# AOCC User guide

**Contents**



# Introduction

"AMD Optimizing C/C++ Compiler" - abbreviated as AOCC is a highly optimized C, C++ and Fortran compiler for x86 targets especially for Zen based AMD processors. This AOCC 1.3.0 is based on LLVM 7.0 release (llvm.org, 19th Sep 2018) and enhanced with various high-level optimizations. It supports Flang as an experimental Fortran compiler along with the default DragonEgg. This manual does not cover the internals of LLVM or AOCC compiler

## Programming language support

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

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

AOCC leverages LLVM's Clang which is the compiler and driver for C and C++ programs and DragonEgg which is the compiler and driver for Fortran programs. Also, introducing Flang as an experimental Fortran frontend in addition to the existing stable DragonEgg.

## C, C++ programs

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

## Fortran programs

*DragonEgg* is the primary Fortran frontend for AOCC. It is a gcc plugin that replaces GCC's optimizers and code generators with those from the LLVM project. DragonEgg that comes with AOCC works with gcc-4.8.2, can target the x86-32/x86-64 and has been successfully tested on various Linux platforms.

Due to this dependency gcc-4.8.2 is packaged as part of AOCC-1.3.0-Fortran-Prerequisites.tar.

GFortran is the actual frontend responsible for preprocessing, parsing and semantic analysis generating the GCC GIMPLE IR. DragonEgg is a GNU plugin, plugging into GFortran compilation flow. It implements the GNU plugin API. With the plugin architecture, DragonEgg becomes the compiler driver, driving the different phases of compilation. DragonEgg is responsible for transforming GIMPLE IR to LLVM IR. DragonEgg driver transfers this transformed LLVM IR to LLVM optimizer for optimization and target code generation.

## Introducing Flang - A new experimental LLVM native Fortran frontend

*Flang* is the Fortran frontend designed for integration with LLVM and suitable for interoperability with Clang/LLVM. Flang consists of two components flang1 and flang2. Flang1 will be invoked by front end driver which is responsible for transforming the Fortran programs in to tokens, then the parser transforms these tokens in to Abstract Syntax Tree (AST). This AST is then transformed in to canonical form which is used to generate ILM code. Then flang2 takes up this ILM code and transforms in to ILI, which is then optimized by internal optimizer. The optimized ILI is then transformed in to LLVM IR. Then, the frontend driver transfers this LLVM IR to LLVM optimizer for optimization and target code generation.

**Note**

1. This is an early drop (alpha quality) for your experimentation and currently supports only 64-bit targets
2. Do send us your feedbacks and the list of improvements you would like to see in Flang if it needs to become AOCC's primary Fortran compiler, either on support forum AMD Server Gurus or email toolchainsupport@amd.com

AOCC 1.3.0 extends the GitHub version found here with enhancements and stability

# How to get help/support, report bugs

If you encounter issues with this release or need assistance using the compiler, please log a ticket at support-channel weblink, http://ontrack.amd.com/projects/CPUP (project name if needed is CPUPerfCompiler) using your AMD SSO login (if you haven't created a AMD SSO login yet, do look at steps mentioned below)

This support-channel is 1:1 support channel between AMD and you. 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 between you and AMD. Under no circumstances will this be shared with anyone else.

Steps to log support/bug report:

1. Click on "Create" button on the top of the web page.
2. Please Select
    - Project: CPUPerfCompiler (CPUP)
    - Issue Type: Bug/New Feature (for feature request)/Task (for Support)
3. Please provide clear reproducible steps and a test case (feel free to provide us code to help us serve you better)
4. Do mention the output of command Clang -v (help us with the AOCC version you are using)
5. You can set the severity based on how this issue impacts you

Email based support: You can also opt for 1x1 support, report issues or seek expert help by emailing toolchainsupport@amd.com . This channel is open to all customers.

## Steps to create a AMD SSO login

Please visit http://ontrack.amd.com/projects/CPUP and you will find a link down below on the login screen, to create a AMD SSO login. Alternately, you can go directly to the weblink https://sso-registration.amd.com/ to place a request. Please make sure you provide your AMD contact person's name and email ID while making this request (the AMD contact person will initiate the process within AMD based on your NDA agreement and so important to provide the AMD contact person's details)

# Using AOCC

Warning: AOCC binaries are suitable to run on Linux systems having glibC version 2.17 and above only

# AOCC optimizer

AOCC includes many target independent and target dependent optimization. Some of these are made default when you use optimization level *-O3* and above. You can read more about these in the command line option section. Some of the other optimizations need whole program analysis and is hence enabled under link-time-optimization (LTO) enabled using -flto. AOCC's preferred linker is 'lld'. Refer the below section for using lld in the complier driver

## Using the compiler

### *C/C++ compiler - To build and run a C/C++ program*

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

### *Using lld linker*

- $ clang [command line flags] -fuse-ld=lld xyz.c -o xyz
- $./xyz

### *Fortran compiler - To build and run Fortran programs:*

- $ gfortran [optimization flags] -fplugin=<absolute path>/dragonegg.so [AOCC optimization flags] -c xyz.f90

- $ clang -O3 -lgfortran -o xyz xyz.o
- $./xyz
- optimization flags: flags that GFortran frontend will use to generate the IR for DragonEgg plugin. It is recommended that you use basic out-of-the-box eg: “-m64 -O2”, preferably least GFortran optimization(“-O0”)
- AOCC optimization flags: Optimization flags DragonEgg plugin will use to generate the optimized LLVM IR and code generation
- Some applications may benefit from optimized libraries. AOCC is known to work seamlessly with these libraries. It is recommended that you evaluate these libraries while building your application with AOCC as they may help boost the performance of your application over and above the compiler optimizations that come with AOCC. In most cases these libraries only need to be linked.
    - clang [command line flags] xyz.cpp <compdir>/AOCC-1.3.0-Compiler/lib -llibamdlibm -o xyz
    - export LD_LIBRARY_PATH=<compdir>/AOCC-1.3.0-Compiler/lib:$LD_LIBRARY_PATH
    - ./xyz

### *Flang - To build and run Fortran programs:*

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

## Command line options

#### Clang - the C, C++ Compiler

C, C++ compiler command line options listing

#### DragonEgg - the Fortran compiler

Fortran compiler command line options listing

#### Flang - the Fortran compiler

Fortran compiler command line options listing