# ユーザーマニュアル

# 開発環境編

# 目次

# 1　はじめに

この度は当社計算機をお買い求め頂きまして誠にありがとうございます。

本マニュアルでは当社計算機で使用する開発環境について概説します。当社の計算機は製品ごとにインストールされている環境は異なります。通常、本マニュアルで紹介する全ての開発環境はインストールされておりません。

本マニュアルをご覧の際は計算機にインストールされているパッケージをご確認の上、ご使用になる開発環境の項目をご参照頂くようお願い致します。

# 2 Intel oneAPI Toolkit

## 2.1 インストールディレクトリ

2023 年現在、Intel 社製アプリケーション開発用ソフトウェアは、Intel oneAPI Toolkit というパッケージで提供されています。Intel oneAPI Toolkit は、基本的な機能を持つ Base Toolkit と、それ以外の追加機能の Toolkit という構成で提供されています。

Intel oneAPI Toolkit のパッケージはインテル レジストレーション・センターのホームページで配布されています。ライセンスをお持ちの場合、Intel oneAPI Toolkit のパッケージを以下のサイトからダウンロードすることができます。

https://registrationcenter.intel.com/

本書では、Intel oneAPI Base Toolkit に科学技術計算用の追加機能を加えた、Intel oneAPI Base Toolkit & HPC Toolkit の組み合わせについて解説を行っていきます。

Intel oneAPI Toolkit は以下のディレクトリにインストールされています。

```
/opt/intel/oneapi
```

Intel oneAPI Toolkit は様々な機能（ツール）から構成されていますが、それぞれについて、ディレクトリ `/opt/intel/oneapi` 以下に

ツール名 / Version

というディレクトリ名でインストールされています。

具体的には、次ページのような構造となっています。

```
> cd /opt/intel/oneapi
> tree -d -L 2
.
├── advisor
|   ├── 2021.1.1
|   ├── 2021.2.0
|   └── latest -> 2021.2.0
├── ccl
|   ├── 2021.1.1
|   ├── 2021.2.0
|   └── latest -> 2021.2.0
├── clck
|   ├── 2021.1.1
|   ├── 2021.2.0
|   └── latest -> 2021.2.0
├── compiler
|   ├── 2021.1.1
|   ├── 2021.2.0
|   └── latest -> 2021.2.0
├── conda_channel

（後略）
```

advisor、ccl、clck、… がツールの種類にあたり、それぞれのツール毎に、Version 2021.1、2021.2.0 がインストールされていることが分かります。( 上記の例で latest は 2021.2.0 を参照しているため、別の Version ではありません)

Intel oneAPI Base Toolkit & HPC Toolkit には複数の開発用ソフトウェアが同梱されていますが、以降では、以下の項目に絞って解説を行っていきます。

- Intel C++ Compiler Classic
- Intel Fortran Compiler Classic

以前 Intel Compiler と呼ばれていた、C 言語、Fortran 言語のコンパイルコマンドが `icc` / `ifort` となっているコンパイラです。

- Intel oneAPI Math Kernel Library

通称、MKL と呼ばれている、科学技術計算向けの数値演算用ライブラリ群です。

- Intel MPI Library

Intel 社が開発した、MPI 計算用バイナリ・ライブラリ群です。

## 2.2 Intel oneAPI Toolkit の設定方法

Intel oneAPI Toolkit の環境設定は、Environment Module を使用して各ユーザーのホームディレクトリのファイルで行っております。設定ファイルは ユーザーのシェルスクリプトが bash の場合[1]は ~/.bashrc 、tcsh の場合は ~/.cshrc ファイルとなります。

Environment Module は、使用するアプリケーションやツールに応じて、PATH 変数や LD_LIBRARY_PATH 等の環境変数を動的に切り替えることが可能なソフトウェアです。

Intel oneAPI Toolkit には、Environment Module 用の設定ファイルが標準で用意されておりますが、HPC SYSTEMS はその設定ファイルを一部カスタマイズした設定ファイルも作成しております。

Intel oneAPI Toolkit 標準の Environment Module 設定ファイルは /opt/intel/oneapi/modulefiles に、HPC SYSTEMS でカスタマイズした設定ファイルは /opt/intel/oneapi/modulefiles-HPCS に、それぞれインストールされています。

Intel oneAPI Toolkit の仕様に関する注意事項

過去の Intel Compiler ( 商品名 Intel Parallel Studio XE )と異なり、Intel oneAPI Toolkit は、Release notes[2] に、以下のように Bash が必要と明記されています :

GNU* Bash is required for local installation and for setting up the environment to use the toolkit.

HPC SYSTEMS が検証した限り、現時点では、tcsh 環境においても本文章で解説している Environment Module を使用する手法では問題が発生しておりませんが、特に理由がない限り ユーザーのシェルスクリプトとして、bash の使用を推奨いたします。

[1] 各ユーザーが使用するシェルは "echo $0" コマンド等で確認できます。
[2] 公式 HP https://software.intel.com/content/www/us/en/develop/articles/intel-oneapi-base-toolkit-system-requirements.html

各ユーザーの .bashrc / .cshrc ファイルでは、以下のように設定されています。ユーザーのシェルが bash でも tcsh でも、Intel oneAPI Toolkit の設定方法は同じですが、bash と tcsh シェルの文法のため最初の２行だけ記述が異なります。以下では / の左側が .bashrc の場合、/ の右側が .cshrc の場合を記述しています。

```
ONEAPIVER=2021.2.0      /     set ONEAPIVER=2021.2.0
#ONEAPIVER=2021.1.1     /     #set ONEAPIVER=2021.2.0

#module use /opt/intel/oneapi/modulefiles
module use /opt/intel/oneapi/modulefiles-HPCS

module load compiler/${ONEAPIVER}
#module load mkl/${ONEAPIVER}
module load mpi/${ONEAPIVER}
```

それぞれの行は以下の意味となっています。

ONEAPIVER = .... ：Intel oneAPI Toolkit の Version。以降 module load コマンドにて、この変数に応じた Intel oneAPI Toolkit の Version を読み込んでいます。

module use .... ：Environment Module 用設定ファイルが存在するディレクトリの指定。
/opt/intel/oneapi/modulefiles ： デフォルトの設定ファイル
/opt/intel/oneapi/modulefiles-HPCS：HPC SYSTEMS 製設定ファイル

module load .... ：module load に続いて Intel oneAPI のツール名と Version を指定し、該当する設定ファイルをロードし、必要な環境設定を行います。
今回の例で指定しているツール名は以下となります。

compiler： Intel C++ / Fortran Compiler Classic
mkl：Intel oneAPI Math Kernel Library[3]
mpi：Intel MPI Library

[3] HPC SYSTEMS 製設定ファイルを使用する場合、module load compiler/${ONEAPIVER} を行うと mkl も自動的に読み込むため、個別に mkl を module load する必要はありません。

