2021.4.12初版
2021.5.12第２版

神戸大学 計算科学教育センター 様

並列計算教育用計算機
利用者マニュアル

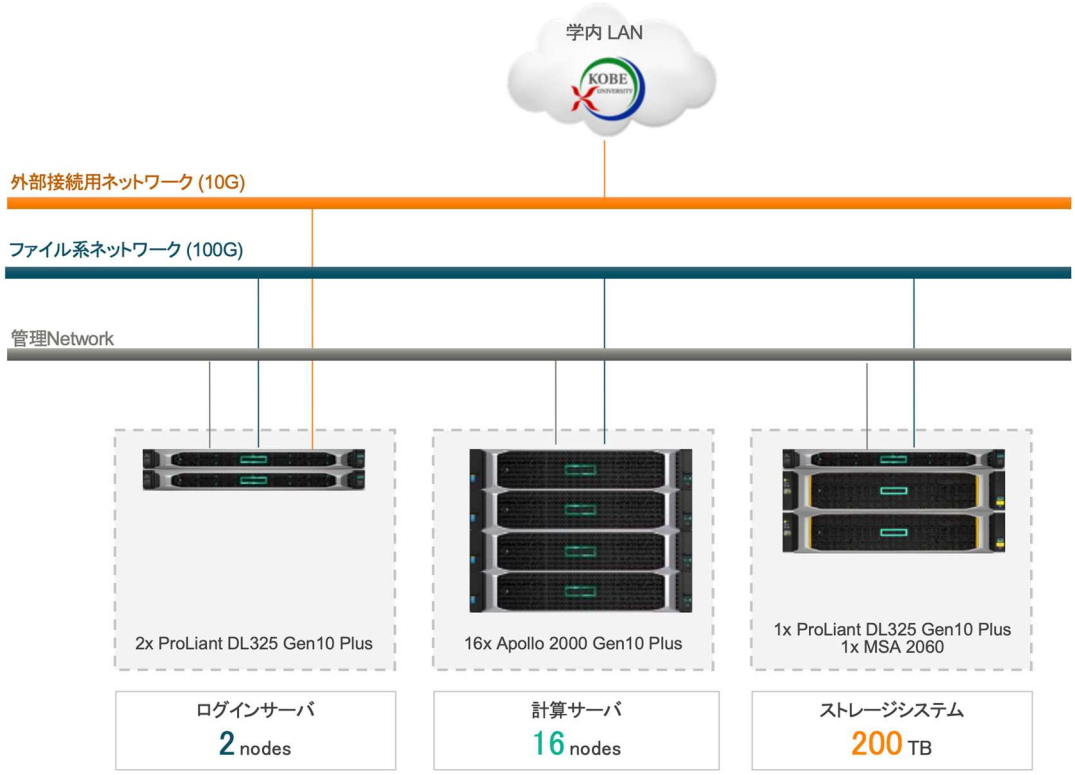
UNIADEX

目次

1. システム概要

1.1. 並列計算教育用計算機システム構成

システム構成図を示します。
ログインサーバ 2 ノードおよび計算サーバ 16 ノード、ストレージシステム 200TB より構成されるシステムです。
ログインサーバおよび計算サーバは AMD APYC CPU が搭載されています。
また、サーバ間を接続する相互接続網 は 100Gbps の InfiniBand です。
ログインサーバは 外部から 10Gbps ネットワーク で ログインできます。
ログインサーバおよび計算サーバのファイルシステムとして、容量 200TB の共有ストレージを有しています。



システム	ホスト名	ハードウェア	ソフトウェア
ログインサーバ	pifront1, pifront2	CPU: AMD EPYC 7542 1CPU (2.35 GHz 32C 155W) MEM: 256GiB (DDR4-3200 32GiB x 8 DIMM) InfiniBand: 1x IB HDR100 2p HCA 10G NW: 1x 10GbE 2p SFP+ NIC	【Operating System】 Red Hat Enterprise Linux 8.2 【開発環境】 Intel oneAPI Toolkit Cray Programming Environment 【ジョブスケジューラ】 Altair PBS Professional
計算サーバ	pinode01～16	CPU: AMD EPYC 7282 2CPU (2.8GHz 16C 120W) MEM: 256GiB (DDR4-3200 16GiB x 16 DIMM InfiniBand: 1x IB HDR100 2p HCA	

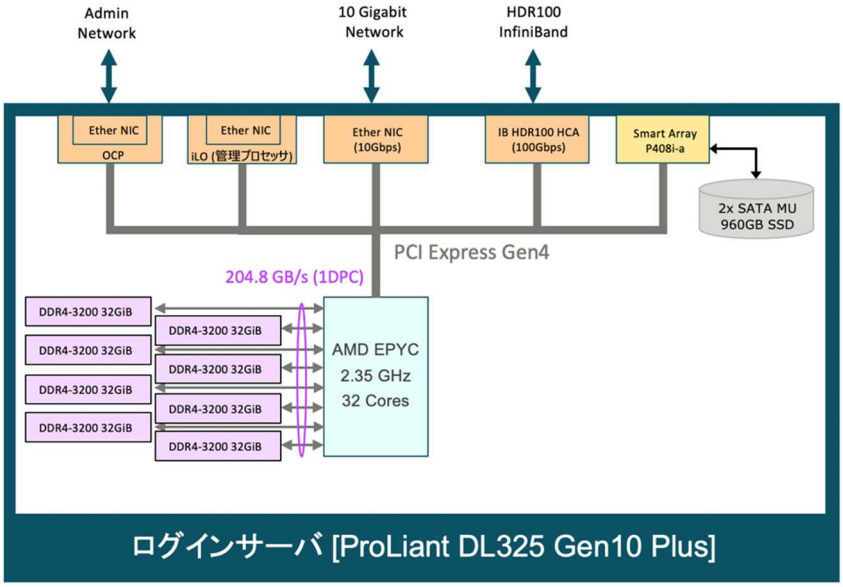
1.2. 各サブシステム構成

ログインサーバ、計算サーバおよびストレージシステムの構成概要を示します。

1.2.1. ログインサーバ

ログインサーバは 2 ノードあり、プログラム開発や並列ジョブの投入に利用します。

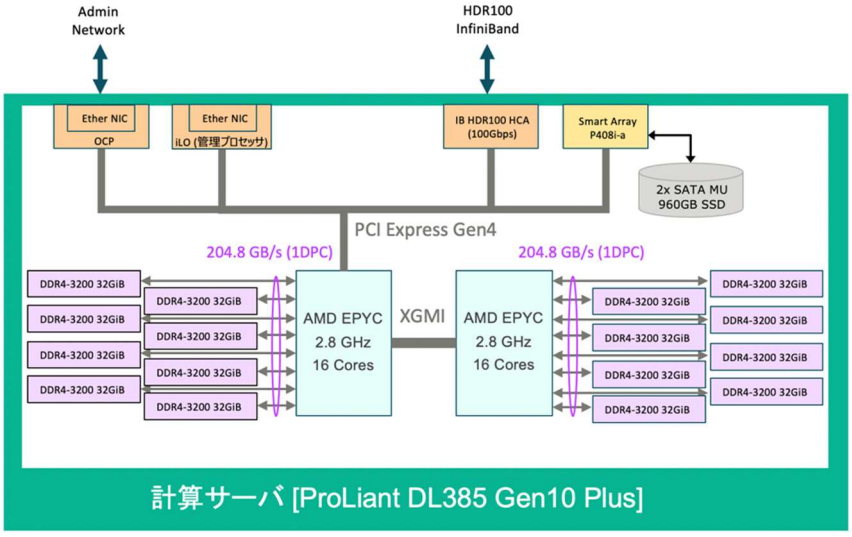
ブロックダイアグラムを以下に示します。



1.2.2. 計算サーバ

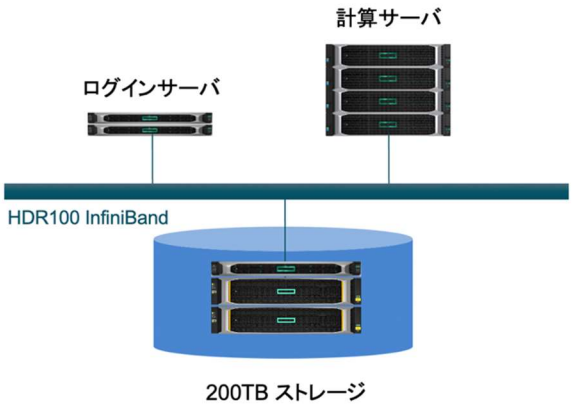
計算サーバは 16 ノードあり、分散メモリ並列計算に利用します。

ブロックダイアグラムを以下に示します。



1.2.3. ストレージシステム

ログインサーバ 2 台 と 計算サーバ 16 台は HDR100 InfiniBand 経由で 200TB のストレージを共有します。



各ユーザには ホーム領域が割り当てられています。

ディレクトリ	ファイルシステム	総容量	備考
/home	NFS	200TB	ホーム領域 ユーザのホームディレクトリの絶対パスは “/home/group/user”

