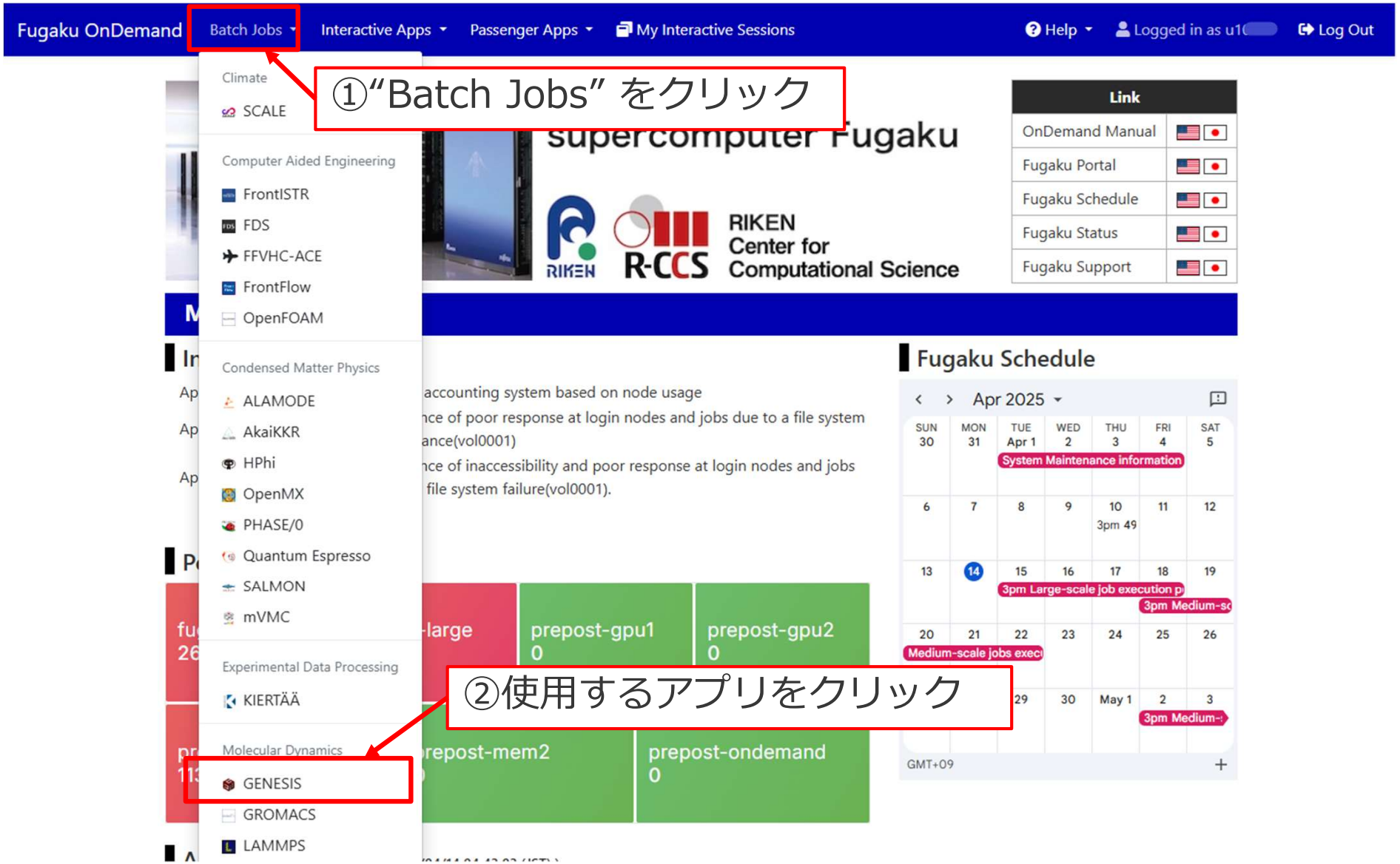
■ スクリプトファイルを指定した会話型ジョブ投入例

```
[_LNlogin]$ cat ./sample.sh
#! /bin/bash
# 通常ジョブでは必要な#PJMで始まる行は会話型ジョブでは不要
echo "HELLO, Fugaku"

[_LNlogin]$ pjsub --interact -g hp250xxx -L "rscgrp=int, elapse=0:10:00" ¥
> --sparam "wait-time=unlimited" ./sample.sh
[INFO] PJM 0000 pjsub Job 4567 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 4567 started.
HELLO, Fugaku
[INFO] PJM 0083 pjsub Interactive job 4567 completed.
# 投入されたスクリプトファイルの実行の終了後、会話型ジョブは終了
```

