HOKUSAI BigWaterfall2 利用手引書

Version 1.12

2025 年 5 月 9 日

理化学研究所

情報セキュリティ・システム部

改版履歴

版	発行年月日	章	改版理由
1.0	2023.12.4	-	新規作成
1.1	2023.12.7	1.2	超並列演算システム、大容量メモリ演算サーバの演算性能の計算式を修正
1.2	2023.12.8	5.7.5	説明文とコマンド実行例を修正
1.3	2023.12.11	5.6	ジョブスクリプト例を修正
1.4	2023.12.12	2.2 5.6	接続先アドレスを追加 ジョブスクリプト例を修正
1.5	2023.12.19	3.2	Lfs quota コマンドを修正
1.6	2023.12.21	5.6	ジョブスクリプト例を修正
1.7	2023.12.22	5.2	ジョブ実行制限を修正
1.8	2023.12.27	5.2 5.3	メモリの単位を GiB 表記に修正 表 5-10 を修正
1.9	2024.4.2	3.2 4.2 5.2 5.4 5.3.2 5.6.1	HPCI 課題の説明を追加 GPU 向けのコンパイル/リンク例を追記 mpc_ht の記述を削除 GPU サーバのジョブ実行リソースを追記 GPU でインタラクティブジョブを実行する際の注釈を追記 ジョブ投入オプションを追記 GPU サーバ(GPU)向けジョブスクリプトの例の記載を追記
1.10	2024.7.1	全体 1.1 1.2 1.6 3.1 3.2 4.1 4.5 5.2 5.3 6 章	用語統一 システム構成図の修正 メモリの単位変更 パーティションの優先度の説明を追記 文字化け修正 GID の確認方法、lfs コマンドの-h オプションのコメント追記 (default)表記の説明、intel ロード時の補足を追記 開発ツールの記載を 6 章から 4.5 に移動 パーティションにおける課題ごとの設定についての記載内容を修正 パーティションについての記載から「mpc_a」と「lmc_a」の記載を削除 設定値の修正 --gpus オプションを追記 Singularity コンテナ追記
1.11	2024.10.25	序論	問い合わせ方法の記載を修正
1.12	2025.5.9	1.6	ジョブ実行順序の設定変更に伴う記載内容の修正

はじめに

本利用手引書は、理化学研究所が導入したスーパーコンピュータ・システム(HOKUSAI BigWaterfall2 システム)の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。

本利用手引書の内容は不定期に更新いたします。最新の利用手引書は、以下の HBW2 ポータルより入手することが可能です。

https://hokusai.riken.jp/

HBW2 ポータルでは、利用可能なアプリケーションのバージョンや実行方法がまとめてあります。また、ssh 公開鍵の登録もここで行います。詳細につきましては「HOKUSAI BigWaterfall2(HBW2) Web ポータル 利用手引書」をご参照ください。

システムの運用状況は HBW2 ポータルや利用者向けメーリングリストでアナウンスします。また、HOKUSAI BigWaterfall2 システムについてのあらゆるご質問は、以下の Web ページをご参照頂き、お問い合わせください。

https://i.riken.jp/supercom/contact/

1. HOKUSAI BigWaterfall2 システムの概要

1.1 システム構成

HOKUSAI BigWaterfall2 システム(以下、HBW2)は、超並列演算システム(MPC)、大容量メモリ演算サーバ(LMC)と、システムの利用入口となるフロントエンドサーバ、共有ストレージから構成されるシステムです。

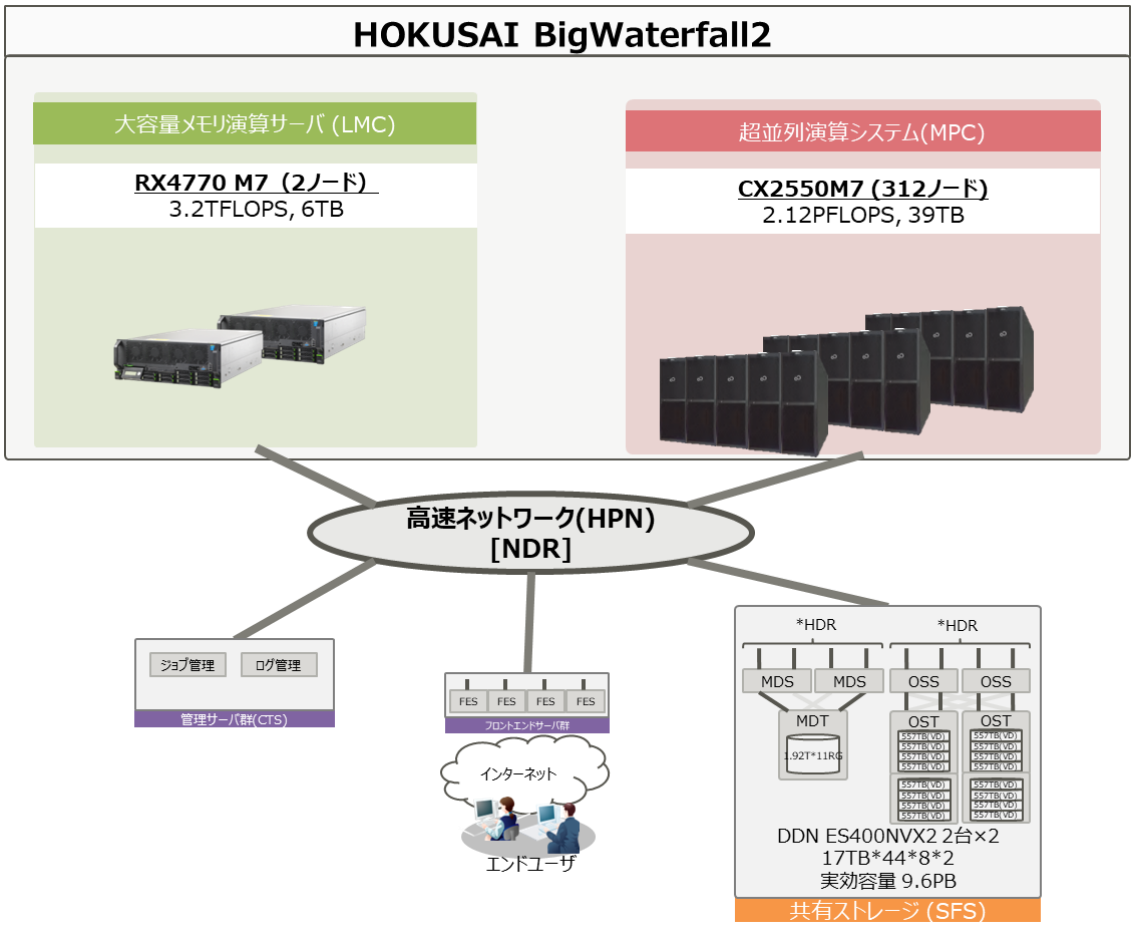

図 1-1 システム構成図

HBW2 の超並列演算システム(MPC)は、PRIMERGY CX2550 M7 を 312 ノードで構成します。1 ノードの理論演算性能は 6.8TFLOPS、主記憶容量は 119GiB です。各ノードは、InfiniBand NDR(400GB/s)で接続され、高速なノード間通信とファイル共有を実現します。

大容量メモリ演算サーバ(LMC)は、PRIMERGY RX4770 M7 を 2 ノードで構成します。1 ノードの理論演算性能は 6.4TFLOPS、主記憶容量は 2.7TiB です。各ノードは、InfiniBand NDR(400.0GB/s)で接続され、高速なノード間通信とファイル共有を実現します。

ストレージ環境は、共有ストレージ(SFS)で構成します。

共有ストレージ(SFS)は、各ユーザーのホームディレクトリや課題グループ用の共有ディレクトリなど、広帯域でオンライン性のあるファイルシステムであり、超並列演算システム(MPC)、大容量メモリ演算サーバ(LMC)およびフロントエンドサーバから参照可能です。利用可能容量は 9.8PB です。

HBW2 へのアクセスは、ssh/scp によるアクセスと HTTPS アクセス(HBW2 ポータル)が可能です。ユーザーはフロントエンドサーバ上にて、プログラムの編集、コンパイル/リンク、バッチジョブの操作、インタラクティブジョブの実行、チューニング、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

1.2.1 超並列演算システム(MPC)

- 演算性能
  CPU: Intel Xeon Max 9480 (1.9GHz) 312 台(624CPU, 34,944 コア)
  理論ピーク性能: 2.12 PFLOPS (1.9GHz × 56 コア × 32FMA × 2CPU × 312 ノード)
- 主記憶
  メモリ容量: 36TiB(119GiB × 312 台)
  メモリバンド幅: 3.26TB/s
- 内蔵ディスク
  ディスク容量: 300TB (480GB × 2 × 312 台)
- インターコネクト
  InfiniBand NDR
  通信性能:ノード間 400GB/s

1.2.2 大容量メモリ演算サーバ(LMC)

- 演算性能
  CPU: Intel Xeon Gold 6418H (2.1GHz) 2 台(8CPU, 192 コア)
  理論ピーク性能: 6.1TFLOPS (2.1GHz × 24 コア × 4CPU × 32FMA)
- メモリ
  メモリ容量: 5.4TiB(2.7TiB × 2 台)
  メモリバンド幅: 1126.4GB/s
- 内蔵ディスク
  ディスク容量: 2.4TB (600GB × 2 × 2 台)
- インターコネクト
  InfiniBand NDR
  通信性能:ノード間 400GB/s