2. システム接続方法

2.1. SSH ログイン

2.1.1. ログイン方法

ログインサーバは ssh によるログインが可能です。

ホスト名 (FQDN)	IP アドレス	サービス	認証方式	備考
pifront.eccse.kobe-u.ac.jp		ssh	公開鍵認証	pifront1 もしくは pifront2 に接続
pifront1.eccse.kobe-u.ac.jp	133.30.94.196	ssh	公開鍵認証	pifront1 に接続
pifront2.eccse.kobe-u.ac.jp	133.30.94.197	ssh	公開鍵認証	pifront2 に接続

PC や ワークステーションから ログインサーバのログインは以下の通りです。

```
$ ssh scuser@pifront.eccse.kobe-u.ac.jp
```

実行例

```
[wsuser@workstation ~]$ ssh scuser@pifront.eccse.kobe-u.ac.jp
Enter passphrase for key '/home/wsuser/.ssh/id_rsa':
[scuser@pifront1 ~]$
```

2.1.2. OpenSSH 設定

OpenSSH を使用している場合、以下のように設定することで、“ssh pifront” や “ssh pifront2” といったホスト名の指定のみでログイン可能です。

設定例

```
[wsuser@workstation ~]$ cat ~/.ssh/config
Host pifront
    HostName pifront.eccse.kobe-u.ac.jp
```

```
    Port 22
    User scuser
Host pifront1
    HostName pifront1.eccse.kobe-u.ac.jp
    Port 22
    User scuser
Host pifront2
    HostName pifront2.eccse.kobe-u.ac.jp
    Port 22
    User scuser
```

上記設定をした場合、“ssh ホスト名”で ログインが可能になります。

実行例

```
[wsuser@workstation ~]$ ssh pifront2
Enter passphrase for key '/home/wsuser/.ssh/id_rsa':
[scuser@pifront2 ~]$
```

2.1.3. ログアウト方法

ssh 接続のログアウトをする場合は “exit” もしくは “logout” コマンドを実行します。

実行例

```
[scuser@pifront2 ~]$ logout
Connection to 133.30.94.196 closed.
[wsuser@workstation ~]$
```

2.2. VNC による GUI 利用

ログインサーバの GUI を VNC (Virtual Networking Computing) で表示することが可能です。

2.2.1. ログインサーバでの vncserver 起動

vncserver を起動します。初回のみ パスワード設定が必要です。

実行例 (初回の VNC 起動)

```
[scuser@pifront1 ~]$ vncserver -geometry 1024x768

You will require a password to access your desktops.

Password: XXXXXXXX
Verify: XXXXXXXX

New 'pifront1:1 (scuser)' desktop is pifront1:1

Creating default startup script /home/scuser/.vnc/xstartup
Creating default config /home/scuser/.vnc/config
Starting applications specified in /home/scuser/.vnc/xstartup
Log file is /home/scuser/.vnc/pifront1:1.log
```

2 回目以降は 設定済みの VNC パスワードをそのまま利用可能です。

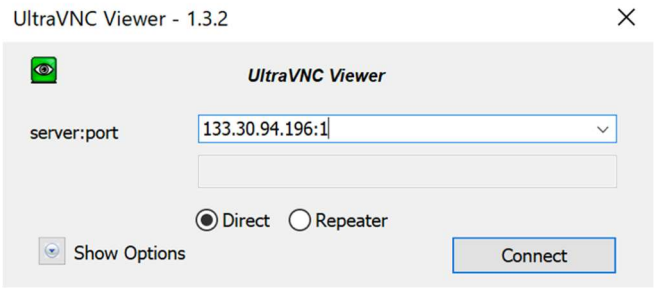
実行例 (2 回目以降の VNC 起動)

```
[scuser@pifront1 ~]$ vncserver -geometry 1024x768

New 'pifront1:1 (scuser)' desktop is pifront1:1

Starting applications specified in /home/scuser/.vnc/xstartup
Log file is /home/scuser/.vnc/pifront1:1.log

[scuser@pifront1 ~]$
```

これで、pifront1 の セッション 1 番 に VNC でアクセスすると、scuser の GUI が利用できるようになります。

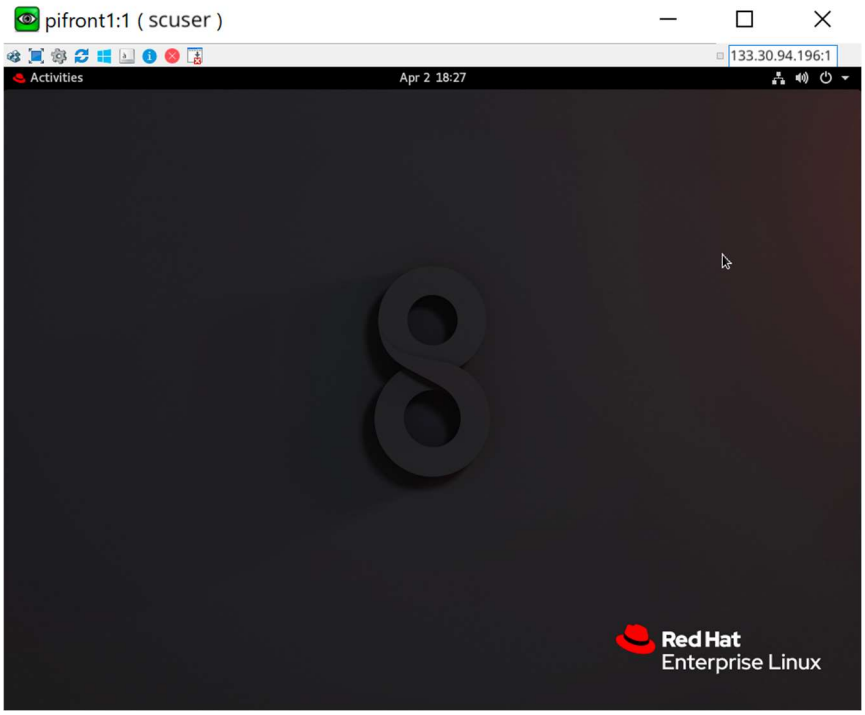
2.2.2. PC/ワークステーションから VNC アクセス方法 (学内)

学内から Windows 環境で UltraVNC Viewer を利用してアクセスする場合、”[IP アドレス]:[セッション番号]”で指定します。今回の場合、pifront1 に セッション 1 番で接続するので、”133.30.94.196:1” を指定して、Connect ボタンを押下します。



フロントサーバ上で VNC サーバを初回に起動した時に設定したパスワードを入力し、”Log On” ボタンを押下します。

VNC が起動します。



2.2.3. PC/ワークステーションから VNC アクセス方法 (学外)

学外から Windows 環境で UltraVNC Viewer を利用してアクセスする場合、ssh のポート転送設定が必要です。
VNC の ポート番号は “5900 番 + VNC セッション番号” といった形で指定します。
下記は pifront1 の 5901 番ポート (VNC セッション#1)を PC/ワークステーション の 25901 番ポートでアクセスできるように設定しています。

pifront1 セッション番号 1 (ポート番号 5901) の設定例

```
[wsuser@workstation ~]$ ssh -L 25901:pifront1:5901 pifront1
```

