極域科学コンピューターシステム
利用の手引き

2025 年 1 月

株式会社 日立製作所

変更履歴

#	日付	変更箇所	変更内容
1	2025/1/23	初版	
2			

※操作で旧システムと異なるもの（コマンド，環境変数など）は赤文字で表示しています。
なお，ハード仕様や製品名の差異は赤文字にしていません。

目次

1　システム概要

1.1 システム構成図

極域科学コンピューターシステムはフロントエンドシステム，演算システム，ファイルシステムの 3 つの部分から構成されています。システム全体の構成図を図 1.1-1 に示します。

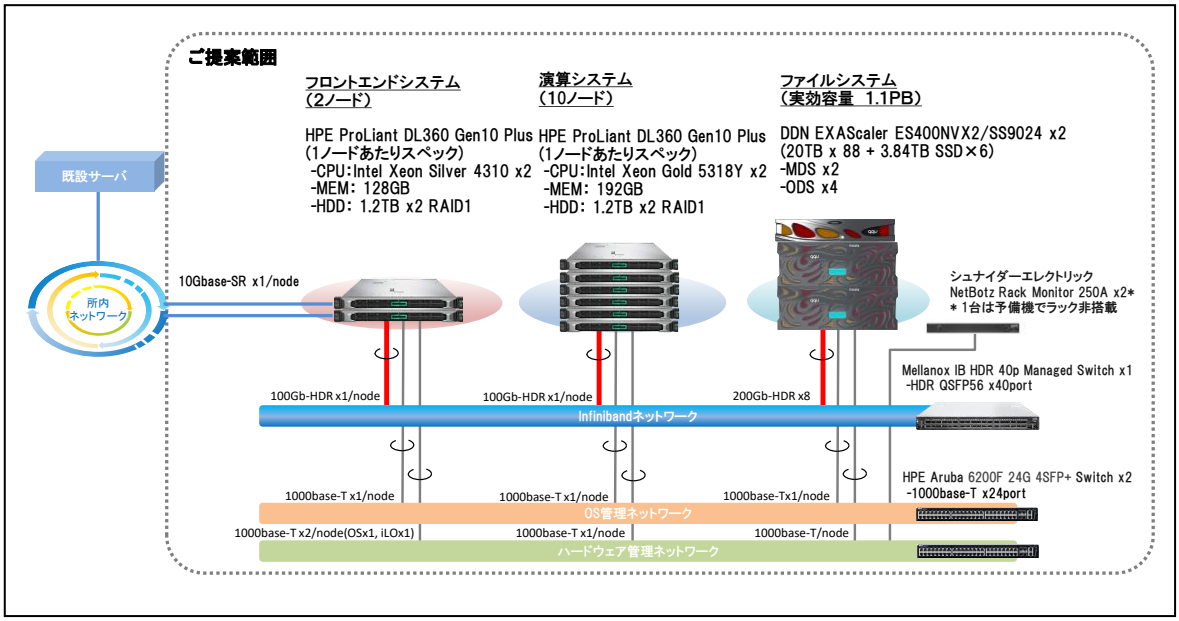


図 1.1-1 システム全体の構成図

1.2　ハードウェア構成

1.2.1 フロントエンドシステム

フロンドエンドシステムは 2 つのサーバから構成されます。2 つのサーバのハードウェア構成を表 1.2-1 に示します。

表 1.2-1　フロントエンドシステムのハードウェア構成

項番	項目	内容
1	機器名	HPE ProLiant DL360 Gen10 Plus
2	ノード数	2
3	CPU タイプ	Intel Xeon Silver 4310 開発コード名 Ice Lake (物理コア数：12，論理コア数：24)
4	CPU 数	2 ソケット/ノード
5	メモリ容量	128GB /ノード
6	ネットワーク	InfiniBand HDR /ノード

1 ノードに CPU を 2 ソケット搭載しており，各ソケットには 12 物理コアがあります。1 物理コアに 2 論理コアを起動しているため，1 ソケットの論理コアが 24 となります。

図 1.1-1 よりノード間のネットワークとして 1000base-T による接続もありますが，管理用として使用しています。ユーザがシステムを利用する際には InfiniBand HDR を使用します。そのため，表 1.2-1 を含めた以下の説明では InfiniBand HDR のみに関する説明をします。

1.2.2 演算システム

演算システムは 10 ノードから構成されます。演算システムのハードウェア構成を表 1.2-2 に示します。

表 1.2-2 演算システムのハードウェア構成

項番	項目	内容
1	機器名	HPE ProLiant DL360 Gen10 Plus
2	ノード数	10
3	CPU タイプ	Intel Xeon Gold 5318Y 開発コード名 Ice Lake (物理コア数：24，論理コア数：48)
4	CPU 数	2 ソケット/ノード
5	メモリ容量	192GB /ノード
6	ネットワーク	InfiniBand HDR /ノード

1 ノードに CPU を 2 ソケット搭載しており，各ソケットには 24 物理コアがあります。1 物理コアに 2 論理コアを起動しています。そのため，1 ノード当たり 48 物理コア(=2 ソケット×24 物理コア)，または 96 論理コア(=2 ソケット×48 論理コア）を利用できます。

1.2.3 ファイルシステム

ファイルシステムのハードウェア構成を表 1.2-3 に示します。

表 1.2-3　ファイルシステムのハードウェア構成

項番	項目	内容
1	機器名	DDN EXAScaler ES400NVX2
2	コントローラ数	2
3	ストレージ実効容量	1.197 PB

1.3　ソフトウェア構成

フロントエンドシステム，演算システム及びファイルシステムにて利用できるソフトウェア構成を説明します。

1.3.1 フロントエンドシステム

フロントエンドシステムにて利用可能なソフトウェア構成を表 1.3-1 に示します。

表 1.3-1　フロントエンドシステムのソフトウェア構成

項番	項目	内容
1	Red Hat Enterprise Linux 9.4	オペレーティングシステム
2	Lustre	共有ファイルシステム
3	PBS Professional(オープンソース版)	バッチ処理ソフトウェア
4	Intel MPI	並列ジョブ実行環境
5	Intel MKL	科学技術計算ライブラリ
6	Intel Fortran	Fortran コンパイラ
7	Intel C/C++	C,C++コンパイラ

利用者の方はフロントエンドシステムからプログラムのコンパイルやジョブのバッチ処理を実行します。詳細に関しては「２．システムへの接続」にて説明します。

1.3.2 演算システム

演算システムにて利用可能なソフトウェア構成を表 1.3-2 に示します。

表 1.3-2　演算システムのソフトウェア構成

項番	項目	内容
1	Red Hat Enterprise Linux 9.4	オペレーティングシステム
2	Lustre	共有ファイルシステム
3	PBS Professional(オープンソース版)	バッチ処理ソフトウェア
4	Intel MPI	並列ジョブ実行環境
5	Intel MKL	科学技術計算ライブラリ
6	Intel Fortran	Fortran コンパイラ
7	Intel C/C++	C,C++コンパイラ

1.3.3 ファイルシステム

ファイルシステムにて利用しているソフトウェア構成を表 1.3-3 に示します。

表 1.3-3　ファイルシステムのソフトウェア構成

項番	項目	内容
1	Linux	オペレーティングシステム
2	Lustre	共有ファイルシステム

図 1.1-1 に示すようにフロントエンドシステム，演算システムとファイルシステムはネットワークにより結合しており，また Lustre を通してファイルの共有をしています。ファイルシステム上のデータの処理はフロントエンドシステム及び演算システムにて実施できます。そのため，ファイルシステムのサーバに利用者が直接ログインすることはありません。

1.4　その他のフリーソフトウェア

上記のソフトウェアの他に，表 1.4-1 に示すフリーソフトウェアが利用可能です。

表 1.4-1　フリーソフトウェアの一覧

項番	項目	内容
1	vi	エディタ
2	emacs	エディタ
3	netpbm	画像用ライブラリ
4	gnuplot	グラフ作成ソフトウェア
5	python	プログラミング言語
6	perl	プログラミング言語
7	NetCDF	ファイル作成ソフトウェア

※項番 7 の NetCDF については 6.2 節にて説明します。

2　システムへの接続

利用者の方はフロントエンドサーバに接続して極域科学コンピューターシステムを使用することになります。以下では，フロントエンドシステムへの接続法，及びフロントエンドシステムの利用法について説明します。

2.1　フロントエンドシステムへの接続

フロントエンドシステムには表 1.2-1 に示す 2 つのサーバがあり，この内の 1 つを使用します。サーバ名と IP アドレスを表 2.1-1 に示します。

表 2.1-1　フロントエンドシステム

項番	サーバ名	IP アドレス
1	crux	133.57.64.40
2	puppis	133.57.64.41

フロントエンドシステムにログインする際には，ssh コマンドを使用して以下のように接続します。crux へのログイン方法を図 2.1-1 に示します。（アカウントが user_name の場合を例に示します。）

```
$ ssh user_name@crux
user_name@crux's password: xxxx
Last login: xxxx
[user_name@crux ~]$
```

図 2.1-1　crux へのログイン方法

2.2　パスワードの変更法

フロントエンドシステムにログインするパスワードは，”passwd”コマンドにより変更可能です。

図 2.2-1 にコマンドの実行例を記載します。ユーザが入力する項目を図 2.2-1 中の青字により示します。変更が反映されるまでに

```
$ passwd
Changing password for user アカウント名.
Current Password:  変更前のパスワードを入力してください
New password:  変更後のパスワードを入力してください
Retype new password:  もう一度変更後のパスワードを入力してください
passwd: all authentication tokens updated successfully.
$
パスワードの変更は終了です。
```

図 2.2-1　パスワードの変更法

2.3　ログインシェルの変更法

フロントエンドシステムのデフォルトのログインシェルは/bin/bash です。フロントエンドシステムにて使用可能なログインシェルを以下に記載します。

- /bin/bash
- /bin/csh
- /bin/sh
- /bin/tcsh

ログインシェルは,"chsh"コマンドにより変更可能です。図 2.3-1 にコマンドの実行例を記載します。ユーザが入力する項目を図 2.3-1 中の青字により示します。

```
$ chsh
Changing shell for アカウント名.
New shell [/bin/bash]:  変更後のログインシェルを入力してください
LDAP Bind Password:  パスワードを入力してください
SASL/GSS-SPNEGO authentication started
SASL/GSS-SPNEGO authentication started
Shell changed.
$
ログインシェルの変更は終了です。
```

図 2.3-1　フロントエンドシステムのログインシェルの変更方法

2.4　フロントエンドシステムの利用法

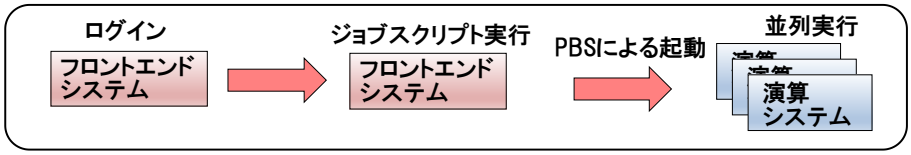
極域科学コンピューターシステムのフロントエンドシステムと演算システムはファイルシステムのディレクトリとファイルを共有していますので，フロントエンドシステムと同様に演算システムにおいてデータの処理ができます。そのため，演算システムの利用も含めて操作は全てフロントエンドシステムにて実施します。演算システムにてプログラムを実行する際は以下のようにします。