使用する Intel oneAPI Toolkit のツールの Version を切り替える場合、各ユーザーのホームディレクトリにおいて、以下のような修正を行います。

例） Intel oneAPI Toolkit の Version 2021.2.0 から 2021.1.1 に変更する場合

ユーザーのシェルが bash の場合は ~/.bashrc 内の ONEAPIVER を修正します。

| 修正前 | 修正後 |
|---|---|
| ONEAPIVER=2021.2.0 | #ONEAPIVER=2021.2.0 |
| #ONEAPIVER=2021.1.1 | ONEAPIVER=2021.1.1 |

ユーザーのシェルが tcsh の場合は ~/.cshrc 内の ONEAPIVER を修正します。

| 修正前 | 修正後 |
|---|---|
| set ONEAPIVER=2021.2.0 | #set ONEAPIVER=2021.2.0 |
| #set ONEAPIVER=2021.1.1 | set ONEAPIVER=2021.1.1 |

次回ユーザーがログインした時からコンパイラのバージョンが変わります。バージョンが変更されたことを、念のため "ifort -V" や "icc -V" コマンドでご確認下さい。

一般的には、Intel C++/Fortran Compiler Classic と MPI のように、Intel oneAPI Toolkit 内のツール群を組み合わせて使用する場合は、全て同じ Version を使用することが普通です。しかし、何らかの理由で特定のツールの Version のみを変更したい場合、それぞれのツール毎に異なる Version を使用する事も可能です。

例えば、Intel C++/Fortran Compiler Classic を Version 2020.2.0、Intel MPI を 2021.1.1 にしたい場合には .bashrc / .cshrc を以下のように修正することで可能です。

```
ONEAPIVER=2021.2.0      /     set ONEAPIVER=2021.2.0
ONEAPIVER2=2021.1.1     /     set ONEAPIVER2=2021.1.1

#module use /opt/intel/oneapi/modulefiles
module use /opt/intel/oneapi/modulefiles-HPCS

module load compiler/${ONEAPIVER}
#module load mkl/${ONEAPIVER}
module load mpi/${ONEAPIVER2}
```

Environment Module の設定ファイルの切り替えは文頭の#を切り替える事で可能となります。
（ **#がついていない行が有効になります** ）

```
#module use /opt/intel/oneapi/modulefiles
module use /opt/intel/oneapi/modulefiles-HPCS
```

 HPC SYSTEMS 製の Environment Module 用設定ファイルの改変内容

HPC SYSTEMS において修正を行った Environment Module 用設定ファイルは、
Intel oneAPI Toolkit の標準添付設定ファイルに関して以下の改変を行っています。

- Environment Module 設定ファイル内の文法の修正
  特定環境下で誤動作する表現を修正しました。動作内容に変更はありません。

- compiler モジュールから mkl モジュールを自動 load するように修正

- compiler モジュールから別コンポーネントを load する際の Version の明確化
  compiler モジュール内で load される、tbb、debugger、compiler-rt（ランタイム）について、呼び出す compiler モジュールと同じ Version のライブラリを明示的に呼び出すように修正しました[4]。

- compiler モジュールから dpl、oclfpga を自動的に load しないように変更
  標準では compiler から dpl（ Intel oneAPI DPC++ Library (oneDPL) ）、oclfpga（FPGA 向けパッケージ）を自動的に load しますが、これらは、HPC 向けアプリケーションでは使用しないため、動作の軽量化のため load しないように修正しました。

- module load した際の画面出力の抑制
  MPI 並列した場合に出力が多く煩雑になる場合があるため、一部出力を抑制しました。

[4] debugger については、compiler と Version 番号が異なるため、同一の Version の Intel oneAPI Toolkit に同梱されている Version を明示的に呼び出すように修正しました。

## 2.3 Intel C++ / Fortran Compiler Classic の使用方法

Intel C++/Fortran Compiler Classic を使いてコンパイルを行う場合には以下のコマンドを使用します。出荷設定では root 以外の全ユーザーが使用できます。

| コマンド | 内容 |
|---|---|
| ifort | Fortran 言語用コンパイラ |
| icc | C 言語用コンパイラ |
| icpc | C++ 用コンパイラ |

コンパイラのバージョンは、-V オプションで確認できます。例：ifort -V

出荷設定では icc / ifort コマンドに以下のエイリアス設定をしています。icc / ifort コマンドを使用する際、以下オプションが自動で適用されます。

| オプション | 内容 |
|---|---|
| -D_FILE_OFFSET_BITS=64 | プログラム上で 2GB を超えるサイズのファイルを取り扱うために使用します。<br>(icc で標準使用します。) |
| -D_LARGEFILE_SOURCE | プログラム上で 2GB を超えるサイズのファイルを取り扱うために使用します。<br>(icc で標準使用します。) |
| -assume buffered_io | ファイルに書き込みを行う際、ディスクにすぐに書き込まず、バッファに蓄積することを指示します。<br>(ifort で標準使用します。) |

icc / ifort でよく使用するコンパイルオプションを以下に列挙します。

| オプション | 内容 |
|---|---|
| -O0<br>-O1<br>-O2<br>-O3 | ソースコードの最適化を制御します。-O0 ～ -O3のうち、１つを選択して指定します。明示的に指定しない場合（デフォルト設定）は -O2 となります。一般的に、-O に続く数字が大きい程パフォーマンスが向上しますが、数値精度に問題が出る場合があります。<br><br>-O0: 最適化なし。パフォーマンスが非常に低下するので注意が必要となります。<br>-O1: 一部の最適化を有効化します。<br>-O2: デフォルト設定。ベクトル化が有効となります。<br>-O3: -O2 より強力な最適化を行います。 |
| -parallel | 作成された実行ファイルがスレッド並列で動作するようになります。 |
| -qopenmp | OpenMP を使用するプログラムをコンパイルする際に使用します。Intel Compiler の旧 Version では、-openmp というオプション名でしたが変更になりました。 |
| -mcmodel | 使用するメモリーモデルを指定します。<br>(例) -mcmodel=medium<br>コードサイズの上限が 2GB になり、データサイズは無制限になります。<br>(例) -mcmodel=large<br>コードサイズ、データサイズともに無制限になります。 |
| -fp-model | 浮動小数点数の扱い方を指定します。<br>(例) -fp-model strict<br>縮約を無効にし、浮動小数点数環境の変更を可能にするプロパティを有効にします。 |
| --diag-disable=10441 | 「remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated～」の注意書きの出力を抑制します。 |

## 2.4 Intel oneAPI Math Kernel Library ( MKL )の使用方法

MKL は、ベクトル演算関数、高速フーリエ変換 (FFT)等、科学技術計算向けでしばしば使用される処理を高速で処理するライブラリ群で、特に Intel 社製 CPU で実行する場合に非常に高い性能を示します。

