アプリケーションのビルド(コンパイラとライブラリ)

(最終更新日: 2025/1/28)

RCCS では GCC (gcc-toolset), AOCC (AMD Optimizing Compilers), NVIDIA HPC SDK について複数のバージョンが利用できます。CUDA (nvcc) についても複数バージョンが導入済みです。MPI 並列を行う場合はこちらのページの内容についてもご確認ください。

GCC

システム標準の GCC はバージョン 8.5 ですが、これとは別に gcc-toolset のものを使うことができます。(/opt 以下にインストールされています。)以下のように module load gcc-toolset/13 のように読み込むことができます。scl コマンドでももちろん読み込み可能です。

```
# (システム標準は 8.5.0)
$ gcc -dumpfullversion
8.5.0
$ module load gcc-toolset/13
$ gcc -dumpfullversion
13.1.1
```

利用可能なバージョンについては module avail gcc-toolset コマンドやパッケージソフト一覧ページで確認できます。

AOCC/AOCL

計算ノードに搭載されている EPYC 7763 を製造している AMD より提供されているコンパイラです。/apl/aocc 以下にインストールされています。module コマンドで module load aocc のように実行することで環境を読み込むことができます。

```
$ module load aocc
$ clang --version
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /lustre/rccs/apl/ap/aocc/4.2.0/bin
```

利用可能なバージョンについては module avail aocc コマンドやパッケージソフト一覧ページで確認してください。

AOCL (AMD Optimizing CPU Libraries) についても /apl/aocl 以下に導入しています。こちらも module で読み込み可能です。BLIS (libblis), libflame がそれぞれ BLAS, LAPACK 相当になります。他にも各種ライブラリが含まれています。

```
$ module load aocc/4.2.0
$ module load aocl/4.2.0-aocc4.2
```

利用可能なバージョンについては module avail aocl コマンドやパッケージソフト一覧ページで確認してください。

Intel oneAPI

RCCS ではライセンスを保持していないため、コンパイラについてはユーザーの皆様に提供することはできません。(MKL や MPI については導入しています。)しかし、2024/4/8 時点では無償のライセンスで自身のホームディレクトリに導入し、利用することが可能です。

以前のコンパイラ(icc, icpc, ifort)で問題無くビルドできていたコードを新しいコンパイラ(icx, icpx, ifx)でビルドしようとするとエラーになるケースがあるようです。巨大なコードの場合はコードの変更無しでは動かないケースの方が多いように思われます。コンパイラオプションや周辺コマンドについては変更されている部分もあるようですのでご注意ください。

導入と設定の方法

Intel oneAPI Base Toolkit はこちらからダウンロードできます。C/C++ コンパイラ、MKL などが含まれています。Linux 向け Online か Offline 版をご利用ください。MPI ライブラリや Fortran のコンパイラが必要な場合は HPC Toolkit も導入する必要があります。こちらよりダウンロードしてください。

bash をご利用されている場合はインストール後に ~/intel/oneapi/setvars.sh を読み込めば大丈夫です。コンパイラや MKL 等の個別パッケージだけを読み込みたい場合は各コンポーネント中の env ディレクトリにあるスクリプトを読み込むこともできます(コンパイラーを読み込む場合: source ~/intel/oneapi/compiler/latest/env/vars.sh)。csh の場合は一手間かかります。いくつか方法はありますが、以下のように module 化する方法が比較的取り回しが良いかと思います。

module 化

(bash の場合も以下の方法はそのまま使えますが、素直に ~/intel/oneapi/setvars.sh を読み込んだ方が普通は楽です。)

csh での設定読み込みはいくつか方法が考えられますが、ここでは module 化する方法を紹介します。oneAPI は ~/intel 以下に導入済とします。

```
$ cd ~/intel/oneapi
$ sh modulefiles-setup.sh
$ cd ~/modulefiles/
$ module use .
$ module save
```

(modulefiles-setup.sh の出力先はオプションで変更可能です。)これを実行すると、oneapi の module を作成し、module の検索パスにそのディレクトリを登録する(module use . の箇所)ことになります。最後に module save でその設定を保存しています。RCCS ではログイン時に save した module 環境を restore するようになっているため、次回以降のログイン時には自動的に設定されます。例えばコンパイラを読み込みたいのであれば、module load compiler/latest を実行することになります。バージョンの詳細等については module avail コマンド等でご確認ください。

コンパイラ等を先に読み込んでから module save を行えば、次回ログイン後すぐにインテルのコンパイラが使えるようになります。

```
$ cd ~/modulefiles/
$ module use .
$ module load compiler/latest
$ module load mkl/latest
$ module load mpi/latest
$ module save
```

module save した環境を破棄したい場合は module saverm を実行してください。なお、環境を保存してしまうと、システム側のデフォルト設定の変更に追随できなくなりますので、その点についてはお気をつけください。

MKL の利用について

MKL のリンクオプションは大変複雑ですが、公式ウェブサイトを利用したり mkl_link_tool コマンドを使うことで簡略化が可能です。mkl_link_tool を引数無しで実行するとオプションが表示されます。利用頻度が高そうなオプションとしては、以下のものがあげられます。

- -c : コンパイラの指定。icc で使うなら intel_c、ifort で使うなら intel_f、gcc で使うなら gnu_c、gfortran で使うならば gnu_f となります。
- -p : 並列設定。-p yes ならば OpenMP 並列有効(-qmkl=parallel 相当; デフォルト)で、-p no だと並列を無効にします(-qmkl=sequential 相当)。
- -m : MPI ライブラリの指定(scalapack などのため)。intelmpi, openmpi などから選べます。デフォルトは intelmpi です。
- --cluster_library : MPI 時のクラスターライブラリの指定。scalapack などが選べます。(例: --cluster_library=scalapack)
- 他では、64 ビット整数版(ilp64)を使ったり、リンク方法を変えて static リンクにしたり、OpenMP のライブラリを変更をするなどの指定もできます。