(1) プログラムのコンパイル

プログラムのコンパイルは以下のようにフロントエンドシステムにログインして実行します。

(2)プログラムの実行

コンパイルして作成したロードモジュールや実行形式のファイルを演算システムにて実行します。実行の際は以下のようにフロントエンドシステムから PBS スケジューラを通してジョブとしてサブミットすることにより演算システムにて実行します。



3　ファイルシステム構成

ディレクトリ構成とファイルシステムの構成に関して説明します。

3.1　ディレクトリの構成

ユーザが使用可能なディレクトリ構成は以下です。

/home/アカウント名

/data/アカウント名

上記はフロントエンドサーバと演算システムにて共有しています。

3.2　ファイルシステム

各ディレクトリの総容量，及び 1 ユーザの利用上限を表 3.2-1 に示します。

表 3.2-1　ファイルシステムの容量

項番	ディレクトリ	総容量	ユーザ利用上限	
			ファイル容量	ファイル数
1	/home	合計	200GB	―
2	/data	1,197TB	25TB	2,500 万

ディレクトリによりファイル容量に利用上限があるため，大容量のデータファイルは/data に作成する方が効率的となります。

4　コンパイラの利用法

極域科学コンピューターシステムでは Intel のコンパイル環境と実行環境が利用できます。この環境として Intel oneAPI 2025(以下 Intel2025)をインストールしています。以下ではコンパイラに関して説明します。

4.1　コンパイラについて

Intel のコンパイル環境では，Intel Fortran, Intel C, Intel C++の 3 種類が利用できます。コンパイラの規格として利用できる範囲を表 4.1-1 に示します。

表 4.1-1 コンパイラの規格

項番	コンパイラ	言語規格	並列関数	
			OpenMP	MPI
1	Intel Fortran	Fortran 2023	6.0	4.0
2	Intel C	C23	6.0	4.0
3	Intel C++	C++23	6.0	4.0

プログラムの並列化の方法として，逐次プログラム，OpenMP による並列プログラム，MPI による並列プログラム，MPI と OpenMP による並列プログラムの 4 種類があります。これら 4 種類のプログラムに関して表 4.1-1 に示すように Intel Fortran, Intel C , Intel C++コンパイラは対応しています。また，Intel Fortran, Intel C, Intel C++コンパイラは，OpenMP 並列化していないプログラムに対してコンパイラの自動並列化機能を適用して，OpenMP と同じように共有メモリシステムに向けた並列化ができます。このことから，プログラムとコンパイル方法の種類について纏めると各言語に対して表 4.1-2 に示す 6 種類があります。

表 4.1-2　プログラムとコンパイル方法の種類

項番	分類	MPI 並列化	OpenMP 並列化	自動並列化
1	逐次	無	無	無
2	自動並列	無	無	有
3	OpenMP 並列	無	有	無
4	MPI 並列	有	無	無
5	MPI 並列+自動並列	有	無	有
6	MPI 並列+OpenMP 並列	有	有	無

以下では，各コンパイラ毎に表 4.1-2 の 6 種類のコンパイル方法について説明します。

なお，Intel Fortran, Intel C, Intel C++コンパイラを使用する場合には共通事項として，図 4.1-1 に示すコマンドを実行して Intel のコンパイル環境を設定してからコンパイルまたは make コマンドを実行してください。

```
$ source /opt/intel/oneapi/setvars.sh
:: initializing oneAPI environment ...
   -bash: BASH_VERSION = 5.1.8(1)-release
   args: Using "$@" for setvars.sh arguments:
:: advisor -- latest
:: ccl -- latest
:: compiler -- latest
:: dal -- latest
:: debugger -- latest
:: dev-utilities -- latest
:: dnnl -- latest
:: dpcpp-ct -- latest
:: dpl -- latest
:: ipp -- latest
:: ippcp -- latest
:: mkl -- latest
:: mpi -- latest
:: pti -- latest
:: tbb -- latest
:: umf -- latest
:: vtune -- latest
:: oneAPI environment initialized ::
$
```

図 4.1-1　コンパイラの環境設定

4.2　Intel Fortran コンパイラ

Fortran プログラムのコンパイル・リンクには，Intel Fortran を使用します。

4.2.1 Intel Fortran の使用方法

MPI 関数を含まない Fortran プログラムのコンパイル・リンクには，Intel Fortran の"ifx"コマンドを使用します。プログラム sample.f をコンパイルしてロードモジュール sample.exe を作成するコマンドの例を図 4.2-1 に示します。

```
$ ifx -o sample.exe sample.f
```

図 4.2-1　MPI 関数を含まない Fortran プログラムのコンパイル例

MPI 関数を含む Fortran プログラムのコンパイル・リンクには，"mpiifx"コマンドを使用します。プログラム sample.f をコンパイルしてロードモジュール sample.exe を作成するコマンドの例を図 4.2-2 に示します。

```
$ mpiifort -o sample.exe sample.f
```

図 4.2-2　MPI 関数を含む Fortran プログラムのコンパイル例

表 4.1-2 に示したプログラムとコンパイル方法に関して表 4.2-1 に纏めます。自動並列化または OpenMP 並列化を適用する際には，コンパイルオプションとして-parallel または-qopenmp を適用します。

表 4.2-1　Fortran プログラムのコンパイル方法

項番	分類	コンパイル方法
1	逐次	ifx
2	自動並列	ifx -parallel
3	OpenMP 並列	ifx -qopenmp
4	MPI 並列	mpiifx
5	MPI 並列+自動並列	mpiifx -parallel
6	MPI 並列+OpenMP 並列	mpiifx -qopenmp

※OpenMP 並列と自動並列の併用は推奨致しません。

4.2.2 Intel Fortran コンパイラのオプション

Intel Fortran にて使用する主要なオプションを表 4.2-2 に纏めます。オプションを指定しない場合は，表 4.2-2 のデフォルトの設定が適用されます。表 4.2-2 に纏めたオプション以外に多くのオプションがあります。オプションの詳細はマニュアル［Intel® Fortran Compiler Developer Guide and Reference］をご参照ください。

表 4.2-2　主要なコンパイルオプション

項番	分類	オプション	内容	デフォルト
1	最適化	-O[0-3]	一般的な最適化オプション -O0：全ての最適化を無効化 -O3：上位の最適化レベル	-O2
2		-ax [code]	code で指定された命令セットを用いたコードを生成 code=CORE-AVX2: AVX2 命令を生成 coce=CORE-AVX512: AVX512 命令を生成	不適用 (-msse2 と同等)
3		-xHost	コンパイルを行うホスト・プロセッサーで利用可能な最上位の命令セット向けのコードを生成	不適用 (-msse2 と同等)
4		-fast	-ipo, -O3, -static, -fp-model fast=1, オプションを設定	不適用
5		-ipo	別々のファイル内で定義されている関数への呼び出しについて関数のインライン展開を実行	-no-ipo
6	浮動小数点	-fp-model= [keyword]	浮動小数点演算のセマンティクスを制御します keyword=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ます。 keyword=fast[=1|2]: 浮動小数点データにより強力な最適化を有効にします	-fp-model=fast=1
7		-prec-div	浮動小数点除算の精度を上げます	-no-perc-div
8	並列処理	-parallel	安全に並列実行できるループのマルチスレッド・コードを生成するよう自動並列化に指示します	不適用
9		-vec-threshold[0-100]	ループのベクトル化のしきい値を設定します 0：計算量にかかわらず常にベクトル化 100：100%性能向上するときのみベクトル化	-vec-threshold100
10		-qopenmp	並列化機能が OpenMP ディレクティブに基づいてマルチスレッド・コードを生成できるようにします	-qno-openmp
11		-qopenmp-simd	OpenMP の SIMD コンパイルを有効にします	-qno-openmp-simd
13	データオプション	-mcmodel= mem_model	特定のメモリーモデルでコード生成とデータ格納をします mem_model=small : コードとデータをアドレス空間の最初の 2GB までに制限 mem_model=medium：コードを最初の 2GB までに制限するが，データの制限は無し mem_model=large：コードとデータ共に制限無し COMMON ブロックとローカルデータのサイズが大きい場合，デフォルトの 2GB の制限ではプログラムのコンパイル時にリンクのエラーが発生することが有り，このオプションにより回避可能です	-mcmodel=small
14		-save	再帰ルーチン内のローカル変数と AUTOMATIC として宣言された変数を	-auto-scalar

項番	分類	オプション	内容	デフォルト
			除き，すべての変数をスタティック・メモリに保存します	
15		-auto-scalar	SAVE 属性を持たない INTEGER，REAL，COMPLEX，および LOGICAL 組込み型のスカラー変数をランタイムスタックに割り当てます	適用
16	言語オプション	-names lowercase	識別子の大文字・小文字の違いを無視し，外部名を小文字に変換するようにコンパイラに指示します	-names as_is
17		-names uppercase	識別子の大文字・小文字の違いを無視し，外部名を大文字に変換するようにコンパイラに指示します	-names as_is
18		-names as_is	コンパイラーは，識別子の大文字と小文字の違いを区別し，外部名の大文字と小文字の違いを保持します	適用
19	リンカオプション	-static	実行ファイルはライブラリをすべて静的にリンクします	ライブラリに依存
20		-static-intel	インテルが提供するライブラリをすべて静的にリンクします	適用 （ただし OpenMP の runtime library は除く）

4.2.3 Intel Fortran コンパイラの使用上の注意点

(1) コンパイルオプション -mcmodel=medium または large

プログラムのデータ量が 2GB を超える場合はコンパイルオプションに-mcmodel=medium または large を使用してください。このオプションを適用した場合，ライブラリは static link ではなく dynamic link となります。static link を使用するとエラーとなります。

(2) コンパイルオプション -ax

フロントエンドシステムと演算システムの CPU は AVX512 に対応していますので，表 4.2-2 の項番 2 の CORE-AVX512 を使用できます。ただし，プログラムにより CORE-AVX2 の方が性能向上を図れる場合がありますので，ご利用の際に調整をしてください。

(3) 計算結果の精度

最適化により計算結果の精度に影響がある場合はコンパイルオプション-fp-model precise を適用して下さい。このオプションにより浮動小数点データの精度に影響する最適化を無効にします。

4.3　Intel C コンパイラ

C プログラムのコンパイル・リンクには，Intel C コンパイラを使用します。

4.3.1 Intel C コンパイラの使用法

MPI 関数を含まない C プログラムのコンパイル・リンクには，Intel　C の”icx”コマンドを使用します。プログラム sample.c をコンパイルしてロードモジュール sample.exe を作成するコマンドの例を図 4.3-1 に示します。

```
$ icx -o sample.exe sample.c
```

図 4.3-1　MPI 関数を含まない C プログラムのコンパイル例

MPI 関数を含む C プログラムのコンパイル・リンクには，”mpiicx”コマンドを使用します。プログラム sample.c をコンパイルしてロードモジュール sample.exe を作成するコマンドの例を図 4.3-2 に示します。

```
$ mpiicx  -o sample.exe sample.c
```

図 4.3-2　MPI 関数を含む C プログラムのコンパイル例

表 4.1-2 に示したプログラムとコンパイル方法に関して表 4.3-1 に纏めます。自動並列化または OpenMP 並列化を適用する際は，コンパイルオプションとして-parallel または-qopenmp を適用します。