1.3 ソフトウェア構成

HBW2 で利用可能なソフトウェア一覧を以下に示します。

表 1-1 ソフトウェア一覧

項目	超並列演算システム(MPC)	大容量メモリ演算サーバ (LMC)	フロントエンドサーバ
OS	Rocky Linux 8(x 312 台) (Linux kernel version 4.18)	Red Hat Enterprise Linux 8 (Linux kernel version 4.18)	Red Hat Enterprise Linux 8 (Linux kernel version 4.18)
コンパイラ	インテル oneAPI ベース & HPC ツールキット マルチノード インテル C/C++および Fortran コンパイラ インテル TBB インテル Distribution for Python		
ライブラリ	インテル oneAPI ベース & HPC ツールキット マルチノード インテル MKL インテル MPI ライブラリ インテル IPP インテル DAL		
ツール	インテル oneAPI ベース & HPC ツールキット マルチノード インテル VTune インテル Advisor インテル Inspector インテル Trace Analyzer & Collector		
アプリケーション	Gaussian, ADF, AMBER, GAMESS,GROMACS, NAMD, ROOT	Gaussian, ADF, AMBER, GAMESS, GROMACS, NAMD, ROOT	GaussView, VMD, ROOT

HBW2 で利用可能なアプリケーションやライブラリなどの最新情報、利用方法は以下の HBW2 ポータルに掲載します。

https://hokusai.riken.jp/

1.4 サービス時間

HBW2 は原則 24 時間サービスを行いますが、システムの定期保守作業、緊急保守作業、設備保守作業(空調機メンテナンス・電源設備点検)を行う必要がある場合など、サービスを休止します。HBW2 の運用状況は HBW2 ポータルや利用者向けメーリングリストでアナウンスします。

1.5 利用規約について

詳細は「スーパーコンピュータ・システム利用規約」をご参照ください。「スーパーコンピュータ・システム利用規約」は以下の URL から参照できます。

https://i.riken.jp/supercom/

1.5.1 利用可能演算時間

課題によって、利用できる演算時間が異なります。課題番号(project ID)、割り当てられた演算時間、使用済み演算時間、割り当てられた演算時間の有効期限は、*listcpu* コマンドで確認できます。

使用済み演算時間が 100%に達した場合、ジョブを新規に投入/実行できなくなります。

```
[username@hokusai1 ~]$ listcpu -p  課題番号(project ID)
[RB999999]
Node/User       Used(h)     Limit(h)     Used(%)     Expiry date
----------      ---------   ----------   ---------   -------------
mpc                0        5115801.6     0.0         2024-03-31
lmc                0        28108.8       0.0         2024-03-31
userA              0            -             -          2024-03-31
userB              0            -             -          2024-03-31
:
```

表 1-2 listcpu コマンドの表示項目

項目	説明
Used (h)	使用済み演算時間 (単位: 時間)
Limit (h)	割り当て演算時間 (単位: 時間)
Use (%)	使用済み演算時間 / 割り当て演算時間 (単位: %)
Expiry date	割り当て演算時間の有効期限

1.6 ジョブの実行順序について

本システムでは、ジョブの優先度をリアルタイムに評価し実行順序を制御しています。優先度の評価は以下の項目を基に行っています。

表 1-3 優先度制御における評価項目

評価順序	評価項目	概要
1	課題毎のフェアシェア値	利用状況によって算出される課題毎のフェアシェア値が高い順に実行

フェアシェアによる優先度制御は、課題毎に割り当てられたリソースに対して、公平にリソースを利用できるようにするための仕組みです。

課題毎にフェアシェア値を持ち、フェアシェア値が一番高い課題から優先的にジョブを実行します。直近の利用量が多いとフェアシェア値が下がり、時間が経つと回復します。そのため割り当てられたリソースを有効に活用するためには、一年を通して継続的に利用することが必要です。

2. HOKUSAI BigWaterfall2 システムへのログイン

2.1 システムログインまでの流れ

HBW2 にログインするためにはアカウント申請、公開鍵登録を実施する必要があります。アカウント申請や公開鍵登録は HBW2 ポータルから実施してください。詳細の手順につきましては「HOKUSAI BigWaterfall2(HBW2) Web ポータル 利用手引書」をご参照ください。

2.2 接続先アドレス

HBW2 に SSH/HTTPS アクセス可能なサーバはフロントエンドサーバです。接続先アドレスを表 2-1 に示します。

フロントエンドサーバは合計 4 台のサーバで構成されており、hokusai.riken.jp にアクセスすると、DNS ラウンドロビンにより、いずれかのログインノードにアクセスすることができます。HBW2 ポータルは、HTTPS アクセスにより、SSH 公開鍵登録やマニュアル参照等が可能です。

表 2-1 接続先アドレス一覧

ホスト名	プロトコル	アクセス用途
hokusai.riken.jp	ssh	• 仮想端末 • ファイル転送
hokusai1.riken.jp		
hokusai2.riken.jp		
hokusai3.riken.jp		
hokusai4.riken.jp		
hokusai.riken.jp	https	• SSH 公開鍵登録 • マニュアル参照 • 各種申請

3. ファイルシステム環境

3.1 ストレージ領域

HBW2 では、以下のストレージ領域が利用可能です。

表 3-1 ストレージ領域一覧

ディレクトリ	サイズ	目的
/home	9.6PB	ホーム領域
/data		データ領域(課題単位、利用負担金の対象)
/tmp_work		一時領域
/arc	申請量	階層型ストレージ領域（課題単位、新規受付停止）

各サーバから各ストレージへのアクセス可否を以下に示します。

表 3-2 ストレージ領域へのアクセス可否一覧

ストレージ種別	フロントエンドサーバ	MPC	LMC
共有ストレージ	○	○	○
階層型ストレージ	○	×	×

○ : 利用可能　　× : 利用不能

3.1.1 共有ストレージ

共有ストレージはユーザーのホーム領域および課題単位のデータ領域となります。ホーム領域は、利用者あたり 4TB ずつ割り当てられます。データ領域は利用負担金の対象となります。データ領域を利用されたい場合は、HBW2 ポータルから申請を行ってください。

表 3-3 共有ストレージの割り当て制限

ディレクトリ	割当ディレクトリ	使用量制限	Inode 数制限
/home	/home/ユーザ名	4TB	1000 万
/data	/data/課題の GID(GID of project)	1TB～	1TB あたり 60 万

3.1.2 階層型ストレージ

階層型ストレージ領域（テープ領域）はキャッシュストレージとテープライブラリ装置からなる長期保存用のストレージとなります。階層型ストレージはフロントエンドサーバから参照可能であり、課題単位での利用となりますが、現在は新規の受付は行っていません。

表 3-4 階層型ストレージの割り当て制限

領域名	使用量制限	Inode 数制限
/arc	テープ 2 巻/8TB～テープ 13 巻/52TB)	テープ 1 巻あたり 40,000 ファイル

主な使用目的は以下のとおりです。

- 大量データの長期保存
- バックアップ

階層型ストレージはテープにデータを保管しますので、サイズの小さなファイル(100MB未満)の格納には不向きです。多くのデータを格納する場合、サイズの小さな複数のファイルを 1 ファイルにアーカイブして格納してください。

3.2 ディスク使用状況と上限値の表示

ディスク使用状況と上限値を表示する場合は、*lfs quota* コマンドを実行してください。/data 領域の使用状況を表示する場合、-p オプションにて課題の GID(GID of project)を指定してください。理研内課題の GID は申請時に通知されました"RB"から始まる課題番号(project ID)の GID になります。HPCI 課題の GID は申請時に通知されました"HP"から始まる課題番号の GID の先頭に 1 を付与し桁上げした数値となります。
課題の GID は id コマンドで確認することができます。

```
[username@hokusai1 ~]$ lfs quota -p ${UID} ${HOME}
Disk quotas for usr username (uid XXXX):
     Filesystem  kbytes   quota   limit   grace   files   quota   limit   grace
 /home/username 8637844       0       0       -   35783       0       0       -
[username@hokusai1 ~]$ lfs quota -p 課題の GID/data/課題番号
Disk quotas for prj XXXXX (pid XXXXX):
     Filesystem  kbytes   quota   limit   grace   files   quota   limit   grace
 /data/RB999999 8637844       0 2000000000    -   35783       0       0       -
```

表 3-5 ディスク使用状況と上限値の表示項目

項目	説明
[kbytes]	ディスクの使用容量と使用上限、使用率をサイズ表示(単位:KB) ※-h オプションを指定してコマンド実行すると見やすい単位に変換して表示されます
[Files]	ファイル(inode)数の使用容量と使用上限、使用率を数量表示(単位:K)

3.3 一時領域

テンポラリファイルを格納する一時領域として、/tmp_work が利用できます。

/tmp_work 配下のデータについては、更新日が 1 週間以前のデータは自動的に削除されます。テンポラリファイルだけを格納する領域として使用してください。

4. コンパイル/リンク

4.1 環境設定

コンパイラ、ライブラリ、アプリケーション、ツールの利用時に必要となる環境変数は、*module* コマンドを使用して容易に設定することができます。

```
$ module  <subcommand>  <subcommand-args>
```

module コマンドのサブコマンドは以下のとおりです。

表 4-1 *module* コマンドのサブコマンド一覧

オプション	説明
avail	利用可能な設定一覧を表示します。
list	利用中の設定一覧を表示します。
load module	設定を読み込みます。
unload module	設定を解除します。
purge	全ての設定を解除します。
switch module1 module2	設定を切り替えます。