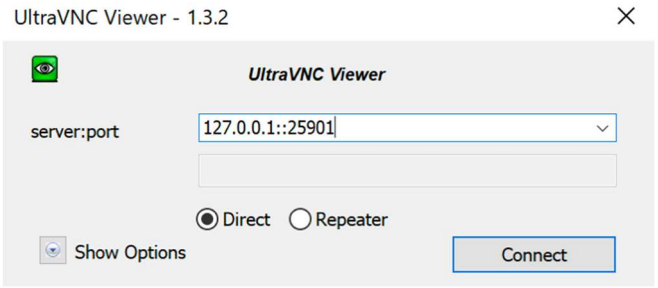
VNC サーバを起動した時に付与されるセッション番号は、その都度変わります。
下記は pifront2 の 5903 番ポート (VNC セッション#3)を PC/ワークステーション の 25901 番ポートでアクセスできるように設定しています。

pifront2 セッション番号 3 (ポート番号 5903) の設定例

```
[wsuser@workstation ~]$ ssh -L 25901:pifront2:5903 pifront2
```

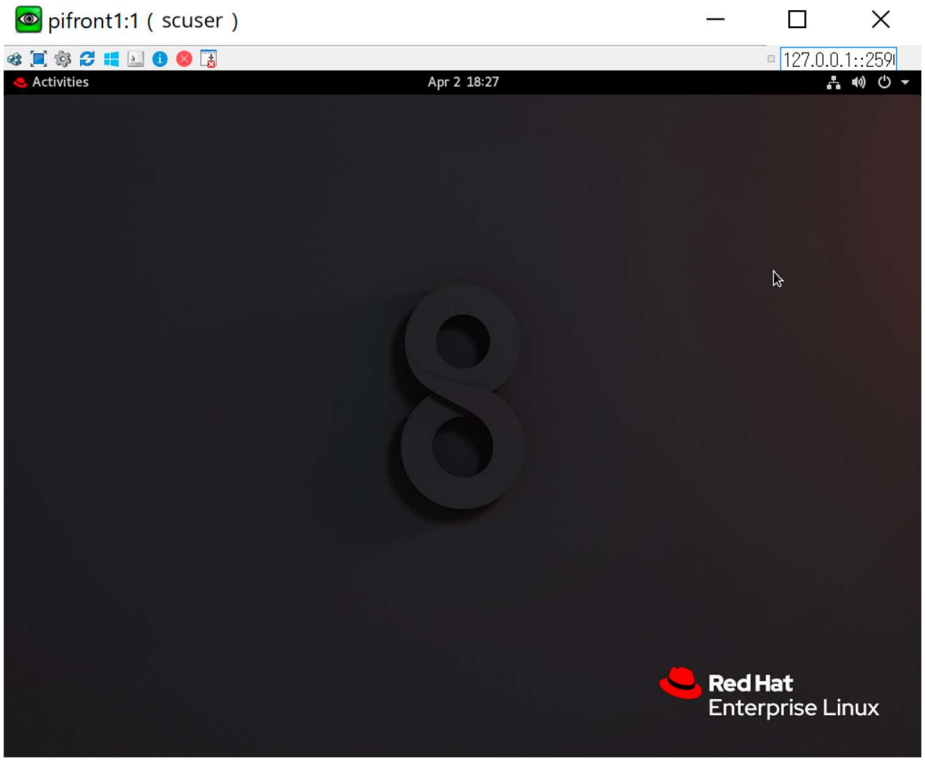
ssh 実行後は、VNC を終了するまで SSH のセッションを開いたままにしておきます。

次に UltraVNC Viewer を起動します。
”[IP アドレス]::[ポート番号]”で指定します。ポート番号にする場合は、”: (コロン)” が 2 個連続で入力されていることに注意してください。
pifront1 の VNC セッション 1 番で接続されている場合、localhost (127.0.0.1) に セッション 1 番が割り当てられたポートで 5901 番で接続するので、”127.0.0.1::1” を指定して、Connect ボタンを押下します。



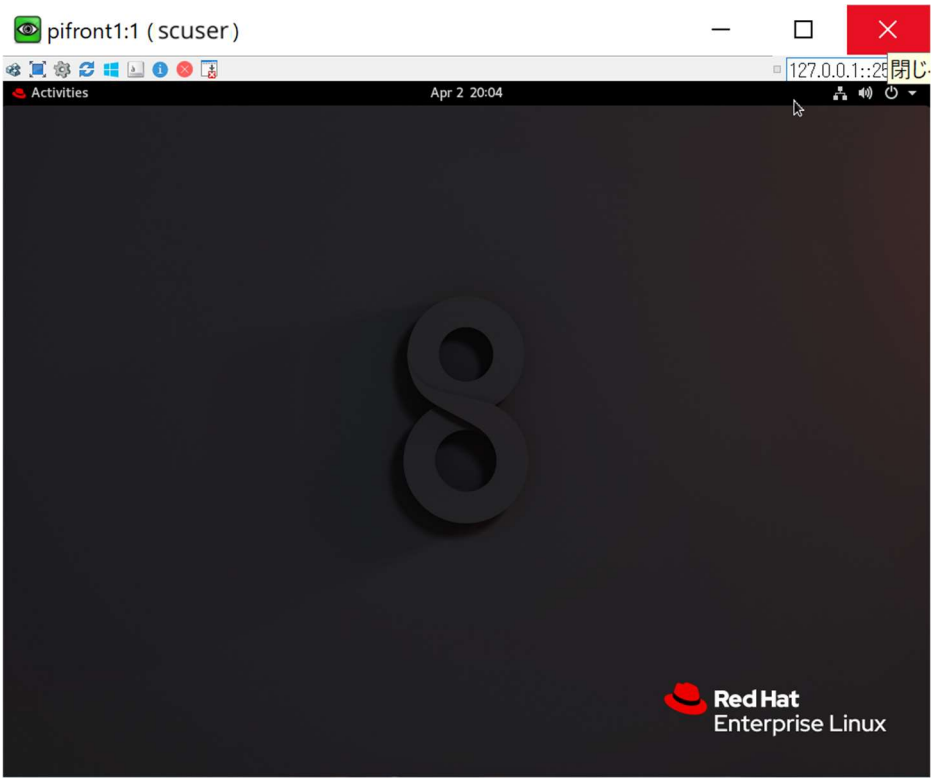
フロントサーバ上で VNC サーバを初回に起動した時に設定したパスワードを入力し、”Log On” ボタンを押下します。

VNC が起動します。


pifront1:1 (scuser)
127.0.0.1::259
Activities
Apr 2 18:27
Red Hat
Enterprise Linux

2.2.4. VNC の終了方法

PC/ワークステーションの VNC Viewer を閉じます。



フロントエンドサーバ上で vncserver を 終了します。

```
[scuser@pifront1 ~]$ vncserver -kill :1
Killing Xvnc process ID 145654
[scuser@pifront1 ~]$
```