※ 会話型ジョブでは、疑似端末を通じてジョブの一連の動作を対話的に行うが、スクリプトファイルが指定された場合、疑似端末は使用されない

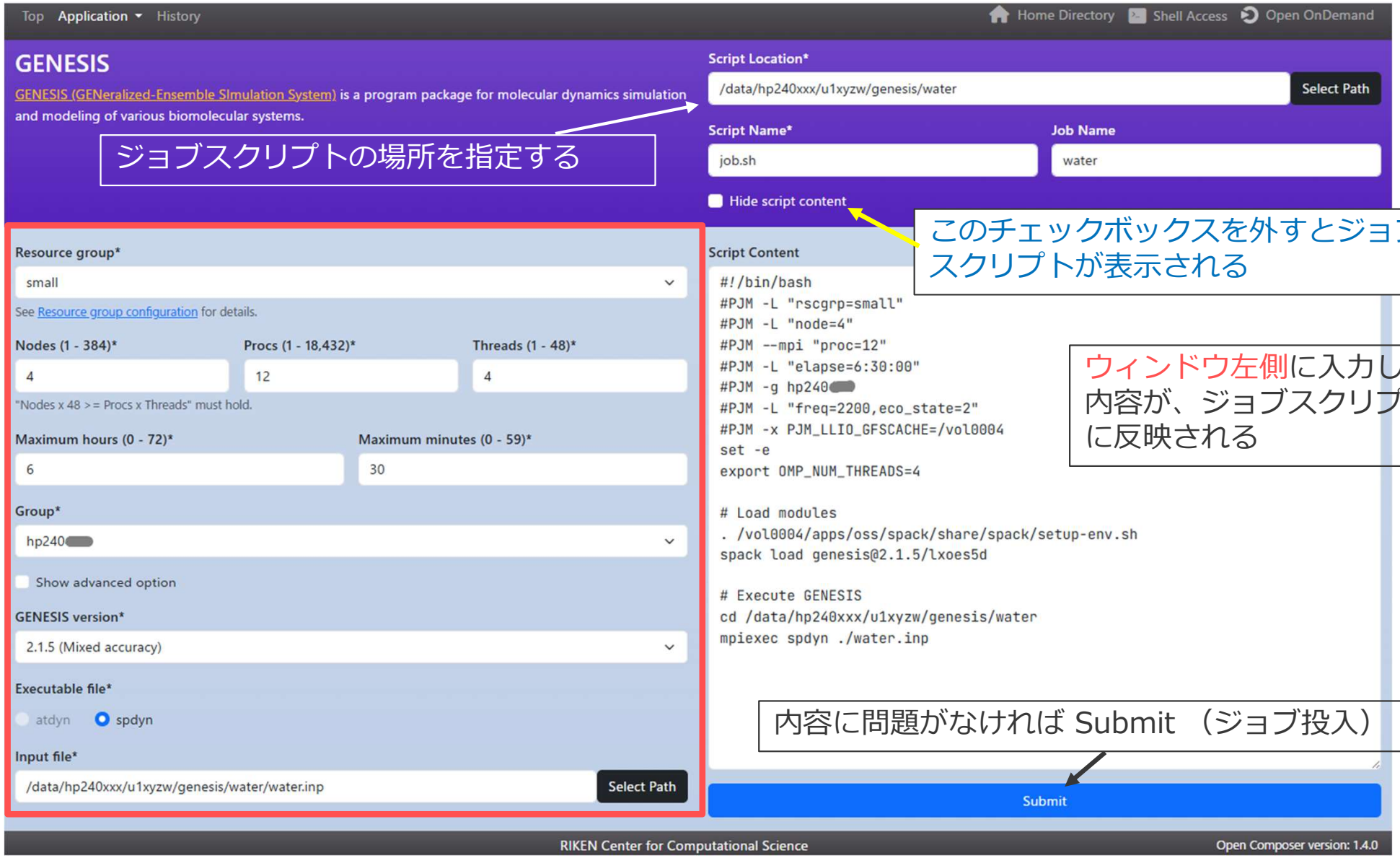
Open Composerによるジョブ投入

Fugaku Open OnDemandからアプリを指定してジョブ投入可能



Open Composerによるジョブ投入

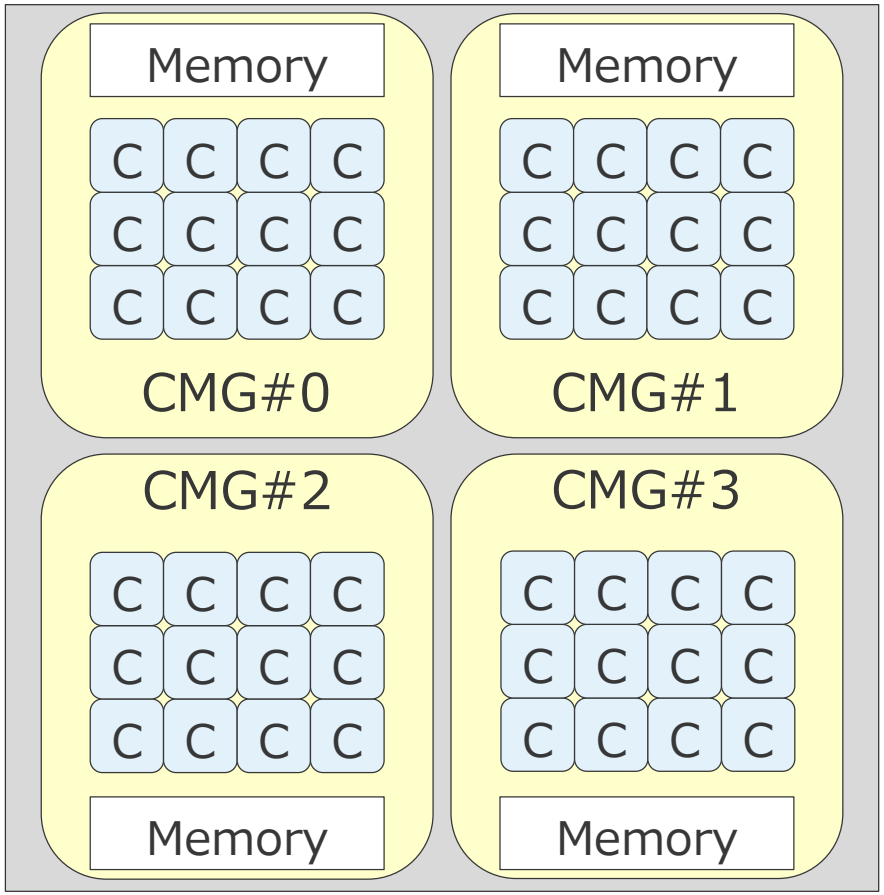
Open Composerの画面ではジョブスクリプトの中身を確認可能



- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

ノード内推奨プロセス数

「富岳」の１ノードの基本構成は以下のようになっている。
特に並列ジョブ実行の際は、以下の数値を元に計算条件を検討する。



※ CMG: Core Memory Group

- A64FXのノード構成
  - 4 CMG / 1 ノード
  - 12コア / 1 CMG
  - 安定的に使用可能なメモリー量の目安：
    約 23.2 GiB / 1 ノード
- ノード内最大スレッド数 = 48
- 推奨される並列方法

```
#PJM --mpi max-proc-per-node=4
          # ノード内推奨プロセス数 = 4
export OMP_NUM_THREADS=12
          # 1プロセスあたり12スレッド
```

並列実行コマンド

計算ノード上でMPIプログラムを実行するには、mpiexecコマンドを使用する。

- MPIプログラムを12並列で実行する場合

```
$ mpiexec -n 12 -std-proc outfile ./a.out
```

- JavaのMPIプログラムを12並列で実行する場合

```
$ mpiexec -n 12 -std-proc outfile java -classpath ./dir JavaTest
```

- MPIプログラムを同時に複数実行する場合
  - MPMD (Multiple Program Multiple Data stream)

```
$ mpiexec -std-proc outfile -n 2 ./a.out : -n 4 ./b.out : -n 6 ./c.out
```

- -appオプションを使用した実行定義ファイル形式による指定も可能
- -n で指定するプロセス数の総和は--mpi proc=N を超えてはならない
- Javaプログラムの場合、MPMDモデルでの実行は保証されていない

標準入力／標準出力／標準エラー出力

mpiexecでは、プログラムの標準入力／標準出力／標準エラー出力用のファイルを指定するオプションが存在する。

■ 全並列プロセスの標準入力を、単一のファイルから読み込む場合

```
$ mpiexec -n 12 -stdin in_file ./a.out
```

■ 各並列プロセスの標準出力をプロセスごとに出力する場合

```
$ mpiexec -n 12 {-stdout-proc | -ofout-proc} stdout_file ./a.out
```

■ 各並列プロセスの標準エラー出力をプロセスごとに出力する場合

```
$ mpiexec -n 12 {-stderr-proc | -oferr-proc} stderr_file ./a.out
```

※ 実際のファイル名は stdout_file.1.0 のように、mpiexec 実行順とランク番号が付加された名前となる

※ 全並列プロセスからの出力先を、指定した単一のファイルとなるオプション（-std, -stdout, -stderrなど）は無効化されているため注意

※ 計算ノードのmpiexecでは、リダイレクトは使用できないので注意

※ デフォルト設定については p.100を参照

出力先ディレクトリの振り分け

大規模なジョブ実行時は、メタ文字を利用した出力ディレクトリの振り分けにより、 1ディレクトリ内のファイル数が多くならないよう調節することで、ファイルシステムの負荷を軽減させることが必要です。