表 4.3-1　C プログラムのコンパイル方法

項番	分類	コンパイル方法
1	逐次	icx
2	自動並列	icx -parallel
3	OpenMP 並列	icx -qopenmp
4	MPI 並列	mpiicx
5	MPI 並列+自動並列	mpiicx -parallel
6	MPI 並列+OpenMP 並列	mpiicx -qopenmp

※OpenMP 並列と自動並列の併用は推奨致しません。

4.3.2 Intel C コンパイラのオプション

Intel C にて使用する主要なオプションを表 4.3-2 に纏めます。オプションを指定しない場合は，表 4.3-2 のデフォルトの設定が適用されます。表 4.3-2 に纏めたオプション以外に多くのオプションがあります。オプションの詳細はマニュアル［Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference］をご参照ください。

表 4.3-2　主要なコンパイルオプション

項番	分類	オプション	内容	デフォルト
1	最適化	-O[0-3]	一般的な最適化オプション -O0：全ての最適化を無効化 -O3：上位の最適化レベル	-O2
2		-ax [code]	code で指定された命令セットを用いたコードを生成 code=CORE-AVX2: AVX2 命令を生成 coce=CORE-AVX512: AVX512 命令を生成	不適用 (-msse2 と同等)
3		-xHost	コンパイルを行うホスト・プロセッサーで利用可能な最上位の命令セット向けのコードを生成	不適用 (-msse2 と同等)
4		-fast	-ipo, -O3, -static, -fp-model fast=1, オプションを設定	不適用
5		-ipo	別々のファイル内で定義されている関数への呼び出しについて関数のインライン展開を実行	-no-ipo
6	浮動小数点	-fp-model= [keyword]	浮動小数点演算のセマンティクスを制御します keyword=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ます。 keyword=fast[=1|2]: 浮動小数点データにより強力な最適化を有効にします	-fp-model=fast=1
7	並列処理	-parallel	安全に並列実行できるループのマルチスレッド・コードを生成するよう自動並列化に指示します	不適用
8		-vec-threshold[0-100]	ループのベクトル化のしきい値を設定します 0：計算量にかかわらず常にベクトル化 100：100%性能向上するときのみベクトル化	-vec-threshold100
9		-qopenmp	並列化機能が OpenMP ディレクティブに基づいてマルチスレッド・コードを生成できるようにします	-qno-openmp
10		-qopenmp-simd	OpenMP の SIMD コンパイルを有効にします	-qno-openmp-simd
11	データオプション	-mcmodel= mem_model	特定のメモリーモデルでコード生成とデータ格納をします mem_model=small : コードとデータをアドレス空間の最初の 2GB までに制限 mem_model=medium：コードを最初の 2GB までに制限するが，データの制限は無し mem_model=large：コードとデータ共に制限無し	-mcmodel=small

4.3.3 Intel C コンパイラの使用上の注意点

(1) コンパイルオプション -mcmodel=medium または large

プログラムのデータ量が 2GB を超える場合はコンパイルオプションに-mcmodel=medium または large を使用してください。このオプションを適用した場合，ライブラリは static link ではなく

dynamic link となります。static link を使用するとエラーとなります。

(2) コンパイルオプション -ax

フロントエンドシステムと演算システムの CPU は AVX512 に対応していますので，表 4.3-2 の項番 2 の CORE-AVX512 を使用できます。ただし，プログラムにより CORE-AVX2 の方が性能向上を図れる場合がありますので，ご利用の際に調整をしてください。

(3) 計算結果の精度

最適化により計算結果の精度に影響がある場合はコンパイルオプション-fp-model precise を適用して下さい。このオプションにより浮動小数点データの精度に影響する最適化を無効にします。

4.4　Intel C++コンパイラ

C++プログラムのコンパイル・リンクには，Intel C++コンパイラを使用します。Intel C++コンパイラはフロントエンドシステムにて利用可能です。

4.4.1 Intel C++コンパイラの使用法

MPI 関数を含まない C++プログラムのコンパイル・リンクには，Intel　C++の”icpx”コマンドを使用します。プログラム sample.cpp をコンパイルしてロードモジュール sample.exe を作成するコマンドの例を図 4.4-1 に示します。

```
$ icpx  -o sample.exe sample.cpp
```

図 4.4-1　MPI 関数を含まない C++プログラムのコンパイル例

MPI 関数を含む C++プログラムのコンパイル・リンクには，”mpiicpx”コマンドを使用します。プログラム sample.cpp をコンパイルしてロードモジュール sample.exe を作成するコマンドの例を図 4.4-2 に示します。

```
$ mpiicpx -o sample.exe sample.cpp
```

図 4.4-2　MPI 関数を含む C++プログラムのコンパイル例

表 4.1-2 に示したプログラムとコンパイル方法に関して表 4.4-1 に纏めます。自動並列化または OpenMP 並列化を適用する際には，コンパイルオプションとして-parallel または-qopenmp を適用します。

表 4.4-1　C++プログラムのコンパイル方法

項番	分類	コンパイル方法
1	逐次	icpx
2	自動並列	icpx -parallel
3	OpenMP 並列	icpx -qopenmp
4	MPI 並列	mpiicpx
5	MPI 並列+自動並列	mpiicpx -parallel
6	MPI 並列+OpenMP 並列	mpiicpx -qopenmp

※OpenMP 並列と自動並列の併用は推奨致しません。

4.4.2 Intel C++コンパイラのオプション

Intel C++にて使用する主要なオプションを表 4.2-2 に纏めます。オプションを指定しない場合は表 4.4-2 のデフォルトの設定が適用されます。表 4.4-2 に纏めたオプション以外に多くのオプションがあります。オプションの詳細はマニュアル［Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference］をご参照ください。

表 4.4-2　主要なコンパイルオプション

項番	分類	オプション	内容	デフォルト
1	最適化	-O[0-3]	一般的な最適化オプション -O0：全ての最適化を無効化 -O3：上位の最適化レベル	-O2
2		-ax [code]	code で指定された命令セットを用いたコードを生成 code=CORE-AVX2: AVX2 命令を生成 coce=CORE-AVX512: AVX512 命令を生成	不適用 (-msse2 と同等)
3		-xHost	コンパイルを行うホスト・プロセッサーで利用可能な最上位の命令セット向けのコードを生成	不適用 (-msse2 と同等)
4		-fast	-ipo, -O3, -static, -fp-model fast=1, オプションを設定	不適用
5		-ipo	別々のファイル内で定義されている関数への呼び出しについて関数のインライン展開を実行	-no-ipo
6	浮動小数点	-fp-model= [keyword]	浮動小数点演算のセマンティクスを制御します keyword=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ます keyword=fast[=1|2]: 浮動小数点データにより強力な最適化を有効にします	-fp-model=fast=1
7	並列処理	-parallel	安全に並列実行できるループのマルチスレッド・コードを生成するよう自動並列化に指示します	不適用
8		-vec-threshold[0-100]	ループのベクトル化のしきい値を設定します 0：計算量にかかわらず常にベクトル化 100：100%性能向上するときのみベクトル化	-vec-threshold100
9		-qopenmp	並列化機能が OpenMP ディレクティブに基づいてマルチスレッド・コードを生成できるようにします	-qno-openmp
10		-qopenmp-simd	OpenMP の SIMD コンパイルを有効にします	-qno-openmp-simd
11	データオプション	-mcmodel= mem_model	特定のメモリーモデルでコード生成とデータ格納をします mem_model=small：コードとデータをアドレス空間の最初の 2GB までに制限 mem_model=medium：コードを最初の 2GB までに制限するが，データの制限は無し mem_model=large：コードとデータ共に制限無し	-mcmodel=small

4.4.3 Intel C++コンパイラの使用上の注意点

(1) コンパイルオプション -mcmodel=medium または large

プログラムのデータ量が 2GB を超える場合はコンパイルオプションに-mcmodel=medium または large を使用してください。このオプションを適用した場合，ライブラリは static link ではなく

dynamic link となります。static link を使用するとエラーとなります。

(2) コンパイルオプション -ax

フロントエンドシステムと演算システムの CPU は AVX512 に対応していますので，表 4.4-2 の項番 2 の CORE-AVX512 を使用できます。ただし，プログラムにより CORE-AVX2 の方が性能向上を図れる場合がありますので，ご利用の際に調整をしてください。

(3) 計算結果の精度

最適化により計算結果の精度に影響がある場合はコンパイルオプション-fp-model precise を適用して下さい。このオプションにより浮動小数点データの精度に影響する最適化を無効にします。

5　最適化情報・デバッグ情報・性能情報の取得

Intel コンパイラによる最適化情報，デバッグ情報，性能情報の取得方法について説明します。Intel Fortran, Intel C, Intel C++コンパイラに共通の方法です。

5.1　最適化情報の取得

コンパイルオプションに"-qopt-report"を指定することにより，最適化レポートを出力することができます。Fortran プログラム sample.f をコンパイルして最適化レポートを出力する例を図 5.1-1 に示します。

```
$ ifx -qopt-report sample.f
```

図 5.1-1　最適化情報の取得例

最適化情報は".optrpt"の拡張子のファイルに出力されます。図 5.1-1 の例では sample.optrpt というファイルがプログラムファイルと同じディレクトリに作成されます。

"-qopt-report"は，最適化情報のレベルに応じて-qopt-report=0 から-qopt-report=5 を設定できます。図 5.1-1 に示す例はデフォルトのレベル 2 が設定され，中レベルの情報を含むレポートが生成されます。-qopt-report=0 を設定すると最適化情報は生成されません。-qopt-report=5 を設定すると最も詳細な最適化情報を出力します。

5.2　デバッグ情報の取得

プログラムの開発・デバッグ時等に有効なデバッグ情報を取得するためのコンパイルオプションがコンパイラに用意さています。デバッグ情報を取得するためのコンパイルオプションを表 5.2-1 に纏めます。

表 5.2-1 の各オプションの指定時には，"-O0"を指定することですべての最適化がオフにされ，最適化が行われる前にプログラムをデバッグ可能になります。オプションの詳細はマニュアル［Intel® Fortran Compiler Developer Guide and Referenc および Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference］をご参照ください。

表 5.2-1　デバッグ情報を取得するための主要なコンパイルオプション

項番	オプション	内容	デフォルト
1	-g	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。このオプションを指定すると，生成されるオブジェクト・ファイルにはデバッグ情報が含まれますが，アセンブリー・ファイルには含まれません。	不適用
2	-debug	詳細なデバッグ情報を生成します。	-debug none (-g を適用しない場合)
3	-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。	-notraceback

5.3　性能情報の取得

Intel 環境において性能情報の取得方法として VTune プロファイラー(旧製品名 VTune Amplifier)を使用する方法と gprof を利用する方法があります。これらの利用方法に関して説明します。なお，この方法は Intel Fortran, Intel C, Intel C++コンパイラに共通です。

5.3.1 VTune プロファイラー

Intel VTune プロファイラーにより性能情報を取得することができます。性能情報を取得する際は"-g"オプションを使用してプログラムをコンパイルします。MPI 関数を含む Fortran プログラムファイル sample.f90 をコンパイルする例を図 5.3-1 に示します。

```
$ mpiifx -g sample.f90
```