例) 利用可能な設定一覧を表示する。

```
[username@hokusai1 ~]$ module avail

-------------------------------------------
/lustre/opt/modulefiles/hokusai/apps ------------------------------------
-----------
adf/2022.103(default)  gamess/2023R2
gaussview/6.0.16       gromacs/2023.3(default)
amber/22(default)      gaussian/16_C.02(default)
gold/2023.2(default)   vmd/1.9.4
-------------------------------------------
/lustre/opt/modulefiles/hokusai/compilers ------------------------------
--------------
boost/1.83.0           fftw-mpi-double/3.3.10
gcc/13.2.0(default)    intelmpi/impi_23.2.0
```

※(default)表示のバージョンは、HBW2 での推奨バージョンです。
バージョン指定無しでモジュールを load した場合は、(default)表示のバージョンの設定を読み込みます。

例) Intel コンパイラを利用する場合

```
[username@hokusai1 ~]$ module load intel
```

※intel を指定すると、intelmpi も合わせて設定を読み込みます。

例) 利用中の設定一覧を表示する。

```
[username@hokusai1 ~]$ module list
  Currently Loaded Modulefiles:
   1) intelmpi/impi_23.2.0   2) intel/23.02.1
```

排他関係にある設定は同時に利用することはできません。利用しようとした場合、以下のエラーメッセージが出力され、設定に失敗します。

```
   [username@hokusai1 ~]$ module load openmpi/4.1.6
   Loading openmpi/4.1.6
     ERROR: openmpi/4.1.6 cannot be loaded due to a conflict.
     HINT: Might try "module unload intelmpi" first.
```

例) 利用中の設定を解除する。

```
[username@hokusai1 ~]$ module purge
[username@hokusai1 ~]$ module list
No Modulefiles Currently Loaded.
```

4.2 コンパイラ

HBW2 のフロントエンドサーバでは、超並列演算システム(MPC)と大容量メモリ演算サーバ(LMC)で実行可能なロードモジュールを作成するためのコンパイラが利用できます。

例) 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けのコンパイル/リンクを行う場合

```
[username@hokusai1 ~]$ module load intel
Loading intel/23.02.1
  Loading requirement: intelmpi/impi_23.2.0
[username@hokusai1 ~]$ module list
Currently Loaded Modulefiles:
   1) intelmpi/impi_23.2.0   2) intel/23.02.1
```

GPU サーバでは、GPU サーバ(GPU)で実行可能なロードモジュールを作成するためのコンパイラが利用できます。

例) GPU サーバ(GPU)向けのコンパイル/リンクを行う場合

```
[username@hokusai1 ~]$ srun --partition gpu_i --account RB999999 --pty $SHELL
[username@bwgpu4 ~]$ module load cuda/12.4

[username@bwgpu4 ~]$ module list
Currently Loaded Modulefiles:
 1) cuda/12.4
```

4.3 コンパイル/リンク概要

コンパイル/リンクのコマンド一覧は以下のとおりです。

表 4-2 超並列演算システム(MPC)、大容量メモリ演算サーバ(LMC)
コンパイル/リンクコマンド一覧

種別	言語処理系	コマンド	自動並列[注1]	OpenMP [注1]
逐次 (非 MPI)	Fortran	ifort	-parallel	-qopenmp
	C	icc		
	C++	icpc		
MPI 並列	Fortran	mpiifort		
	C	mpiicc		
	C++	mpiicpc		

注 1: 自動並列、OpenMP オプションはデフォルトでは無効です。

4.3.1 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けコンパイル/リンク

逐次プログラムのコンパイル/リンク

逐次プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンクする場合、*ifort*/*icc*/*icpc* コマンドを使用します。

```
ifort/icc/icpc[option] file [...]
```

例1)逐次 Fortran プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ ifort seq.f
```

例2)逐次 C プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ icc seq.c
```

スレッド並列プログラムのコンパイル/リンク

スレッド並列プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンクする場合、*ifort*/*icc*/*icpc* コマンドを使用します。

```
ifort/icc/icpc thread-option [option] file [...]
```

例 1) Fortran プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けに自動並列を有効にしてコンパイル/リンク

```
[username@hokusai1 ~]$ ifort -parallel para.f
```

例 2) C プログラムをアプリケーション超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けに自動並列を有効にしてコンパイル/リンク

```
[username@hokusai1 ~]$ icc -parallel para.c
```

例 3) OpenMP Fortran プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ ifort -qopenmp omp.f
```

例 4) OpenMP C プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ icc -qopenmp omp.c
```

例 5) OpenMP Fortran プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けに自動並列も有効にしてコンパイル/リンク

```
[username@hokusai1 ~]$ ifort -parallel -qopenmp omp_para.f
```

例 6) OpenMP C プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けに自動並列も有効にしてコンパイル/リンク

```
[username@hokusai1 ~]$ icc -parallel -qopenmp omp_para.c
```

MPI 並列プログラムのコンパイル/リンク

MPI 並列プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンクする場合は、*mpiifort*/*mpiicc*/*mpiicpc* コマンドを使用します。

```
mpiifort/mpiicc/mpiicpc [option] file [...]
```

例 1) MPI Fortran プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ mpiifort mpi.f
```

例 2) MPI C プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ mpiicc mpi.c
```

例 3) ハイブリッド並列(MPI+OpenMP)Fortran プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ mpiifort -qopenmp mpi_omp.f
```

例 4) ハイブリッド並列(MPI+OpenMP)C プログラムを超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けにコンパイル/リンク