- ランク番号が1000ごとに標準出力／標準エラー出力の出力先ディレクトリを変更する場合の指定方法
  - デフォルトでは、下記の設定に合わせてディレクトリが自動的に作られる
  - `-std-proc`: 標準出力と標準エラー出力を同じファイルに出力するオプション

```
$ mpiexec -std-proc ./output.%j/%/1000R/stdout_err ./a.out
```

- mpiexecコマンドで使用可能な主なメタ文字

メタ文字	意味
%n	ジョブ名
%j, %J, %b, %s	順にジョブID, サブジョブID, バルク番号、ステップ番号
%R	ランク番号 出力先がmpiexecコマンド単位の場合は、空文字になる
%/*N*R	ランク番号を数値*N*単位で切り捨て 上記の例では*N*=1000として使用

ファイルシステムに関する注意事項

ファイルシステムのスローダウンや計算ノードのダウンを避けるため、
下記の注意事項を厳守すること

- 1,000 本以上の小規模ジョブを同一ディレクトリ上で実行し、そのジョブ内でファイルやディレクトリの作成・削除をしないこと
  - 同じディレクトリに同時にアクセスしないようジョブ毎に出力ディレクトリを分けるようにしてください。（予めログインノードでディレクトリを作成しておくことを推奨）
- 複数のプロセスから単一ファイルへアクセスをしないこと
- １つのディレクトリ配下のファイルやディレクトリ数は100,000個まで

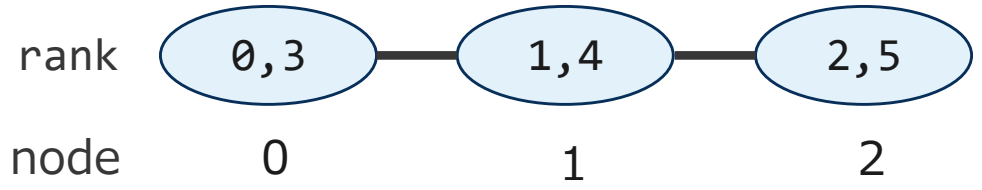
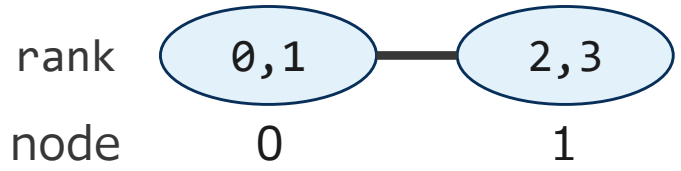
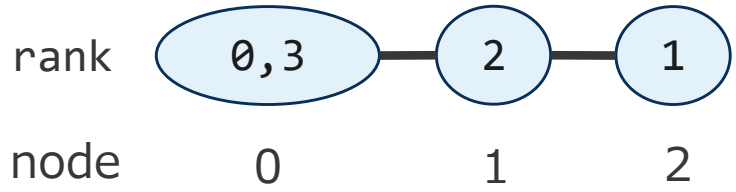
- 上記注意事項が守られない場合
  - ファイルシステムに高い負荷がかかる
  - システム側でジョブのIOを強制的に中断することがある

- 上記の利用規則は変更の可能性がある。最新の規則については、ユーザブリーフィングを参照すること。

MPIランクの割り当て方法（1次元）

ランクの割り当て方法は、pjsubコマンドの--mpiオプションにより指定する。

■ MPIランク割り当てのルール

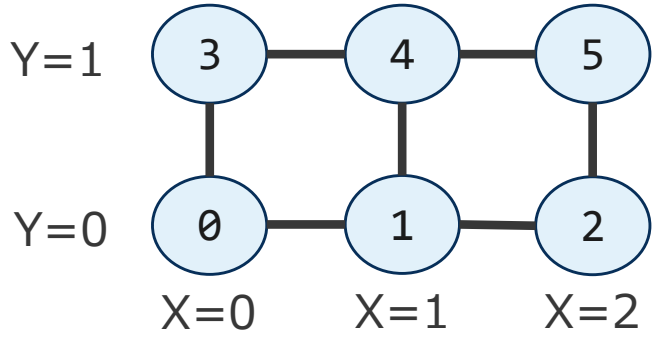
指定方法	ランク割り当て方法
rank-map-bynode	先頭の計算ノードから順繰り形式でランクを自動割り当て 
rank-map-bychip (デフォルト設定)	先頭の計算ノードから1ノードあたり*n*ランクを自動割り当て  $n = \frac{\text{プロセス数}}{\text{ノード数}}$
rank-map-hostfile	ランク割り当てを*hostfile*で指定  ホストマップファイル例 (0) # rank 0 (2) # rank 1 (1) # rank 2 (0) # rank 3

MPIランクの割り当て方法（2次元）

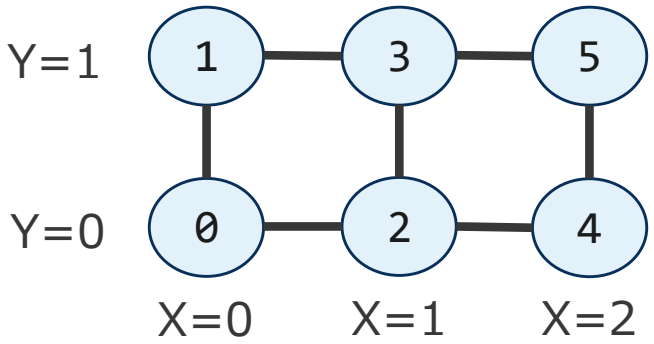
プロセスの形状（使用するノードの形状）が2次元／3次元の場合、
どの軸方向から割り当てていくかを指定可能(rankmap)。

■ rank-map-bynode

```
#PJM --mpi "shape=3x2"
#PJM --mpi rank-map-bynode=XY
```

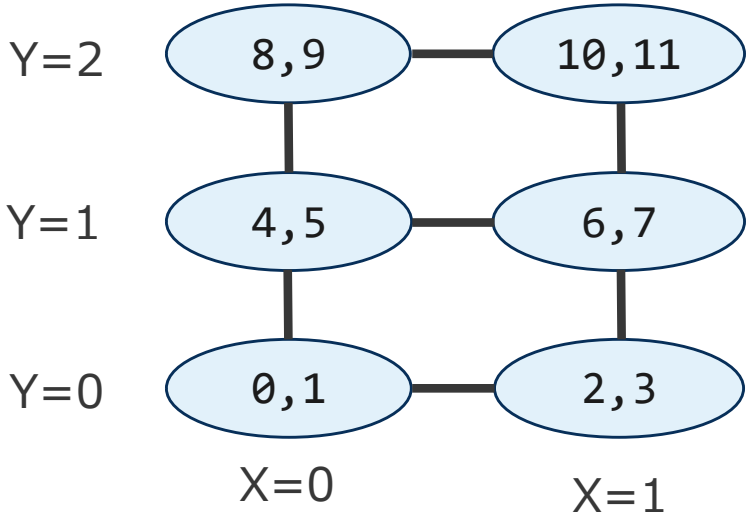



```
#PJM -L "node=3x2"
#PJM --mpi rank-map-bynode=YX
```



