AMD

AOCC Install Guide

Publication # **57223**	Revision: **3.1**
Issue Date: **July 2021**	

Contents

List of Tables

Revision History

Date	Revision	Description
July 2021	3.1	• Updated Table 1. • Updated Chapter 4 and 5.
March 2021	3.0	Initial version.

Chapter 1 Prerequisite

The following software packages must be installed prior to the AOCC installation:

Table 1. Prerequisites

Package Name	Version(s)	Notes
GCC	5.1.0 or later	C/C++ compiler
libstdc++	6 or later	GNU Standard C++ Library V3
libncurses-dev	5.9 or later	Provides libtinfo, which is a low level terminfo library
zlib	1.2.7 or later	Compression library
Libxml2	2 or later	Parses the XML documents
libquadmath	4.8 or later	GCC Quad-Precision Math Library
python	3.x	Python library

Notes:

1. *For a better performance, it is recommended to use the latest versions of Glibc and Binutils.*
2. *AOCC compiler binaries are suitable only for the Linux® systems having Glibc version 2.17 or later.*

Chapter 2 Installing AOCC on Linux®

Note: *This installation does not required root or sudo permission.*

To install *aocc-compiler-<ver>.tar*, execute the following commands:

1. `cd <compdir>`
2. `tar -xvf aocc-compiler-<ver>.tar`
3. `cd aocc-compiler-<ver>`
4. `bash install.sh`

   It will install the compiler and display the AOCC setup instructions.
5. `source <compdir>/setenv_AOCC.sh`

   This will setup the shell environment for using AOCC C, C++, and Fortran compiler where the command is executed.

Chapter 3 Using AOCC

Ensure the following:

- Run the bash command `<compdir>/aocc-compiler-<ver>/AOCC-prerequisites-check.sh` to check if you have all the prerequisites and your shell environment is configured correctly.
  - If there are failing checks, correct them (repeat any of the above steps that you may have missed) and run `prerequisites_check.sh` again.
  - Repeat until `AOCC-prerequisites-check.sh` displays **Check:PASSED**.

    Note: *You could proceed if the packages mentioned in the warnings during the failing checks are not required for your run.*

- The compiler is installed and your environment is configured to the current release of AOCC. At any point, you can execute the command `source <compdir>/setenv_AOCC.sh` to set the environment variables for the installed compiler.

  The distributable AOCC runtime libraries are available under **AOCCRuntimeLibraries** folder along with the end user license agreement in the compiler installed path.

To use AOCC, execute the following commands:

1. `source <compdir>/setenv_AOCC.sh`
2. To build and run a C, execute the following commands:

```
–   $ clang [command line options] xyz.c -o xyz
–   $ ./xyz
```

3. To build and run a C++ program, execute the following commands:

```
–   $ clang++ [command line options] xyz.cpp -o xyz
–   $ ./xyz
```

   For more information on the optimization options, read the options listed in the *Clang - the C, C++ Compiler Guide*.

4. To build and run Fortran programs, execute the following commands:

```
–   $ flang [command line options] xyz.f90 -o xyz
–   $ ./xyz
```

   For information on the optimization options, read the options listed in *Flang - the Fortran Compiler Guide*.

3.1 Libraries

Some applications will benefit from the optimized libraries. AOCC will work seamlessly with these libraries. It is recommended that you evaluate these libraries while building your application with AOCC. They will boost the performance of your application over the compiler optimizations that come with AOCC. For more information on AMD Optimizing CPU Libraries (AOCL), refer *https://developer.amd.com/amd-aocl/*.

3.1.1 Configuring Library Path

Execute the following command to configure the library path:

```
•   For 64-bit Library
    export LD_LIBRARY_PATH=<compdir>/aocc-compiler-<ver>/lib:$LD_LIBRARY_PATH
•   For 32-bit Library
    export LD_LIBRARY_PATH=<compdir>/aocc-compiler-<ver>/lib32:$LD_LIBRARY_PATH
•   For other AMD optimizing CPU libraries
    export LD_LIBRARY_PATH=<Path to AMD optimizing CPU Libraries>:$LD_LIBRARY_PATH
```

3.1.2 Generate Vector Library Calls

Execute one of the following commands to generate the vector library calls from AOCC:

```
•   $ clang [command line flags] xyz.c -fveclib=AMDLIBM -o xyz.out
•   $ clang [command line flags] xyz.c -mllvm -vector-library=AMDLIBM -o xyz.out
```

3.1.3 Linker Option

Execute the following command to link AMDLIBM with the linker:

```
$ clang [command line flags] xyz.c -L<compdir>/aocc-compiler-<ver>/lib -lalm -o xyz.out
```

Execute the following command to link other AMD optimizing CPU libraries with linker:

```
$ clang [command line flags] xyz.c -L<Path to AMD optimizing CPU Libraries> -l<library name>
-o xyz.out
```

3.2 Upgrading AMD LibM (ALM)

This is required only when you are upgrading AMD LibM from the *AMD portal*.

