Build Applications (Compilers and Libraries)

(Last update: Jul 11, 2025)

On RCCS supercomputer system, multiple versions of GCC (gcc-toolset),AOCC (AMD Optimizing Compilers), and NVIDIA HPC SDK are available. Multiple versions of CUDA (nvcc) are also installed.For MPI library and runtime environment, please see this page.

GCC

GCC 8.5 is the system default gcc. In addition to the one, GCC from gcc-toolset packages are also available (installed in /opt). You can load the environment with "module load gcc-toolset/13" command for example. You can also use standard "scl" command.

```
# (system default versioin is 8.5.0)
$ gcc -dumpfullversion
8.5.0
$ module load gcc-toolset/13
$ gcc -dumpfullversion
13.1.1
```

Full list of available GCC versions can be found with "module avail gcc-toolset" command or on theInstalled Applications page.

AOCC/AOCL

AOCC (AMD Optimizing C/C++ and Fortran Compilers) is provided by AMD, the manufacturer of EPYC 7763 installed in the computation nodes. Multiple versions of AOCC are installed under /apl/aocc. You can load AOCC setting with "module load aocc" command. (If you omit version name, default version will be loaded.)

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

Full list of available versions can be found with "module avail aocc" command or onInstalled Applications page in this website.

AOCL (AMD Optimizing CPU Libraries) is also installed in /apl/aocl. You can load this with module command. BLIS (libblis) and libflame correspond to BLAS and LAPACK, respectively. Other libraries such as FFTW are also included.

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

Full list of available versions can be found with "module avail aocl" command orInstalled Applications page in this website.

Intel oneAPI

RCCS can't provide compilers to users, since we don't have the license. (MKL and MPI are installed.) However, you can install Intel oneAPI compilers in your home directory (as of Jul 11, 2025).

Please be careful when you use newer compilers (icx, icpx, ifx). Those ones seem to be different from classic compilers (icc, icpc, ifort). Old and/or complicated programs often suffer from their differences. In the case of huge software, you many need to modify some codes or configurations to use newer compilers. Please also note that there are some changes in compiler options etc.

Installation and Setup

Intel oneAPI Base Toolkit can be downloaded from this website.C/C++ compilers, MKL, and other packages are included. Linux Online or Offline version can be used for RCCS system. If you need Fortran compiler and/or MPI library, please also download HPC Toolkit.

If you are using bash or zsh, you just need to run "source ~/intel/oneapi/setvars.sh" after the installation. If you need only the specific component like compiler or MKL, please source configuration script in env directory of each component. (For example, "source ~/intel/oneapi/compiler/latest/env/vars.sh" to load compilers.) In case of csh, this straightforward procedure is not available. There are several ways to load the environment, we here recommend to use "module" command.

Make Private Modules

(This procedure also works for bash/zsh. However, loading "~/intel/oneapi/setvars.sh" is already simple and easy enough.)

Assume oneAPI is installed under ~/intel. You can create module files with "modulefiles-setup.sh" script.

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

(Output path of modulefiles-setup.sh can be modified with option.) Then, you need to register that path for the module search path ("module use ." command). The module search path is saved with "module save" command. The saved module path is loaded next time when you log in ("module restore" command is implicitly invoked). You can load compiler with "module load compiler/latest" command. Please check "module avail" command for details about available versions.

If you load compilers before "module save" command, you can use oneAPI compilers immediately after the log in.

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

The saved module environment can be removed with "module saverm" command. Please note that once the environment is saved, it will not be able to follow changes in the system's default settings.

MKL

Compiler and linker options for MKL are complicated. Please useoffical advisor or "mkl_link_tool" command to prepare them. Running "mkl_link_tool" without arguments will show you the available options. Following options are expected to be used frequently.