```
[username@hokusai1 ~]$ mpiicc -qopenmp mpi_omp.c
```

オプションによるプログラムの最適化(Fortran/C/C++)

最適化は演算結果に影響を与える場合があります。最適化を施した場合は実行結果が正しいかどうか検証してください。以下に記載するオプションが全てではありません。詳細は各コンパイルコマンドのマニュアル(man コマンド)をご参照ください。

表 4-3 一般的な最適化オプション

コンパイル オプション	説明
-O0	最適化は行われません 。このオプションは、アプリケーション開発の初期段階およびデバッグ時に使用します。アプリケーションが正常に動作することを確認した後は、より高度なオプションを使用してください。
-O1	サイズを考慮した最適化を行います。オブジェクトのサイズを増やす傾向が ある最適化を省略します。多くの場合、最小限のサイズで最適化されたコー ドが作成されます。 コードサイズが大きいため、メモリーページングが問題になる巨大なサーバー / データベース・アプリケーションにおいて、このオプションは効果的です。
-O2	最速化します(デフォルト設定)。ベクトル化と実行速度を改善する多くの 最適化を有効にします。多くの場合、/O1 (-O1) よりも速いコードを作成します。
-O3	-O2 の最適化に加えて、スカラー置換、ループアンロール、分岐を除去するコード反復、 効率的にキャッシュを使 用するループ・ブロッキング、 データ・プリフェッチ機能など、強力なループ最適化およびメモリアクセス最適化を行います。 -O3 オプションは、特に浮動小数点演算を多用するループや大きな データセットを処理するループを含むアプリケーションに推奨します。この最適化は、場合によって-O2 の最適化よりもアプリケーションの実行が遅くなることがあります。
-fast	プログラム全体の速度を最大限にします。次のオプションを設定します。 -ipo, -O3, -no-prec-div, -static, -fp-model fast=2, -xHost * -static は MPI プログラムのリンク時には指定できません。
-qopt-report[=*n*]	最適化レポートを作成し、標準エラー出力します。n には、0(レポートなし)から 5(最大限の情報)の範囲で詳細レベルを指定します。デフォルトは 2 です。
-qopt-report-phase[=*list*]	最適化フェーズごとのレポートを生成します。指定可能な list は man を参照してください。
-qopt-report-help	上記の-qopt-report-phase で利用可能なすべてのフェーズオプションを表示します。コンパイルは行いません。
-qopt-report-routine=string	string で指定された文字列を含む関数またはサブルーチンに関するレポートを生成します。デフォルトでは、すべての関数およびサブルーチンのレポートが生成されます。

表 4-4 並列パフォーマンスオプション

コンパイル オプション	説明
-qopenmp	OpenMP 指示句がある場合、その指示によるマルチスレッド・コードが生成されます。スタックのサイズを増やさなければならないことがあります。
-parallel	自動パラレライザーは、安全に並列実行可能な構造のループを検出し、そのループに対するマルチスレッド・コードを自動生成します。
-par-threshold[*n*]	パフォーマンス向上の可能性に基づいて、ループの自動並列化のしきい値を設定します(n=0 から n=100。デフォルト: n=100)。 0 – 計算量にかかわらず並列化を行います。 100 – パフォーマンス向上が確実に見込める場合にのみ、ループを並列化します。 -parallel とともに使用する必要があります。
-guide[=*n*]	ガイド付き自動並列化。ループのベクトル化や自動並列化を促進するアドバイスを生成します。オブジェクト・ファイルや実行ファイルは生成されません。自動並列化のアドバイスは、-parallel オプションが同時に指定された場合に提供されます。n には 1 から 4 までの値で、アドバイスのレベルを指定します。4 は最も詳細なレベルのアドバイスです。n が省略された場合、デフォルトは 4 です。
-qopt-matmul	コンパイラが生成する行列乗算(matmul)ライブラリ呼び出しを有効にします。行列乗算におけるループの入れ子(該当する場 合)を特定し、ループを matmul ライブラリ呼び出しに置換して、パフォーマンスを向上します。このオプションは、-O3 と-parallel が指定されると、デフォルトで有効になります。また、このオプションは、-O2 オプション以上が指定されない限り、効果はありません。
-coarray=shared	共有メモリーシステムにおいて Fortran 2008 標準の Co-Array を有効にします(Fortran のみ)。

表 4-5 プロセッサ専用最適化オプション

コンパイル オプション	説明
-x*target*	*target* で指定された命令セットをサポートするインテルプロセッサ向けの専用コードを生成します。 超並列演算システム(MPC)向けに最適化する場合、オプション-xCORE-AVX512 を指定してください。 GPU 演算サーバ向けに最適化する場合、オプション-xCORE-AVX2 を指定してください。
-xhost	コンパイルを行うホスト・プロセッサで利用可能な最上位の命令セッ トを利用したコードを生成します。これは、最適な-x オプションに対応します。

表 4-6 プロシージャ間最適(IPO)オプションとプロファイルに基づく最適化オプション

コンパイル オプション	説明
-ip	単一ファイルの最適化を行います。現在のソースファイルを対象にしたインライン展開を含むプロシージャ間の最適化です。
-ipo[=*n*]	インライン展開およびその他のプロシージャ間の最適化を複数のソースファイルに対して行います。オプションの n 引数には、コンパイル時に生成するオブジェクトファイルの最大数を指定します。デフォルトの n は 0 です(コンパイラが最適なファイル数を自動選択)。
-ipo-jobs[*n*]	プロシージャ間の最適化(IPO)のリンクフェーズで、同時に実行するジョブ数を指定します。デフォルトは 1 ジョブです。
-finline-functions -inline-level=2	インライン展開をコンパイラの判断に任せます。このオプションは、-O2 および−O3 を指定すると有効になります。
-inline-factor=*n*	インライン化される関数の合計サイズと最大サイズを指定します。n のデフォルト値は 100(100%、スケール係数 1)です(Fortran のみ)。
-prof-gen	プロファイル最適化で参照する動的なパフォーマンスデータを生成するため、プログラムにインストルメント・コードを埋め込みます。
-prof-use	最適化中に-prof-gen オプションで生成した実行ファイルのプロファイリング情報を参照します。

表 4-7 浮動小数点演算最適化オプション

コンパイル オプション	説明
-fp-model *name*	浮動小数点演算における演算モデルを制御します。特定の最適化を制限して浮動小数点結果の一貫性を強化します。
-ftz[-]	メインプログラムをこのオプションでコンパイルすると、プログラム全体で SSE 命令によるデノーマル結果をゼロにフラッシュします。-O0 を除き、デフォルトはオンです。
-fimf-precision:*name*	算術ライブラリ関数の精度を制御します。デフォルトはオフです(コンパイラは、デフォルトのヒューリスティックを使用)。name の値は、high、medium、low のいずれかです。精度を下げるとパフォーマンスの向上につながります。また、その逆についても同じことが言えます。算術ライブラリのルーチンの多くは、 インテル製マイクロプロセッサ向けにより高度に最適化されています。
-fimf-arch-consistency=*true*	算術ライブラリ関数が、同じアーキテクチャの異なるインテルプロセッサにおいて一貫 した結果を生 成します。ランタイム・パフォーマンスが低下することがあります。デフォルトは “false”(オフ)です。
-prec-div	浮動小数点除算の精度を上げます。精度を上げるとパフォーマンスに影響します。
-prec-sqrt	平方根計算の精度を上げます。精度を上げるとパフォーマンスに多少影響します。

表 4-8 きめ細かなチューニングオプション

コンパイル オプション	説明
-unroll[*n*]	ループをアンロールする最大回数を設定します。-unroll0 はループアンロールを無効にします。デフォルトは、-unroll で、コンパイラの判断に任せます。
-qopt-prefetch[=*n*]	ソフトウェア・プリフェッチのレベルを制御します。n には 0(プリフェッチなし)から 4(強力なプリフェッチ)までの値です。高レベルの最適化が有効な場合、デフォルトは 2 です。
-qopt-block-factor=*n*	ループのブロッキング最適化におけるブロッキング係数を指定します。 ブロッキング係数は、1 ブロックのループ反復数です。ループ・ブロッキングは-O で有効になり、キャッシュ中のデータ利用率を高めるように設計されています。
-qopt-streaming-stores =*mode*	ストリーミング・ストアを生成するかどうかを指定します。 *mode* の値は次のとおりです。 always: アプリケーションがメモリ上のデータ再利用がほとんどないことを仮定して、キャッシュをバイパスするストリーミング・ストアの生成をコンパイラに指示します。 never: ストリーミング・ストアの生成を無効にします。 auto: ストリーミング・ストアの生成をコンパイラの判断に任せます。
-fno-alias	プログラムでエイリアシングしないことを前提に最適化します。デフォルトではオフです。
-fno-fnalias	関数内でエイリアシングしないことを前提に最適化します。デフォルトではオフです。
-fexceptions	-fexceptions: C++のデフォルトです。例外処理テーブルの生成を有効にします。 -fno-exceptions: C、Fortran のデフォルトです。コードサイズが小さくなります。C++では、例外指定は解析されますが、無視されます。構造化例外処理(try ブロックや throw 文)を使用していると、一連の呼び出し中の関数が-fno-exceptions でコンパイルされている場合、エラーが発生します。
-vec-threshold *n*	パフォーマンス向上の可能性に基づいて、ループのベクトル化のしきい値 n を設定します(n=0 から n=100。デフォルト : n =100)。 0 – 計算量にかかわらずベクトル化されます。 100 – パフォーマンス向上が確実に見込める場合にのみループはベクトル化されます。

4.4 数値計算ライブラリ

数値計算ライブラリとして BLAS/LAPACK/ScaLAPACK が利用可能です。

4.4.1 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)の数値計算ライブラリ

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向け BLAS/LAPACK/ScaLAPACK を利用する場合、以下のコンパイル/リンクオプションが利用可能です。

表 4-9 BLAS/LAPACK/ScaLAPACK オプション一覧

ライブラリ	並列性	指定オプション	備考
BLAS	逐次	-qmkl=sequential	
	スレッド並列	-qmkl=parallel	
LAPACK	逐次	-qmkl=sequential	
	スレッド並列	-qmkl=parallel	
ScaLAPACK	MPI 並列	-qmkl=cluster	

例 1) 逐次版 BLAS/LAPACK を利用する。

```
$ ifort -qmkl=sequential blas.f
```

例 2) スレッド並列版 BLAS/LAPACK を利用する。

```
$ ifort  -qmkl=parallel blas.f
```

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