3. module コマンドによる環境設定方法

module コマンドを利用することで、コンパイラやアプリケーションで使用する環境設定をセットすることができます。
利用可能な module 環境は `module avail` で確認できます。

```
$ module avail
------------------------------------ /share/spack/share/spack/modules/linux-rhel8-zen -------------------------------------
berkeley-db-18.1.40-gcc-8.3.1-5q2ol2j  ncurses-6.2-gcc-8.3.1-mefq7nl            py-setuptools-50.3.2-gcc-8.3.1-aspmcvt
bzip2-1.0.8-gcc-8.3.1-p2kq6k2          openblas-0.3.13-gcc-8.3.1-atpqev5        py-threadpoolctl-2.0.0-gcc-8.3.1-e2j4vfw
cmake-3.19.5-gcc-8.3.1-sdnk3x5         openssl-1.1.1j-gcc-8.3.1-5rvzi7x         python-3.8.8-gcc-8.3.1-xbx7faq
diffutils-3.7-gcc-8.3.1-5cxx2jc        perl-5.32.1-gcc-8.3.1-ezyqegs            readline-8.0-gcc-8.3.1-v2z56yf
expat-2.2.10-gcc-8.3.1-hgy7ksk         pkgconf-1.7.3-gcc-8.3.1-zwn7zfd          sqlite-3.34.0-gcc-8.3.1-ut7bdqe
gdbm-1.18.1-gcc-8.3.1-ugrzsga          py-cython-0.29.21-gcc-8.3.1-6e3f7yw      tar-1.32-gcc-8.3.1-h2wggm2
gettext-0.21-gcc-8.3.1-t6n3pyr         py-joblib-0.14.0-gcc-8.3.1-oaspy2t       util-linux-uuid-2.36-gcc-8.3.1-t25i5ee
libbsd-0.10.0-gcc-8.3.1-4ikaln3        py-numpy-1.20.1-gcc-8.3.1-m52nwlz        xz-5.2.5-gcc-8.3.1-favghf3
libffi-3.3-gcc-8.3.1-w6vkdly           py-pybind11-2.5.0-gcc-8.3.1-6idzm6c      zlib-1.2.11-gcc-8.3.1-ycvigzu
libiconv-1.16-gcc-8.3.1-k4uqlcr        py-scikit-learn-0.24.1-gcc-8.3.1-fl2pjhk
libxml2-2.9.10-gcc-8.3.1-e53xwao       py-scipy-1.6.1-gcc-8.3.1-ueumwgw

------------------------------------------- /usr/share/Modules/modulefiles --------------------------------------------
dot  hmpt/2.23  module-git  module-info  modules  MPInside/4.2.50  mpiplace/2.00  mpt/2.23  null  perfboost  use.own

---------------------------------------------- /share/modulefiles -----------------------------------------------
ParaView/5.9.0

---------------------------------------------- /share/oneAPI/modulefiles ----------------------------------------------
advisor/2021.1.1        compiler/2021.1.1       dnnl-cpu-gomp/2021.1.1  init_opencl/2021.1.2          mkl32/2021.1.1
advisor/latest          compiler/2021.1.2       dnnl-cpu-gomp/latest    init_opencl/latest            mkl32/latest
ccl/2021.1.1            compiler/latest         dnnl-cpu-iomp/2021.1.1  inspector/2021.1.1            mpi/2021.1.1
ccl/latest              compiler32/2021.1.1     dnnl-cpu-iomp/latest    inspector/latest              mpi/latest
clck/2021.1.1           compiler32/2021.1.2     dnnl-cpu-tbb/2021.1.1   intel_ipp_intel64/2021.1.1    oclfpga/2021.1.2
clck/latest             compiler32/latest       dnnl-cpu-tbb/latest     intel_ipp_intel64/latest      oclfpga/latest
compiler-rt/2021.1.1    dal/2021.1.1            dnnl/2021.1.1           intel_ippcp_intel64/2021.1.1  tbb/2021.1.1
compiler-rt/2021.1.2    dal/latest              dnnl/latest             intel_ippcp_intel64/latest    tbb/latest
compiler-rt/latest      debugger/10.0.0         dpct/2021.1.1           itac/2021.1.1                 vpl/2021.1.1
compiler-rt32/2021.1.1  debugger/latest         dpct/latest             itac/latest                   vpl/latest
compiler-rt32/2021.1.2  dev-utilities/2021.1.1  dpl/2021.1.2            mkl/2021.1.1                  vtune/2021.1.2
compiler-rt32/latest    dev-utilities/latest    dpl/latest              mkl/latest                    vtune/latest
```

module 環境をロードしたい場合は、`module load <モジュール名>` を実行します。

Intel コンパイラ の 設定例

```
$ module load compiler
```

ロードした module 環境を初期化したい場合は `module purge` を実行します。

```
$ module purge
```

ジョブスクリプト内で module コマンドを実行する場合は、スクリプト内で module コマンドの初期設定を行う必要があります。

ジョブスクリプト内 Intel oneAPI 設定例

```
source /etc/profiled.d/modules.sh
module load compiler
```

csh 系の場合は “/etc/profile.d/modules.csh” に置き換えてください。

4. 開発環境の利用

本システムで利用できる開発環境は以下の通りです。

	Intel oneAPI Toolkit	HPE Cray Programming Environment	HPE MPI
コンパイラ	Intel Compiler Classic	Cray Compiling Environment (CCE)	
数値計算ライブラリ	Intel oneMKL	Cray Scientific and Math Libraries (CSML)	
MPI 通信ライブラリ	Intel MPI	Cray Message Passing Toolkit (CMPT)	HPE MPI

4.1. Intel oneAPI Toolkit の利用

4.1.1. Intel コンパイラ classic の利用

4.1.1.1. 環境設定

module コマンドで compiler モジュールをロードします。

```
$ module load compiler
```

4.1.1.2. コンパイル／リンクの概要

コンパイル／リンクの書式は以下の通りです。

```
$ command [options] sourcefile […]
```

コンパイルには言語に応じて以下のコマンドを利用します。

言語処理系	コマンド	コマンド形式
Fortran	`ifort`	`$ ifort [options] sourcefile`
C	`icc`	`$ icc [options] sourcefile`
C++	`icpc`	`$ icpc [options] sourcefile`

[options] には、最適化オプション、並列化オプション、ライブラリのリンクオプション、その他のコンパイラオプションが入ります。なお、`--help` オプションを指定すると、コンパイラオプションの一覧が表示されます。

Fortran コンパイル実行例

```
$ ifort -march=core-avx2 -O3 program.f
```

4.1.1.3. コンパイルの主なオプション

最適化のオプションは以下の通りです。

オプション	説明
最適化レベルオプション	
-O0	全ての最適化を無効にします。デバッグを行う時に指定してください。
-O1	保守的な最適化を行います。
-O2	ソフトウェアパイプラインによる最適化を行います。デフォルトの最適化レベルです。何も指定しないとこのオプションで最適化されます。
-O3	積極的な最適化を行います。-O2 の最適化に加えて、ループの交換やデータプリフェッチを行います。
CPU アーキテクチャに対する最適化オプション	
-march=core-avx2	AVX2 命令セットを持つ AMD EPYC CPU 向けの最適化を行います。
最適化レポートオプション	
-qopt-report=n	最適化レポートを生成します。デフォルトでは、レポートは.optrpt 拡張子を持つファイルに出力されます。n には、0 (レポートなし) から 5 (最も詳しい) の詳細レベルを指定します。デフォルトは 2 です。

[補足]

AMD EPYC CPU は AVX512 命令セットを持たないので、Intel プロセッサ向け 最適化 の -xCORE-AVX512 は用いることはできません。

その他、主なオプションは以下の通りです。

オプション	説明
浮動小数点演算、プロシージャ間解析オプション	
-fp-model precise	浮動小数点データの精度に影響する最適化を無効にし、中間結果をソースで定義された精度まで丸めます。
-ipo	複数のソースファイルにあるプロシージャ間の解析、最適化を行います。リンク時にもオプションとして指定してください。インライン展開を有効にし、プログラムのなかで小さいサイズの関数を何回も呼び出しているような場合に性能向上の効果があります。
デバック用オプション	
-g	-g オプションはオブジェクト・ファイルのサイズを大きくするシンボリック・デバッグ情報をオブジェクト・ファイルに生成するようにコンパイラに指示します。
-traceback	このオプションは、ランタイム時に致命的なエラーが発生したとき、ソースファイルのトレースバック情報を表示できるように、オブジェクト・ファイル内に補足情報を生成するようにコンパイラに指示します。 致命的なエラーが発生すると、コールスタックの 16 進アドレス (プログラム・カウンター・トレース) とともに、ソースファイル、ルーチン名、および行番号の相関情報が表示されます。 マップファイルとエラーが発生したときに表示されるスタックの 16 進アドレスを使用することで、エラーの原因を特定できます。 このオプションを指定すると、実行プログラムのサイズが増えます。

プログラムの実行時にエラーが発生した場合、診断メッセージが出力されます。より詳細な情報であるトレースバックを表示してデバッグするためには “-g -traceback” オプションが有効です。

4.1.1.4. メモリモデル指定

次のいずれかのメモリモデルを使用して実行バイナリを作成します。

メモリモデル	説明
-mcmodel=small	コードとデータのすべてのアクセスが、命令ポインター (IP) 相対アドレス指定で行われるように、コードとデータはアドレス空間の最初の 2GB までに制限されます。 -mcmodel オプションを指定しない場合、デフォルトでこちらが指定されます。
-mcmoddel=medium	コードはアドレス空間の最初の 2GB までに制限されますが、データは制限されません。コードは IP 相対アドレス指定でアクセスできますが、データのアクセスは絶対アドレス指定を使用する必要があります。
-mcmodel=large	コードもデータも制限されません。コードもデータもアクセスは絶対アドレス指定を使用します。

IP 相対アドレス指定は 32 ビットのみ必要ですが、絶対アドレス指定は 64 ビット必要です。これは、コードサイズとパフォーマンスに影響します。(IP 相対アドレス指定の方が多少速くアクセスできます)。
プログラム内の共通ブロック、グローバルデータ、静的データの合計が 2GB を越えるとき、リンク時に次のエラーメッセージが出力されます。

```
<some lib.a library>(some .o): In Function <function>:
  : relocation truncated to fit: R_X86_64_PC32 <some symbol>
.....................
  : relocation truncated to fit: R_X86_64_PC32 <some symbol>
```