ここでは mkl_link_tool コマンドの利用例をいくつか示します。(--quiet をつけて出力を簡略化しています)

■ gfortran で OpenMP 並列無しの場合(-c gnu_f -p no)

コンパイル時用入力(-opts): (先頭の $ は入力しないでください)

```
$ mkl_link_tool -opts -c gnu_f -p no --quiet
```

結果:

```
-m64  -I"${MKLROOT}/include"
```

リンク時用入力(-libs): (先頭の $ は入力しないでください)

```
$ mkl_link_tool -libs -c gnu_f -p no --quiet
```

結果:

```
-m64  -L${MKLROOT}/lib -Wl,--no-as-needed -lmkl_gf_lp64 -lmkl_sequential -lmkl_core -lpthread -lm -ldl
```

■ ifort で openmpi を使って scalapack まで含めてリンクする場合

コンパイル時用入力(-opts): (先頭の $ は入力しないでください)

```
$ mkl_link_tool -opts -c intel_f -p yes -m openmpi --cluster_library=scalapack --quiet
```

結果:

```
-I"${MKLROOT}/include"
```

リンク時用入力(-libs): (先頭の $ は入力しないでください)

```
$ mkl_link_tool -libs -c intel_f -p yes -m openmpi --cluster_library=scalapack --quiet
```

結果:

```
-L${MKLROOT}/lib -lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lmkl_blacs_openmpi_lp64 -liomp5
```

-xHost オプションについて

最近のバージョンであれば AMD の CPU 向けであっても -xHost オプションの利用で問題が起きるケースはほとんど無いと思われます。ただし、クラシックコンパイラ(icpc)で -xHost を指定した場合に pragma simd のディレクティブがうまく効かないケースが確認されています(なお、icpx の場合は -xHost で問題ありませんでした)。そのような場合は -xHost のかわりに -march=core-avx2 を指定するとうまくいくようです。

NVIDIA HPC SDK

NVIDIA HPC SDK も対応する module を読み込むことで利用できるようになります。OpenACC や CUDA Fortran が必要な場合にはこちらが必要になることがあります。また、PGI コンパイラを要求するソフトについてはこちらを使うことで対応できる場合があるかもしれません。

```
$ module load nvhpc/23.9-nompi
$ nvc -dumpversion
23.9
```

基本的には -nompi のパッケージをご利用ください。(含まれている MPI にはキューイングシステムへの対応が入っていないため、問題が起こる可能性があります。)MPI 版を使いたい場合は openmpi/4.1.6/nv23 などの module 版をご利用ください(nvhpc/23.9-nompi と openmpi/4.1.6/nv23 の両方の module を読み込んでください)。こちらは純正のビルドではありませんが、キューイングシステム対応と CUDA-aware 対応を有効にしたバージョンとなっております。

コンパイラだけ別のものを使いたい場合は byo (bring-your-own compiler)版を指定してください。

CUDA

CUDA 環境はデフォルト設定でも読み込まれていますが、それとは別のバージョンを使うこともできます。module purge した後で module load するか、module switch コマンドでバージョンを切り替えることができます。

```
$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
$ module purge
```

```
$ module load cuda/11.8
$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

利用可能なバージョンは module avail cuda コマンドやパッケージソフト一覧ページで確認できます。

各種ツール(autoconf, cmake)やライブラリ(boost, openblas)

システム標準のものとは別に新しいバージョンが /apl 以下に導入されている場合があります。module avail コマンドでは /apl/modules/util 以下にリストされている場合がほとんどだと思われます。mkl については /apl/modules/oneapi/mkl 以下になります。必要なバージョンを module コマンドで読み込むようにしてください。

OpenBLAS, Boost, Ninja を読み込む例:

```
$ module load openblas/0.3.26-lp64
```

```
$ module load boost/1.84.0
```

```
$ module load ninja/1.11.1
```

依存関係等の理由でうまく読み込めない場合は一旦 module purge してリセットするなどして対応してください。

コンパイラやライブラリの選択について

GCC(特に新しいバージョン)、Intel Compiler、AOCC についてはどれでビルドしても大きな速度差がでない、というケースが多く確認できています。一方で、特定のコンパイラでは妙に速度が出ない、あるいは安定しないコードもあります。長期に渡って実行することを予定しているプログラムであれば、特定のコンパイラでバージョンを変えるだけでなく、他のコンパイラも試してみても良いかもしれません。

あまりこだわりが無い、そこまで速度を求めていないのであれば、ひとまず GCC (gcc, g++, gfortran)を使ってしまうのが無難かと思います。新しいバージョンを使うと速度が出る場合が多いですので、module load gcc-toolset/13 のように読み込んで試すと良いかもしれません。gcc-toolset を使ったバイナリをジョブで使う場合、実行時ライブラリについてはシステム標準のもので十分ですので module load gcc-toolset/* による読み込みは不要です。

BLAS についても MKL よりも OpenBLAS を使った方が少し速いケースが確認されています。AOCL に含まれる BLIS や libflame を使った場合も同様の事象があるかもしれません。MKL の場合は export MKL_ENABLE_INSTRUCTIONS=AVX2 で明示的に AVX2 を指定した方が良いようなケースもあるかもしれません。