```
$ mpiifort -qmkl=cluster scalapack.f
```

上記以外の組み合わせについては、以下 URL を参照してください。

https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

4.5 開発ツール

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けに以下のツールが利用できます。

- インテル VTune Amplifier XE(パフォーマンス分析ツール)
- インテル Inspector XE(エラー検出ツール)
- インテル Advisor XE(スレッド化アシスタント)
- インテル Trace Analyzer & Collector(MPI 通信パフォーマンス分析ツール)

5. バッチジョブ・インタラクティブジョブ

5.1 ジョブ管理システム概要

HBW2 ではジョブ管理システムによりバッチジョブおよびインタラクティブジョブの実行が制御されます。ユーザーはジョブ実行に必要なパーティション名、ノード数、コア数、経過時間などを指定し、ジョブ管理システムにジョブ実行を依頼します。

HBW2 で実行可能なジョブは以下の 2 種類です。

表 5-1 ジョブの種類

ジョブ形式	用途
バッチジョブ	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、オプションによりジョブの自動再実行が可能。
インタラクティブジョブ	ユーザー端末からのデータ入力により、会話形式でジョブを実行する。 主にデバッグ等での利用を想定。 ノードダウンなどの異常発生時、ジョブは再実行されない。

バッチジョブは、以下の 3 種類に分類されます。

表 5-2 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	「5.5.1 通常ジョブ投入」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	「5.5.2 ステップジョブ投入」参照
アレイジョブ	1 度のジョブ投入で、複数の同じ通常ジョブを投入し実行するジョブ	「5.5.3 アレイジョブ投入」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 5-3 ジョブ操作コマンド一覧

機能	コマンド名
ジョブ投入	sbatch/srun
ジョブ参照	squeue/sacct
ジョブ削除	scancel

5.2 ジョブ実行リソース

ジョブを投入する際は、ジョブを実行するハードウェアを指定する「パーティション」を選択します。

5.2.1 パーティション

パーティションはジョブを実行するハードウェアを指定するオプションです。

パーティションにおける課題ごとの設定

各課題は、以下の計算資源を同時に使用することができます。

表 5-4 各サブシステムにおける課題ごとの設定

ジョブ実行先：サブシステム		実行コア数	投入ジョブ本数
超並列演算システム	(MPC)	7168	5000
大容量メモリ演算サーバ	(LMC)	192	100
GPU サーバ	(GPU)	448	100

表 5-5 パーティションごとの設定

パーティション	ノード数	最大 経過時間	最大 コア数	最大 ノード数	ノードあたりの メモリ量
mpc	312	24h	7168	64	112GiB
mpc_l	156	72h	896	8	112GiB
lmc	2	24h	192	2	2880GiB
gpu	3	72h	112	1	448GiB
gpu_i	1	24h	112	1	448GiB

*デフォルト経過時間は 1h です。

5.3 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、以下の 3 つのオプションを指定します。

- 基本オプション
- ジョブ資源オプション

5.3.1 基本オプション

ジョブ投入時に指定する基本オプションは以下のとおりです。

表 5-6 ジョブ投入基本オプション

オプション	説明
--account <project ID>	ジョブ実行に利用する課題番号（project ID）を指定 (複数課題を持つユーザーは指定必須)
--mail-user	メールの送信先を指定
--mail-type	メール通知を指定
BEGIN	ジョブ開始時にメール通知を指定
END	ジョブ終了時にメール通知を指定
REQUEUE	ジョブ再実行時にメール通知を指定
--job-name <name>	ジョブ名を指定
--output <filename>	標準出力を指定されたファイルに出力 （-e 指定無しの場合、標準エラー出力も同じファイルに出力）
--error <filename>	標準エラー出力を指定されたファイルに出力
--requeue	障害発生時ジョブを再実行することを指定 (デフォルト: --no-requeue)
--dependency	ステップジョブとしてジョブを投入
afterany:<jobid>	指定したジョブ(jobid)終了後に投入
afterok:<jobid>	指定したジョブ(jobid)が正常終了後に投入
afternotok:<jobid>	指定したジョブ(jobid)が異常終了後に投入
after:<jobid[+minute]>	指定したジョブ(jobid)開始後時間の経過で投入
singleton	同一ユーザー・同一ジョブ名のジョブを逐次実行
--array <start-end>	アレイジョブとしてジョブを投入
--export=<env>	ジョブ投入時の環境変数をジョブ実行環境に継承
--begin=<time>	ジョブ開始時刻を指定

5.3.2 ジョブ資源オプション

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。

表 5-7 ジョブ資源オプション(共通)

オプション	説明
--partition <partition_name>	キュー（パーティション）を指定
--nodes <num>	ノード数を指定
--ntasks <num>	プロセス数を指定
--ntasks-per-node <num>	ノードあたりのプロセス数を指定
--cpus-per-task <num>	プロセスあたり確保する論理 CPU コア数を指定
--ntasks-per-core=<num>	CPU コアあたりのプロセス数を指定
--mem=<num[M|G|T]>	ノードあたりのメモリ量を指定
--mem-per-cpu=<num[K|M|G|T]>	CPU コアあたりのメモリ量を指定 超並列演算システム(MPC)の最大: 112G 大容量メモリ演算サーバ(LMC)の最大: 2880G
--cpus-per-gpu=<num>	GPU あたりの CPU コア数を指定
--mem-per-gpu=<num[K|M|G|T]>	GPU あたりのメモリ量を指定 GPU サーバ(GPU)の最大: 448G
--gpus=<num>	GPU の使用枚数を指定 ※GPU あたり CPU:28 コア、メモリ:112G
--time	経過時間を指定
<mm>	分で指定
<mm:ss>	分、秒で指定
<hh:mm:ss>	時間、分、秒で指定
<days-hh>	日数、時間で指定
<days-hh:mm>	日数、時間、分で指定
<days-hh:mm:ss>	日数、時間、分、秒で指定

メモリ量を指定する場合、以下の単位が使用可能です。

表 5-8 メモリ量で使用可能な単位

単位	説明
KiB	キビバイト
MiB	メビバイト
GiB	ギビバイト
TiB	ティビバイト

コアあたりのメモリ割当量のデフォルトは、以下のとおりです。

表 5-9 コアあたりのメモリ割当量のデフォルト

システム	コアあたりのメモリ割当量
超並列演算システム(MPC)	1G
大容量メモリ演算サーバ(LMC)	30G
GPU サーバ(GPU)	4G

コアあたりのメモリ割当量より多いメモリ量を要求した場合、要求メモリ量に応じて演算時間が計算されるので注意してください。HBW2 システムより CPU コアあたりのメモリ量は少なくなっています。

生成するプロセス数やスレッド数は MPI オプションや環境変数 OMP_NUM_THREADS を使用して明に指定するようにしてください。省略すると意図しないプロセス数、スレッド数でプログラムが実行される可能性があり、性能劣化を招く恐れがあります。

5.4 インタラクティブジョブ実行

インタラクティブジョブを実行するためには、*srun* コマンドを実行します。ジョブ管理システムがインタラクティブジョブに対して計算資源を割り当てることでジョブの実行が開始されます。

計算ノードにログインするインタラクティブジョブの投入は、ジョブ投入オプションは引数として以下の通り指定します。

```
srun --partition パーティション名 --account projectname --pty $SHELL
```

ログインノードではプロセス数制限があるため、多数のプロセス等のリソースが必要なコンパイルを実行する場合は、インタラクティブジョブを利用されることを推奨します。

例1) 超並列演算システム(MPC)へログインしプログラムを実行する例

```
[username@hokusai1 ~]$ srun --account RB999999 --pty $SHELL
[username@bwmpc001 ~]$ ./a.out
Hello world
[username@bwmpc001 ~]$ exit
exit
[username@hokusai1 ~]$
```

フロントエンドサーバから計算ノードに直接プログラムを実行する場合、ジョブ投入オプションは引数として以下の通り指定します。ジョブ投入オプションは実行するプログラムに合わせ、「5.3 ジョブ投入オプション」を参照し指定してください。

```
srun --partition=mpc --account projectname [option] 実行プログラム
```

例2) 超並列演算システム(MPC)へ直接プログラムを実行する例

```
[username@hokusai1 ~]$ module load intel
[username@hokusai1 ~]$ srun --partition=mpc --account RB999999 --nodes=1 --
ntasks=2 ./a.out

Hello world
[username@hokusai1 ~]$
```

srun コマンド実行前に実行するプログラムに合わせ、環境設定を実施してください。
なお、GPU サーバでインタラクティブジョブを実行する際は、
partition=gpu_i を指定してください。

5.5 バッチジョブ投入

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、利用するパーティション、経過時間、ノード数などの資源を記載したオプションを記述した上で、実行するコマンド列を記載します。ユーザーはジョブスクリプトを *sbatch* コマンドで投入します。投入されたジョブはジョブ管理システムにより資源の空き状況やジョブ間の優先度などを元に自動で実行開始されます。

5.5.1 通常ジョブ投入

通常ジョブを投入する場合、*sbatch* コマンドの引数にバッチジョブとして実行するジョブスクリプトを指定します。

```
sbatch [option] [job-script]
```

- ジョブスクリプトを指定しない場合、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、ジョブスクリプト内もしくは標準入力にディレクティブを用いて指定することが可能です。
- 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) 通常ジョブ投入例

```
[username@hokusai1 ~]$ sbatch run.sh
Submitted batch job  1234.
```

5.5.2 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。

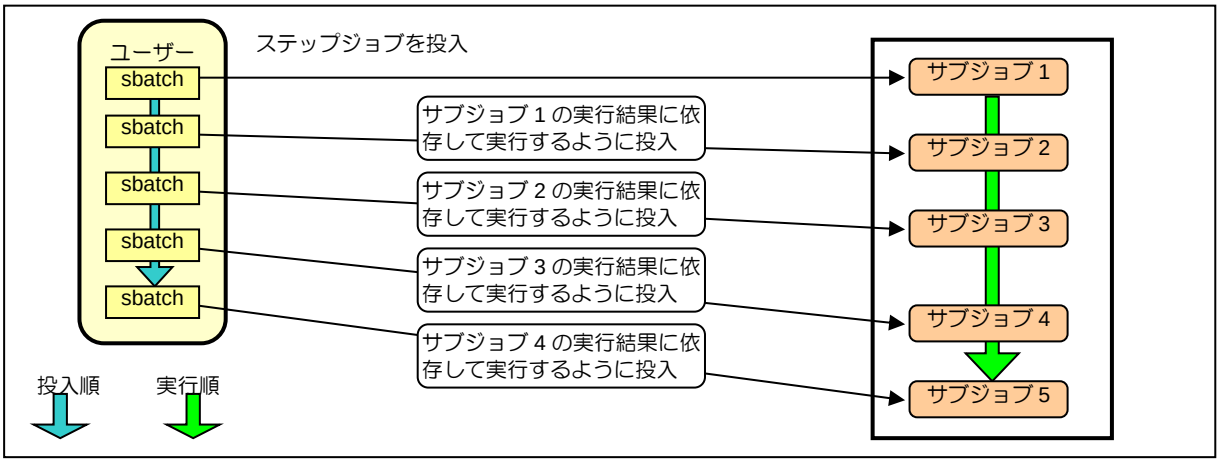

図 5-1 ステップジョブイメージ

ステップジョブの投入形式は以下のとおりです。

```
sbatch --dependency=[option] [job-script]
```

例 1-1) 指定したジョブ(jobid)が終了後にジョブを投入

```
[username@hokusai1 ~]$ sbatch step1.sh
Submitted batch job 1234
[username@hokusai1 ~]$ sbatch --dependency=afterany:1234 step2.sh
Submitted batch job 1235
[username@hokusai1 ~]$ sbatch --dependency=afterany:1235 step3.sh
Submitted batch job 1236
```

例 1-2) 同一ユーザー・同一ジョブ名のジョブを逐次実行(同時に 1 本のみ実行可能)