図 5.3-1 コンパイル方法

性能情報を取得する際に"vtune"コマンドを使用します。このコマンドを使用するための環境設定をする方法を図 5.3-2，図 5.3-3 に示します。実行しているシェルが bash の場合は図 5.3-2 の方法を使用し，シェルが csh，または tcsh の場合は図 5.3-3 の方法を使用してください。

```
# bash をご使用の場合
$ source /opt/intel/oneapi/vtune/latest/vtune-vars.sh
```

図 5.3-2　bash における環境設定法

```
# csh, tcsh をご使用の場合
$ source /opt/intel/oneapi/vtune/latest/vtune-vars.csh
```

図 5.3-3　csh, tcsh に置かる環境設定法

コンパイルしたロードモジュールを実行する際に"vtune"コマンドを使用して性能データを取得します。実行例を図 5.3-4 に示します。

```
$ mpirun -n 2 vtune -result-dir my_result -data-limit=0 -collect hotspots ./a.out
```

図 5.3-4　vtune を使用したロードモジュールの実行例

図 5.3-4 ではロードモジュール a.out を mpirun により実行しています。性能データはディレクトリ”my_result.実行ノード名”に出力されます。
（実際には上記のような実行ではなく，7 章の実行方法に示すように PBS スケジューラを用いて実行してください。）

ロードモジュールを実行した後，図 5.3-5 に示すように”vtune”コマンドを使用して性能データから性能情報を取得します。

```
$ vtune -report summary -report-knob show-issues=false -r my_result.実行ノード名
```

図 5.3-5　vtune を使用した性能情報の取得例

Intel VTune プロファイラーの利用に関する詳細はマニュアル［Intel® VTune™ Profiler User Guide］をご参照ください。

5.3.2 gprof

gprof によりファイルのプロファイルを取るためのコンパイルとリンク方法について Intel Fortran を例として図 5.3-6 に示します。Intel C，Intel C++における gprof の使用方法は Intel Fortran と同じです。図 5.3-6 では main.f，sub1.f，sub2.f，sub3.f からロードモジュール sample.exe を作成する例を示します。

```
$ ifx -o sample.exe -p main.f sub1.f sub2.f sub3.f
```

図 5.3-6　gprof のコンパイルとリンク方法

コンパイラにて作成したロードモジュール sample.exe の実行方法と実行した後のプロファイルの出力方法を図 5.3-7 に示します。

```
$ ./sample.exe
$ gprof sample.exe > profile.txt
```

図 5.3-7　gprof を適用したロードモジュールの実行とプロファイルの出力法

(a) 実行方法は通常の実行と同じです。ロードモジュールを実行した結果 gmon.out ファイルが生成されます。

(b) gmon.out があるディレクトリでロードモジュール名を引数として gprof を実行するとロードモジュールを実行した際のプロファイル結果が表示されます。
上記の例では結果をファイル profile.txt に出力しています。

なお，ロードモジュールを実行する際には，7 章の実行方法に示すように PBS スケジューラを用いて実行してください。

6　ライブラリの利用

Intel 環境では，CPU に最適化された BLAS, LAPACK, ScaLAPCK 等の科学技術計算ライブラリとして MKL (Math Kernel Library)が用意されています。また，極域科学コンピューターシステムにはデータ入出力ライブラリとして NetCDF をインストールしました。これらのライブラリの利用法に関して以下に説明します。

6.1　科学技術計算ライブラリ(Intel MKL)

アプリケーションと Intel MKL をリンクするにはコンパイルオプションに"-mkl"を指定します。使用機能毎の指定方法を表 6.1-1 に示します。-mkl を指定すると MKL は dynamic link されます。static link をする際にはリンクオプションとして-mkl -static-intel を指定してください。MKL の詳細はマニュアル［Developer Guide for Intel® oneAPI Math Kernel Library Linux］をご参照ください。

表 6.1-1　MKL のリンク方法

項番	使用機能	形式	コンパイルオプション
1	Fourier 変換，BLAS, LAPCK	OpenMP 並列化向けライブラリ	-qopenmp -mkl
2		シーケンシャルライブラリ	-mkl=sequential
3	擬似乱数	シーケンシャルライブラリ	-mkl=sequential
4	ScaLAPCK	MPI 並列向けライブラリ	-mkl=cluster

6.2　NetCDF(Network Common Data Form)

NetCDF は配列を用いる科学技術データの作成，アクセス，共有をサポートするデータフォーマットのソフトウェアライブラリです。

Intel Fortran 用の NetCDF を /usr/local/netcdf-fortran にインストールしました。ライブラリをリンクして使用する際には，シェルの設定ファイルに図 6.2-1 及び図 6.2-2 に示す環境変数の設定が必要です。

```
# bash を使用する場合
export NETCDFFORTRANPATH=/use/local/netcdf-fortran
export PATH=${NETCDFFORTRANPATH}/bin:${PATH}
export LD_LIBRARY_PATH=${NETCDFFORTRANPATH}/lib:${LD_LIBRARY_PATH}
```

図 6.2-1　bash における NetCDF の設定

```
# csh, tcsh を使用する場合
setenv NETCDFFORTRANPATH /use/local/netcdf-fortran
setenv PATH ${NETCDFFORTRANPATH}/bin:${PATH}
setenv LD_LIBRARY_PATH ${NETCDFFORTRANPATH}/lib:${LD_LIBRARY_PATH}
```

図 6.2-2　csh，tcsh における NetCDF の設定

Intel Fortran 用のコンパイル時の include ファイルとリンク時の library は以下にあります。

include: -I/usr/local/netcdf-fortran/include

library: -L/usr/local/netcdf-fortran/lib -lnetcdf

また，上記とは別に NetCDF のパッケージもインストールしています。こちらは Fortran，C，C++ に対応しています。こちらの NetCDF を使用する際のコンパイル時の include ファイルとリンク時の library は以下となります。

include: -I/usr/include

library: -L/usr/lib64 -lnetcdf

7　演算システムにおける実行方法

コンパイルして作成したロードモジュールまたはインストールされている実行形式のファイルといったアプリケーションを演算システムにて実行する方法を説明します。アプリケーションを演算システムにて実行するには PBS スケジューラを使用します。

7.1　PBS コマンドによる実行方法

演算システムにて実行するアプリケーションのファイル一つ一つをここではタスクとします。一つまたは複数のタスクと環境変数やファイル設定のコマンドを纏めてジョブとします。フロントエンドシステムや演算システムの PBS スケジューラではこのジョブを一つの単位として実行します。

PBS スケジューラにジョブを投入するときの例を図 7.1-1 に示します。

```
$ qsub submit.sh
1307.crux
```

図 7.1-1 qsub コマンドの例

図 7.1-1 ではジョブを投入するコマンド”qsub”により submit.sh というスクリプトを投入しています。qsub コマンドによりジョブを投入した後，ジョブ ID が戻ってきます。図 7.1-1 の例では 1307 です。スクリプトの記載方法に関しては，7.4 節にて説明します。

投入したジョブの状態を確認する時には”qstat”コマンドを使用します。引数としてジョブ ID を指定すると，該当するジョブの状態を表示します。引数としてジョブ ID を指定しない場合，投入されている全てのジョブが表示されます。引数としてジョブ ID を指定しない場合の例を図 7.1-2 に示します。

```
$ qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
1307.crux         P_hpl            sysop1           01:41:34 R P1
1308.crux         P_hpl            sysop1           01:42:04 R P1
1309.crux         P_hpl            sysop1           01:41:31 R P1
1310.crux         P_hpl            sysop1           01:41:38 R P1
1311.crux         P_hpl            sysop1           01:41:24 R P1
1312.crux         P_hpl            sysop1           01:41:41 R P1
1313.crux         P_hpl            sysop1           01:41:04 R P1
1314.crux         P_hpl            sysop1           01:42:03 R P1
1315.crux         P_hpl            sysop1           01:41:22 R P1
1316.crux         P_hpl            sysop1           01:41:55 R P1
```

図 7.1-2　qstat コマンドの例

図 7.1-2 の右から 2 番の列にジョブの状況を表示しており，”R”は実行中，”Q”はジョブをサブミットしていますが，実行待ちであることを表します。この他，“qstat -a”により経過時間の表示ができますし，”qstat -fx ジョブ ID”によりジョブ実行の詳細を見ることができます。

投入したジョブをキャンセルする際には”qdel”コマンドを使用します。qdel コマンドの引数としてジョブ ID をしていします。ジョブ ID として 1307.crux をキャンセルする例を図 7.1-3 に示します。qsub コマンドを実行したときのジョブ ID または qstat コマンドにより実行状況を確認したときのジョブ ID を指定してください。

```
$ qdel 1307.crux
```

図 7.1-3 qdel コマンドの例

7.2 実行形式の種類

4 章においてコンパイル方法を説明した際，表 4.1-2 に示す 6 種類の方法がありました。演算システムにて実行する際には，自動並列化と OpenMP による並列化は同じ実行方法ですので，区別が不要となります。そのため，分類は①逐次実行，②自動並列化または OpenMP 並列化，③MPI 並列化，④MPI 並列化と自動並列化または OpenMP 並列化の 4 種類です。ここでは自動並列や OpenMP 並列により並列化されて実行する 1 つの単位をスレッドとします。また，MPI により並列化されて実行する 1 つの単位を MPI プロセスとします。そのため，①の逐次実行と②の自動並列または OpenMP 並列の場合はスレッドを起動して実行します。③の MPI 並列化のみの場合は MPI プロセスを起動して実行します。④の MPI 並列化と自動並列化または OpenMP 並列化を併用する場合は，複数の MPI プロセスを起動し，各 MPI プロセスの中において複数のスレッドを起動することになります。

次の節では，この MPI プロセスまたはスレッドを起動して演算システムにて実行する方法について説明します。

7.3 演算システムの構成と実行方法

演算システムの構成を図 7.3-1 に示します。またノード内の CPU やメモリの構成を表 7.3-1 に示します。

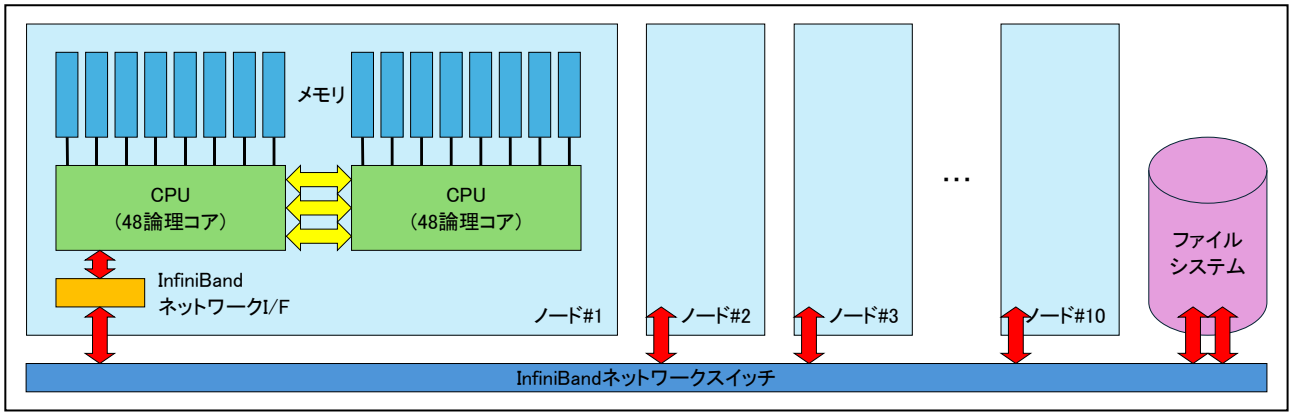


図 7.3-1 演算システムの構成

表 7.3-1　演算システムのノード構成

項番	分類	項目	数値
1	CPU ソケット Xeon Gold 5318Y	物理コア数	24
2		論理コア数	48
3		L1 キャッシュ容量[KB]	48
4		L2 キャッシュ容量/物理コア[MB]	1.25
5		L3 キャッシュ容量/ソケット[MB]	36
6	メモリ	メモリ容量/ソケット[GB]	96
7		メモリ転送性能/ソケット[GB/s]	140.7
8	ノード	CPU ソケット数	2
9		物理コア数	48
10		論理コア数	96
11		メモリ容量[GB]	192
12		メモリ転送性能[GB/s]	281.5