■ rank-map-bychip

```
#PJM --mpi "proc=12"
#PJM --mpi "shape=2x3"
#PJM --mpi rank-map-bychip:XY
```



■ rankmapが省略された時は
XY (2D)／XYZ (3D)となる。

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

ノード時間積の使用状況確認

accountjコマンドで、課題に割り当てられたノード時間積と電力消費量についてグループ全体およびユーザーごとの使用状況を確認できる。

■ 例：自分自身が所属するグループの使用状況を表示する

```
[_LNlogin]$ accountj
COLLECTDATE : 2022-03-29 16:59:10   unit[Ms,MWh]
*-----[ SUBTHEME ]-----------------------------------------------------------*
SUBTHEME                PARENT      LIMIT(N)     USAGE(N)     LIMIT(E)     USAGE(E)
rist                  Y21-RIST        11,668       11,644    unlimited          228
*-----[ SUBTHEME_PERIOD ]---------------------------------------------------*
                        PERIOD      LIMIT(N)     USAGE(N)     LIMIT(E)     USAGE(E)
rist                         1         5,188          720          ---           0
rist                         2         6,480        6,456          ---           0
*-----[ GROUP ]-------------------------------------------------------------*
GROUP                   PARENT      LIMIT(N)     USAGE(N)     LIMIT(E)     USAGE(E)
rist                      rist     unlimited        7,176    unlimited          228
```

割り当て量　　使用分　　消費電力

- デフォルト設定では所属するすべてのグループの使用状況が表示
  - 特定のグループの使用状況の確認には -g *グループ名* を指定
- ノード時間積のみを表示するには、オプション -r 1 を使用
  - ノード時間積の表示単位は、オプション-s（秒）、-h（時間）で変更可能

ノード時間積の使用履歴確認

pjstataコマンドにより、過去に実行したジョブごとのノード時間積の使用履歴が確認できる。

- 例： 自分自身(user1)の投入したジョブのうち、2021年4月26日以降4月28日以前に開始したジョブの情報を表示する場合

```
[_LNlogin]$ pjstata -d 20210426:20210428
JOBID   SNO BLKNO GENNO JOB_NAME  MD JTYPE USER     GROUP    ST   RSC_UNT  RSC_GRP EC  PC
367430            0     g11F2DB.s NM BT    user1    group1   EXT  unit1    small   0   0
369249            0     parmer8K1 NM BT    user1    group1   EXT  unit1    small   0   0
```

下へ続く

```
ERR_CD JOB_START       JOB_END         ELAPSE_TIM  NODE_NUM  RATE  ACCT_RSC ACCT_ECON   PERIOD_NUM
0      04/26 16:12:17 04/26 16:23:32  0000:11:15  1         100%  675      7.518036    2
0      04/27 13:18:41 04/27 13:20:05  0000:01:24  2         100%  168      1.815250    2
```

- -dオプションはジョブの開始日による絞り込みを行う
- ジョブ終了日による絞り込みには-tオプションを用いる
- RSC_GRPは実行時のリソースグループ名を表示
- ACCT_RSCは当該ジョブで利用されたノード時間積をノード秒の単位で表示

ディスクの使用状況確認

accountdコマンドにより、グループ全体およびユーザーごとのディスク使用状況を確認できる。

■ 例：自分自身が所属するグループのデータ・シェア領域、および自身のホーム領域のディスク使用状況を表示する

```
[_LNlogin]$ accountd
COLLECTDATE : 2022/06/21 18:13:59    unit[GiB]
USER : u1xxxx
*----------------------------------[GROUP]-----------------------------------*
GROUP         VOLUME      LIMIT      USAGE  AVAILABLE      FILES   USE_RATE
*hp220xxx     vol030y   409,600     43,953    365,647  7,093,815      10.7%
*-----------------------------------[USER]-----------------------------------*
USER          VOLUME      LIMIT      USAGE  AVAILABLE      FILES   USE_RATE
U1xxxx        vol030y        20          1         19        409       0.9%
```

割り当て量　使用分　未使用分　使用率

■ VOLUME欄のvolに続く最初の２桁はvolumeを表す
例）vol0300 ⇒　/vol0003

■ -Eオプションを指定すると、データ領域、シェア領域、ホーム領域の具体的なパスが表示される。

ディスクの使用状況確認

accountdコマンドの-iオプションにより、グループ全体およびユーザーごとのi-node使用状況を確認できる。

- 例：自分自身が所属するグループのデータ・シェア領域、および自身のホーム領域のi-node使用状況を表示する

```
[_LNlogin]$ accountd -i
COLLECTDATE : 2022/02/21 18:13:59
USER : u1xxxx
*----------------------------------[GROUP]-----------------------------------*
GROUP          VOLUME        ILIMIT        IUSED         IFREE      USE_RATE
*hp220xxx      vol030y    120,000,000   67,096,622    52,903,378         55.9%
*-----------------------------------[USER]-----------------------------------*
USER           VOLUME        ILIMIT        IUSED         IFREE      USE_RATE
u1xxxx         vol030y       200,000           409       199,591          0.2%
```

割り当て量　使用分　未使用分　使用率

- .localなどのホーム領域の隠しディレクトリ下にあるファイルがi-node制限にかかることがあるので留意すること。
- 同一ディレクトリ配下に作成するファイル・ディレクトリ数は10万までとすること。適宜tarコマンドでファイルをまとめることを推奨。

ノード空き状況確認

pjshowrscコマンドにより、システム全体のノード使用状況（混雑具合）が確認できる。

■ 例：システム全体のノード使用状況をリソースグループ別に表示

```
[_LNlogin]$ pjshowrsc --rscgrp
[ CLST: fugaku-comp ]
[ RSCUNIT: rscunit_ft01 ]
RSCGRP              NODE
                   TOTAL      FREE     ALLOC
large             110560     31088     79472
int                20720      3614     17106
small              20720      3614     17106
```

各リソースグループに割り当てられているノード数（故障ノードは除く）

現在利用可能なノード数

現在ジョブに割り当てられているノード数

※ 複数のリソースグループが資源を共有している場合、ALLOCには当該リソースグループおよび資源を共有するリソースグループで使用されているノード数が表示される

※ シングルアカウントユーザーは -g，--gname などのオプションにより、グループを指定する必要がある

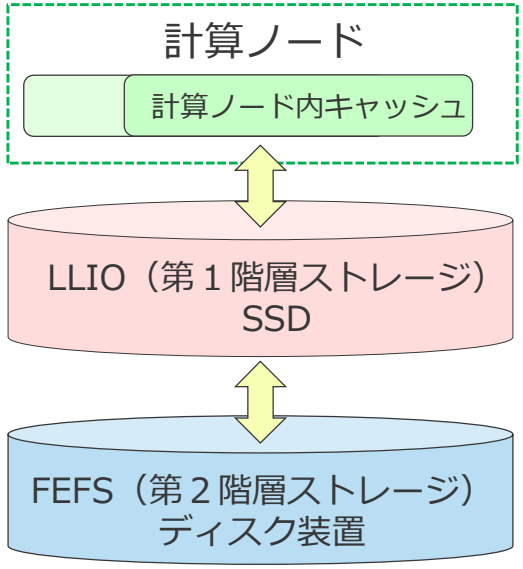
- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