```
[username@hokusai1 ~]$ sbatch step1.sh
Submitted batch job 1234
[username@hokusai1 ~]$ sbatch --dependency=singleton step2.sh
Submitted batch job 1235
```

例 1-3) 指定したジョブ(jobid)が正常終了後にジョブを投入

```
[username@hokusai1 ~]$ sbatch step1.sh
Submitted batch job 1234
[username@hokusai1 ~]$ sbatch --dependency=afterok:1234 step2.sh
Submitted batch job 1235
```

例 1-4) 指定したジョブ(jobid)が異常終了後にジョブを投入

```
[username@hokusai1 ~]$ sbatch step1.sh
Submitted batch job 1234
[username@hokusai1 ~]$ sbatch --dependency=afternotok:1234 step2.sh
Submitted batch job 1235
```

例 2) 複数の条件を指定してステップジョブを投入

```
[username@hokusai1 ~]$ sbatch step1.sh
Submitted batch job 1234
[username@hokusai1 ~]$ sbatch --dependency=afterok:1234 step2.sh
Submitted batch job 1235
[username@hokusai1  ~]$  sbatch  --dependency=afterok:1234,afternotok:1236
step3.sh
Submitted batch job 1236
[username@hokusai1 ~]$ sbatch --dependency=afterany:1234:1235,after:1236+1
step4.sh
Submitted batch job 1237
```

表 5-10 ステップジョブオプション

オプション	説明
afterany:<jobid>	指定したジョブが終了後に投入
afterok:<jobid>	指定したジョブが正常終了後(終了ステータスが 0)に投入
afternotok:<jobid>	指定したジョブが異常終了後(終了ステータスが 0 以外)に投入
after:<jobid[+minute]>	指定したジョブの開始後、指定した時間の経過後に投入
singleton	同一ユーザー・同一ジョブ名のジョブを逐次実行

5.5.3 アレイジョブ投入

アレイジョブは、複数の同じバッチジョブを同時に投入し、実行するジョブです。

例えば、ジョブのパラメータを変えて、それぞれの実行結果を確認したい場合、通常のバッチジョブであれば、ジョブを 1 つ 1 つ投入する必要がありますが、アレイジョブを使えば、1 度に複数パターンの投入ができます。

アレイジョブの投入形式は以下のとおりです。

```
sbatch --array 0-2:1 jobscript
```

アレイジョブでは、サブジョブごとにジョブの入出力ファイルを変えられるようにジョブスクリプトを作成する必要があります。このために、サブジョブごとに設定されるアレイ番号を使います。アレイ番号は環境変数 SLURM_ARRAY_TASK_ID で参照することが可能です。

各アレイジョブには、ジョブ ID_タスク ID という形式のジョブ ID に加え、通常のジョブと同様のジョブ ID も付与されます。ただし、通常のジョブ ID はジョブの実行が開始された順に付与されるため、タスク ID の順番とジョブ ID の順番が一致しないことがあります。

5.5.4 バッチジョブの標準出力と標準エラー出力

バッチジョブの標準出力ファイルと標準エラー出力ファイルは、ジョブ投入ディレクトリもしくは、ジョブ投入時に指定されたファイルに出力されます。

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。ジョブ投入時に、標準出力ファイル、標準エラー出力ファイルを指定しなかった場合は、以下のファイルに出力されます。

slurm-XXXXX.out --- 標準出力・エラーファイル
(XXXXX はジョブ投入時に割り当てられたジョブ ID)

5.5.5 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* コマンドにてジョブスクリプトを作成します。

(1) 先頭行は "#!" に続けて、シェルのパスを指定してください。
[記述例]

```
#!/bin/bash
```

ログインシェルが bash 以外の場合、ジョブスクリプト内で module コマンドを実行する場合は、「#!/bin/sh -l」を指定する必要があります。

(2) 2 行目以降にジョブ投入オプションをディレクティブ"**#SBATCH**"を用いて指定します。
[記述例]

```
#SBATCH --nodes=1              ノード数を指定
#SBATCH --time=1:00:00         経過時間を指定
```

(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、プログラム実行を指定します。Slurm では srun コマンドで実行プログラム(task)を指定することにより、ジョブ投入オプションに従ってノードやプロセス等のリソース割当が行われます。
[記述例]

```
export OMP_NUM_THREADS=20      環境変数を設定
srun ./a.out                   プログラムを実行
```

5.5.6 MPI プログラムの実行

srun オプション

ジョブスクリプトファイルに記載しているジョブ投入オプションは引き継がれます。そのため、srun コマンドも sbatch 同様のオプションがありますが、MPI プログラム実行時にオプションを記載する必要はありません。

表 5-11 srun オプション

オプション		説明
--cpu-bind=<type>		タスクを CPU にバインドします
	verbose	実行する前にバインディングを詳細に報告する
	rank	タスクランクごとに自動的にバインドする
	rank_ldom	ランクごとに NUMA ローカリティドメインにバインドする ※ノード内の全コア使用時のみ指定可能

5.6 バッチジョブスクリプト例

5.6.1 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向けジョブスクリプト

5.6.1.1 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向け単一ノード逐次ジョブ用スクリプト

以下のジョブ実行を想定した記述方法を説明します。

・パーティション : mpc
・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 : 60 分
・課題番号(project ID) : RB999999

```
[username@hokusai1 ~]$ vi mpc-seq.sh
#!/bin/sh
#------ slurm option --------#
#SBATCH --partition=mpc
#SBATCH --account=RB999999
#SBATCH --nodes=1
#SBATCH --time=1:00:00
#------- Program execution -------#
module load intel
srun ./a.out
```

5.6.1.2 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)単一ノードスレッド並列スクリプト

以下のジョブ実行を想定した記述方法を説明します。

・パーティション　　: mpc
・ノード数　　: 1 ノード
・プロセス数(スレッド数)　　: 1 プロセス(10 スレッド)
・経過時間　　: 60 分
・課題番号(project ID)　　: RB999999

```
[username@hokusai1 ~]$ vi mpc-para.sh
#!/bin/sh
#------ slurm option --------#
#SBATCH --partition=mpc
#SBATCH --account=RB999999
#SBATCH --nodes=1
#SBATCH --cpus-per-task=10
#SBATCH --time=1:00:00
#------- Program execution -------#
module load intel

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun --cpus-per-task=${SLURM_CPUS_PER_TASK} ./a.out
```

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)でスレッド並列ジョブを実行する場合、環境変数 OMP_NUM_THREADS の指定は必須です。指定するメモリ量により、確保するコア数が変わる場合があり、省略すると意図しないスレッド数で実行される可能性があります。

OMP_NUM_THREADS　:スレッド数(環境変数${SLURM_CPUS_PER_TASK}を指定、または cpus-per-task で指定した値を記載する)

5.6.1.3 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向け単一ノード 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。

・パーティション : mpc
・ノード数 : 1
・プロセス数(スレッド数) : 20 プロセス(1 スレッド)
・経過時間 : 60 分
・課題番号(project ID) : RB999999

```
[username@hokusai1 ~]$ vi mpc-single-mpi.sh
#!/bin/bash
#------ slurm option --------#
#SBATCH --partition=mpc
#SBATCH --account=RB999999
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --time=1:00:00
#------- Program execution -------#
module load intel

srun ./a.out
```

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)で MPI 並列ジョブを実行する場合、指定するメモリ量により、確保するコア数が変わる場合があり、省略すると意図しないプロセス数で実行される可能性があります。

5.6.1.4 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向け単一ノードハイブリッド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。

・パーティション : mpc
・ノード数 : 1
・プロセス数(スレッド数) : 2 プロセス(10 スレッド)
・コア数 : 20 コア(2 x 10)
・経過時間 : 60 分
・課題番号(project ID) : RB999999

```
[username@hokusai1 ~]$ vi mpc-single-hybrid.sh
#!/bin/bash
#------ slurm option --------#
#SBATCH --partition=mpc
#SBATCH --account=RB999999
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10
#SBATCH --time=1:00:00
#------- Program execution -------#
module load intel

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun --cpus-per-task=${SLURM_CPUS_PER_TASK} ./a.out
```

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)でハイブリッド並列ジョブを実行する場合、環境変数 OMP_NUM_THREADS は指定必須です。指定するメモリ量により、確保するコア数が変わる場合があり、省略すると意図しないプロセス数/スレッド数で実行される可能性があります。

OMP_NUM_THREADS : スレッド数(環境変数${SLURM_CPUS_PER_TASK} を指定、または cpus-per-task で指定した値を記載する)

5.6.1.5 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向け複数ノード 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。

・パーティション : mpc
・ノード数 : 2
・プロセス数(スレッド数) : 80 プロセス(1 スレッド)
・ノードあたりのプロセス数 : 40 プロセス
・経過時間 : 90 分
・課題番号(project ID) : RB999999

```
[username@hokusai1 ~]$ vi mpc-multi-mpi.sh
#!/bin/sh
#------ slurm option --------#
#SBATCH --partition=mpc
#SBATCH --account=RB999999
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=1:30:00
#------- Program execution -------#
module load intel

srun ./a.out
```

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)で MPI 並列ジョブを実行する場合、指定するメモリ量により、確保するコア数が変わる場合があり、省略すると意図しないプロセス数で実行される可能性があります。

5.6.1.6 超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)向け複数ノードハイブリッド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。

・パーティション : mpc
・ノード数 : 2
・プロセス数(スレッド数) : 4 プロセス(20 スレッド)
・ノードあたりのプロセス数 : 2 プロセス
・経過時間 : 1 時間 30 分
・課題番号(project ID) : RB999999