図 7.3-1 より 1 ノードには CPU が 2 ソケットあり，各 CPU ソケットは物理コアが 24 有ります。各物理コアには論理コアが 2 つあります。そのため，表 7.3-1 の項番 2 に示すように 1CPU ソケットには論理コアが 48 あります。また，項番 10 のように 1 ノードには論理コアが 96 あります。

図 7.3-1 より各 CPU ソケットはメモリと繋がっています。1 CPU ソケットの物理 CPU コア 24 はこのメモリ容量とメモリ転送性能を共有して使用します。また，2 つの CPU ソケット間は内部のネットワークにより結合されています。

表 7.3-1 の項番 3~5 では，CPU ソケットのキャッシュ構成を示しています。1 物理コアには L1 キャッシュが 48KB，L2 キャッシュが 1.25MB 搭載されています。この L1，L2 キャッシュを 1 物理コア中の 2 論理コアは共有して使用します。また，L3 キャッシュは 1 CPU ソケット内の 24CPU コアにより共有します。

次の節では，上記の構成の演算システムを用いて PBS ジョブを実行する方法について説明します。

7.4　実行方法

演算システムを使用して，7.2 節にて説明した 1 つのスレッドや MPI プロセスを 7.3 節にて説明した論理コアに割り当てて実行します。1 ノード内にあるすべての論理コアを使用するとアプリケーションの並列スケーラビリティやシステムの特性により，返って実行時間がながくなることがあります。ここでは，システムの特性を説明しながら，アプリケーションの実行方法の例を示します。

システム特性を考慮した論理コアの一般的な利用方法として 1 物理コアの 2 論理コアの内 1 つを使う場合と 2 つを使う場合，1 ノード内で 2 ソケットの論理コアを均等に使う場合と 1 ソケットを纏めて利用する場合があります。それぞれの利点と欠点を表 7.4-1 に示します。

表 7.4-1　ノード内論理コアの利用法

項番	実行方法	利点	欠点
1	1 物理コア中の 1 論理コアのみを利用	・1 物理コアに搭載されている L1，L2 キャッシュを 1 論理コアにて利用するため, 1 論理コアの利用キャッシュ容量が大きく，キャッシュヒット率が向上 ・1 ソケットに起動する論理コアが減ると, 1 論理コアが使用できるメモリ転送性能が大きい	・1 物理コアの 2 論理コアで実行するスレッド間で，共有するデータに関して L1，L2 キャッシュヒット率が低下
2	1 物理コアの 2 論理コアを利用	・1 物理コアの 2 論理コアで実行するスレッド間で，共有するデータのキャッシュヒット率が向上	・1 物理コアに 2 論理コアを起動するため 1 論理コア当たりの L1, L2 キャッシュ容量が少なく，キャッシュヒット率低下 ・1 ソケットに起動する論理コアが増えると, 1 論理コアが利用できるメモリ転送性能が低下
3	1 ソケットの論理コアを優先して利用	・スレッドや MPI プロセス間で L3 キャッシュやメモリを共有して実行時間を短縮 ・時間のかかるソケット間のデータ転送を最小化	・1 ソケットを重点的に利用すると各 CPU ソケットに繋がるメモリ転送性能を十分利用できない場合有り
4	2 ソケットの論理オアを均等に利用	・メモリやキャッシュの共有を極力減らすため, 各 CPU ソケットに繋がるメモリ転送性能やキャッシュ容量を使用して実行時間を短縮	・スレッドや MPI プロセス間で L3 キャッシュやメモリを共有できず実行時間が長い ・時間のかかるソケット間のデータ転送が多発

表 7.4-1 の利用法を取り入れて 7.2 にて説明した 4 つの分類の実行方法を表 7.4-2 に示します。

表 7.4-2　スレッドと MPI プロセスの割り付け方による実行方法

項番	分類	MPI 並列	自動並列または OpenMP 並列	実行方法	論理コア/物理コア	ソケット/ノード
(a)	逐次実行	無	無	1 物理コアに 1 スレッド	1	1
(b)	自動並列/OpenMP 並列	無	有	1 物理コアに 1 スレッド	1	1(優先)
(c)		無	有	1 物理コアに 2 スレッド	2	1(優先)
(d)		無	有	2 ソケットに均等配置	1 または 2	2(均等)
(e)	MPI 並列	有	無	1 物理コアの 1 スレッド	1	1(優先)
(f)		有	無	1 物理コアの 2 スレッド	2	1(優先)
(g)		有	無	2 ソケットに均等配置	1 または 2	2(均等)
(h)	MPI 並列+自動並列/OpenMP 並列	有	有	1 物理コアの 1 スレッド	1	1(優先)
(i)		有	有	1 物理コアの 2 スレッド	2	1(優先)
(j)		有	有	2 ソケットに均等配置	1 または 2	2(均等)

表 7.4-1 の項番 1 に示す 1 物理コアの 1 論理コアの利用と項番 3 の 1 ソケット優先を合わせた実行は表 7.4-2 の(a), (b), (e), (h)に相当します。表 7.4-1 の項番 2 に示す 1 物理コアの 2 論理コアの利用と項番 3 の 1 ソケット優先を合わせた実行は表 7.4-2 の(c), (f), (i)に相当します。また，表 7.4-1 の項番 4 に相当する実行は表 7.4-2 の(d), (g), (j)に相当します。

一般的な利用では，キャッシュヒット率向上による実行時間短縮と，ソケット間のデータ転送の多発による実行時間延長の回避の 2 点から，1 物理コアに 1 スレッドを割り当てる(a), (b) , (e), (h)による実行を推奨します。この実行の後，更に 1 物理コアに 2 スレッド，または 2 ソケットの均等配置による実行により実行時間の短縮の有無を確認するのが良いと考えます。

以下では(a)~(j)の 10 のケースについて PBS にて実行するジョブスクリプトの例を示しながら，スクリプトの作成方法を説明します。本システムでは Intel 環境を使用しており，(e)～(j)にて使用する MPI は Intel MPI です。以下のスクリプトでは MPI プログラムの実行に mpirun を使用します。

(a) 1 物理コアに 1 スレッドを起動する実行：逐次実行

1 物理コアの 1 論理コアに 1 スレッドを起動して実行する方法です。1 ノードを使用して実行します。ロードモジュール sample.exe を実行する PBS ジョブスケジューラ向けジョブスクリプトの例を図 7.4-1 に示します。

```
#!/bin/bash

#PBS -q P1
#PBS -l select=1:ncpus=96
#PBS -j oe

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

./sample.exe
```

図 7.4-1　1 物理コアに 1 スレッドを起動する実行：逐次実行の PBS ジョブスクリプト例

図 7.4-1 の各行のコマンドの説明は以下です。

“#PBS –q P1” にて使用する PBS のキューを P1 に指定します。

（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=1:ncpus=96” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。1 ノードを確保するため select=1 としています。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定し

ます。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
最後にアプリケーション(ここでは./sample.exe)の実行を指示します。

(b)1 物理コアに 1 スレッドを起動する実行：自動並列/OpenMP 並列

1 物理コアの 1 論理コアに 1 スレッドを起動して実行する例を説明します。1 ノードを使用して実行します。ロードモジュール sample.exe を 48 スレッドにて実行するときの PBS ジョブスケジューラ向けジョブスクリプトを図 7.4-2 に示します。

```
#!/bin/bash

#PBS -q P1
#PBS -l select=1:ncpus=96
#PBS -j oe

export OMP_STACKSIZE=16M
export KMP_AFFINITY=granuality=fine,compact,1,0
export OMP_NUM_THREADS=48

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

./sample.exe
```

図 7.4-2 1 物理コアに 1 スレッドを起動して実行：自動並列/OpenMP 並列

図 7.4-2 の各行のコマンドの説明は以下です。
“#PBS –q P1” にて使用する PBS のキューを P1 に指定します。
（PBS キューについては表 7.5-3 を参照）
“#PBS -l select=1:ncpus=96”にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。1 ノードを確保するため select=1 としています。
“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。
環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足しるとアプリケーションの実行を中断することがあります。)
環境変数 KMP_AFFINITY=granuality=fine,compact,1,0 にて 1 物理コアに 1 スレッドを割り付けることを指定します。

環境変数 OMP_NUM_THREADS=48 にてスレッド数を 48 に指定します。
（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）
“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
最後にアプリケーション(ここでは./sample.exe)の実行を指示します。

(c)1 物理コアに 2 スレッドを起動する実行：自動並列/OpenMP 並列

1 物理コアの 2 論理コアに 2 スレッドを起動して実行する例を説明します。1 ノードを使用して実行します。ロードモジュール sample.exe を 48 スレッドにて実行するときの PBS ジョブスケジューラ向けジョブスクリプトを図 7.4-3 に示します。

```
#!/bin/bash

#PBS -q P1
#PBS -l select=1:ncpus=96
#PBS -j oe

export OMP_STACKSIZE=16M
export KMP_AFFINITY=granuality=fine,compact
export OMP_NUM_THREADS=48

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

./sample.exe
```

図 7.4-3　1 物理コアに 2 スレッドを起動して実行：自動並列/OpenMP 並列

図 7.4-3 の各行のコマンドの説明は以下です。
“#PBS –q P1” にて使用する PBS のキューを P1 に指定します。
（PBS キューについては表 7.5-3 を参照）
“#PBS -l select=1:ncpus=96” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。1 ノードを確保するため select=1 としています。
“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。
環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変

更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)
環境変数 KMP_AFFINITY=granuality=fine,compact にて 1 物理コアに 2 スレッドを割り付けることを指定します。
環境変数 OMP_NUM_THREADS=48 にてスレッド数を 48 に指定します。
（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）
“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
最後にアプリケーション(ここでは./sample.exe)の実行を指示します。

(d)2 ソケットにスレッドを均等に配置する実行：自動並列/OpenMP 並列
2 ソケットの論理コアにスレッドを起動して実行する例を説明します。1 ノードを使用して実行します。ロードモジュール sample.exe を 48 スレッド(1 ソケットに 24 スレッド)にて実行するときの PBS ジョブスケジューラ向けジョブスクリプトを図 7.4-4 に示します。

```
#!/bin/bash

#PBS -q P1
#PBS -l select=1:ncpus=96
#PBS -j oe

export OMP_STACKSIZE=16M
export KMP_AFFINITY=granuality=fine,scatter
export OMP_NUM_THREADS=48

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

./sample.exe
```