Intel oneAPI Toolkit の仕様に関する注意事項

Intel oneAPI Math Kernel Library を始めとした Intel oneAPI Toolkit に含まれている各種ツール、ライブラリは、全て Intel 社製 CPU 向けに最適化されています。
非 Intel 社製 x86_64 互換 CPU 環境においては、Intel 社製 CPU 環境に比べてパフォーマンスが十分に発揮できない場合がありますが、こちらは Intel oneAPI Toolkit の仕様となります。

MKL の BLAS / LAPACK 等を使用する場合の使用例は以下のとおりです。

(例 1) Intel LP64 インターフェースを使用したダイナミックリンクでのコンパイル

```
ifort myprog.f -I${MKLROOT}/include -L${MKLROOT}/lib/intel64
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm -ldl
```

(例 2) Intel LP64 インターフェースを使用したスタティックリンクでのコンパイル

```
ifort myprog.f -I${MKLROOT}/include  -Wl,--start-group
${MKLROOT}/lib/intel64/libmkl_intel_lp64.a ${MKLROOT}/lib/intel64/libmkl_intel_thread.a
${MKLROOT}/lib/intel64/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm -ldl
```

※${MKLROOT}は Environment Module による環境設定時に設定される環境変数です。

MKL はビルド状況に応じて、オプションやリンクするライブラリを選択、変更する必要があり、リンクオプションは複雑です。Intel 社は公式サイトで、状況に応じたリンクオプションをナビゲートするサイト、Intel® Math Kernel Library Link Line Advisor サイトを開設しています。具体的なリンクオプションは次のサイトを参照ください。

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

## 2.5 Intel MPI Library の使用方法

Intel MPI Library は、並列・分散プロセス間のメッセージ機能を提供する MPI 規格を実装したバイナリおよびライブラリ群で、世間では Intel MPI と呼ばれていることが多いソフトウェアです（本節でも、以降、Intel MPI と呼称します）。特徴としては、以下が挙げられます。

・MPICH、MVAPICH2 をベースに Intel 社が開発を行っている。
・MPI-3.1 規格に準拠している。
・特に Intel 社製ハードウェアに向けた調整がなされている。

過去の MPI-2 規格に準拠した MPICH2、OpenMPI で動作していたソースコードは、ほとんどの場合、修正無しにそのままコンパイルできます。

なお、過去に MPICH2 や OpenMPI 等でビルドを行ったアプリケーションバイナリはそのままでは Intel MPI 環境で動作しません。Intel MPI で実行するためには、ソースコードから Intel MPI でリビルドを行う必要があります。

**本節では oneAPI HPC Toolkit に同梱されている Intel MPI について概説します。**

### 2.5.1 Intel MPI によるコンパイル

Intel MPI は、内部で使用するコンパイラの種類によってコンパイルコマンドが異なっています。過去に慣例的に使用されていた mpif90 や mpicc コマンドを使用すると、Intel Compiler C++/Fortran Classic コンパイラが使用されない点にご注意ください。

| コマンド | 内容 |
|---|---|
| mpiifort | Intel Fortran Compiler Classic ( ifort )を使用した Fortran 用コマンド |
| mpiicc | Intel C++ Compiler Classic ( icc ) を使用した C 用コマンド |
| mpiicpc | Intel C++ Compiler Classic ( icpc ) を使用した C++用コマンド |
| | |
| mpif77 | GNU Fortran Compiler g77 を使用した Fortran 用コマンド |
| mpif90 | GNU Fortran Compiler gfortran を使用した Fortran 用コマンド |
| mpicc | I_MPI_CC 変数等で指定している場合は、指定の C Compiler<br>指定の無い場合は、GNU C Compiler ( gcc )<br>を使用した C 言語用コマンド |
| mpigcc | GNU C Compiler ( gcc )を使用した C 言語用コマンド |
| mpigxx | GNU C/C++ Compiler ( g++ )を使用した C/C++言語用コマンド |

コンパイル時のオプションについては、基本的に、内部で使用されるコンパイラのオプションをそのまま使用できます。

ただし、static リンクについては注意が必要となります。近年、OS のシステムの関係や、またはアプリケーション開発の都合上、ライブラリのリンク時に shared リンク（dynamic リンク）を使用するケースが大変多くなっています。コンパイル時のオプションで static オプションを使用することは可能ですが、アプリケーションが shared リンクを想定しており対応が難しいケースや、リンクに必要なライブラリ群が shared リンク用のパッケージしかないというケースが見受けられます。このような場合、全てのライブラリを static でリンクしたバイナリの作成は困難です。このような状況下では、仮に static コマンドを使用してアプリケーションのビルドを行っても、作成したノード外で実行する、ノード環境を変更（更新）するなどのケースでは、アプリケーションが正常に動作しない場合があります。

### 2.5.2 Intel MPI によるバイナリ実行方法（実行ノード内）

Intel MPI を使用して、プログラムをローカル環境（プロンプトで作業している計算機）で実行する場合の実行方法は以下の通りです。実行には mpirun コマンドを使用します。

```
mpirun -np [並列数] [option] [実行ファイル]
```

[並列数] には、並列計算に使用するプロセスの数が入ります。通常、指定したプロセス数と計算に使用される CPU コア数は等しくなります[5]。例えば 4 コアを使用する並列計算の場合、mpirun -np 4 … のように指定します。

[option] の項目は実行時のオプションです。オプションの詳細な解説は、OS 上の man コマンドで man mpirun と実行するか、Intel MPI のドキュメントでご確認下さい。

[実行ファイル] 等、ファイル指定時にはパスの指定を正確に行う必要があります。特に同名のファイルが複数ある場合や、ノード間でプロセスを起動する場合には、安全に実行するため、ファイル指定を絶対パス等で明示的に記述することを強く推奨します。

例えば、仮に、バイナリ test を -i オプションで入力ファイル、-o オプションで出力ファイルを指定して実行するケースを考えます。バイナリ test が /usr/local/bin に存在し、入力ファイルとして /home/hpc/test_data/inp 、出力ファイルとして /home/hpc/test_out/out を指定したい場合、下記のようなコマンドになります。

```
mpirun -np 4 /usr/local/bin/test -i /home/hpc/test_data/inp
-o /home/hpc/test_out/out
```

また、バイナリ test がカレントディレクトリに存在する場合は以下のコマンドになります。

```
mpirun -np 4 ./test -i /home/hpc/test_data/inp -o /home/hpc/test_out/out
```

[5] OpenMP 等でスレッド並列を組み合わせる、いわゆる Hybrid 並列の場合、トータルで使用される CPU コア数は、（プロセス並列数）×（スレッド並列数）となります。

### 2.5.3 Intel MPI によるバイナリ実行方法（複数ノード）

Intel MPI では、前節で記載の通り、mpirun コマンド実行時に計算に使用するノードを指定しない場合、プロンプトでコマンドを入力した計算機でバイナリが実行されます。そのため、複数のノードを指定してバイナリを実行する場合には、計算に使用するノードを指定する必要があります。