```
[username@hokusai1 ~]$ vi mpc-multi-hybrid.sh
#!/bin/sh
#------ slurm option --------#
#SBATCH --partition=mpc
#SBATCH --account=RB999999
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=5
#SBATCH --time=1:30:00
#------- Program execution -------#
module load intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun --cpus-per-task=${SLURM_CPUS_PER_TASK} ./a.out
```

超並列演算システム(MPC)/大容量メモリ演算サーバ(LMC)でハイブリッド並列ジョブを実行する場合、環境変数 OMP_NUM_THREADS は指定必須です。指定するメモリ量により、確保するコア数が変わる場合があり、省略すると意図しないプロセス数/スレッド数で実行される可能性があります。

OMP_NUM_THREADS :スレッド数(環境変数${SLURM_CPUS_PER_TASK} を指定、または cpus-per-task で指定した値を記載する)

5.6.1.7 GPU サーバ(GPU)向け単一ノード逐次ジョブ用スクリプト

以下のジョブ実行を想定した記述方法を説明します。

・パーティション : gpu
・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 : 60 分
・課題番号(project ID) : RB999999

```
[username@hokusai1 ~]$ vi gpu-seq.sh
#!/bin/bash
#------ slurm option --------#
#SBATCH --partition=gpu
#SBATCH --gpus=1
#SBATCH --account=RB999999
#SBATCH --nodes 1
#SBATCH --time 01:00:00

#------- Program execution -------#
module load intel
srun ./a.out
```

5.7 ジョブ状態表示

投入したジョブ状態やリソース情報を確認する場合、*squeue* コマンドを使用します。

```
squeue [option] [JOBID [JOBID …]]
```

表 5-12 squeue コマンドオプション一覧

オプション	説明
なし	実行待ち、実行中のジョブ情報を表示
--long	実行待ち、実行中の詳細なジョブ情報を表示
--account <account>	指定した課題のジョブ情報を表示
--jobs <jobid>	指定したジョブ ID のジョブ情報を表示
--partition <partitionname>	指定したパーティションのジョブ情報を表示
--states <states>	指定したジョブステータスのジョブ情報を表示
--nodelist <nodename>	指定したノードに割り当てられたジョブ情報を表示
--iterate <seconds>	指定した間隔（秒単位）でジョブ情報を表示

5.7.1 ジョブ状態表示

squeue 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。

```
[username@hokusai1 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              1234       mpc   job.sh username  R       0:04      1 bwmpc185
```

表 5-13 ジョブ情報の表示項目

項目	説明
JOBID	ジョブ ID
PARTITION	ジョブのキュー名
NAME	ジョブ名
USER	ユーザー名
ST	ジョブの現在の状態(表 5-14 ジョブの状態一覧参照)
TIME	ジョブの実行時間
NODES	ジョブ実行に使用するノード数
NODELIST	ジョブが実行されるホスト名のリスト

表 5-14 ジョブの状態一覧

状態	説明
PD	ジョブ実行待ち
CF	ジョブ実行に必要な資源を獲得中
R	ジョブ実行中
CG	ジョブの終了待ち
CD	ジョブ終了
CA	ジョブキャンセルによる終了
F	ゼロ以外の終了コードまたはその他の障害状態で終了
NF	割り当てられたノードのいずれかの故障による終了
TO	実行時間制限による終了
RJT	投入が受付けられなかった状態

5.7.2 詳細ジョブ情報の表示(sacct --format オプション)

sacct コマンドの--format オプションを指定すると、指定した項目のジョブ情報を表示します。表示する項目は--helpformat オプションで確認することができます。

```
[username@hokusai1                ~]$              sacct              -X               --
format=JobID,Partition,Account,Submit,Start,End,CPUTimeRAW,AllocTres%40,State -j 1440
JobID          Partition     Account               Submit               Start                 End
CPUTimeRAW                                  AllocTRES       State
------------ ---------- ---------- ------------------- ------------------- ------------------- ---
------- ---------------------------------------- ----------
1234                   mpc       RB999999 2023-11-16T13:04:05 2023-11-16T13:04:18 2023-11-16T13:09:19
301          billing=8,cpu=1,mem=8800M,node=1  COMPLETED
```

表 5-15 sacct コマンドオプション一覧

オプション	説明
-X	ジョブ割り当て自体に関連する統計のみを表示
--accounts	指定した課題の情報を表示
--jobs	指定したジョブ ID の情報を表示
--helpformat	--fomat オプションで表示する項目の確認
--fomat	指定した情報を表示
--starttime	指定した日時(YYYY-MM-DD[THH:MM[:SS]])からのジョブ情報を表示

表 5-16 詳細ジョブ情報の表示項目

項目		説明
Account		課題番号(project ID)
Partition		パーティション
Submit		ジョブの投入日時
Start		ジョブの開始時間
End		ジョブ終了時間
Elapsed		実行経過時間
CPUTimeRAW		経過時間(CPU 秒)
AllocTres		割り当てられたリソース
	billing	演算時間計算に用いられるコア数(実際に使用されたコア数)
	cpu	割り当てコア数
	mem	割り当てメモリ量
	node	割り当てられたノード数
State		ジョブの状態

5.7.3 終了ジョブ情報の表示(sacct コマンド)

sacct コマンドを実行すると、投入済みジョブに加え、既に実行が終了したジョブも表示されます。--starttime オプションを指定すると、コマンド実行日以前の情報を表示することが可能です。

```
[username@hokusai1 ~]$ sacct
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
12345          test_job        mpc   username          4  COMPLETED      0:0
```

表 5-17 終了ジョブ情報の表示項目

項目	説明
JobID	ジョブ ID
JobName	ジョブ名
Partition	ジョブ実行したパーティション
Account	課題番号(project ID)
AllocCPUS	割り当て CPU 数
State	ジョブの終了状態
ExitCode	ジョブの終了コード

5.7.4 パーティションの表示

sinfo コマンドを実行すると、ユーザーが利用可能なパーティションを表示します。

```
[username@hokusai1 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT   NODES(A/I/O/T) NODELIST
mpc*         up 1-00:00:00      0/312/0/312 bwmpc[001-312]
mpc_l        up 3-00:00:00      0/156/0/156 bwmpc[001-156]
lmc          up 1-00:00:00          0/2/0/2 bwlmc[1-2]
```

表 5-18 パーティションの表示項目

項目	説明
PARTITION	キュー名（パーティション名）
AVAIL	キューの状態
TIMELIMIT	最大実行時間
NODES	ノードの状態を表示します。 A:allocated I:idle O:other T:total
NODELIST	キュー（パーティション）に割り当てられたノード

5.7.5 ジョブ投入時に指定可能なジョブ資源の上限表示(sinfo コマンド)

sinfo コマンドの「--format」オプションを指定すると、パーティション毎に利用可能なノード数、ノード当たりの CPU 数、最大経過時間を表示します。

```
sinfo --format="%.6P %.5a %.11l %.10s %.6D %.18B %.8m"
```

以下のように表示された場合、超並列演算システム(MPC)の mpc パーティションでは 1 から 12 ノードが利用でき、ジョブは最大 24 時間(1 日)、1 ノード当たりの CPU 数は 112 コアで実行できることを示しています。

表示された上限を超えた値を指定したジョブも投入することが可能ですが、ジョブ実行はされません。そのため、*scancel* コマンドを実行しジョブを削除してください。

```
PARTIT AVAIL   TIMELIMIT   JOB_SIZE  NODES  MAX_CPUS_PER_NODE    MEMORY
  mpc*    up  1-00:00:00       1-12    260                112    128300
 mpc_l    up  3-00:00:00        1-6    156                112    128300
   lmc    up  1-00:00:00          1      2                 96   3095859
```

表 5-19 ジョブ資源の上限表示項目

項目	説明
PARTIT	パーティション
AVAIL	パーティションの状態表示
TIMELIMIT	最大経過時間
JOB_SIZE	割り当てノード数
NODES	ノード数
MAX_CPUS_PER_NODE	ノードあたりの最大 CPU 数
MEMORY	ノード当たりのメモリ量(MB)

5.8 ジョブキャンセル

投入済みのジョブをキャンセルする場合、*scancel* コマンドを実行します。