LLIO (Lightweight Layered IO-Accelerator) とは、並列分散ファイルシステムであるFEFSと計算ノードの間に位置する、SSDを使用した高性能なファイルシステム、またはそれを実現する技術である。

- 「富岳」のストレージ構成
  - 第１階層ストレージ
    - LLIOによって管理される、計算実行中に使用可能になるSSDストレージ
  - 第２階層ストレージ
    - FEFSによって管理される、データ保管可能なディスクストレージ

※第3階層ストレージについては説明省略

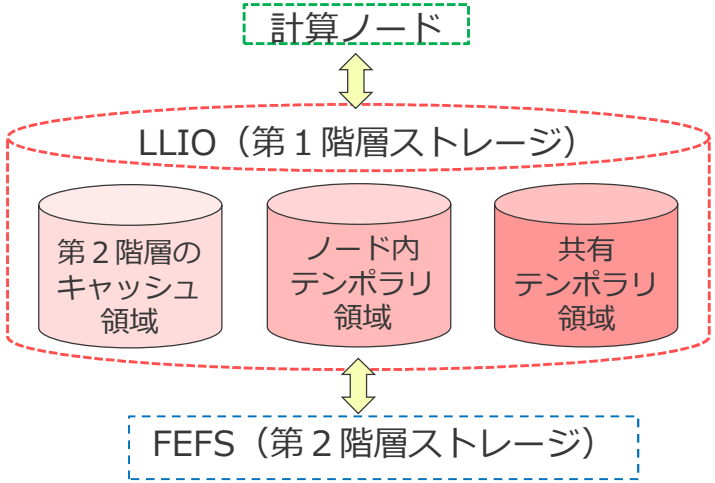
- LLIOの重要性について
  - 多数ノードで計算する場合にIO負荷が実行時間に与える影響が顕著になる場合があり、IO負荷軽減が求められる
  - LLIOは以下の機能によってIO負荷を軽減できる
    - 一時ファイルを第１階層（SSD)で扱うことでデータの読み書きにかかる時間を短縮する
    - ファイル出力を非同期に第２階層に書き出すことでIO処理にかかる時間を隠蔽する



LLIOが第1階層ストレージに提供する3つの領域

LLIOが第1階層ストレージに提供する3つの領域と、各領域の特徴は以下の通りである。

- 第2階層ストレージのキャッシュ領域
- ノード内テンポラリ領域
- 共有テンポラリ領域



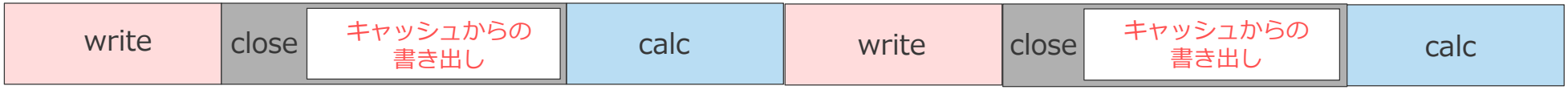
領域名	参照範囲	適用事例	最大容量	容量の指定法
ノード内 テンポラリ領域	同一ジョブの同一ノード内	ランク・ノードごとに独立した中間ファイルや一時ファイルの作成・参照	全領域で87GiB/node	pjsubのオプションにて指定 ※指定しない場合0MiB
共有 テンポラリ領域	同一ジョブの全ての計算ノード	ランク・ノード間でのファイルを参照、巨大なファイルの操作		pjsubのオプションにて指定 ※指定しない場合0MiB
第2階層ストレージのキャッシュ領域	同一ジョブの全ての計算ノード	標準出力・標準エラー出力など、第2階層に保存されるファイルの出力		87GiB – (ノード内テンポラリ領域 + 共有テンポラリ領域) ※128MiB以上の容量が必要 ※ジョブのキャッシュ利用量に対して小さい場合、キャッシュミス頻発の可能性あり

より良い
IO性能
スケール

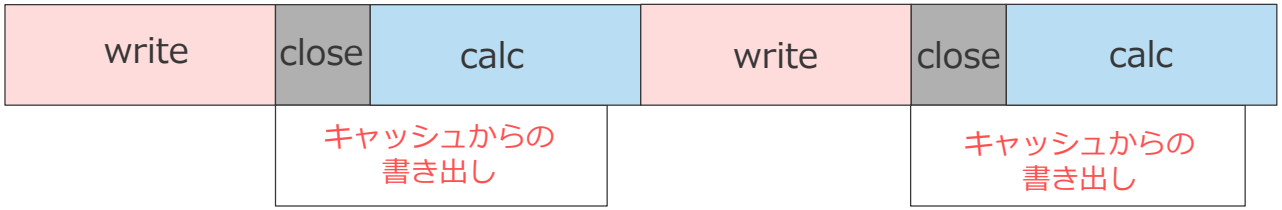
非同期クローズ

LLIOが提供する機能の一つである、非同期クローズについて説明する。

同期クローズ



非同期クローズ



- 同期クローズ
  - クローズ終了の時点で、第1階層および第2階層ストレージへの書出し終了が保証される
- 非同期クローズ
  - 計算ノード内キャッシュから第1階層ストレージおよび第2階層ストレージへの書出し終了を待たずにプログラム中のファイルクローズを終了し、次の計算フェーズを始める
  - ジョブの経過時間制限内であれば、ジョブ終了時にはそれらのファイルの書出しは保証される

非同期クローズ

■ 非同期クローズの有効化/無効化

```
$ pjsub --llio async-close=on sample.sh
```

- --llio async-close=on 非同期クローズ
- --llio async-close=off 同期クローズ (デフォルト値)