Complete the following steps to perform an upgrade:

1. Extract the latest AMD LibM package.
2. Overwrite *aocc-compiler-<ver>/lib/libalm.so* and *aocc-compiler-<ver>/lib/libalm.a* with the latest versions of *libalm.so* and *libalm.a* respectively.
3. Similarly, overwrite *aocc-compiler-<ver>/include/amdlibm.h* and *amdlibm_vec.h* with the latest versions of *amdlibm.h* and *amdlibm_vec.h* respectively.

3.3 OpenMP Debugging Support (OMPD)

Note: *This is available in AOCC 2.3 or later.*

The AOCC 3.1 installation includes OMPD for debugging C/C++ OpenMP programs through a *gdb* plugin with limited functionality.

Note: *Debugging the code that runs on an offloading device is not supported.*

Complete the following steps to use OMPD for debugging C/C++ OpenMP programs through a *gdb* plugin:

Note: *For using the OMPD plugin, Python 3.5 or later is required.*

1. Add folders **ompd** and **lib** to your **LD_LIBRARY_PATH** using this command:

```
$ export LD_LIBRARY_PATH=<compdir>/aocc-compiler-<ver>/ompd:<compdir>/aocc-compiler-
<ver>/lib:$LD_LIBRARY_PATH
```

2. Set OMP_DEBUG to enabled.

```
$ export OMP_DEBUG=enabled
```

3. Compile the program to be debugged with **-g** and **-fopenmp** options as follows for a sample C source file *xyz.c*:

```
$ <compdir>/aocc-compiler-<ver>/bin/clang -g -fopenmp xyz.c -o xyz.out
```

Note: *The program to be debugged needs to have a dynamic link dependency on 'libomp.so' under <compdir>/aocc-compiler-<ver>/lib for OpenMP-specific debugging to work correctly. The user can check this using ldd on the generated binary i.e. xyz.out.*

4. Debug the binary *xyz.out* by invoking *gdb* with the plugin as follows:

```
$ gdb -x <compdir>/aocc-compiler-<ver>/ompd/__init__.py ./xyz.out
```

Note: *The plugin <compdir>/aocc-compiler-<ver>/ompd/__init__.py must be used.*

3.3.1 OMPD Commands

The following table describes the OMPD commands:

Table 2. OMPD Commands

Command	Description
`help ompd`	It lists the subcommands available for OpenMP specific debugging.
`ompd init`	• It must be run first to load the *libompd.so* available in the $LD_LIBRARY_PATH environment variable and to initialize the OMPD library. • It starts the program run and the program stops at a temporary breakpoint at the OpenMP internal location *ompd_dll_locations_valid()*. • You can continue from the temporary breakpoint for debugging. • You can place breakpoints at the OpenMP internal locations *ompd_bp_thread_begin* and *ompd_bp_thread_end* to catch the begin and end events • *ompd_bp_task_begin* and *ompd_bp_task_end* breakpoints can be used to catch the beginning and ending of the events • *ompd_bp_parallel_begin* and *ompd_bp_parallel_end* can be used to catch the beginning and ending of the parallel events.

3.3.2 OMPD Subcommands

The following table lists the OMPD subcommands that can used inside gdb:

Table 3. OMPD Subcommands

Subcommand	Description
`ompd init`	Finds and initializes the OMPD library.
`ompd bt`	Used to turn the filter on or off for the bt output on or off. You must specify the **on continued** option to trace the worker threads back to the master threads.
`ompd icvs`	Displays the values of the Internal Control Variables.
`ompd parallel`	Displays the details of the current and enclosing parallel regions.
`ompd step`	Executes step and skip runtime frames as much as possible.
`ompd threads`	Provides the details of the current threads.

Chapter 4 Supported Operating Systems (OS)

The following OS are supported in this release:

- RHEL 8.x
- SLES 15
- Ubuntu® 20.04 LTS
- CentOS 8.x
- Other Linux® flavors/versions with glibc 2.17 or higher

Chapter 5 Known Issues and Limitations

This release has the following known issues and limitations:

- AOCC binaries can run optimally only on Linux® systems having *glibc* version 2.17 or later.
- Currently, Flang supports only 64-bit targets.