```
scancel JOBID [JOBID…]
```

キャンセルするジョブのジョブ ID を scancel の引数に指定します。

```
[username@hokusai1 ~]$ scancel 12345
```

5.9 環境変数

5.9.1 スレッド並列、MPI 実行に関する環境変数

スレッド並列および MPI プログラム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 5-20 実行時環境変数

環境変数	説明
OMP_NUM_THREADS	OpenMP もしくは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 OMP_NUM_THREADS にスレッド数を指定します。
KMP_AFFINITY	スレッドのコアバインドを制御します。
I_MPI_PIN_DOMAIN	MPI プロセスのコアバインドを制御します。

5.9.2 ジョブに関する環境変数

ジョブ内では、ジョブ管理システムにより、以下の環境変数が設定されます。

表 5-21 ジョブ内で参照可能な環境変数

環境変数	説明
SLURM_JOB_ID	ジョブ ID
SLURM_JOB_NAME	ジョブ名
SLURM_SUBMIT_DIR	*sbatch* コマンド実行時のカレントディレクトリ
SLURM_ARRAY_JOB_ID	アレイジョブ ID(アレイジョブのみ設定)
SLURM_ARRAY_TASK_ID	アレイジョブのタスク番号(アレイジョブのみ設定)

また、ジョブ投入時にオプションで指定する資源量の一部は、ジョブ内で以下の環境変数に設定されます。ジョブ投入時に指定されなかった項目は、自動で適切な値が設定されます。

表 5-22 ジョブ内で参照可能な環境変数

環境変数	説明
SBATCH_PARTITION	パーティション名 (-p の値)
SLURM_JOB_NUM_NODES	ノード数 (-N の値)
SLURM_JOB_CPUS_PER_NODE	1 ノードあたりのコア数 カンマ区切りでノード毎のコア数
SLURM_MEM_PER_NODE	1 ノードあたりのメモリ量 --mem の値
SLURM_MEM_PER_CPU	1 コアあたりのメモリ量 --mem-per-cpu の値

ログインノード上でジョブ投入前に設定した環境変数は、指示がない限り、ジョブ内には引き継がれません。ジョブ内に引き継ぐためには、*sbatch* コマンドの--export オプションを指定します。

6. Singularity コンテナ

6.1 Singularity イメージの取得

Singularity イメージは利用者自身で用意する必要があります。
dockerhub や Singularity ライブラリなどから取得できます。
Dockerhub：https://hub.docker.com/
Singularity ライブラリ：https://cloud.sylabs.io/library
以下は dokcerhub から ubuntu の sif イメージを取得します。

```
[username@hokusai1 ~]$ singularity pull docker://ubuntu:24.04
[username@hokusai1 ~]$ ls -l ubuntu_24.04.sif
```

6.2 Singularity の"Command Line Interface(CLI)"について

Command Line Interface には Run, Exec, Shell が指定できます。各 CLI については以下を参照ください。

・Run

コンテナ内でユーザ定義のデフォルトコマンドを実行することができます。sif ファイルの作成時に指定できる %runscript が実行されます。Docker や Podman でいう CMD に近いものです。%runscript が指定されていない場合は Shell が立ち上がります。

```
[username@bwmpcxxx ~]$ singularity run ubuntu_24.04.sif << EOF
hostname
date
EOF
bwgpu4
Thu Jun 27 11:59:12 JST 2024
[username@bwmpcxxx ~]$
```

・Exec

コンテナ内でコマンドを実行することができます。Docker や Podman のようにコンテナを作成してから exec するのではなく、sif ファイルに直接コマンドを投げることが可能です。

```
[username@bwmpcxxx ~]$ singularity exec lolcow.sif cowsay "Fresh from the
library!"
 _________________________
< Fresh from the library! >
 -------------------------
        ¥   ^__^
         ¥  (oo)¥_______
            (__)¥       )¥/¥
                ||----w |
                ||     ||
[username@bwmpcxxx ~]$
```

・Shell

コンテナ内で Shell を実行する. %runscript などは実行されず Shell が起動する。デフォルトの Shell は Bash が起動するが, --shell で Shell を指定することも可能。

```
[username@bwmpcxxx ~]$ singularity shell lolcow.sif
Singularity>
```

6.3 Ubuntu の sif イメージを利用したインタラクティブジョブ実行

```
[username@hokusai1 ~]$ srun --partition=mpc --account=RBxxxxxx --nodes=1 --
ntasks=1 --pty bash
[username@bwmpcxxx ~]$ singularity run ubuntu_24.04.sif
Singularity> cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04 LTS"
NAME="Ubuntu"
(snip)

Singularity> exit
```

6.4 Ubuntu の sif イメージを利用したバッチジョブ実行

バッチジョブで Ubuntu sif イメージを MPC 計算ノードで実行。

```
[username@hokusai1 ~]$ cat go.sh
#!/bin/sh

#SBATCH --partition=mpc
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH -J JOB
#SBATCH -o %x_%j.log
#SBATCH -e %x_%j.err
#SBATCH --time=1:00:00
#SBATCH --account=RBxxxxxxx

singularity run ubuntu_24.04.sif << EOS
hostname
echo
cat /etc/os-release

EOS

[username@hokusai1 ~]$ sbatch go.sh
```

6.5 CPU 版(MPC) Singularity Ubuntu 環境構築

Singularity イメージは利用者の環境でも構築できますが、ログインノードでも作成できます。

6.5.1 Ubuntu 構築設定ファイル作成

```
[username@hokusai1 ~]$ cat ubuntu.def
BootStrap: docker
From: ubuntu:24.04

%post
    apt-get update
    apt-get -y upgrade
    apt-get -y install build-essential vim
```

6.5.2 Ubuntu 環境イメージ作成

```
[username@hokusai1 ~]$ singularity build --fakeroot ubuntu.img ubuntu.def
INFO:    Starting build...
Getting image source signatures
Copying blob d4c3c94e5e10 done
Copying config 2abc4dfd83 done
Writing manifest to image destination
Storing signatures
2024/05/17 11:22:20  info unpack layer:
sha256:d4c3c94e5e10ed15503bda7e145a3652ee935c0b2e9de9b5c98df7ec0a0cd925
INFO:    Running post scriptlet
+ apt-get update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
(snip)
Processing triggers for libc-bin (2.31-0ubuntu9.15) ...
INFO:    Creating SIF file...
INFO:    Build complete: ubuntu.img

[username@hokusai1 ~]$ ls -l ubuntu.img
-rwxr-x--- 1 username username 160432128 May 17 11:26 ubuntu.img
```

6.6 GPU ノード向け Singularity コンテナの取得

GPU ノードにインタラクティブジョブでログインし、最新の PyTorch コンテナ SIF ファイルを取得します。
NGC や dokcerhub、Singularity のライブラリなどから SIF ファイルを取得できます。
NGC カタログ:
https://catalog.ngc.nvidia.com/?filters=&orderBy=weightPopularDESC&query=&page=&pageSize=

GPU ノードのコンテナ向けバッチジョブでは、「--gpus」で GPU の使用枚数を指定します。
「--gpus=1」は 1GPU、「--gpus=2」 は 2GPU、「--gpus=4」は 4GPU を使用できます。

「--gpus=1」を指定した場合、CPU は 28 コア確保されます。

```
NGC から取得する場合は docker://nvcr.io/{イメージ名}を指定
[username@hokusai1 ~]$ srun -p gpu_i --account=RBxxxxxx --nodes=1 --gpus=1 --
time=5:00:00 --pty bash
[username@bwgpu4 ~]$ singularity pull pytorch:24.01-py3.sif
docker://nvcr.io/nvidia/pytorch:24.01-py3
```

6.7 インタラクティブジョブ Singularity コンテナ内で Shell 起動

singularity コマンド実行時「--nv」オプションを指定すると GPU 環境が設定されます。

```
[username@hokusai1 ~]$ srun -p gpu_i --account=RBxxxxxx --nodes=1 --gpus=1 --time=5:00:00
--pty bash
[username@bwgpu4 ~]$ singularity run --nv pytorch:24.01-py3.sif
INFO:    Converting SIF file to temporary sandbox...
(snip)
NOTE: CUDA Forward Compatibility mode ENABLED.
  Using CUDA 12.3 driver version 545.23.08 with kernel driver version 535.129.03.
  See https://docs.nvidia.com/deploy/cuda-compatibility/ for details.

Singularity> nvidia-smi
Thu Apr 18 16:26:02 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.129.03             Driver Version: 535.129.03   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA H100 80GB HBM3          On  | 00000000:1C:00.0 Off |                    0 |
| N/A   28C    P0              72W / 700W |      4MiB / 81559MiB |     0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
(snip)
Singularity>
```

6.8 インタラクティブジョブ PyTorch コンテナで CuPy を使用した実行例

```
[username@hokusai1 ~]$ srun -p gpu_i --account=RBxxxxxx --nodes=1 --gpus=1 --
time=5:00:00 --pty bash
$ singularity pull pytorch:24.01-py3.sif docker://nvcr.io/nvidia/pytorch:24.01-
py3
コンテナ内で Shell を起動する。
[username@bwgpu4 ~]$ singularity run --nv pytorch:24.01-py3.sif
(snip)
Singularity> pip freeze|grep cupy
cupy==13.1.0
Singularity> pip freeze|grep Cython
Cython==0.29.37

Singularity> cat cupy_ex.py
import time
import cupy as cp

start_time = time.time()

a = cp.random.rand(50000, 50000)
b = cp.random.rand(50000, 50000)

result = cp.dot(a, b)

end_time = time.time()

print(f"CuPy Time: {end_time - start_time} seconds")

Singularity> python cupy_ex.py
CuPy Time: 331.78111362457275 seconds
```

6.9 バッチジョブ PyTorch コンテナの GPU 実行例

```
[username@hokusai1 ~]$ cat go.sh
#!/bin/sh

#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --gpus=1
#SBATCH --ntasks-per-gpu=1
#SBATCH -J GPU
#SBATCH -o %x_%j.log
#SBATCH -e %x_%j.err
#SBATCH --time=1:00:00
#SBATCH --account=RBxxxxxxx

singularity run --nv pytorch:24.01-py3.sif << EOS
python cupy_ex.py
echo
cat /etc/os-release

EOS
[username@hokusai1 ~]$ sbatch go.sh
```