■ 非同期クローズ使用時の注意事項

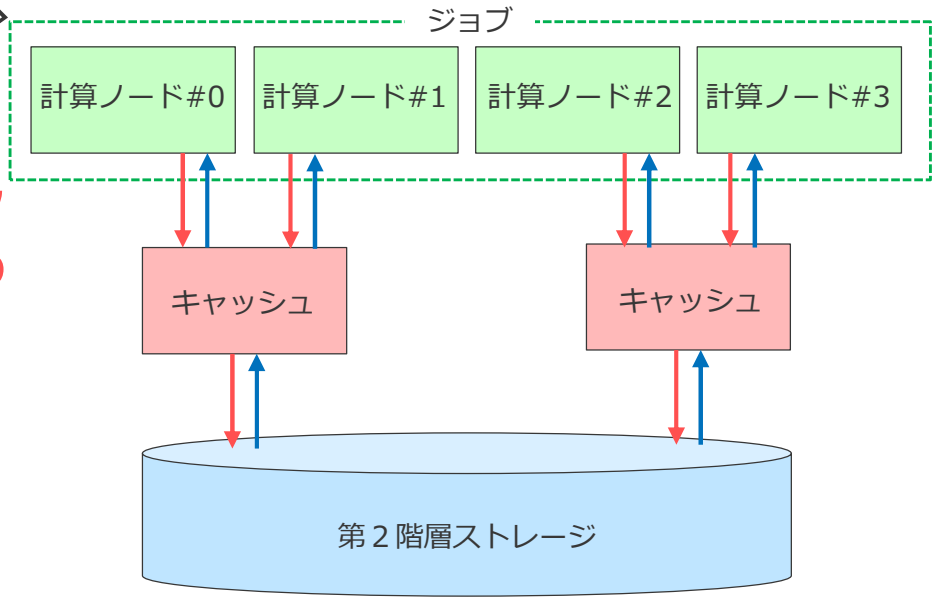
- 計算ノードがダウンした場合やジョブの経過時間制限を超過した場合、キャッシュから第2階層ストレージへの転送は中断される。キャッシュから第2階層ストレージへの書き出しは保証されない

- 第2階層ストレージへの書き出しに失敗した場合は、標準エラー出力に未書出しファイルの一覧が出力される
  - pjsub のオプション --llio uncompleted-fileinfo-path を指定すると、指定したパスに未書出しファイルの一覧を出力できる

第２階層ストレージのキャッシュ領域

LLIOが提供する第２階層ストレージのキャッシュ領域の概要と、その使い方について説明する。

- 第２階層ストレージのキャッシュ領域とは
  - 第１階層ストレージ上に第２階層ストレージがキャッシュされた領域
  - キャッシュを経由した第２階層ストレージへの書き出しを計算処理中に非同期に行うことで高速化が可能（非同期クローズが有効な場合）
  - 同一ジョブに割り当てられた全ての計算ノードから参照可能
- 利用方法
  - パス名：/vol0x0y/data/<*groupname*>
    - ログインノードにおける第2階層ストレージのパス名と同じ
  - ジョブから第2階層ストレージへのアクセスは、基本的に第2階層ストレージのキャッシュ領域を介して行われる
    ※第2階層ストレージに直接アクセスしたい場合は、2ndfs領域（パス名：/2ndfs/<*groupname*>）を用いる。



第２階層ストレージのキャッシュ領域

- 生存期間
  - ジョブが終了する（エラーによる異常終了を含む）、または削除されるまで
  - キャッシュ領域の容量オーバーによりデータが追い出された場合、ジョブが読み込むファイルが第2階層ストレージ上で削除された場合、Direct I/Oが実行された場合もキャッシュ領域の当該データは削除される
  - `pjdel` でジョブを削除する場合、オプション `--llio-flush` によって、経過時間制限値の範囲内で、ファイルへの書き出し完了を待ち合わせられる

- 共通ファイル配布機能
  - コマンド `llio_transfer` を利用して、アクセスが集中するファイルを第2階層ストレージのキャッシュ領域に共通ファイルとして配布する
    - 実行ファイルや入力データなどの共有ファイルに対しては利用を推奨
  - 実行ファイル `a.out` を共通ファイルとして配布する例

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

  - 実行ファイル `a.out` をキャッシュ領域から削除する例

```
$ llio_transfer --purge ./a.out
```

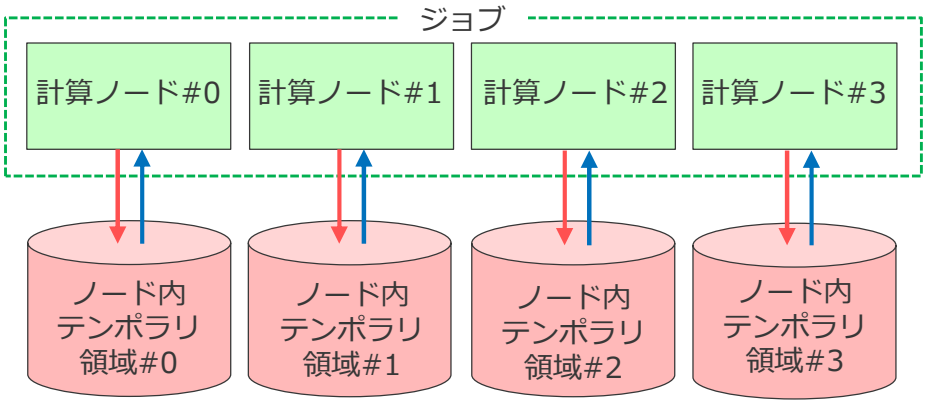
第２階層ストレージのキャッシュ領域

- 共通ファイル配布機能に関する注意事項
  - 読み込みを行うだけのファイルが対象
  - 転送できる共通ファイル数の上限は16,384
  - 全ての計算ノードが同時にopenできる共通ファイルは1,024まで
  - 第2階層ストレージのキャッシュ上にコピーした共通ファイルがジョブスクリプトの途中で不要となる場合は、キャッシュ容量確保のため、`llio_transfer --purge` で適宜削除するとよい
  - 第２階層ストレージ上の共通ファイルのコピー元を変更したり削除したりしてはならない
    - 元ファイルを変更した場合、第2階層ストレージのキャッシュにコピーした共通ファイルの内容が不定になる可能性がある
    - 元ファイルを削除した場合、共通ファイルを `llio_transfer --purge` で削除できなくなる
  - 共通ファイルに対するファイルロックは未サポート
  - `llio_transfer --purge` で共通ファイルを削除した直後は、コピー元ファイルの削除や更新等の操作がエラーで失敗する場合がある
  - `llio_transfer` 使用前に対象のファイルをオープンすると、キャッシュデータが作成され `llio_transfer` がエラーになる場合がある

ノード内テンポラリ領域

LLIOが提供するノード内テンポラリ領域の概要と、その使い方について説明する。

- ノード内テンポラリ領域とは
  - 同一ジョブにおいて、同一計算ノード内で動作するプロセスからのみ参照可能な一時領域
- 利用方法
  - pjsub コマンドのオプションで、`--llio localtmp-size` で領域サイズを指定する必要がある
  - パス名は、ジョブ内で環境変数 `${PJM_LOCALTMP}` により参照する



- 生存期間
  - ジョブ開始時に使用可能になり、ジョブ終了時に削除

ノード内テンポラリ領域

- 利用例
  - ジョブ投入時のオプション例

```
[_LNlogin]$ pjsub --llio localtmp-size=10Gi jobscript.sh
```