例えば、カレントディレクトリにあるバイナリ test を、計算ノード node01 で 1 プロセス、計算ノード node02 で 3 プロセス実行させる場合は下記のコマンドとなります。

```
mpirun -np 1 -host node01 ./test : -np 3 -host node02 ./test
```

計算に使用する複数のノードに対して、計算プロセスを均等に割り振って実行する場合には、-ppn オプションを使用する事も可能です。例えば、カレントディレクトリのバイナリ test を、計算ノード node01、node02、node03、node04 でそれぞれ 2 プロセスずつ、計 8 プロセスで実行させる場合、次のコマンドとなります[6]。

```
mpirun -ppn 2 -hosts node01,node02,node03,node04 ./test
```

Intel MPI では、実行時に各ノードに割り振られるプロセスは CPU（コア）に Bind される設定がデフォルトとなっています。プロセスを CPU（コア）に Bind するオプションを明示的に設定する必要はありません。

計算に使用するノードをファイルで指定する事も可能です。まず、計算に使用するノード名と、それぞれのノードに割り振るプロセス数を指定したファイルを作成します。例えば、node01 に 2 プロセス、node02 に 4 プロセス、node03 に 2 プロセスを実行させる場合の設定ファイルは次のようになります。

```
node01:2
node02:4
node03:2
```

[6] -np オプションと -ppn オプションの同時使用は誤動作する場合があるため、推奨しません。

この設定ファイルを machinelist という名前で作成した後、-machinefile オプションで、そのファイルを指定します。この設定ファイルとバイナリ test がカレントディレクトリにある場合、これを実行するには次のコマンドとなります。

```
mpirun -machinefile ./machinelist ./test
```

こうすると、設定ファイルに沿ってノードにプロセスが割り振られて並列計算が行われます。上記の例の machinelist の場合では、計 8 プロセスでの並列計算が実行されます。

また、-machinefile オプションは、-np オプションと組み合わせることも可能です。具体的には次のように指定します。

```
mpirun -machinefile ./machinelist -np 4 ./test
```

この 2 つのオプションを併用した場合、 -np で指定された並列数分のプロセスを、-machinefile で指定した設定ファイルの先頭から順番に割り振る形で計算が行われます。上記の例の場合、node01、node02 で 2 プロセスずつ、計 4 プロセスでの並列計算が実行されます。

Intel MPI では、-machinefile で指定する設定ファイルを、計算実行時に自動的に参照させる機能を持っておりません。ジョブスケジューラーを使用しない場合、計算を実行する度に毎回指定する必要があります。

### 2.5.4 ジョブスケジューラーとの連携

Intel MPI は、ジョブスケジューラー( LSF / PBS / Grid Engine 等) を使用してプログラムをジョブとして投入すると、自動的に連携を行います。ジョブスケジューラーを使用して MPI ジョブを投入する方法につきましては、各ジョブスケジューラーのマニュアルをご確認下さい。

## 2.6 ドキュメント

Intel oneAPI Toolkit のドキュメントは、基本的にインターネットで Intel 社のウェブサイトにアクセスし、オンラインでマニュアルを参照する形式をとっています。そのため、計算機内の Intel oneAPI Toolkit のインストールディレクトリには、基本的にマニュアルがインストールされていません。

Intel oneAPI Toolkit のマニュアルについては以下のサイトから検索することができます。

https://www.intel.com/content/www/us/en/developer/tools/oneapi/documentation.html

また、Intel oneAPI Toolkit の代理店のエクセルソフト(株)のサイトより、各種ドキュメントが配布されています。以下 URL をご参照下さい。

https://www.xlsoft.com/jp/products/intel/tech/documents.html?tab=1

## 2.7 サポート期間

当社で販売している Intel oneAPI Toolkit の有償商品には標準で 1 年間のサポートサービスが付帯しています。サポート期間内はユーザーに以下の権利があります。

(1) Intel oneAPI Toolkit のパッケージの更新版がリリースされた際、新しいパッケージを入手して計算機にインストールすることができます。また、過去に Intel 社が販売した旧バージョンのパッケージについてもダウンロードしてインストールする事が可能です。パッケージはインテル レジストレーション・センターから配布されます。

(2) Intel 社の技術サポートサービスを受けられます。

詳細は Intel oneAPI Toolkit の代理店のエクセルソフト(株)の次のページをご覧ください。

https://www.xlsoft.com/jp/products/intel/purchase/intel_license.html?tab=1

サポート期間が期限切れを迎えた後もこれらの対応を希望する場合には、サポートサービス更新（SSR）を購入する必要があります。なお、サポート期限切れとなった後も、既に計算機にインストールされている oneAPI Toolkit 群を使用することや、コンパイルしたバイナリを使用することは可能です。

Intel oneAPI Toolkit のサポート期間はインテル レジストレーション・センターにログインして確認いただけます。また、計算機で使用されているライセンスファイルがあれば、そこにもサポート期間が記述されています。ライセンスファイルは以下をご確認下さい。

・ライセンスファイルは /opt/intel/licenses 以下に置かれています。

確認箇所は次の例を参考にしてください：

・ファイル例：(この例では 2022/2/22 までです)

```
PACKAGE IC45FB71A INTEL 2022.0222 XXXXXXXXXXXX COMPONENTS="ArBBL ¥
```

# 3 Intel Parallel Studio XE

## 3.1 インストールディレクトリ

Intel Parallel Studio XE は、2020 年まで、Intel 社から発売されていた、Intel Compiler 等の Intel 社製アプリケーション開発用ソフトウェアのパッケージです。

Intel Parallel Studio XE には複数の開発用ソフトウェアが同梱されていますが、本文章では、Intel Compiler 及び、Intel Compiler に同梱されている MKL に絞って解説を行っていきます[7]。

Intel Parallel Studio XE のパッケージはインテル レジストレーション・センターのホームページで配布されています。ライセンスをお持ちの場合、Intel Parallel Studio XE は以下のサイトからパッケージをダウンロードすることができます。

https://registrationcenter.intel.com/

Intel Parallel Studio XE は発売された西暦年が Version となっています。そこに含まれる Intel Compiler は、Intel Parallel Studio XE の Version（西暦年）の末尾２桁が Major Version となります。

例： Intel Parallel Studio XE 2018 → Intel Compiler 18.0.X　X は update の回数

計算機にセットアップされている Intel Parallel Studio XE は、以下のディレクトリにインストールされています。

| パッケージ | ディレクトリ |
|---|---|
| Intel Parallel Studio XE 20XX<br>例） Intel Parallel Studio XE 2018 | /opt/intel/psxe20XX<br>/opt/intel/psxe2018 |

このインストールディレクトリは、Intel Parallel Studio XE のデフォルトとは異なっております。変更した理由は、複数の Intel Compiler（ Parallel Studio XE ）を混在させた場合に、一部の環境変数が重複し、不具合が発生するためです[8]。

[7] Intel Parallel Studio XE が Cluster Edition の場合、Intel MPI も同梱されていますが、こちらについては、4 章にて解説しています。

