AOCC User Guide
© 2020 - 2021 Advanced Micro Devices, Inc. All rights reserved.

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale. Any unauthorized copying, alteration, distribution, transmission, performance, display or other use of this material is prohibited.

Trademarks

AMD, the AMD Arrow logo, AMD-V, AMD Virtualization, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

Windows is a registered trademark of Microsoft Corporation.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Document Information

Software Version: 3.0
Document Version: 1.0
Last Updated on: 12th March 2021
Contents

1 Introduction ........................................................................................................................................... 1
2 Programming Language Support ........................................................................................................... 1
   2.1 C and C++ Programs ....................................................................................................................... 1
   2.2 Fortran Programs ............................................................................................................................. 1
3 Support ................................................................................................................................................ 2
   3.1 Logging a Ticket ............................................................................................................................... 2
   3.2 Creating an AMD SSO Login .......................................................................................................... 2
4 Using AOCC .......................................................................................................................................... 3
   4.1 Prerequisite ..................................................................................................................................... 3
   4.2 AOCC Optimizer .............................................................................................................................. 3
   4.3 Using the Compiler .......................................................................................................................... 3
      4.3.1 C and C++ ................................................................................................................................. 3
      4.3.2 Flang .......................................................................................................................................... 3
   4.4 Using an LLD linker ......................................................................................................................... 3
   4.5 Using the Libraries .......................................................................................................................... 3
      4.5.1 Configuring Library Path ......................................................................................................... 4
      4.5.2 Generate Vector Library Calls ................................................................................................. 4
      4.5.3 Linker Option ............................................................................................................................ 4
   4.6 OpenMP Debugging Support (OMPD) ............................................................................................. 4
      4.6.1 OMPD Commands .................................................................................................................... 5
      4.6.2 OMPD Subcommands .............................................................................................................. 5
   4.7 Command Line Options .................................................................................................................... 6
1 Introduction

AMD Optimizing C/C++ Compiler (AOCC) is a highly optimized C, C++, and Fortran compiler for x86 targets, especially for Zen based AMD processors. It supports Flang as the default Fortran front-end compiler. This document does not cover the internals of Low Level Virtual Machine (LLVM) or AOCC.

AOCC 3.0 is based on LLVM 12 trunk (llvm.org, 22nd Oct 2020) with Flang as a Fortran front-end added with F2008, Real 128 features. AOCC 3.0 also includes the support for OpenMP Debugging Interface (OMPD) APIs.

2 Programming Language Support

LLVM is a compiler infrastructure covering many programming languages and target processors. AOCC focuses on the following:

- C, C++, and Fortran programming languages
- Target dependent (x86 targets especially AMD processors) and target independent optimizations

AOCC leverages LLVM Clang, which is the compiler and driver for C and C++ programs. Flang is the compiler and driver for the Fortran programs.

2.1 C and C++ Programs

Clang is not just a C and C++ frontend, which compiles the program to LLVM intermediate representation (IR). It is also the driver, which ensures that it invokes all the required LLVM optimization passes and targets code generation to generating the binaries. You can use Clang as an end-to-end driver or if you are an advanced user, you can do a multi-pass compilation calling for each of the compilation phases manually. The AOCC package comes with all the tools and libraries essential for using clang as an end-to-end driver or to perform a manual multi-pass compilation.

2.2 Fortran Programs

Flang is the Fortran frontend designed for an integration with LLVM and is suitable for an interoperability with Clang/LLVM. Flang consists of the following two components:

- flang1: It is invoked by the front-end driver, which is responsible for transforming the Fortran programs into tokens. Then, the parser transforms these tokens into Abstract Syntax Tree (AST). This AST is then transformed into canonical form, which is used to generate the ILM code.
- flang2: It picks up the ILM code from flang1 and transforms it into ILI, which is then optimized by the internal optimizer. The optimized ILI is then transformed into LLVM IR. The frontend driver transfers this LLVM IR to the LLVM optimizer for optimization and target code generation.

Note: AOCC's flang extends the GitHub version with enhancements and stability.
3 Support

If you encounter any issues with this release or need assistance using the compiler, you must log a ticket in the AMD support portal using your AMD SSO login.