この場合は、-mcmodel=medium を指定してコンパイル/リンクして下さい。

4.1.1.5. スレッド並列 (OpenMP と自動並列化)

OpenMP、自動並列化によるスレッド並列によりプログラムの高速化ができます。

オプション	説明
自動並列化	
-parallel	自動並列化の指定です。OpenMP 指示行はコメント行として扱います。
-qopt-report=n -qopt-report-phase=par	-qopt-report=n の出力を自動並列化に限定します。
-parallel -par-threshold[n]	自動並列化の閾値を設定します。(n は 0 から 100 までデフォルトは 75)。 -par-threshold0:性能向上が確実でないループでも並列化 -par-threshold100:性能向上が確実なループのみ並列化
OpenMP	
-qopenmp	自動並列化をせずに、OpenMP 指示行のみ有効にします。
-qopt-report=n -qopt-report-phase=openmp	-qopt-report=n の出力を OpenMP 並列化に限定します。

4.1.2. MPI 通信ライブラリ Intel MPI の利用

Intel MPI を使った MPI プログラムのコンパイルは MPI ライブラリをリンクする専用コマンドを利用します。

4.1.2.1. 環境設定

module コマンドで mpi モジュールをロードします。

```
$ module load mpi
```

4.1.2.2. コンパイル／リンク方法

言語処理系	Intel コンパイラ classic 利用	GNU コンパイラ利用
Fortran	$ mpiifort [options] sourcefile	$ mpif90 [options] sourcefile
C	$ mpiicc [options] sourcefile	$ mpicc [options] sourcefile
C++	$ mpiicpc [options] sourcefile	$ mpicxx [options] sourcefile

-show オプションをつけると、実行されるコマンドの詳細を確認できます。

mpiifort コマンドの確認

```
$ mpiifort -show
ifort  -I'/opt/intel/oneapi/mpi/2021.1.1//include'  -I'/opt/intel/oneapi/mpi/2021.1.1/include'  -
L'/opt/intel/oneapi/mpi/2021.1.1/lib/release' -L'/opt/intel/oneapi/mpi/2021.1.1/lib' -Xlinker --
enable-new-dtags -Xlinker -rpath -Xlinker '/opt/intel/oneapi/mpi/2021.1.1/lib/release' -Xlinker -
rpath -Xlinker '/opt/intel/oneapi/mpi/2021.1.1/lib' -lmpifort -lmpi -ldl -lrt -lpthread
```

mpif90 コマンドの確認

```
$ mpif90 -show
gfortran               -I'/opt/intel/oneapi/mpi/2021.1.1/include/gfortran/8.2.0'               -
I'/opt/intel/oneapi/mpi/2021.1.1/include'    -L'/opt/intel/oneapi/mpi/2021.1.1/lib/release'    -
L'/opt/intel/oneapi/mpi/2021.1.1/lib'  -Xlinker  --enable-new-dtags  -Xlinker  -rpath  -Xlinker
'/opt/intel/oneapi/mpi/2021.1.1/lib/release'          -Xlinker          -rpath          -Xlinker
'/opt/intel/oneapi/mpi/2021.1.1/lib' -lmpifort -lmpi -lrt -lpthread -Wl,-z,now -Wl,-z,relro -Wl,-
z,noexecstack -Xlinker --enable-new-dtags -ldl
```

4.1.3. 数値計算ライブラリ oneMKL の利用

4.1.3.1. Intel OneMKL 概要

数値計算ライブラリとして Intel oneMKL (oneAPI Math Kernel Library) を提供します。

- BLAS および BLAS 拡張
- LAPACK および LAPACK 拡張
- ScaLAPACK
- ベクトル数学 (VML)
- ベクトル統計 (乱数ジェネレータ)
- ベクトル統計 (サマリー統計)
- データ適合
- FFT/DFT
- スパース BLAS
- スパースソルバー

4.1.3.2. 環境設定

module コマンドで mkl モジュールをロードします。

```
$ module load mkl
```

すでに compiler モジュールをロードしている場合は compiler-rt および tbb モジュールを アンロードしてから mkl モジュールをロードします。

```
$ module load compiler
$ module unload compiler-rt tbb
$ module load mkl
```

4.1.3.3. コンパイル／リンク方法

Intel oneMKL のリンク方法は 用途に応じてオプションを選んでください。

Intel コンパイラ classic (`ifort`, `icc`, `icpc`) のコンパイラオプションとして `-mkl` オプションを利用できます。

用途	オプション	備考
逐次	`-mkl=sequential`	
スレッド並列	`-mkl=parallel`	OpenMP や Intel TBB を使用する場合
MPI 並列	`-mkl=cluster`	Intel MPI を使用する場合

`-mkl` オプションを使わず必要なライブラリを明示的にリンクする場合は、以下のように指定します。

用途	オプション
逐次	`-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm`
スレッド並列	`-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm`
ScaLAPACK スレッド並列	`-lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm`
64 ビット整数型対応 スレッド並列	`-lmkl_intel_ilp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm`

4.2. HPE MPI の利用

Intel MPI だけではなく、HPE MPI (旧 SGI MPT) を使用した MPI プログラムのコンパイルを利用できます。
HPE MPI を使った MPI プログラムのコンパイルにおいても MPI ライブラリをリンクする専用コマンドを利用します。
※旧システム(vizfront, vizcore)での SGI MPT は、HPE MPI に名称が変更されました。

4.2.1. 環境設定

module コマンドで mpt モジュールをロードします。

```
$ module load mpt
```

4.2.2. コンパイル、リンク方法

mpif90、mpicc、mpicxx コマンドが用意されており、環境変数を指定することで Intel コンパイラ classic を利用することができます。

言語処理系	使用コンパイラ	実行方法
Fortran	GNU コンパイラ	`$ mpif90 [options] sourcefile`
	Intel コンパイラ Classic	`$ module load compiler` `$ mpif90 [options] sourcefile`
C	GNU コンパイラ	`$ mpicc [options] sourcefile`
	Intel コンパイラ Classic	`$ module load compiler` `$ MPICC_CC=icc mpicc [options] sourcefile`
C++	GNU コンパイラ	`$ MPI_LIB="mpi+ -lmpi" mpicxx [options] sourcefile`
	Intel コンパイラ Classic	`$ module load compiler` `$ MPICXX_CXX=icpc MPI_LIB="mpi+ -lmpi" mpicxx [options] sourcefile`

`-show` オプションをつけると、実行されるコマンドの詳細を確認できます。

mpif90 コマンドの確認

```
$ mpif90 -show
gfortran  -I/opt/hpe/hpc/mpt/mpt-2.23/include  -I/opt/hpe/hpc/mpt/mpt-2.23/include  -lpthread  -
L/opt/hpe/hpc/mpt/mpt-2.23/lib -lmpi
```

Intel コンパイラ classic の モジュールを ロードすると、`ifort` に切り替わります。

```
$ module load compiler
$ mpif90 -show
ifort -I/opt/hpe/hpc/mpt/mpt-2.23/include -lpthread -L/opt/hpe/hpc/mpt/mpt-2.23/lib -lmpi
```

5. ジョブスケジューラの利用

5.1. ジョブスケジューラの概要

本システムでは、効率的なジョブ運用を実現するため、PBS Professional (以降 PBS) が導入されています。
ユーザはジョブ開始時に必要なリソース情報など実行条件を指定し、PBS に対してジョブ実行を指示します。
ジョブ実行形式には、インタラクティブ実行と、ジョブスクリプトを使用したバッチ実行があります。
インタラクティブ実行は対話形式、バッチ実行はスクリプトに記述されたコマンド実行形式です。
ジョブスクリプトの作成方法は、「ジョブスクリプトの作成」をご参照ください。

ユーザがジョブ操作に用いるコマンドは以下の通りです。

機能	コマンド
バッチジョブ投入	`$ qsub <job script>`
対話形式によるジョブ投入	`$ qsub -I`
実行ジョブの確認	`$ qstat <jobid>`
実行ジョブのキャンセル	`$ qdel <jobid>`