  - 簡単なプログラム実行例
    - ノード内テンポラリ領域を prog1 の一時的なデータの格納先とする
    - prog1 の出力を prog2 の入力とし、ノード内テンポラリ領域に結果を出力
    - ノード内テンポラリ領域の結果を第２階層にコピー

```
$ prog1 -o ${PJM_LOCALTMP}/out.data
$ prog2 -i ${PJM_LOCALTMP}/out.data -o ${PJM_LOCALTMP}/result.data
$ cp ${PJM_LOCALTMP}/result.data ${PJM_JOBDIR}/result_${PJM_JOBID}.data
```

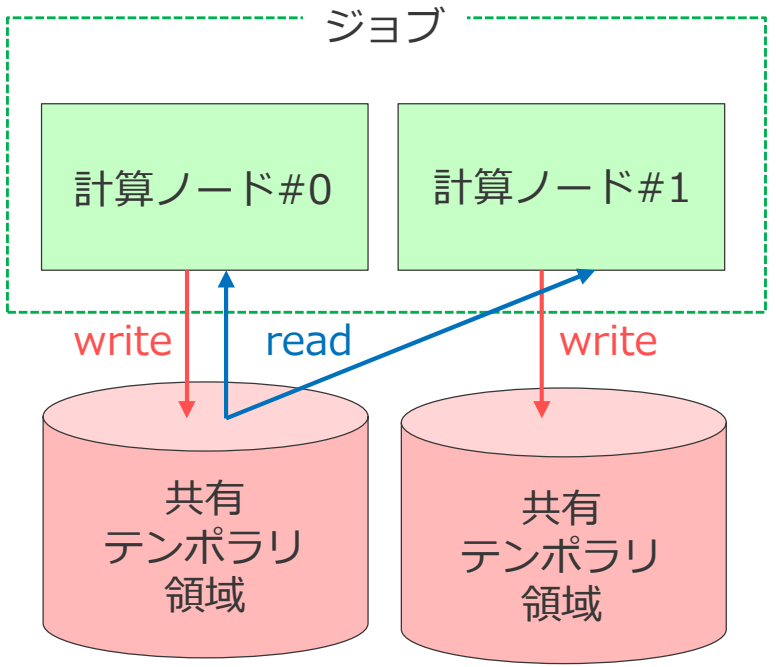
  - ノード内テンポラリ領域に第2階層ファイルをコピーする例
    - ノード内の1プロセスのみを利用してコピーを行う

```
$ mpiexec sh -c 'if [ ${PLE_RANK_ON_NODE} == 0 ]; then cp -rf ./data/ ${PJM_LOCALTMP} ; fi'
```

共有テンポラリ領域

LLIOが提供する共有テンポラリ領域の概要と、その使い方について説明する。

- 共有テンポラリ領域とは
  - 同一ジョブに割り当てられた全ての計算ノードから参照できる一時領域
- 利用方法
  - pjsub コマンドのオプションで、`--llio sharedtmp-size` で1ノード当たりの領域サイズを指定する必要がある
  - パス名は、ジョブ内で環境変数 `${PJM_SHAREDTMP}` により参照する
- 生存期間
  - ジョブ開始時に使用可能になり、ジョブ終了時に削除



共有テンポラリ領域

- 利用例
  - ジョブ投入時のオプション例

```
[_LNlogin]$ pjsub --llio sharedtmp-size=10Gi jobscript.sh
```

※--llio sharedtmp-sizeで指定するのは1ノード当たりの領域サイズ

  - 簡単なプログラム実行例
    - 共有テンポラリ領域を prog1 の一時的なデータの格納先とする
    - prog1 の出力を prog2 の入力とし、共有テンポラリ領域に結果を出力
    - 共有テンポラリ領域の結果を第２階層にコピー

```
$ prog1 -o ${PJM_SHAREDTMP}/out.data
$ prog2 -i ${PJM_SHAREDTMP}/out.data -o ${PJM_SHAREDTMP}/result.data
$ cp ${PJM_SHAREDTMP}/result.data ${PJM_JOBDIR}/result_${PJM_JOBID}.data
```

領域の選択に関する補足

領域の選択時に留意すべきことを補足する。

- ノード内テンポラリ領域と共有テンポラリ領域の選択
  - ノード内テンポラリ領域の特徴
    - ストレージ容量はノード単位での制限を受ける
    - メタデータ処理に際し別のジョブの影響を受けにくい
  - 共有テンポラリ領域の特徴
    - 計算実行のノード規模により確保される総容量が変わるため、大きなファイルを扱うことができる
    - メタデータ処理で別のジョブの影響を受けやすい
  - ファイル容量等で問題が無ければ、ノード内テンポラリ領域を用いることを推奨
  - 共有テンポラリ領域を用いる場合は特に、多数のプロセスから同一ファイルにアクセスしないよう注意が必要

- MPI-IO利用時の領域選択
  - 第2階層ストレージのキャッシュ領域と共有テンポラリ領域の両方が利用可能
  - 性能の観点から、共有テンポラリの利用を推奨

ファイル数の制限

LLIOを利用する上で、１つのジョブが利用できるファイル数の制限について以下に示す。

- １つのジョブが同時に open できる最大ファイル数は3つの領域の合計で1,024 * (計算ノード数)
- ノード内テンポラリ領域に作成可能なファイル数は計算ノードあたり1000万ファイルまで
- 1つのファイルを複数のプロセスから利用する場合、以下の制限を超えるとI/Oがスローダウン、またはSIOがダウンする
  - 同一ファイルを利用するプロセスが存在するノード数が7,000以下
  - 同一ファイルを利用するプロセスの総数が28,000以下
- 上記について、共通ファイル配布機能やストライプ機能を利用した場合、１つのファイルが別ファイルとして数えられる場合がある

I/O プロファイリング

ジョブ投入時に`--llio perf`オプションを指定することで、LLIO性能情報をジョブの出力結果として得ることが可能である。

■ 使用方法

```
[_LNlogin]$ pjsub --llio perf jobscript.sh
```

■ ジョブ終了後、下記のファイル名でLLIO性能情報が出力される

```
(jobname).(jobid).llio_perf
```

■ 詳細は利用手引書 利用およびジョブ実行編『8.7. I/Oのプロファイリング』及び「富岳」利用セミナー 中級編（MPI・LLIO編）を参照

I/O プロファイリング

スケーラブルなHPC向けのI/O評価ツール Darshan がspackにより提供されており、ジョブのI/O解析のために利用可能である。

■ プロファイリングデータの取得

```
[_CNlogin]$ spack load darshan-runtime scheduler=fj
```

- ジョブスクリプト例については、利用手引書 利用およびジョブ実行編『8.7. I/Oのプロファイリング』を参照
- プロファイリングデータの出力先は2ndfs領域を推奨

  ※プロファイリングデータはすべての計算ノードからアクセスされるため、数千ノード以上で利用し、データ領域に出力した場合、LLIOの制限を超過し、ノードがスローダウンする可能性がある

■ プロファイリングデータの分析

```
[_LNlogin]$ spack load darshan-util@3.4.0 arch=linux-rhel8-cascadelake
[_LNlogin]$ darshan-parser --file-list (ユーザー名)_(実行ファイル名)_(JOBID).darshan
```