図 7.4-4 2 ソケットにスレッドを均等に配置する実行：自動並列/OpenMP 並列

図 7.4-4 の各行のコマンドの説明は以下です。
“#PBS –q P1” にて使用する PBS のキューを P1 に指定します。
（PBS キューについては表 7.5-3 を参照）
“#PBS -l select=1:ncpus=96” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有しま

す。1 ノードを確保するため select=1 としています。
“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。
環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)
環境変数 KMP_AFFINITY=granuality=fine,scatter にて物理コア，キャッシュ，メモリの共有を極力さけるようにスレッドを起動します。そのため，2 つのソケットを均等に使用する配置にてスレッドを起動します。
環境変数 OMP_NUM_THREADS=48 にてスレッド数を 48 に指定します。
（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）
“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
最後にアプリケーション(ここでは./sample.exe)の実行を指示します。

(e)1 物理コアに 1 スレッドを起動する実行：MPI 並列
1 物理コアの 1 論理コアに 1 スレッドを起動する MPI 並列実行の例を説明します。ロードモジュール sample.exe を 1 ノードに 48MPI プロセス，2 ノードで 96MPI プロセスを起動して実行するときの PBS ジョブスケジューラ向けジョブスクリプトを図 7.4-5 に示します。

```
#!/bin/bash

#PBS -q P2
#PBS -l select=2:ncpus=96:mpiprocs=48
#PBS -j oe

export OMP_STACKSIZE=16M
#export I_MPI_DAPL_TRANSLATION_CACHE=on
#export I_MPI_DAPL_RDMA_RNDV_WRITE=on
export I_MPI_PIN_PROCESSOR_LIST=allcores

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

mpirun -n 96 ./sample.exe
```

図 7.4-5　1 物理コアに 1 スレッドを起動する実行：MPI 並列

図 7.4-5 の各行のコマンドの説明は以下です。

“#PBS –q P2” にて使用する PBS のキューを P2 に指定します。

（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=2:ncpus=96:mpiprocs=48” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。mpiprocs=48 として 1 ノードに起動する MPI プロセス数を指定します。select=2 により 96 論理コアを確保して，MPI プロセス数 48 となるノードを 2 ノード指定します。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)

環境変数 I_MPI_DAPL_TRANSLATION_CACHE=on (通信時にキャッシュを利用する)，および，環境変数 I_MPI_DAPL_RDMA_RNDV_WRITE=on (データ長の長い通信時(Rendezvous 通信時)に他のノードのメモリに直接アクセスする)は、DAPL(Direct Access Programming Library)のサポートが廃止されたため、指定しません。

環境変数 I_MPI_PIN_PROCESSOR_LIST=allcores にて，すべての物理コアの番号を定義します。この番号に合わせて MPI プロセスを起動します。(I_MPI_PIN_PROCESSOR_LIST は Intel MPI の環境変数です。)

“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。

“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。

mpirun -n 96 ./sample.exe にて MPI 並列化されたロードモジュールを実行します。-n にて実行する全 MPI プロセス数を指定します。ここでは 96(=48MPI プロセス数/ノード×2 ノード)となります。

(f)1 物理コアに 2 スレッドを起動する実行：MPI 並列

1 物理コアの 2 論理コアに 2 スレッドを起動する MPI 並列実行の例を説明します。ロードモジュール sample.exe を 1 ノードに 96MPI プロセス，2 ノードで 192MPI プロセスを起動して実行するときの PBS ジョブスケジューラ向けジョブスクリプトを図 7.4-6 に示します。

```
#!/bin/bash

#PBS -q P2
#PBS -l select=2:ncpus=96:mpiprocs=96
#PBS -j oe

export OMP_STACKSIZE=16M
#export I_MPI_DAPL_TRANSLATION_CACHE=on
#export I_MPI_DAPL_RDMA_RNDV_WRITE=on
export I_MPI_PIN_PROCESSOR_LIST=all

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

mpirun -n 192 ./sample.exe
```

図 7.4-6　1 物理コアに 2 スレッドを起動する実行：MPI 並列

図 7.4-6 の各行のコマンドの説明は以下です。

“#PBS –q P2” にて使用する PBS のキューを P2 に指定します。

（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=2:ncpus=96:mpiprocs=96” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。mpiprocs=96 として 1 ノードに起動する MPI プロセス数を指定します。select=2 により 96 論理コアを確保して，MPI プロセス数 96 となるノードを 2 ノード指定します。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)

環境変数 I_MPI_DAPL_TRANSLATION_CACHE=on (通信時にキャッシュを利用する)，および，環境変数 I_MPI_DAPL_RDMA_RNDV_WRITE=on (データ長の長い通信時(Rendezvous 通信時)に他のノードのメモリに直接アクセスする)は、DAPL(Direct Access Programming Library)のサポートが廃止されたため、指定しません。

環境変数 I_MPI_PIN_PROCESSOR_LIST=all にて，すべての論理コアの番号を定義します。この番号に合わせて MPI プロセスを起動します。(I_MPI_PIN_PROCESSOR_LIST は Intel MPI の環境変数です。)

“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。

“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。

mpirun -n 192 ./sample.exe にて MPI 並列化されたロードモジュールを実行します。-n にて実行する全 MPI プロセス数を指定します。ここでは 192(=96MPI プロセス数/ノード×2 ノード)となります。

(g)2 ソケットにて均等に MPI プロセスを配置する実行：MPI 並列

2 ソケットの論理コアに MPI プロセスを起動して実行する例を説明します。ロードモジュール sample.exe を 1 ノードに 48MPI プロセス(各ソケットに 24MPI プロセス)，2 ノードで 96MPI プロセスを起動して実行するときの PBS ジョブスケジューラ向けジョブスクリプトを図 7.4-7 に示します。

```
#!/bin/bash

#PBS -q P2
#PBS -l select=2:ncpus=96:mpiprocs=48
#PBS -j oe

export OMP_STACKSIZE=16M
export OMP_NUM_THREADS=1
#export I_MPI_DAPL_TRANSLATION_CACHE=on
#export I_MPI_DAPL_RDMA_RNDV_WRITE=on
export I_MPI_PIN_PROCESSOR_LIST=all
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PIN_ORDER=scatter

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

mpirun -n 96 ./sample.exe
```

図 7.4-7　2 ソケットにて均等に MPI プロセスを配置する実行：MPI 並列

図 7.4-7 の各行のコマンドの説明は以下です。

“#PBS –q P2” にて使用する PBS のキューを P2 に指定します。

（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=2:ncpus=96:mpiprocs=48” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。mpiprocs=48 として 1 ノードに起動する MPI プロセス数を指定します。select=2 により 96 論理コアを確保して，MPI プロセス数 80 となるノードを 2 ノード指定します。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)

環境変数 OMP_NUM_THREADS=1 にてスレッド数を 1 に指定します。ここでは MPI プロセスのみを起動するため OpenMPI スレッド数を 1 にします。
（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）
環境変数 I_MPI_DAPL_TRANSLATION_CACHE=on（通信時にキャッシュを利用する），および，環境変数 I_MPI_DAPL_RDMA_RNDV_WRITE=on（データ長の長い通信時(Rendezvous 通信時)に他のノードのメモリに直接アクセスする)は、DAPL(Direct Access Programming Library)のサポートが廃止されたため、指定しません。
環境変数 I_MPI_PIN_PROCESSOR_LIST=all にて，すべての論理コアの番号を定義します。この番号に対して 1 つの MPI プロセスを起動します。
環境変数 I_MPI_PIN_DOMAIN=omp により 1 MPI プロセスと，そこから起動するスレッドを 1 つのドメインとして扱います。また，この環境変数により 1 MPI プロセスが持つスレッド数が OpenMP と同じであることを定義します。
寛容変数 I_MPI_PIN_ORDER=scatter によりメモリやキャッシュの共有を避けることを優先して MPI プロセスを論理コアに配置します。そのため，ソケットに均等になるように MPI プロセスが配置されます。(I_MPI_PIN_PROCESSOR_LIST, I_MPI_PIN_ORDER は Intel MPI の環境変数です。)
“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
mpirun -n 96 ./sample.exe にて MPI 並列化されたロードモジュールを実行します。-n にて実行する全 MPI プロセス数を指定します。ここでは 96(=48MPI プロセス数/ノード×2 ノード)となります。

(h)1 物理コアに 1 スレッドを起動する実行：MPI 並列+自動並列/OpenMP 並列

1 物理コアの 1 論理コアに 1 スレッドを起動する MPI 並列＋自動並列/OpenMP 並列の実行の例を説明します。ロードモジュール sample.exe を 1 ノード 4MPI プロセスとして各 MPI プロセスに 12 スレッドを起動します。これを 2 ノードで 8MPI プロセスとして実行するときの PBS スケジューラ向けジョブスクリプトを図 7.4-8 に示します。

```
#!/bin/bash

#PBS -q P2
#PBS -l select=2:ncpus=96:mpiprocs=4
#PBS -j oe

export OMP_STACKSIZE=16M
export OMP_NUM_THREADS=12
#export I_MPI_DAPL_TRANSLATION_CACHE=on
#export I_MPI_DAPL_RDMA_RNDV_WRITE=on
export I_MPI_PIN_DOMAIN=omp:platform
export I_MPI_PIN_ORDER=range

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

mpirun -n 8 ./sample.exe
```

図 7.4-8　1 物理コアに 1 スレッドを起動する実行：MPI 並列+自動並列/OpenMP 並列

図 7.4-8 の各行のコマンドの説明は以下です。

“#PBS –q P2” にて使用する PBS のキューを P2 に指定します。
（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=2:ncpus=96:mpiprocs=4” にて 1 ノードの 80 論理コアを指定することにより，1 ノードを占有します。mpiprocs=4 として 1 ノードに起動する MPI プロセス数を指定します。select=2 により 96 論理コアを確保して，MPI プロセス数 4 となるノードを 2 ノード指定します。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)

環境変数 OMP_NUM_THREADS=12 にてスレッド数を 12 と指定します。
（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）

環境変数 I_MPI_DAPL_TRANSLATION_CACHE=on (通信時にキャッシュを利用する)，および，環境変数 I_MPI_DAPL_RDMA_RNDV_WRITE=on (データ長の長い通信時(Rendezvous 通信時)に他のノードのメモリに直接アクセスする)は、DAPL(Direct Access Programming Library)のサポートが廃止されたため、指定しません。

環境変数 I_MPI_PIN_DOMAIN=omp:platform により 1 MPI プロセスと，そこから起動するスレッドを 1 つのドメインとして扱います。また，この環境変数により 1 MPI プロセスが持つスレッド数が OpenMP と同じであることを定義します。ここでは OMP_NUM_THREADS=12 により OpenMP のスレッド数を 12 に設定しています。また，I_MPI_PIN_DOMAIN=platform を指定しているため，1MPI