5.2. キュー構成

本システムで利用可能なキュー構成は以下の通りです。

キュー名	最大 CPU コア数	最大実行時間	実行ホスト
debug	32	30 分	pinode01-16
large	512	12 時間	pinode01-16

5.3. ジョブ投入方法

基本的なジョブ投入方法は、

```
$ qsub <job script>
```

です。`job script` は ジョブスクリプト (ジョブ投入用シェルスクリプト) です。
また、スクリプトによるバッチ実行ではなく、インタラクティブ実行は 以下のコマンドを利用します。

```
$ qsub -I
```

ジョブの投入が成功すると、投入ジョブに対してジョブ ID が割り当てられます。インタラクティブ実行時は `exit` コマンドでジョブを終了します。

5.3.1. ジョブ投入例

バッチ実行時には、ジョブ投入（`qsub` コマンド実行）を行ったディレクトリに係わらず、実行ユーザのホームディレクトリがカレントディレクトリとなります。したがってプログラムを実行するディレクトリが異なる、または固定的な場所をアクセスす

るようなジョブの場合には、実行コマンドまたは作成したジョブスクリプト内で cd を実行し、ディレクトリを明示してください。

バッチ実行のジョブ内では"PBS_O_WORKDIR"という環境変数に qsub コマンドを実行したディレクトリがセットされています。
スクリプト内に"cd $PBS_O_WORKDIR"を記載しておくことにより、ジョブ投入を行ったディレクトリに変更する事ができます。

5.3.1.1. ジョブスクリプトの作成

任意のテキストエディタでスクリプトを作成ください。

```
$ cat sample_script.sh
#!/bin/bash
#PBS -q debug
#PBS -l select=1:ncpus=1
#PBS -N sample_job
#PBS -j oe
cd ${PBS_O_WORKDIR}
echo “Hello sample job.”
```

5.3.1.2. ジョブの投入

作成したジョブスクリプトを用いて qsub コマンドでジョブを投入します。

```
$ qsub ./sample_script.sh
104.pifront1
```

104 が PBS で割り当てられたジョブ ID です。

5.3.1.3. ジョブの終了

ジョブの終了後、ジョブスクリプトが出力した内容を確認することができます。

```
$ cat ./sample_job.o104
Hello sample job.
```

5.3.2. ジョブの投入 qsub コマンド

qsub コマンドの主なオプションは以下の通りです。ジョブ投入時に指定する、もしくは ジョブスクリプト内に記載ください。

オプション	説明
-q queue_name	ジョブを投入するキューの指定。 指定がない場合は、デフォルトキュークラスである“debug”へ投入されます。
-N name	ジョブ名の指定。指定しない場合はジョブスクリプトの名前が自動的に設定されます。アルファベット、数字、一部記号(+ - _ .)が使用可能です。空白文字は使用できません。
-a date_time	指定した日時にジョブを開始するように指定します。date_time は[[[[CC]YY]MM]DD]hhmm[.SS]というような形式で指定します。CCYY は西暦、MMDD は月日、hhmm.SS は時分秒です。
-j oe | eo	stdout(標準出力)、stderr(標準エラー出力)の出力ファイルを統合。"oe"を指定した場合は、これらの出力をまとめて stdout に出力します。"eo"を指定した場合は、出力をまとめて stderr に出力します。
-o path_name	stdout の出力ファイル名を指定。無指定だと"ジョブ名.o ジョブ ID"となります。投入時のディレクトリに対する相対パス、絶対パスが指定できます。
-e path_name	stderr の出力ファイル名を指定。無指定だと"ジョブ名.e ジョブ ID"となります。投入時のディレクトリに対する相対パス、絶対パスが指定できます。
-m mail_option	メールの送信の設定。mail_option には以下のものがあります。 a： ジョブが中止された場合に送信 b： ジョブの実行が開始された際に送信 e： ジョブの実行が終了した際に送信 n： メール送信しない
-M mail_address	メール受信者の設定。指定しない場合、ジョブを投入したユーザーアカウントが受信者となります。
-l keyword=value	ジョブのリソース(コア数、メモリ、実行時間など)要求の指定。利用するノード数、コア数の指定の仕方は「qsub コマンド select 文について」をご覧ください。
-l walltime=hh:mm:ss	ジョブの実行時間制限の指定。hh:mm:ss(時:分:秒)の形式で指定してください。例えば実行時間を 1 時間に制限する場合は、次のように記述します。-l walltime=1:00:00

qsub コマンドの詳細については、オンラインマニュアル (man qsub)、または PBS 付属のユーザガイドを参照ください。

5.3.2.1. qsub コマンドの select 文について

ジョブで利用するリソース数（ノード数、CPU コア数など）を指定する場合は select 文を用います。
ジョブスクリプト内もしくはジョブ投入オプションに以下を指定します。

```
-l select=N1:ncpus=N2
```

MPI ジョブの場合は 以下を指定します。

```
-l select=N1:ncpus=N2:mpiprocs=N3
```

select 文で 指定する N1、N2、N3 の説明は、以下の通りです。

Nx	説明
N1	利用するノード数を指定します。
N2	計算ノード内の CPU コア数を指定します。 本システムの計算ノードは 32 コアを搭載していますので、N2 は 32 以下となります。
N3	計算ノード内の MPI プロセス数を指定します。 MPI ジョブの場合は N3=N2 となります。 MPI と OpenMP のハイブリッドジョブの場合は、N3=N2÷(OpenMP スレッド数) です。

ノードごとに N2、N3 の値が変わる場合には、

```
-l select=N1:ncpus=N2:mpiprocs=N3+M1:ncpus=M2:mpiprocs=M3
```

のように + でつなげて指定してください。

5.3.3. ジョブスクリプトの作成

バッチ実行のジョブを投入する際に使用するジョブスクリプトについて、以下にサンプルを示します。
ジョブスクリプトのファイル名は任意で構いません。ジョブスクリプトは qsub コマンドの入力ファイルですので、実行属性は必要ありません。

5.3.3.1. 並列化されていない逐次ジョブ

2 行目から 6 行目のように"#PBS"で始まる行は qsub コマンドに指定するオプションをスクリプト内に記述するものです。(#PBS 行は、シェルにはコメント行として解釈されますが、qsub コマンドにはオプション指定行として解釈されます)。

1CPU を使用した逐次ジョブのスクリプト例

1	#!/bin/bash	bash スクリプトを宣言
2	#PBS -q debug	キュー名指定
3	#PBS -l select=1:ncpus=1	利用するリソースを指定
4	#PBS -N serial_JOB	ジョブ名を指定
5	#PBS -o serial_out_file	標準出力（stdout）を出力するファイルを指定
6	#PBS -j oe	標準エラー出力を 標準出力に結合する指定
7	source /etc/profile.d/modules.sh	
8	module load compiler	Intel コンパイラ環境を読み込み
9	cd ${PBS_O_WORKDIR}	実行ファイルがあるディレクトリへ移動
10	./a.out > log.txt 2>&1	プログラムの実行

10 行目のようにプログラムの標準出力、標準エラー出力を明示的に指定するファイルへ出力することもできます。

5.3.3.2. スレッド並列ジョブ (OpenMP)

OpenMP の並列数(OMP_NUM_THREADS 環境変数)は、デフォルトで qsub 実行時の-l select の ncpus オプションで指定された値が自動的に設定されます。

OpenMP 8 スレッド並列ジョブのスクリプト例

1	#!/bin/bash	
2	#PBS -q debug	
3	#PBS -l select=1:ncpus=8	利用するノード数、CPU 数（並列度）を指定
4	#PBS -N omp_JOB	
5	#PBS -o omp_out_file	
6	#PBS -j oe	
7	source /etc/profile.d/modules.sh	
8	module load compiler	Intel コンパイラ環境を読み込み
9	export OMP_NUM_THREADS=8	OpenMP の並列数を明示的に変更する場合。 ncpu と同じ値なら不要。
10	export KMP_AFFINITY=disabled	Intel コンパイラの AFFINITY を無効化
11	cd ${PBS_O_WORKDIR}	
12	dplace ./a.out > log.txt 2>&1	プログラムの実行

自動並列化/OpenMP 並列化ジョブの場合は、性能を安定させるために実行プログラムの前に dplace コマンドを指定してください。
dplace コマンドは、OS のプロセススケジュールによる CPU 間のプロセス移動を避けるために、プロセス/スレッドをコアに固定するコマンドです。