[8] Compiler の man の参照先 が重複し、使用している Version の設定が反映されなくなります。

## 3.2 Intel Compiler の使用方法

Intel Compiler を使うには以下コマンドを使用します。出荷設定では root 以外の全ユーザーが使用できます。

| コマンド | 内容 |
|---|---|
| ifort | Fortran 言語用コンパイラ |
| icc | C 言語用コンパイラ |
| icpc | C++ 用コンパイラ |

コンパイラのバージョンは、-V オプションで確認できます。例： ifort -V

出荷設定では icc / ifort コマンドに以下のエイリアス設定をしています。icc / ifort コマンドを使用する際、以下オプションが自動で適用されます。

| オプション | 内容 |
|---|---|
| -D_FILE_OFFSET_BITS=64 | プログラム上で 2GB を超えるサイズのファイルを取り扱うために使用します。<br>(icc で標準使用します。) |
| -D_LARGEFILE_SOURCE | プログラム上で 2GB を超えるサイズのファイルを取り扱うために使用します。<br>(icc で標準使用します。) |
| -assume buffered_io | ファイルに書き込みを行う際、ディスクにすぐに書き込まず、バッファに蓄積することを指示します。<br>(ifort で標準使用します。) |

Intel Compiler で多用されるコンパイルオプションを次に示します。

| オプション | 内容 |
|---|---|
| -O0<br>-O1<br>-O2<br>-O3 | ソースコードの最適化を制御します。-O0 ～ -O3 のうち、１つを選択して指定します。明示的に指定しない場合（デフォルト設定）は -O2 となります。一般的に、-O に続く数字が大きい程パフォーマンスが向上しますが、数値精度に問題が出る場合があります。<br><br>-O0：最適化なし。パフォーマンスが非常に低下するので注意が必要となります。<br>-O1：一部の最適化を有効化します。<br>-O2：デフォルト設定。ベクトル化が有効となります。<br>-O3：-O2 より強力な最適化を行います。 |
| -parallel | 作成された実行ファイルがスレッド並列で動作するようになります。 |
| -qopenmp | OpenMP を使用するプログラムをコンパイルする際に使用します。Intel Compiler の旧 Version では、-openmp というオプション名でしたが変更になりました。 |
| -mcmodel | 使用するメモリーモデルを指定します。<br>(例) -mcmodel=medium<br>コードサイズの上限が 2GB になり、データサイズは無制限になります。<br>(例) -mcmodel=large<br>コードサイズ、データサイズともに無制限になります。 |
| -fp-model | 浮動小数点数の扱い方を指定します。<br>(例) -fp-model strict<br>縮約を無効にし、浮動小数点数環境の変更を可能にするプロパティを有効にします。 |

## 3.3 コンパイラバージョンの変更

Intel Compiler の環境設定は各ユーザーのホームディレクトリのファイルで行っています。具体的には、ユーザーのシェルスクリプトが bash の場合[9]は ~/.bashrc 、tcsh の場合は ~/.cshrc ファイル内で、使用コンパイラのパッケージを設定しています。

複数の Intel Compiler コンパイラがインストールされている場合、使用する Intel Compiler のバージョンを変更するには、次のように変更します。

・ユーザーのシェルが bash の場合は ~/.bashrc 内の COMPILER を修正します。

例） Intel Compiler を Version 18.0 から 17.0 に変更する場合

| 修正前 | 修正後 |
|---|---|
| COMPILER=INTEL18.0 | #COMPILER=INTEL18.0 |
| #COMPILER=INTEL17.0 | COMPILER=INTEL17.0 |
| #COMPILER=INTEL15.0 | #COMPILER=INTEL15.0 |
| #COMPILER=PGI17 | #COMPILER=PGI17 |
| #COMPILER=PGI16 | #COMPILER=PGI16 |
| #COMPILER=PGI15 | #COMPILER=PGI15 |

・ユーザーのシェルが tcsh の場合は ~/.cshrc 内の COMPILER を修正します。

例） Intel Compiler を Version 18.0 から 17.0 に変更する場合

| 修正前 | 修正後 |
|---|---|
| set COMPILER=INTEL18.0 | #set COMPILER=INTEL18.0 |
| #set COMPILER=INTEL17.0 | set COMPILER=INTEL17.0 |
| #set COMPILER=INTEL15.0 | #set COMPILER=INTEL15.0 |
| #set COMPILER=PGI17 | #set COMPILER=PGI17 |
| #set COMPILER=PGI16 | #set COMPILER=PGI16 |
| #set COMPILER=PGI15 | #set COMPILER=PGI15 |

[9] 各ユーザーが使用するシェルは "echo $0" コマンド等で確認できます。

次回ユーザーがログインした時からコンパイラのバージョンが変わります。バージョンが変更されたことを、念のため "ifort -V" や "icc -V" でご確認下さい。

コンパイラバージョン変更時の注意

上記のインテルコンパイラのバージョンを変更すると、環境変数 LD_LIBRARY_PATH 等が変わります。コンパイラのバージョン変更後に以前コンパイルしたプログラムを動作すると、実行ファイルが正しいライブラリを参照できず、正常に動作しない場合があります。その際は実行するプログラムをコンパイルし直すか、使用するコンパイラのバージョン毎に新しいユーザーを作成するなどをしてご対応下さい。

## 3.4 Intel Math Kernel Library

Intel Compiler には Intel Math Kernel Library (MKL)パッケージが同梱されています。MKL は、ベクトル演算関数、高速フーリエ変換 (FFT)等、科学技術計算でしばしば使用される処理を高速で処理するライブラリ群で、特に Intel 社製 CPU で実行する場合に非常に高い性能を示します。

MKL の BLAS / LAPACK 等を使用する場合の使用例を次に示します。

(例 1) Intel LP64 インターフェースを使用したダイナミックリンクでのコンパイル

```
ifort myprog.f -L${MKLPATH} -I${MKLINCLUDE}
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm
```

(例 2) Intel LP64 インターフェースを使用したスタティックリンクでのコンパイル

```
ifort myprog.f -L${MKLPATH} -I${MKLINCLUDE}  -Wl,--start-group
${MKLPATH}/libmkl_intel_lp64.a ${MKLPATH}/libmkl_intel_thread.a
${MKLPATH}/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
```

※${MKL_PATH}、${MKLINCLUDE}は使用する Intel Compiler のバージョンで変わります。

MKL はビルド状況に応じて、オプションやリンクするライブラリを選択、変更する必要があり、リンクオプションは複雑です。Intel 社は公式サイトで、状況に応じたリンクオプションをナビゲートするサイト、Intel® Math Kernel Library Link Line Advisor サイトを開設しています。具体的なリンクオプションは以下のサイトを参照ください。