プロセス内のスレッドが論理コア番号に沿って起動することを指示しています。1 ノード内にはソケット 0 とソケット 1 の 2 つのソケットがあります。ソケットの 0 の 24 物理コアの各 1 論理コアが論理コア番号の 0～23，ソケット 1 の 24 物理コアの各 1 論理コアが論理コア番号の 24～47 に対応します。ソケット 0 の 20 物理コアの残りの 1 論理コアずつが論理コア番号の 48～71 に対応し，ソケット 1 の 24 物理コアの残りの 1 論理コアずつが論理コア番号の 72～95 に対応します。そのため，1 ノードに 49 スレッド以上を起動する設定にすると，1 物理コアの 2 論理コアを使用することが発生します。
環境変数 I_MPI_PIN_ORDER=range により MPI プロセスが論理コア番号に沿って起動することを指示しています。環境変数 I_MPI_PIN_DOMAIN=platform によりスレッドを論理コア番号に沿って起動しており，そのスレッドを含む MPI プロセスの単位が論理コア番号に沿うように並べます。そのため，I_MPI_PIN_DOMAIN=omp:platform, I_MPI_PIN_ORDER=range により，図 7.4-8 のスクリプトの MPI プロセスとスレッドの配置は表 7.4-3 となります。

表 7.4-3　図 7.4-8 のスクリプトによる MPI プロセスとスレッドの配置

No.	MPI プロセス	ノード	ソケット	論理コア番号
1	0	node01	0	0~11
2	1		0	12~23
3	2		1	24~35
4	3		1	36~47
5	4	node02	0	0~11
6	5		0	12~23
7	6		1	24~35
8	7		1	36~47

(上記の I_MPI_PIN_DOMAIN, I_MPI_PIN_ORDER は Intel MPI の環境変数です。)
“cd ${PBS_O_WORKDIR}” にてジョブを投入した時のカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
mpirun -n 8 ./sample.exe にて MPI 並列化されたロードモジュールを実行します。-n にて実行する全 MPI プロセス数を指定します。ここでは 8(=4MPI プロセス数/ノード×2 ノード)となります。

(i)1 物理コアに 2 スレッドを起動する実行：MPI 並列+自動並列/OpenMP 並列
1 物理コアの 2 論理コアに 2 スレッドを起動する MPI 並列＋自動並列/OpenMP 並列の実行の例を説明します。ロードモジュール sample.exe を 1 ノード 8MPI プロセスとして各 MPI プロセスに 12 スレッドを起動します。これを 2 ノードで 16MPI プロセスとして実行するときの PBS スケジューラ向けジョブスクリプトを図 7.4-9 に示します。

```
#!/bin/bash

#PBS -q P2
#PBS -l select=2:ncpus=96:mpiprocs=8
#PBS -j oe

export OMP_STACKSIZE=16M
export OMP_NUM_THREADS=12
#export I_MPI_DAPL_TRANSLATION_CACHE=on
#export I_MPI_DAPL_RDMA_RNDV_WRITE=on
export I_MPI_PIN_DOMAIN=omp:compact
export I_MPI_PIN_ORDER=compact

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

mpirun -n 16 ./sample.exe
```

図 7.4-9　1 物理コアに 2 スレッドを起動する実行：MPI 並列+自動並列/OpenMP 並列

図 7.4-9 の各行のコマンドの説明は以下です。

“#PBS –q P2” にて使用する PBS のキューを P2 に指定します。
（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=2:ncpus=96:mpiprocs=8” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。mpiprocs=8 として 1 ノードに起動する MPI プロセス数を指定します。select=2 により 96 論理コアを確保して，MPI プロセス数 8 となるノードを 2 ノード指定します。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)

環境変数 OMP_NUM_THREADS=12 にてスレッド数を 12 と指定します。
（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）

環境変数 I_MPI_DAPL_TRANSLATION_CACHE=on (通信時にキャッシュを利用する)，および，環境変数 I_MPI_DAPL_RDMA_RNDV_WRITE=on (データ長の長い通信時(Rendezvous 通信時)に他のノードのメモリに直接アクセスする)は、DAPL(Direct Access Programming Library)のサポートが廃止されたため、指定しません。

環境変数 I_MPI_PIN_DOMAIN=omp により 1 MPI プロセスとそこから起動するスレッドを 1 つのドメインとして扱います。また，この環境変数により 1 MPI プロセスが持つスレッド数が OpenMP と同じであることを定義します。ここでは OMP_NUM_THREADS=12 により OpenMP スレッド数を 12 に設定しています。I_MPI_PIN_DOMAIN=compact を指定しているため，1MPI プロセス内のスレッド

は 1 物理コアに 2 スレッド起動するように指示しています。1 ノード内にはソケット 0 とソケット 1 の 2 つのソケットがあります。ソケット 0 の論理コア番号は 0~23 と 48~71 であり，ソケット 1 の論理コア番号は 24~47 と 72~95 です。

I_MPI_PIN_ORDER=compact は隣接する MPI プロセスのドメイン(スレッドを含む 1 MPI プロセス)がキャッシュやメモリを極力共有するように同じソケット内に配置します。そのため，I_MPI_PIN_DOMAIN=omp:compact, I_MPI_PIN_ORDER=compact により図 7.4-9 のスクリプトの MPI プロセスとスレッドの配置は表 7.4-4 となります。

表 7.4-4 図 7.4-9 のスクリプトによる MPI プロセスとスレッドの配置

No.	MPI プロセス	ノード	ソケット	論理コア番号
1	0	node01	0	0~5, 48~53
2	1			6~11, 54~59
3	2			12~17, 60~65
4	3			18~23, 66~71
5	4		1	24~29, 72~77
6	5			30~35, 78~83
7	6			36~41, 84~89
8	7			42~47, 90~95
9	8	node02	0	0~5, 48~53
10	9			6~11, 54~59
11	10			12~17, 60~65
12	11			18~23, 66~71
13	12		1	24~29, 72~77
14	13			30~35, 78~83
15	14			36~41, 84~89
16	15			42~47, 90~95

（上記の I_MPI_PIN_DOMAIN，I_MPI_PIN_ORDER は Intel MPI の環境変数です。）

“cd ${PBS_O_WORKDIR}” はジョブを投入したカレントディレクトリに移動することを指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。

“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。

mpirun -n 16 ./sample.exe にて MPI 並列化されたロードモジュールを実行します。-n にて実行する全 MPI プロセス数を指定します。ここでは 16(=8MPI プロセス数/ノード×2 ノード)となります。

(j) 2 ソケットにて均等に MPI プロセスを配置する実行：MPI 並列+自動並列/OpenMP 並列

2 ソケットの論理コアに MPI プロセスを起動する MPI 並列＋自動並列/OpenMP 並列の実行の例を説明します。ロードモジュール sample.exe を 1 ノードに 4MPI プロセス(各ソケットに 2MPI プロセス)として各 MPI プロセスに 6 スレッドを起動します。これを 2 ノードで 8MPI プロセスを起動して実行するときの PBS スケジューラ向けジョブスクリプトを図 7.4-10 に示します。

```
#!/bin/bash

#PBS -q P2
#PBS -l select=2:ncpus=96:mpiprocs=4
#PBS -j oe

export OMP_STACKSIZE=16M
export OMP_NUM_THREADS=6
#export I_MPI_DAPL_TRANSLATION_CACHE=on
#export I_MPI_DAPL_RDMA_RNDV_WRITE=on
export I_MPI_PIN_DOMAIN=omp:platform
export I_MPI_PIN_ORDER=scatter

cd ${PBS_O_WORKDIR}
source /opt/intel/oneapi/setvars.sh

mpirun -n 8 ./sample.exe
```

図 7.4-10　2 ソケットにて均等に MPI プロセスを配置する実行：MPI 並列+自動並列/OpenMP 並列

図 7.4-10 の各行のコマンドの説明は以下です。

“#PBS –q P2” にて使用する PBS のキューを P2 に指定します。

（PBS キューについては表 7.5-3 を参照）

“#PBS -l select=2:ncpus=96:mpiprocs=4” にて 1 ノードの 96 論理コアを指定することにより，1 ノードを占有します。mpiprocs=4 として 1 ノードに起動する MPI プロセス数 4 を指定します。select=2 により 96 論理コアを確保して，MPI プロセス数 4 となるノードを 2 ノード指定します。

“#PBS –j oe” により標準エラー出力を標準出力ファイルに出力することを指示します。

環境変数 OMP_STACKSIZE=16M にてスレッドのスタックサイズをデフォルト 4MB から 16MB に変更します。(stack size が不足するとアプリケーションの実行を中断することがあります。)

環境変数 OMP_NUM_THREADS=6 にてスレッド数を 6 と指定します。

（自動並列化の場合も環境変数 OMP_NUM_THREADS にて指定します。）

環境変数 I_MPI_DAPL_TRANSLATION_CACHE=on (通信時にキャッシュを利用する)，および，環境変数 I_MPI_DAPL_RDMA_RNDV_WRITE=on (データ長の長い通信時(Rendezvous 通信時)に他のノードのメモリに直接アクセスする)は、DAPL(Direct Access Programming Library)のサポートが廃止されたため、指定しません。

環境変数 I_MPI_PIN_DOMAIN=omp:platform により 1 MPI プロセスと，そこから起動するスレッドを 1 つのドメインとして扱います。この環境変数により 1 MPI プロセスが持つスレッド数が OpenMP と同じであることを定義します。ここでは OMP_NUM_THREADS=6 により OpenMP スレッド数を 6 に設定しています。また，I_MPI_PIN_DOMAIN=platform を指定しているため，1MPI プロセス内の

スレッドが論理コア番号に沿って起動することを指示しています。1 ノード内にはソケット 0 とソケット 1 の 2 つのソケットがあります。ソケットの 0 の 24 物理コアの各 1 論理コアが論理コア番号の 0～23，ソケット 1 の 24 物理コアの各 1 論理コアが論理コア番号の 24～47 に対応します。ソケット 0 の 24 物理コアの残りの 1 論理コアずつが論理コア番号の 48～71 に対応し，ソケット 1 の 24 物理コアの残りの 1 論理コアずつが論理コア番号の 72～95 に対応します。そのため，1 ノードに 49 スレッド以上を起動する設定にすると，1 物理コアの 2 論理コアを使用することが発生します。
環境変数 I_MPI_PIN_ORDER=scatter によりメモリやキャッシュの共有を避けることを優先して MPI プロセスを配置します。そのため，ソケットに均等になるように MPI プロセスが配置されます。環境変数 I_MPI_PIN_DOMAIN でスレッドを論理コア番号に沿って起動しており，そのスレッドを含む MPI プロセス単位の配置方法を指定しています。そのため，I_MPI_PIN_DOMAIN=omp:platform, I_MPI_PIN_ORDER=scatter により図 7.4-10 のスクリプトによる MPI プロセスとスレッドの配置は表 7.4-5 となります。

表 7.4-5　図 7.4-10 のスクリプトによる MPI プロセスとスレッドの配置

No.	MPI プロセス	ノード	ソケット	論理コア番号
1	0	node01	0	0~5
2	1		1	24~29
3	2		0	12~17
4	3		1	36~41
5	4	node02	0	0~5
6	5		1	24~29
7	6		0	12~17
8	7		1	36~41

（上記の I_MPI_PIN_DOMAIN, I_MPI_PIN_ORDER は Intel MPI の環境変数です。）
“cd ${PBS_O_WORKDIR}” にてジョブを投入したカレントディレクトリへの移動を指定します。これを省略すると，/home/user 名のホームディレクトリにて IO やロードモジュールの実行をします。
“source /opt/intel/oneapi/setvars.sh” にて Intel Fortran, Intel C/C++でコンパイルしたプログラムの実行に必要な環境変数やライブラリの設定をします。
mpirun -n 8 ./sample.exe にて MPI 並列化されたロードモジュールを実行します。-n にて実行する全 MPI プロセス数を指定します。ここでは 8(=4MPI プロセス数/ノード×2 ノード)となります。

