AMD ROCm™

HIP Programming Guide

Publication # **1.0** Revision: **1217**
Issue Date: **December 2020**

Specification Agreement

This Specification Agreement (this "Agreement") is a legal agreement between Advanced Micro Devices, Inc. ("AMD") and "You" as the recipient of the attached AMD Specification (the "Specification"). If you are accessing the Specification as part of your performance of work for another party, you acknowledge that you have authority to bind such party to the terms and conditions of this Agreement. If you accessed the Specification by any means or otherwise use or provide Feedback (defined below) on the Specification, You agree to the terms and conditions set forth in this Agreement. If You do not agree to the terms and conditions set forth in this Agreement, you are not licensed to use the Specification; do not use, access or provide Feedback about the Specification. In consideration of Your use or access of the Specification (in whole or in part), the receipt and sufficiency of which are acknowledged, You agree as follows:

1. You may review the Specification only (a) as a reference to assist You in planning and designing Your product, service or technology ("Product") to interface with an AMD product in compliance with the requirements as set forth in the Specification and (b) to provide Feedback about the information disclosed in the Specification to AMD.

2. Except as expressly set forth in Paragraph 1, all rights in and to the Specification are retained by AMD. This Agreement does not give You any rights under any AMD patents, copyrights, trademarks or other intellectual property rights. You may not (i) duplicate any part of the Specification; (ii) remove this Agreement or any notices from the Specification, or (iii) give any part of the Specification, or assign or otherwise provide Your rights under this Agreement, to anyone else.

3. The Specification may contain preliminary information, errors, or inaccuracies, or may not include certain necessary information. Additionally, AMD reserves the right to discontinue or make changes to the Specification and its products at any time without notice. The Specification is provided entirely "AS IS." AMD MAKES NO WARRANTY OF ANY KIND AND DISCLAIMS ALL EXPRESS, IMPLIED AND STATUTORY WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, TITLE OR THOSE WARRANTIES ARISING AS A COURSE OF DEALING OR CUSTOM OF TRADE. AMD SHALL NOT BE LIABLE FOR DIRECT, INDIRECT, CONSEQUENTIAL, SPECIAL, INCIDENTAL, PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, LOST PROFITS, LOSS OF CAPITAL, LOSS OF GOODWILL) REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE) AND STRICT PRODUCT LIABILITY OR OTHERWISE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

4. Furthermore, AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur.

5. You have no obligation to give AMD any suggestions, comments or feedback ("Feedback") relating to the Specification. However, any Feedback You voluntarily provide may be used by AMD without restriction, fee or obligation of confidentiality. Accordingly, if You do give AMD Feedback on any version of the Specification, You agree AMD may freely use, reproduce, license, distribute, and otherwise commercialize Your Feedback in any product, as well as has the right to sublicense third parties to do the same. Further, You will not give AMD any Feedback that You may have reason to believe is (i) subject to any patent, copyright or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any product or intellectual property incorporating or derived from Feedback or any Product or other AMD intellectual property to be licensed to or otherwise provided to any third party.

6. You shall adhere to all applicable U.S. import/export laws and regulations, as well as the import/export control laws and regulations of other countries as applicable. You further agree to not export, re-export, or transfer, directly or indirectly, any product, technical data, software or source code received from AMD under this license, or the direct product of such technical data or software to any country for which the United States or any other applicable government requires an export license or other governmental approval without first obtaining such licenses or approvals; or in violation of any applicable laws or regulations of the United States or the country where the technical data or software was obtained. You acknowledge the technical data and software received will not, in the absence of authorization from U.S. or local law and regulations as applicable, be used by or exported, re-exported or transferred to: (i) any sanctioned or embargoed country, or to nationals or residents of such countries; (ii) any restricted end-user as identified on any applicable government end-user list; or (iii) any party where the end-use involves nuclear, chemical/biological weapons, rocket systems, or unmanned air vehicles. For the most current Country Group listings,

or for additional information about the EAR or Your obligations under those regulations, please refer to the U.S. Bureau of Industry and Security's website at *http://www.bis.doc.gov/*.
7. The Software and related documentation are "commercial items", as that term is defined at 48 C.F.R. §2.101, consisting of "commercial computer software" and "commercial computer software documentation", as such terms are used in 48 C.F.R. §12.212 and 48 C.F.R. §227.7202, respectively. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.7202-1 through 227.7202-4, as applicable, the commercial computer software and commercial computer software documentation are being licensed to U.S. Government end users (a) only as commercial items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions set forth in this Agreement. Unpublished rights are reserved under the copyright laws of the United States.
8. This Agreement is governed by the laws of the State of California without regard to its choice of law principles. Any dispute involving it must be brought in a court having jurisdiction of such dispute in Santa Clara County, California, and You waive any defenses and rights allowing the dispute to be litigated elsewhere. If any part of this agreement is unenforceable, it will be considered modified to the extent necessary to make it enforceable, and the remainder shall continue in effect. The failure of AMD to enforce any rights granted hereunder or to take action against You in the event of any breach hereunder shall not be deemed a waiver by AMD as to subsequent enforcement of rights or subsequent actions in the event of future breaches. This Agreement is the entire agreement between You and AMD concerning the Specification; it may be changed only by a written document signed by both You and an authorized representative of AMD.