- I/Oデータの分析方法については利用手引書 利用およびジョブ実行編『8.8. I/Oの最適化』を参照

- はじめに
- 利用について
- コンパイラ
- 数学ライブラリ
- モジュール
- Spack
- スクリプト言語
- ジョブ実行
- MPIジョブ実行
- 計算資源の使用状況確認
- LLIO
- 高度な利用

ラージページ

ラージページ機能は、ページサイズ（メモリ管理の単位）を拡張することで、OSのアドレス変換処理によるコストを低減し、メモリアクセス性能を向上させる機能である。

■ ページサイズの比較

「富岳」ノーマルページ	「富岳」ラージページ
64 KiB	2 MiB

- 「富岳」では、富士通コンパイラを利用時、ラージページ使用がデフォルト設定
- ラージページの副作用：静的データ領域で2倍または3倍のメモリ領域を使用

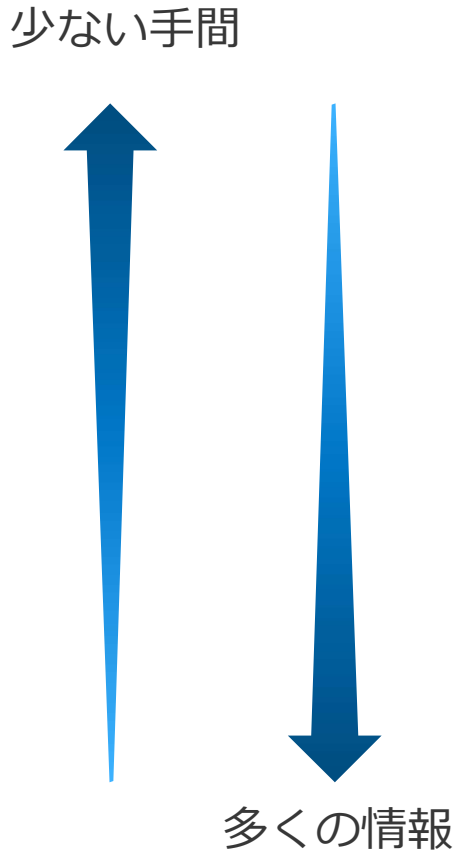
■ メリット・デメリット

	ノーマルページ	ラージページ
メモリ初期化コスト／メモリ使用効率	コスト小／高効率	コスト大／低効率
アクセスするページ数／TLBミス率	多い／高い	少ない／低い
有効なアプリケーションタイプ	メモリ使用量の少ないアプリケーション	大規模なメモリを扱うアプリケーション

プロファイラ

富士通コンパイラでは3つのプロファイラが利用可能。

- プロファイラの種類
  - 基本プロファイラ
    - fippコマンドにより測定
    - 手続・ループごとの実行時間・性能情報、スレッド間・プロセス間インバランスの確認
  - 詳細プロファイラ
    - fappコマンドにより測定
    - 指定した区間ごとの実行時間・性能情報、MPI関数ごとの実行時間の確認
  - CPU性能解析レポート
    - fappコマンドにより測定を複数回実施
    - 演算機・メモリ等のビジー時間・待ち時間・キャッシュミス率等のハードウェアモニタ情報の確認
      - ハードウェアの活用度の確認に有用



※各プロファイラの使い方は「利用手引書 言語開発環境編」または「プロファイラ使用手引書」を参照

より高度に利用したい人のために

- アプリケーションのタイプ別CPU性能チューニング
  https://www.fugaku.r-ccs.riken.jp/doc_root/ja/programming_guides/app-tuning-pattern-Nihongo.pdf
- 富岳利用セミナー（中級編・実践編）
  - 中級編（CPU単体性能の最適化手法：SIMD、ソフトウェアパイプライニングなど）
  - 中級編（MPI・LLIO編）
  - 中級編（CPU単体性能の最適化手法：演算の効率化、キャッシュチューニングなど）
  - 実践編（ハンズオン）

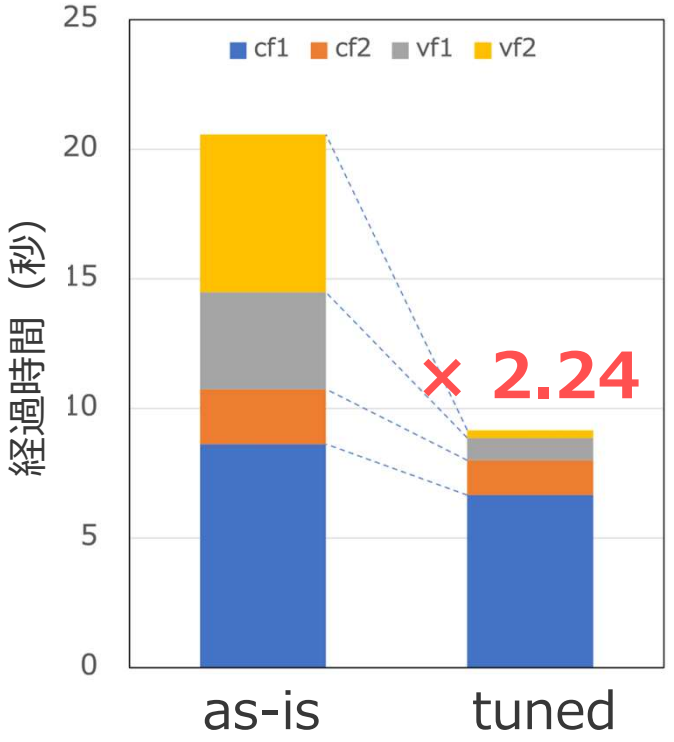
  ※開催済みのテキストは、富岳ウェブサイトの『講習会資料』から入手可能
- R-CCS/RIST Joint Seminar on Advanced use of Supercomputer Fugaku and Arm computer systems（旧名称：A64FX向けチューニング技術検討会）（英語での開催）
  https://www.hpci-office.jp/events/seminars#FugakuAndArm

高度化支援のご案内

RISTは、「富岳」を中核とするHPCIシステムの利用研究課題を対象にアプリソフトの移植・高速化・高並列化等の支援を無償で実施中

- 支援対象課題
  - 一般・若手課題、産業課題、臨時募集課題
    - 参考：https://www.hpci-office.jp/using_hpci/project_categories_overview
  - 「富岳」成果創出加速プログラム課題
- 支援内容
  - 移植支援
    - 各計算機資源を実際に使用した動作確認等
  - 高速化支援、高並列化支援
    - 実効性能分析（プログラムの性能調査、ホットスポット・ロードバランスの調査、、並列性能調査）
    - 性能向上策の提案
- 支援申込方法
  - HPCIポータルサイトを参照
    https://www.hpci-office.jp/user_support/tuning_support

是非ご活用ください



支援例：FFVHC-ACEのカーネルプログラムに対する支援の効果
* 出典: https://www.hpci-office.jp/pages/e_meeting_A64FX_210427/#topic-2,