7.5　PBS の利用法

これまでスクリプトの記載方法について説明をしました。この中で使用していました PBS のオプションやキュー名等について以下に纏めます。

7.5.1 PBS スクリプトの主要なオプション

PBS のコマンドは”#PBS”により始まります。これまで使用したオプションや有効なオプションを表 7.5-1 に示します。

表 7.5-1　PBS の主要なオプション

項番	オプション	内容
1	#PBS -q キュー名	投入するキュー名を指定します。
2	#PBS -N ジョブ名	ジョブ名を指定します。
3	#PBS -l select=1:ncpus=96	1 ノードの 96 論理コアを確保します。このジョブにより 1 ノードを占有します。
4	#PBS -l select=α:ncpus=96:mpiprocs=β	ncpus=96 により 1 ノードの 96 論理コアを確保します。この設定によりノードを占有して他のジョブとノードを共有することを避けます。 mpiprocs=β により 1 ノードに MPI プロセス数 β を起動します。 select=α の設定にてノード数 α を確保します。(合計 MPI プロセス数は α × β です。)
5	#PBS -l walltime=hh:mm:ss	実行時間を hh:mm:ss に設定します。 (これ以上時間が経過する場合は，中断します。)
6	#PBS -l mem=γgb	メモリサイズγ[GB]確保します。 (mb と記載すれば MB 単位にメモリを確保します)
7	#PBS -V	ジョブを投入したユーザの環境変数の設定をジョブに引き継ぐことを指定します。
8	#PBS -o ファイル名	標準出力の出力先のファイル名を指定します。指定が無い場合は，”ジョブ名”.O”ジョブ ID”となります。
9	#PBS -e ファイル名	標準エラー出力の出力先をファイル名に指定します。指定が無い場合は，”ジョブ名”.e”ジョブ ID”となります
10	#PBS -j oe|eo	oe：標準出力のファイルに標準エラー出力も出力します。 eo：標準エラー出力のファイルに標準出力も出力します。
11	#PBS -m	メールによる通知を指定します。
12	#PBS -M メールアドレス	電子メールの宛先を指定します。
13	#PBS -W depend=afterany:ジョブ ID	投入したジョブは先行する”ジョブ ID”のジョブが正常/異常を問わず終了後に実行されることを指定します。
14	#PBS -W depend=afterok:ジョブ ID	投入したジョブは先行する”ジョブ ID”のジョブが正常終了後に実行されることを指定します。

また，PBS に関連して使用した環境変数を表 7.5-2 に示します。

表 7.5-2　PBS の環境変数

項番	変数	内容
1	${PBS_O_WORKDIR}	ジョブを投入した際のディレクトリを示します。 cd ${PBS_O_WORKDIR}とすることによりジョブを投入したディレクトリに移動します。PBS のデフォルトの設定では，ユーザのホームディレクトリがジョブのディレクトリです。

7.5.2 キュー構成

演算システムにて実行できる PBS のキューを表 7.5-3 に示します。

表 7.5-3　PBS のキュー

項番	キュー名	ノード数制限	メモリ容量上限 (1 ノード)　[GB]	最長実行時間 [hh:mm:ss]	最大同時 実行ジョブ数
1	P1	1	180	24:00:00	10
2	P2	2	180	24:00:00	5
3	P3	4	180	24:00:00	2
4	P4	8	180	24:00:00	1

PBS のジョブスクリプトでは”#PBS -q”の後にキュー名を指定します。このキュー名を指定することにより，ジョブに対して表 7.5-3 のノード数制限，メモリ容量上限，最長実行時間と最大同時実行ジョブ数が適用されます。使用するノード数に応じて PBS ジョブスクリプトのキュー名を変更してください。最長実行時間が設定されていまので，この時間よりも長くなるジョブはノード数を増やして実行してください。

7.5.3 PBS ジョブの実行順序

PBS ジョブの実行順序は原則として投入された順番に実行します。その上で，演算システムのノードを効率的に利用するため，その時点で空いている演算システムのノードで実行可能な他のジョブを優先して実行することがあります。この優先するジョブはジョブの最長実行時間を基に選択されます。最長実行時間は，表 7.5-3 に示すようにデフォルトの設定では 24 時間です。PBS のオプションとして表 7.5-1 の項番 5 によりジョブ毎に設定が可能です。最長実行時間の設定例を図 7.5-1 に示します。

```
#PBS -l walltime = 00:00:50          #50 秒
#PBS -l walltime = 00:10:00          #10 分
#PBS -l walltime = 01:00:00          #1 時間
```

図 7.5-1　最長実行時間の設定例

演算システムに数ノードの空き時間ができた場合，投入されているジョブの中から，実行順序を変えて最長実行時間の短いジョブを先に空いているノードにて実行することがあります。このように最長実行時間を適切に設定することにより，先にジョブを実行することがあるため，短い期間にジョブの結果を得ることができます。そのため，ジョブ毎に”#PBS -l walltime=”の設定をされることを推奨します。

7.5.4 メールによる通知

投入したジョブの実行開始，終了[正常/異常]を電子メールにて通知することができます。qsub コマンドの “-m”オプションまたはジョブスクリプトの PBS オプションにて指定します。指定方法を表 7.5-4 に纏めます。

表 7.5-4　PBS のメールによるジョブの開始/終了通知

項番	コマンドライン	ジョブスクリプト記述	内容
1	-m a	#PBS -m a	異常終了時に通知
2	-m b	#PBS -m b	実行開始時
3	-m e	#PBS -m e	正常終了時
4	-M メールアドレス	#PBS -M メールアドレス	電子メールの宛先を指定します。

※指定が重なった場合，ジョブスクリプトの記述よりコマンドラインの指定が優先されます。

通知するメールの宛先を省略した場合はジョブを投入したホストの/var/spool/mail に配送されます。

7.6　環境変数

これまでに説明したスクリプトの中で使用した環境変数を以下に纏めます。

7.6.1 自動並列または OpenMP 並列の環境変数

Intel 環境を使用していますので, 通常の OpenMP のように OMP_から始ます環境変数の他に KMP_から始まる環境変数があります。スクリプト内にて使用した環境変数を表 7.6-1 に纏めます。他にも多くの環境変数がありますので，詳細はマニュアル［Intel® Fortran Compiler Developer Guide and Reference，及び Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference]をご参照ください。

表 7.6-1　自動並列/OpenMP 並列の主要な環境変数

項番	環境変数	内容
1	OMP_NUM_THREADS	自動並列や OpenMP 並列領域で使用する最大スレッド数を設定します
2	OMP_STACKSIZE	スレッドのプライベート・スタックとして使用する各 OpenMP スレッドに割り当てるバイト数を設定します(自動並列化の際も使用します) (推奨するサイズは 16M です)
3	KMP_AFFINITY	論理コアにスレッドをバインドします（自動並列化の際も使用します) granularity=fine: 最も細かい論理コアの粒度でスレッドを割り付けることを指定します compact：フリー・スレッド・コンテキストの OpenMP スレッド <*n*>+1 は，OpenMP スレッド <*n*> が割り当てられたスレッド・コンテキストにできる限り近いスレッド・コンテキストに割り当てられます scatter：compact の逆であり，システム全体にわたってスレッドが均等に分配されます

7.6.2 MPI に関する環境変数とオプション

Intel MPI では，MPI プロセスやスレッドを CPU の論理コアに割り当てる際に環境変数を使用しています。MPI 並列のみを使用する場合は，各 MPI プロセスを論理コアに配置する方法です。また，MPI 並列と自動並列または OpenMP 並列を使用する場合は，1 つの MPI プロセスから起動されるスレッドを 1 つの纏まりとして，MPI プロセス単位に論理コアに配置する方法です。Intel MPI に関する主要な環境変数を表 7.6-2 に纏めます。旧システムで使用していた I_MPI_DAPL から始まる環境変数(表 7.6-2 の項番 1,2)は，DAPL(Direct Access Programming Library)のサポート廃止により廃止になりました。指定した場合は MPI プログラムを実行した時に警告メッセージが出力されます。後継として I_MPI_SHM から始まる環境変数(表 7.6-2 の項番 7,他)がありますが，システムで最適値が自動選択されるため指定する必要はありません。他にも多くの環境変数がありますので, 詳細はマニュアル［Intel® MPI Library Developer Reference for Linux* OS]をご参照ください。

表 7.6-2　MPI に関する主要な環境変数

項番	環境変数	内容
1	~~I_MPI_DAPL_TRANSLATION_CASHE~~	~~DAPL 通信にてパスのメモリ登録キャッシュの on/off を指定します~~(DAPL 廃止のため環境変数も廃止)
2	~~I_MPI_DAPL_RDMA_RNDV_WRITE~~	~~DAPL 通信のパスで rendezvous 通信の際に RDMA によりデータを直接書き込むことのお on/off を指定します~~(DAPL 廃止のため環境変数も廃止)
3	I_MPI_PIN_PROCESSOR_LIST	CPU 内の MPI プロセスのマッピング規則を定義します all: 全ての論理コアを指定します allcores：すべての物理コアを指定します（デフォルト値です） (1 物理コアに 1 論理コアしか起動しないハイパースレッディングが無効な場合は，allcores と all は等価です
4	I_MPI_PIN_DOMAIN	ノード上の論理コアがオーバーラップしないサブセット(ドメイン)を定義し，ドメインあたり 1 つの MPI プロセスにすることで，ドメインへ MPI プロセスをバインドするルールを設定します (I_MPI_PIN_DOMAIN 環境変数が定義されている場合，MPI プロセスは I_MPI_PIN_PROCESSOR_LIST 環境変数の設定は無視されます) omp：ドメインサイズは，OMP_NUM_THREADS 環境変数の値を設定します platform：ドメインのメンバは，BIOS で定義される番号付けに従って並べられます
5	I_MPI_PIN_ORDER	I_MPI_PIN_DOMAIN 環境変数の値で指定されたドメインへ MPI プロセスの順番割り当てを定義します range：MPI プロセスは BIOS で定義される番号付けに従って並べられます compact：隣接するドメインが共有リソース(FSB(バス)，キャッシュ及び物理コア)を最大限に共有するようにドメインが配置されます（設定が無い場合のデフォルト値です） scatter：compact とは逆に隣接するドメインが共有リソースを最小限に共有するようにドメインが配置されます
6	I_MPI_FABRICS	MPI で使用するファブリックを指定します shm:ノード内通信で共有メモリを使用する ofi:OpenFabrics Interfaces ネットワーク(InfiniBand)を使用する shm:ofi　両方を使用する(デフォルト値)
7	I_MPI_SHM	ノード内通信で使用する共有メモリ通信方法を指定します auto:自動選択(デフォルト値) icx: IceLake に最適化された通信方法を選択する disable | no | off | 0: 共有メモリ通信を使用しない

また，(e)～(j)のスクリプトでは mpirun のオプションとして表 7.6-3 を使用しました。他のオプション等，詳細はマニュアル [Intel® MPI Library Developer Reference for Linux* OS]をご参照ください。

表 7.6-3　mpirun のオプション

項番	オプション	内容
1	-n	起動する全 MPI プロセス数を指定します

以上