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

## 3.5 ドキュメント

Intel Parallel Studio XE のマニュアルは以下にインストールされています。使用方法や仕様の詳細が知りたい場合には参照ください。

| パッケージ | マニュアルのディレクトリ |
|---|---|
| Intel Parallel Studio XE 20XX<br>例）Intel Parallel Studio XE 2018 | /opt/intel/psxe20XX/documentation_20XX<br>/opt/intel/psxe2018/documentation_2018 |

以上のディレクトリにあるドキュメントですが、基本的に Intel 社のオンラインマニュアルを参照する形式をとっています。そのため、計算機がインターネットに接続できない環境下では参照できる情報が多くないことにご注意ください。

また、Intel Parallel Studio XE の代理店のエクセルソフト(株)のサイトより、各種ドキュメントが配布されています。以下 URL をご参照下さい。

https://www.xlsoft.com/jp/products/intel/studio_xe/index.html#documents

## 3.6 サポート期間

当社で販売している Intel Parallel Studio XE のライセンスは 1 年間のサポート期間があります。保障期間内は以下の権利があります。

(1) Intel Parallel Studio XE のパッケージがバージョンアップした際、新しいパッケージを入手して計算機にインストールすることができます。新しいパッケージはインテル レジストレーション・センターから配布されます。

(2) Intel 社のテクニカルサポート対応が受けられます。

ライセンスが切れた場合は上記の対応が受けられなくなります。ライセンス期限後も引き続きこれらの対応を受ける場合は有償でライセンス更新を行う必要があります。サポート期限が切れた後も既に計算機にインストールされているコンパイラを使用することや、コンパイルしたプログラムを使用することは可能です。

Intel Parallel Studio XE のサポート期間はインテル レジストレーション・センターでログインして確認できます。また、計算機で使用されているライセンスファイルにも記述されています。ライセンスファイルは以下をご確認下さい。

・ライセンスファイルは /opt/intel/licenses 以下に置かれています。

確認箇所は以下の例を参考にしてください。

・ファイル例：(この例では 2019/2/19 までです)

```
PACKAGE IC45FB71A INTEL 2019.0219 XXXXXXXXXXXX COMPONENTS="ArBBL ¥
```

# 4 Intel MPI Library（非 Intel oneAPI 同梱版）

Intel MPI Library は、並列・分散プロセス間のメッセージ機能を提供する MPI 規格を実装したバイナリおよびライブラリ群で、世間では Intel MPI と呼ばれていることが多いソフトウェアです（本章でも、以降、Intel MPI と呼称します）。特徴としては、以下の点が挙げられます。

- MPICH、MVAPICH2 をベースに Intel 社が開発を行っている。
- MPI-3.1 規格に準拠している。
- 特に Intel 社製ハードウェアに向けた調整がなされている。

過去の MPI-2 規格に準拠した MPICH2、OpenMPI で動作していたソースコードは、ほとんどの場合、修正が必要無くそのままコンパイルできます。なお、過去に MPICH2 や OpenMPI 等でビルドを行ったアプリケーションバイナリはそのままでは Intel MPI 環境で動作しません。Intel MPI で実行するためには、ソースコードから Intel MPI でリビルドを行う必要があります。

**本項では Intel oneAPI Toolkit リリース前に販売されていた、Intel oneAPI HPC Toolkit に同梱されていない Intel MPI Library について解説を行います。**

## 4.1 インストールディレクトリ

Intel MPI は、Intel Parallel Studio XE とディレクトリ構造が統一されています[10]。Intel MPI の Major Version と、Intel Parallel Studio XE の Version（西暦年）は共通しているため、例えば、Intel MPI 2018 は Intel Parallel Studio XE 18.0 とディレクトリ構造が同じとなっています。

計算機にセットアップされている IntelMPI は、2.1 章で説明している Intel Parallel Studio XE との整合性をとるため、以下のディレクトリにインストールされています。

| パッケージ | ディレクトリ |
|---|---|
| Intel MPI 20XX<br>例） Intel MPI 2018 | /opt/intel/psxe20XX<br>/opt/intel/psxe2018 |

[10] Intel MPI Version 2017 以降

## 4.2 Intel MPI のコンパイルコマンド

以下のコマンドで、MPI を使用したソースコードのコンパイルを行うことが可能です。Intel MPI は、内部で使用するコンパイラの種類によってコンパイルコマンドが異なっており、今まで慣例的に使用されてきた mpif90 や mpicc を使用すると、Intel Compiler が使用されない点にご注意ください。

| コマンド | 内容 |
|---|---|
| mpiifort | Intel Fortran Compiler ( ifort )を使用した Fortran 用コマンド |
| mpiicc | Intel C Compiler ( icc ) を使用した C 用コマンド |
| mpiicpc | Intel C/C++ Compiler ( icpc ) を使用した C++用コマンド |
| | |
| mpif77 | GNU Fortran Compiler g77 を使用した Fortran 用コマンド |
| mpif90 | GNU Fortran Compiler gfortran を使用した Fortran 用コマンド |
| mpicc | I_MPI_CC 変数等で指定している場合は、指定の C Compiler<br>指定の無い場合は、GNU C Compiler ( gcc )<br>を使用した C 言語用コマンド |
| mpigcc | GNU C Compiler ( gcc )を使用した C 言語用コマンド |
| mpigxx | GNU C/C++ Compiler ( g++ )を使用した C/C++言語用コマンド |

コンパイル時のオプションについては、基本的に、内部で使用されるコンパイラのオプションをそのまま使用できます。

ただし、static リンクについては注意が必要となります。近年、OS のシステムの関係や、またはアプリケーション開発の都合上、ライブラリのリンク時に shared リンク（dynamic リンク）を使用するケースが大変多くなっています。コンパイル時のオプションで static オプションを使用することは可能ですが、アプリケーションが shared リンクを想定しており対応が難しいケースや、リンクに必要なライブラリ群が shared リンク用のパッケージしかないというケースが見受けられます。このような場合、全てのライブラリを static でリンクしたバイナリの作成は困難です。このような状況下では、仮に static コマンドを使用してアプリケーションのビルドを行っても、作成したノード外で実行する、ノード環境を変更（更新）するなどのケースでは、アプリケーションが正常に動作しない場合があります。

## 4.3 実行コマンド

Intel MPI を使用して、プログラムをローカル環境（プロンプトで作業している計算機）で実行する場合の実行方法は以下の通りです。実行には mpirun コマンドを使用します。

```
mpirun -np [並列数] [option] [実行ファイル]
```

[並列数] には、並列計算に使用するプロセスの数が入ります。通常、指定したプロセス数と計算に使用される CPU コア数は等しくなります[11]。例えば 4 コアを使用する並列計算の場合、mpirun -np 4 … のように指定します。

[option] の項目は実行時のオプションです。オプションの詳細な解説は、OS 上の man コマンドで man mpirun と実行するか、Intel MPI のドキュメントでご確認下さい。

[実行ファイル] 等、ファイル指定時にはパスの指定を正確に行う必要があります。特に同名のファイルが複数ある場合や、ノード間でプロセスを起動する場合には、安全に実行するため、ファイル指定を絶対パス等で明示的に記述することを強く推奨します。