5.3.3.3. MPI 並列ジョブ (HPE MPI)

ジョブ実行時に qsub オプションで確保するリソース数を、-l select の ncpus オプション（CPU 数、並列度）、mpiprocs オプション（MPI プロセス数）で指定してください。

MPI 8 並列ジョブ (2 ノード x 4 MPI) のスクリプト例

```
#!/bin/bash
#PBS -q large
#PBS -l select=2:ncpus=4:mpiprocs=4          利用するノード数、CPU 数（並列度）、MPI プロセス数を指定
#PBS -N hpempt_JOB
#PBS -o hpempt_out_file
#PBS -j oe
source /etc/profile.d/modules.sh
module load compiler                         Intel コンパイラ環境を読み込み
module load mpt                              HPE MPI 環境を読み込み
cd ${PBS_O_WORKDIR}
mpiexec_mpt dplace -s1 ./a.out > log.txt 2>&1   プログラムの実行
```

HPE MPI 並列ジョブを実行するには mpiexec_mpt コマンドを利用します。MPI の並列度を-np オプションで指定しなくても、自動的に PBS の mpiprocs オプションで指定した値が使用されます。
HPE MPI を利用した MPI 並列プログラムの実行には、性能を安定させるためにプログラムの前に dplace -s1 コマンド（要オプション）を指定してください。
dplace コマンドは、OS のプロセススケジュールによる CPU 間のプロセス移動を避けるために、プロセス/スレッドをコアに固定するコマンドです。-s オプションを指定すると、最初の n 個のプロセスを CPU に配置しません。
mpiexec_mpt では 1+N（並列数）個のプロセスが起動されますが、最初のプロセスは実際にはインアクティブな状態ですので、-s1 を指定してインアクティブなプロセスを CPU に割り当てないようにします。

5.3.3.4. MPI 並列ジョブ (Intel MPI)

ジョブ実行時に qsub オプションで確保するリソース数を、-l select の ncpus オプション（CPU 数、並列度）、mpiprocs オプション（MPI プロセス数）で指定してください。

MPI 8 並列ジョブ (2 ノード x 4MPI) のスクリプト例

```
#!/bin/bash
#PBS -q large
#PBS -l select=2:ncpus=4:mpiprocs=4
#PBS -N intelmpi_JOB
#PBS -o intelmpi_out_file
#PBS -j oe
source /etc/profile.d/modules.sh
module load compiler
module load mpi
cd ${PBS_O_WORKDIR}
mpirun ./a.out > log.txt 2>&1
```

利用するノード数、CPU 数（並列度）、MPI プロセス数を指定

Intel コンパイラ環境を読み込み
Intel MPI 環境を読み込み

プログラムの実行

Intel MPI 並列ジョブを実行するには mpirun コマンドを利用します。MPI の並列度を-np オプションで指定しなくても、自動的に PBS の mpiprocs オプションで指定した値が使用されます。

5.3.3.5. MPI+OpenMP ハイブリッド並列ジョブ (HPE MPI)

ジョブ実行時に qsub オプションで確保するリソース数を、-l select の ncpus オプション（CPU 数、並列度）、mpiprocs オプション（MPI プロセス数）で指定してください。

ハイブリッド並列ジョブ (2 ノード x 2MPI x 8 スレッド) のスクリプト例

```
#!/bin/bash
#PBS -q large
#PBS -l select=2:ncpus=16:mpiprocs=2                    利用するリソースを指定
#PBS -N hybrid_JOB
#PBS -o hybrid_out_file
#PBS -j oe
source /etc/profile.d/modules.sh
module load compiler                                    Intel コンパイラ環境を読み込み
module load mpt                                         HPE MPI 環境を読み込み
export OMP_NUM_THREADS=8                                OpenMP の並列数を指定
export KMP_AFFINITY=disabled                            Intel コンパイラの AFFINITY を無効化
cd ${PBS_O_WORKDIR}
mpiexec_mpt omplace -nt ${OMP_NUM_THREADS} ./a.out > log.txt 2>&1    プログラムの実行
```

HPE MPI でハイブリッド並列ジョブを実行するには mpiexec_mpt コマンド、omplace コマンドを利用します。
各ノードの OpenMP の並列度は omplace コマンドの-nt オプションの後に指定してください。
omplace コマンドは、OS のプロセススケジュールによる CPU 間のプロセス移動を避けるために、プロセス/スレッドをコアに固定するコマンドです。ハイブリッドジョブでスレッド並列を最適な CPU 配置にする事で、性能が向上する可能性があります。

5.3.3.6. MPI+OpenMP ハイブリッド並列ジョブ (Intel MPI)

ジョブ実行時に qsub オプションで確保するリソース数を、-l select の ncpus オプション（CPU 数、並列度）、mpiprocs オプション（MPI プロセス数）で指定してください。

ハイブリッド並列ジョブ (2 ノード x 2MPI x 8 スレッド) のスクリプト例

```
#!/bin/bash
#PBS -q S
#PBS -l select=2:ncpus=16:mpiprocs=2                利用するリソースを指定
#PBS -N hybrid_JOB
#PBS -o hybrid_out_file
#PBS -j oe
source /etc/profile.d/modules.sh
module load compiler                                Intel コンパイラ環境を読み込み
module load mpi                                     Intel MPI 環境を読み込み
export OMP_NUM_THREADS=8                            OpenMP の並列数を指定
cd ${PBS_O_WORKDIR}
mpirun ./a.out > log.txt 2>&1                       プログラムの実行
```

5.3.4. ジョブの確認方法

ジョブの確認には qstat コマンドを用います。以下に出力例を示します。

```
$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
106.pifront1      mpi256job1       user1            17:31:36 R large
107.pifront1      hyb256job2       user2            01:19:28 R large
108.pifront1      hyb256job2       user2                   0 Q large
109.pifront1      mpi256job2       user1                   0 Q large
```

qstat コマンドにより参照可能な情報は以下の通りです。

表示の属性	説明
Job id	投入時に割り当てられたジョブ ID
Name	ジョブ名
User	投入ユーザ
Time Use	経過した CPU 時間
S	ジョブステータス
Queue	ジョブが属するキュー

またジョブステータスは以下の状態を示します。

ジョブステータス	説明
Q	キューに投入され実行待ち
R	実行中
E	実行済みで終了処理中
S	サーバによって中断中

qstat コマンドの主なオプションは以下の通りです。

オプション	説明
-u username	指定したユーザのジョブの状態を表示します。
-f JOBID	ジョブの詳細な状況を表示します。 JOBID を省略すると、全てのジョブについての詳細な状況を表示します。
-Q	キューの状況を表示します。
-s	ジョブコメントおよびその他の情報が表示されます。 キューイング中のジョブは実行待ちになっている理由が表示されます。

5.3.5. キューの状態確認方法

本システムで設定されているキュークラスの状態を確認したい場合は、”qstat -Q” コマンドを利用します。

qstat コマンドの実行例

```
$ qstat -Q
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
debug                0     0 yes yes     0     0     0     0     0     0 Exe*
large                0     4 yes yes     3     1     0     0     0     0 Exe*
school               0     0  no  no     0     0     0     0     0     0 Exe*
```

qstat -Q コマンドにより参照可能な情報は以下の通りです。

表示の属性	説明
Max	Maximum (該当キューで実行可能なジョブ数の最大値)
Tot	Total (該当キューに投入されたジョブの総数)
Ena	Enabled (ジョブの受付可否)
Str	Started (ジョブの実行可否)
Que	Queued (待機状態のジョブ数)
Run	Running (実行状態のジョブの数)
Hld	Hold (保留状態のジョブの数)
Wat	Waiting (要求された時間まで待機状態のジョブの数)
Trn	Transit (移行状態のジョブの数)
Ext	Exiting (実行済みで終了状態のジョブの数)
Type	Execution あるいは Routing (Execution は 該当キューが実行可能であるという意味)

詳細についてはオンラインマニュアル (man qstat)、または PBS 付属の User’s Guide を参照してください。

5.3.6. ジョブのキャンセル方法

ジョブをキャンセルするには qdel コマンドを利用します。実行中のジョブ及び、待機中のジョブをキャンセルする事が可能です。qdel コマンドでキャンセル可能なジョブは自分の投入したジョブに限ります。

```
$ qdel [JOBID [JOBID…]]
```