**Note:** If required, the project name is CPUPerfCompiler.

This support channel between AMD and you is 1:1. So, you can feel free to share programs or code snippets to help us assist you better. AMD assures you that these discussions and code snippets will strictly remain confidential. They will not be shared with anyone else under any circumstances.

3.1 Logging a Ticket

Complete the following steps to log a ticket in the AMD support portal:

1. Click the **Create** button on the top of the Web page.
2. Please select the following options from the respective drop-down lists:
   - **Project:** CPUPerfCompiler (CPUP)
   - **Issue Type:** Bug/New Feature (for feature request)/Task (for Support)
3. Provide clear reproducible steps and a test case.
   **Note:** You can provide us the code to help us serve you better.
4. Mention the output of command `clang -v` (provide the AOCC version you are using).
5. Select the severity based on the impact.

3.2 Creating an AMD SSO Login

Complete the following steps to create an AMD SSO login:

2. Click **here**. The SSO registration page is displayed.
3. Fill in all the mandatory details and click **Register**.

   **Note:** Provide the AMD contact person’s name and email ID accurately. He/she will initiate the process in AMD based on your NDA agreement.
4 Using AOCC

4.1 Prerequisite
The AOCC binaries can run optimally only on the Linux systems using glibc version 2.17 and later.

4.2 AOCC Optimizer
AOCC includes many optimization independent and dependent targets. A few of these are made default when you use an optimization level 03 and above. You can read more about these in the command line option section. A few other optimizations need a whole program analysis. Hence, they are enabled under Link Time Optimization (LTO) using -flto. The AOCC preferred linker is LLD. Refer the below section for using lld in the compiler driver.

4.3 Using the Compiler

4.3.1 C and C++
To build and run a C or C++ program, execute the following commands:

- $ clang [command line flags] xyz.c -o xyz.out
- $./xyz.out

- $ clang++ [command line flags] xyz.cpp -o xyz.out
- $./xyz.out

4.3.2 Flang
To build and run Fortran programs, execute the following commands:

- $ flang [command line flags] xyz.f90 -o xyz.out
- $./xyz.out

4.4 Using an LLD linker
To use an LLD linker, execute the following commands:

- $ clang [command line flags] -fuse-ld=lld xyz.c abc.c -o xyz [here -fuse-ld=lld is optional as this option is default]
- $./xyz

4.5 Using the 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.
4.5.1 Configuring Library Path
Execute the following command to configure the library path:

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

4.5.2 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`

4.5.3 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`
- `/xyz.out`

4.6 OpenMP Debugging Support (OMPD)
The AOCC 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.

Add folders ompd and lib to your LD_LIBRARY_PATH

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

Set OMP_DEBUG to enabled.

- `export OMP_DEBUG=enabled`

Compile the program to be debugged with `-g` and `-fopenmp` options as shown 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`. 
Debug the binary xyz.out by invoking gdb with the plugin as shown below.

Please note that plugin ‘<compdir>/aocc-compiler-<ver>/ompd/__init__.py’ should be used.

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

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.

### 4.6.1 OMPD Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>help ompd</td>
<td>It lists the subcommands available for OpenMP specific debugging.</td>
</tr>
</tbody>
</table>
| 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. |

### 4.6.2 OMPD Subcommands

The following is the list OMPD subcommands that can used inside gdb:

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ompd init</td>
<td>It finds and initializes the OMPD library.</td>
</tr>
<tr>
<td>ompd bt</td>
<td>It is 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.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ompd icvs</td>
<td>It displays the values of the Internal Control Variables.</td>
</tr>
<tr>
<td>ompd parallel</td>
<td>It displays the details of the current and enclosing parallel regions.</td>
</tr>
<tr>
<td>ompd step</td>
<td>It executes step and skip runtime frames as much as possible.</td>
</tr>
<tr>
<td>ompd threads</td>
<td>It provides the details of the current threads.</td>
</tr>
</tbody>
</table>

### 4.7 Command Line Options

**Clang - the C, C++ Compiler**
- Refer OPTIONS section in Clang – the C, C++ Compiler document

**Flang - the Fortran Compiler**
- Refer OPTIONS section in Flang – the Fortran Compiler document