例えば、バイナリ test を -i オプションで入力ファイル、-o オプションで出力ファイルを指定して実行するケースを考えます。バイナリ test が/usr/local/bin に存在し入力ファイルとして/home/hpc/test_data/inp 、出力ファイルとして/home/hpc/test_out/out を指定したい場合、次のコマンドになります。

```
mpirun -np 4 /usr/local/bin/test -i /home/hpc/test_data/inp
-o /home/hpc/test_out/out
```

また、バイナリ test がカレントディレクトリに存在する場合は次のコマンドになります。

```
mpirun -np 4 ./test -i /home/hpc/test_data/inp -o /home/hpc/test_out/out
```

[11] OpenMP 等でスレッド並列を組み合わせる、いわゆる Hybrid 並列の場合、トータルで使用されるコア数は、（プロセス並列数）×（スレッド並列数）となります。

## 4.4 計算実行時における計算ノードの指定

Intel MPI では、前節で解説した通り、mpirun コマンド実行時に計算に使用するノードを指定しない場合、プロンプトでコマンドを入力した計算機でプログラムが実行されます。そのため、複数のノードを指定してプログラムを実行する場合には、計算に使用するノードを指定する必要があります。

カレントディレクトリにあるプログラム test を、計算ノード node01 で 1 プロセス、計算ノード node02 で 3 プロセス実行させる場合は下記のコマンドとなります。

```
mpirun -np 1 -host node01 ./test : -np 3 -host node02 ./test
```

並列計算に使用する複数のノードに対して、計算プロセスを均等に割り振って実行する場合には、-ppn オプションを使用する事も可能です。カレントディレクトリのプログラム test を、計算ノード node01、node02、node03、node04 でそれぞれ 2 プロセスずつ、計 8 プロセスで実行させる場合、下記のコマンドとなります[12]。

```
mpirun -ppn 2 -hosts node01,node02,node03,node04 ./test
```

なお、Intel MPI では実行時に各ノードに割り振られるプロセスは、CPU（コア）に Bind される設定がデフォルトとなっています。プロセスを CPU（コア）に Bind するオプションを明示的に設定する必要はありません。

計算に使用するノードをファイルで指定する事も可能です。まず、計算に使用するノード名と、それぞれのノードに割り振るプロセス数を指定したファイルを作成します。例えば、node01 に 2 プロセス、node02 に 4 プロセス、node03 に 2 プロセスを実行させる場合の設定ファイルは以下のようになります。

```
node01:2
node02:4
node03:2
```

[12] -np オプションと -ppn オプションの同時使用は誤動作する場合があるため、推奨しません。

設定ファイルを作成した後、-machinefile オプションで、作成したファイルを指定します。この設定ファイルをカレントディレクトリに machinelist という名前で保存し、カレントディレクトリのプログラム test を実行させる場合、下記のコマンドとなります

```
mpirun -machinefile ./machinelist ./test
```

この場合、設定ファイルに書かれた割り振り方を使用して並列計算が行われます。今回の例で示した machinelist の場合は、計 8 プロセスでの並列計算が実行されます。

また、-machinefile オプションは、-np オプションと組み合わせることも可能です。具体的には以下のように指定します。

```
mpirun -machinefile ./machinelist -np 4 ./test
```

この 2 つのオプションを併用した場合、 -np で指定された並列数分のプロセスを、-machinefile で指定した設定ファイルの上から順番に割り振る形で計算が行われます。上記の例の場合、node01、node02 で 2 プロセスずつ、計 4 プロセスでの並列計算が実行されます。

Intel MPI では、-machinefile で指定する設定ファイルを、計算実行時に自動的に参照させる機能を持っておりません。ジョブスケジューラーを使用しない場合、計算を実行する度に毎回指定する必要があります。

## 4.5 ジョブスケジューラーとの連携

Intel MPI は、ジョブスケジューラー( LSF / PBS / Grid Engine 等) を使用してプログラムをジョブとして投入すると、自動的に連携を行います。ジョブスケジューラーを使用して MPI ジョブを投入する方法につきましては、各ジョブスケジューラーのマニュアルをご確認下さい。

# 5 CUDA

CUDA は NVIDIA 社が提供する GPU を演算処理で使用するための統合開発環境です[13]。本項では計算機にセットアップされている CUDA について概説します。

## 5.1 インストールパッケージ

CUDA の各パッケージを以下ディレクトリにインストールしています。

| バージョン | ディレクトリ |
|---|---|
| CUDA Driver | .run ファイルをダウンロードし、インストール |
| CUDA Toolkit | CUDA Version X.Y の場合<br>/usr/local/cuda-X.Y<br><br>上記ディレクトリへのシンボリックリンクとして<br>/usr/local/cuda が作成されます。 |

CUDA の各パッケージは出荷するシステムに合わせた、最新の安定版をインストールしております。セットアップされている CUDA のバージョンは以下コマンドで確認できます。

CUDA Driver ····· nvidia-smi コマンド
CUDA Toolkit ····· nvcc コマンド (root 以外のユーザーで使用可能)

## 5.2 使用するコマンド

GPU 用ソースファイルを CUDA でコンパイルする場合には、以下のコマンドを使用します。出荷設定では root 以外の全ユーザーが使用できます。

| コマンド | 内容 |
|---|---|
| nvcc | C 言語 / C++ 用 コンパイラ（CUDA） |

[13] https://developer.nvidia.com/cuda-downloads にて配布

## 5.3 環境設定

弊社でセットアップをした計算機において、環境設定は
① Environment Module を使用している場合
② source コマンドを使用している場合
の２つのケースがあります。以下では、それぞれの方法について解説します。

① Environment Module を使用している場合

各アプリケーションの設定ファイルは、各ユーザーのホームディレクトリにある .bashrc/.cshrc 内で、module load コマンドを使用して読み込まれます。読み込まれる設定ファイルは、以下のようになります。( X.Y は CUDA の Version を表します。)

/home/.common/modulefiles/CUDA/CUDA-X.Y
例：CUDA Version 11.1 の場合
/home/.common/modulefiles/CUDA/CUDA-11.1

② source コマンドを使用している場合

以下の設定ファイルを 各ユーザーの .bashrc/.cshrc 内で source しています。
( X.Y は CUDA の Version を表します。)

bash 用 /home/.common/600-CUDA X.Y.sh
tcsh 用 /home/.common/600-CUDA X.Y.csh
例：CUDA Version 9.0 の場合
bash 用 /home/.common/600-CUDA9.0.sh
tcsh 用 /home/.common/600-CUDA9.0.csh