例えば JOBID が 105 の場合、

```
$ qdel 105
```

でジョブがキャンセルされます。

6. HPE Cray Programming Environment の利用

6.1. HPE Cray Programming Environment (CPE) 概要

HPE Cray Programming Environment (CPE) は コンパイラ (CCE)、数値計算ライブラリ (CSML)、MPI 通信ライブラリ (CMPT)、開発環境設定 (CENV)、プロファイラ (CPMAT) が含まれる 統合開発環境です。
なお、CPE の利用にあたっては、コンパイル・リンクは ログインサーバ (pifront1, pifront2) 上で、ビルドしたプログラムの実行は ジョブとして、キューに投入し、計算サーバ pinode01-16 上で実行します。

6.2. 環境設定

6.2.1. ログインサーバ上の環境設定

CPE を利用するには module コマンドで PrgEnv-cray をセットします。

```
$ module restore PrgEnv-cray
```

ロードされている開発環境は `module list` コマンドを実行し、確認できます。

```
$ module list
Currently Loaded Modulefiles:
 1) cpe-cray                4) craype-x86-rome                 7) cray-mpich/8.1.3(default)
 2) cce/11.0.3(default)     5) libfabric/1.10.2pre1(default)   8) cray-libsci/20.12.1.2(default)
 3) craype/2.7.5(default)   6) craype-network-ofi
```

6.2.2. 計算サーバ上の環境設定

CPE を利用するには module コマンドで PrgEnv-cray をセットします。
また 後述の aprun コマンドを利用するため、module コマンドで cray-pals をセットします。

```
$ module restore PrgEnv-cray
$ module load cray-pals
```

6.3. コンパイル・リンク方法

6.3.1. コンパイラ CCE の利用

CCE (Cray Compiling Environment) は コンパイラ群です。
コンパイラのコマンドは以下の形式です。

言語処理系	コマンド	オンラインマニュアル
Fortran	`$ ftn [options] sourcefile`	`$ man crayftn`
C	`$ cc [options] sourcefile`	`$ man craycc`
C++	`$ CC [options] sourcefile`	`$ man crayCC`

MPI プログラムのコンパイルは、自動的にリンクされるので、明示的にリンクオプションを付ける必要はありません。
OpenMP プログラムのコンパイルはデフォルトでは無効です。明示的に指定する必要があります。

6.3.1.1. Fortran コンパイラ 主なオプション

オプション	説明
並列化・最適化など	
`-o FILENAME`	出力ファイル名を指定します。省略時は a.out が設定されます。
`-h omp`	OpenMP 指示行による並列化を有効にします。
`-h autothread`	自動並列化を行います。
`-O[0-3]`	自動最適化レベルの指定 (デフォルトは 2、最高 3)

6.3.1.2. C、C++コンパイラ 主なオプション

オプション	説明
並列化・最適化など	
`-o FILENAME`	出力ファイル名を指定します。省略時は a.out が設定されます。
`-fopenmp`	OpenMP 指示行による並列化を有効にします。(デフォルト無効)
`-O[0-3]`	自動最適化レベルの指定 (デフォルトは 2、最高 3)
`-ffast-math`	浮動小数点の誤差を一定許容することで、より踏み込んだ最適化を行います。

6.4. 数値計算ライブラリ CSML の利用

CSML (Cray Scientific and Math Libraries) は 数値計算ライブラリ群です。
以下のライブラリが利用可能です。

BLAS	ScaLAPACK
BLACS	FFTW3
LAPACK	IRT

なお、FFTW は 別モジュールとなっています。
`cray-fftw` モジュールをロードします。

```
$ module load cray-fftw
```

`module list` で `fftw` モジュールが 追加されていることを確認します。

```
$ module list
Currently Loaded Modulefiles:
 1) cpe-cray                3) craype/2.7.5(default)    5) libfabric/1.10.2pre1(default)   7) cray-mpich/8.1.3(default)   9) cray-fftw/3.3.8.8(default)
 2) cce/11.0.3(default)     4) craype-x86-rome          6) craype-network-ofi              8) cray-libsci/20.12.1.2(default)
```

`cray-libsci` や `cray-fftw` がロードされると、これらのパッケージに関する全てのヘッダとライブラリが ftn、cc、CC でのコンパイルおよびリンクに追加されます。

PrgEnv-cray モジュールに加えて 他の モジュールをロードする場合、LD_LIBRARY_PATH を以下のように設定します。

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CRAY_LD_LIBRARY_PATH
```

cray-fftw モジュールや LD_LIBRARY_PATH はジョブ実行する際のスクリプト 内においても指定ください。

6.5. プログラム実行方法

ビルドしたプログラムの実行は ジョブとして キューに投入します。
ログインサーバでは直接プログラムを実行しないようにしてください。
また、ジョブとしてプログラムを起動するには aprun コマンドを用いて起動します。
aprun コマンドでプログラムを起動しないと、計算サーバでプログラムが実行されないのでご注意ください。

6.5.1. プログラム起動コマンド aprun

```
$ aprun [options] exec_program
```

主なオプションは以下の通りです。

オプション	説明
-n 並列数	MPI 並列数を指定します。
-d スレッド数	OpenMP スレッド数を指定します。
-N ノードあたりの並列数	ノードごとに配置する MPI プロセス数を指定します。
-S ソケットあたりの並列数	CPU ソケットごとに配置する MPI プロセス数を指定します。
--cc depth	プロセス/スレッドの配置方法を指定します。 depth: -d オプションで指定したプロセス/スレッドを連続した CPU コアに固定します。

6.5.2. ジョブスクリプト例

ジョブスクリプト内の select 文では ノード数のみ指定してください。

6.5.2.1. 並列化されていない逐次ジョブ

1CPU を使用した逐次ジョブのスクリプト例

```
 1 #!/bin/bash                              bash スクリプトを宣言
 2 #PBS -q debug                            キュー名指定
 3 #PBS -l select=1                         利用するノード数を指定
 4 #PBS -N serial_JOB                       ジョブ名を指定
 5 #PBS -o serial_out_file                  標準出力（stdout）を出力するファイルを指定
 6 #PBS -j oe                               標準エラー出力を 標準出力に結合する指定
 7 module restore PrgEnv-cray               CPE 環境の呼び出し
 8 module load cray-pals                    aprun コマンドの呼び出し
 9 cd ${PBS_O_WORKDIR}                      実行ファイルがあるディレクトリへ移動
10 aprun ./a.out > log.txt 2>&1             プログラムの実行
```

6.5.2.2. スレッド並列ジョブ

OpenMP 8 スレッド並列ジョブのスクリプト例

```
#!/bin/bash
#PBS -q debug
#PBS -l select=1                                利用するノード数を指定
#PBS -N omp_JOB
#PBS -o omp_out_file
#PBS -j oe
module restore PrgEnv-cray                      CPE 環境の呼び出し
module load cray-pals                           aprun コマンドの呼び出し
export OMP_NUM_THREADS=8                        OpenMP の並列数を明示的に指定
cd ${PBS_O_WORKDIR}
aprun -d 8 --cc depth ./a.out > log.txt 2>&1    プログラムの実行
```

6.5.2.3. MPI 並列ジョブ

MPI 8 並列ジョブ (2 ノード x 4MPI) のスクリプト例

```
#!/bin/bash
#PBS -q large
#PBS -l select=2                                利用するノード数を指定
#PBS -N cmpt_JOB
#PBS -o cmpt_out_file
#PBS -j oe
module restore PrgEnv-cray                      CPE 環境の呼び出し
module load cray-pals                           aprun コマンドの呼び出し
cd ${PBS_O_WORKDIR}
aprun -n 8 -N 4 ./a.out > log.txt 2>&1          プログラムの実行
```

6.5.2.4. MPI+OpenMP ハイブリッド並列ジョブ

ハイブリッド並列ジョブ (2 ノード x 2MPI x 8 スレッド) のスクリプト例

```
#!/bin/bash
#PBS -q S
#PBS -l select=2                                利用するノード数を指定
#PBS -N hybrid_JOB
#PBS -o hybrid_out_file
#PBS -j oe
module restore PrgEnv-cray                      CPE 環境の呼び出し
module load cray-pals                           aprun コマンドの呼び出し
cd ${PBS_O_WORKDIR}
export OMP_NUM_THREADS=8                        OpenMP の並列数を指定
aprun -n 4 -N 2 -d 8 --cc depth ./a.out > log.txt 2>&1  プログラムの実行
```