- -c : Specify compiler. intel_c => icc, ifx => ifx, gnu_c => gcc, gnu_f => gfortran
- -p : Parallel setting. OpenMP is enabled with "-p yes". (Default; corresponding to -qmkl=parallel of intel compilers.) To disable OpenMP, please try "-p no". (Corresponding to -qmkl=sequential of intel compilers.)
- -m : MPI library setting (for scalapack etc). MPI libraries such as Intel MPI (intelmpi) and Open MPI (openmpi) are available. Default is intelmpi.
- --cluster_library : Choice of cluster library in case of MPI. Scalapack (scalapack) etc. can be specified. (e.g. --cluster_library=scalapack)
- For other options, you can request 64-bit integer version (ilp64), static links, and openmp library other than libiomp5.

We here show some example usage of mkl_link_tool. (--quiet is added to simplify the output.)

■ gfortran without OpenMP parallelism (-c gnu_f -p no)

compilation options (-opts): (don't type the first $.)

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

result:

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

linker options (-libs): (don't type the first $.)

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

result:

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

■ ifort + scalapack with openmpi

compilation options (-opts): (don't type the first $.)

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

result:

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

linker options (-libs): (don't type the first $.)

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

result:

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

-xHost option

-xHost option of the Intel compilers' recent versions is usually not a problem for AMD processors. However, in classic compiler (icpc), there is a case where "pragma simd" compiler directive does not work as expected when -xHost is specified. (NOTE: icpx is free from the issue. -xHost works as expected.) In such case, you should try "-march=core-avx2" instead of "-xHost". At least for the case of icpc, this replacement works fine.

NVIDIA HPC SDK

Modulefile for NVIDIA HPC SDK is also available. If you need OpenACC (for GPU) and CUDA Fortran compilers, you may want to try this one. For old software which require PGI compilers, you can try with this new SDK.

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

Please use -nompi version first. (MPI version lacks support for the queuing system. It may cause some disadvantages.) If you need MPI version, please try module like "openmpi/4.1.6/nv23". (You need to load nvhpc/23.9-nompi and openmpi/4.1.6/nv23 modules.) Although this openmpi is not a official build, the MPI library supports our queuing system. CUDA-aware MPI is also active in this installation.

If you want to use your own compiler with NVIDIA HPC SDK environment, please try "byo" (bring-your-own compiler) version of module.

CUDA

CUDA should be already loaded when you logged in. You can use different version of CUDA. You can "switch" CUDA version (e.g. module switch cuda/11.8). Or, clear the environment first and then load the version you want.

```
$ 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
```

Full list of available versions can be found with "module avail cuda" command orInstalled Applications page in this website.

Other Tools (autoconf, cmake) and Libraries (boost, openblas)

There can be newer versions of libraries/tools in /apl directory. In the "module avail" command, those libraries/tools would be listed in /apl/modules/util section. For MKL, they are installed in /apl/modules/oneapi/mkl directory. You may need to specify the software version explicitly.

Examples for OpenBLAS, Boost, and Ninja:

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

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

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

If the module load command fails due to the dependencies, please run "module purge" before loading the target module,

Tips on Compiler and Library Selection

Binaries built with GCC (especially newer versions), Intel Compiler, and AOCC often don't show significant difference in performance. On the other hand, some of software are strangely slow or unstable with certain compilers. If you are planning to run for a long period of time, it may be better to try other compilers in addition to changing versions with a particular compiler.

If you can ignore the minor performance differences or you are not looking for that much speed, we recommend you to try GCC (gcc, g++, gfortran) first. The latest version of GCC can be the good first choice (e.g. module load gcc-toolset/13); newer version is often better than old versions. You need to load gcc-toolset module when you build binaries. But you usually don't need to load the module when you run the program.

For BLAS, sometimes OpenBLAS is bit faster than MKL. BLIS and libflame in AOCL might be the fastest in some cases (not yet carefully investigated). For MKL, explicit declaration of "export MKL_ENABLE_INSTRUCTIONS=AVX2" may be required to achieve the best performance for some cases (depending of software and mkl version).