## 5.4 GPU の動作確認

GPU の状態は以下のように確認いただけます。

(1) nvidia-smi コマンド

計算機に搭載している各 GPU の GPU 温度・GPU 使用率・メモリ使用率等の情報を取得できます。nvidia-smi には以下のようなオプションがあります。

| オプション | 内容 |
|---|---|
| -q | GPU の全ての情報を出力します |
| -q -d XXX<br><br>XXX：対象 | XXX で指定した情報のみ出力します。<br>このオプションで指定可能な項目には以下があります。<br>MEMORY, UTILIZATION, ECC, TEMPERATURE, POWER, CLOCK, COMPUTE, PIDS, PERFORMANCE, SUPPORTED_CLOCKS, PAGE_RETIREMENT, ACCOUNTING, ENCODER_STATS, SUPPORTED_GPU_TARGET_TEMP, VOLTAGE FBC_STATS, ROW_REMAPPER |
| -i 0 | 複数の GPU が搭載されている場合、0 番として認識している GPU に対して処理を実行します。 |
| -l 2 | 2 秒おきに出力しつづけます。<br>Ctrl+C で出力を止められます。 |

(2) deviceQuery

nvidia-smi と同様に GPU の状態や構成情報を出力します。

deviceQuery プログラムは CUDA Toolkit パッケージ内のサンプルプログラムであり、次のディレクトリにあります。

/usr/local/cuda/samples/1_Utilities/deviceQuery

# 6 Tips

## 6.1 実行プロセスのノードへの割り振り

実行するプログラムによっては、実行プロセスのノードへの割り振り方がプログラムの性能に大きな影響を与えることがあります。以下に、その例と対策を示します。

(例) CPU-メモリ間の大量のデータ転送により律速しているプログラム

CPU 処理性能を十分に発揮させるには、それに見合った速度でデータをメモリと送受信できる必要があります。アルゴリズム上、メモリと CPU コア（のレジスタ）の間のデータ転送速度が性能支配的なプログラムにおいては、各プロセスが使用可能な転送速度を勘案しながらノードへのプロセス群の割り当てを選択することが肝要です。ノードの上では、そのノードで稼働するプロセス群が、CPU-メモリ間のデータ転送帯域を分け合って使用します。したがって、各プロセスが使用できるデータ転送速度は、「ノードの CPU-メモリ間のデータ転送速度 ÷ 稼働しているプロセス数」 となります。

例えば、各ノードに 8 コアを有する 4 ノードクラスタ環境で、計 16 プロセス並列を行う場合について、以下の２種類の実行例を考えます。

実行例 A：2 ノードを使用し、各ノードで 8 個ずつプロセス起動する場合
実行例 B：4 ノードを使用し、各ノードで 4 個ずつプロセス起動する場合

各ノード上の CPU-メモリ間のデータ転送速度を 100 とした場合、それぞれの実行例で利用可能な、各 CPU コア-メモリ間のデータ転送速度は、次表のようになります。

| ノード | 実行例 A | 実行例 B |
|---|---|---|
| node01 | 100/8 = 12.5 | 100/4 = 25 |
| node02 | 100/8 = 12.5 | 100/4 = 25 |
| node03 | | 100/4 = 25 |
| node04 | | 100/4 = 25 |

このように、実行例 B は実行例 A に比べ、各プロセスで利用可能な、CPU コア-メモリ間のデータ転送速度が 2 倍になります。CPU-メモリ間の大量のデータ転送により律速しているプログラ

ムにおいては、このデータ転送速度がプログラム全体の計算速度に大きく影響するため、上記の実行例 B の方が速く動作する可能性があります。また、そのようなプログラムでは、使用するノード数を増やさないまま並列数を増やしても計算速度がほとんど向上しないケースもありえます。

以上の例は一般論ですので、実際の動作についてはお客様ご自身でご確認いただいた上でご検討下さい。

# 7　トラブルシューティング

現時点で判明している不具合は特にありません。

# 付録A

## A.1 環境設定ファイルの配置

弊社でセットアップをした計算機において、環境設定は

① Environment Module を使用している場合
② source コマンドを使用している場合

の２つのケースがあります。多くの場合、Intel oneAPI Toolkit を使用している場合には①を、Intel Parallel Studio XE を使用している場合には②を使用しています。以下では、それぞれの方法について解説します。

### A.1.1 環境設定に Environment Module を使用している場合

各アプリケーションの設定ファイルは、各ユーザーのホームディレクトリにある .bashrc/.cshrc 内で、module load コマンドを使用して読み込まれます。読み込まれる設定ファイルは、以下に存在します。

Intel oneAPI Toolkit：

/opt/intel/oneapi/modulefiles ： Intel oneAPI Toolkit 標準の設定ファイル
/opt/intel/oneapi/modulefiles-HPCS：HPC SYSTEMS 製設定ファイル

その他アプリケーション：

/home/.common/modulefiles

### A.1.2　環境設定に source コマンドを使用している場合

弊社でセットアップをした各アプリケーションのシェル環境設定ファイルは /home/.common ディレクトリに置かれています。設定ファイルは bash 用（拡張子が.sh）と csh 用（拡張子が.csh）の 2 種類が用意されています。ディレクトリ内は、コンパイラや MPI のバージョンに応じて以下のように配置されています。

① /home/.common/{コンパイラ名・Version}ディレクトリ

このディレクトリはコンパイラ名・コンパイラバージョンごとに作成されます。例えば Intel Parallel Studio XE では、/home/.common/INTEL19.0、といった名前になり、以下にコンパイラに対応したシェル環境設定ファイルが配置されます。さらに、そのコンパイラを用いてコンパイル・リンクを行ったアプリケーション用のシェル環境設定ファイルが配置されます。

② /home/.common/INTEL*/IntelMPI ディレクトリ

Intel Compiler が設定されている環境上において、Intel MPI を使用するためのシェル環境設定ファイルが配置されます。さらに、Intel Compiler 、Intel MPI を用いてコンパイル・リンクを行ったアプリケーションのシェル環境設定ファイルが配置されます。

③ /home/.common ディレクトリ

このディレクトリ直下には、コンパイラ・MPI に依存しないアプリケーションのシェル環境設定ファイルが配置されます。

各ユーザーの ~/.cshrc と ~/.bashrc において定義されている変数 COMPILER、MPI により、使用するコンパイラ、MPI が指定されます。指定されたコンパイラ、MPI に関するシェル環境設定ファイルが、ファイル名昇順で /home/.common 以下から読み込まれます。読み込まれる順番は、前述の箇条書きの ①、 ②、最後に③の順になります。

# 付録B

## B.1　HPC システムズ お問い合わせ先


弊社ホームページ　https://www.hpc.co.jp/support/

サポート案内やお問い合わせの多い内容など様々な情報を掲載しております。
是非ご活用ください。

**HPC システムズ株式会社**
〒108-0022　東京都港区海岸 3-9-15 LOOP-X 8 階

HPC 事業部


【営業】03-5446-5531　【サポート】03-5446-5532

お電話によるサポート受付は祝日、弊社指定休日を除く月曜日から金曜日の 9:30～17:30 とさせて頂きます。


【FAX】03-5446-5550


【電子メール】hpcs_support@hpc.co.jp