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. In addition, any stated support is planned and is also subject to change. 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.

[This page left blank intentionally]

Table of Contents

[This page is intentionally left blank]

Chapter 1 Introduction

HIP is a C++ Runtime API and Kernel Language that allows developers to create portable applications for AMD and NVIDIA GPUs from a single source code.

1.1 Features

The key features include:

- HIP is very thin and has little or no performance impact over coding directly in CUDA mode.
- HIP allows coding in a single-source C++ programming language including features such as templates, C++11 lambdas, classes, namespaces, and more.
- HIP allows developers to use the "best" development environment and tools on each target platform.
- The *HIPIFY* tools automatically convert source from CUDA to HIP.
- Developers can specialize in the platform (CUDA or AMD) to tune for performance or handle tricky cases.

New projects can be developed directly in the portable HIP C++ language and can run on either NVIDIA or AMD platforms. Additionally, HIP provides porting tools, which make it easy to port existing CUDA codes to the HIP layer, with no loss of performance as compared to the original CUDA application.

Thus, the HIP source code can be compiled to run on either platform. Platform-specific features can be isolated to a specific platform using conditional compilation.

NOTE: HIP is not intended to be a drop-in replacement for CUDA, and developers should expect to do some manual coding and performance tuning work to complete the port.

1.2 Accessing HIP

HIP is open source in GitHub and the repository maintains several branches.

The HIP repository maintains several branches. The important branches are:

- Release branch: This is the stable branch. All stable releases are listed with release tags.
- Main branch: This is the branch, with new features still under development. While this may be of interest to many, it should be noted that this branch and the features under development might not be stable.

For more information, refer to *https://github.com/ROCm-Developer-Tools/HIP*

1.2.1 Release Tagging

HIP releases are typically naming convention for each ROCM release to help differentiate them.

- rocm x.yy: These are the stable releases based on the development branch.

1.3 HIP Portability and Compiler Technology

HIP C++ code can be compiled with either AMD or NVIDIA GPUs. On the AMD ROCm platform, HIP provides a header and runtime library built on top of the HIP-Clang compiler. The HIP runtime implements HIP streams, events, and memory APIs, and is an object library that is linked with the application.

On the NVIDIA CUDA platform, HIP provides a header file, which translates from the HIP runtime APIs to CUDA runtime APIs. The header file contains mostly inline functions and, thus, has a very low overhead developers coding in HIP should expect the same performance as coding in native CUDA. The code is then compiled with nvcc, the standard C++ compiler provided with the CUDA SDK. Developers can use any tools supported by the CUDA SDK including the CUDA profiler and debugger.

Thus, HIP provides source portability to either platform. HIP provides the hipcc compiler driver which will call the appropriate toolchain depending on the desired platform. The source code for all headers and the library implementation is available on GitHub.

Chapter 2 Installing HIP

2.1 Installing Pre-built Packages

HIP can be easily installed using pre-built binary packages using the package manager for your platform.

2.2 Prerequisites

HIP code can be developed either on AMD ROCm platform using the HIP-Clang compiler, or a CUDA platform with nvcc installed.

2.3 AMD Platform

For HIP installation instructions, refer to the ROCm Installation Guide at

https://rocmdocs.amd.com/en/latest/Installation_Guide/Installation-Guide.html

HIP-Clang is the compiler for compiling HIP programs on the AMD platform.

To build HIP-Clang manually, use the following instructions:

```
git clone -b rocm-4.0.x https://github.com/RadeonOpenCompute/llvm-project.git
cd llvm-project
mkdir -p build && cd build
cmake -DCMAKE_INSTALL_PREFIX=/opt/rocm/llvm -DCMAKE_BUILD_TYPE=Release -
DLLVM_ENABLE_ASSERTIONS=1 -DLLVM_TARGETS_TO_BUILD="AMDGPU;X86" -
DLLVM_ENABLE_PROJECTS="clang;lld;compiler-rt" ../llvm
make -j
sudo make install
```

To build the ROCm device library,

```
export PATH=/opt/rocm/llvm/bin:$PATH
git clone -b rocm-4.0.x https://github.com/RadeonOpenCompute/ROCm-Device-Libs.git
cd ROCm-Device-Libs
mkdir -p build && cd build
CC=clang CXX=clang++ cmake -DLLVM_DIR=/opt/rocm/llvm -DCMAKE_BUILD_TYPE=Release -
DLLVM_ENABLE_WERROR=1 -DLLVM_ENABLE_ASSERTIONS=1 -DCMAKE_INSTALL_PREFIX=/opt/rocm ..
make -j
sudo make install
```

2.4 NVIDIA Platform

HIP-nvcc is the compiler for HIP program compilation on NVIDIA platform.

- Add the ROCm package server to your system as per the OS-specific guide available *here*.
- Install the "hip-nvcc" package. This will install CUDA SDK and the HIP porting layer.

```
apt-get install hip-nvcc
```

2.5 Default paths and environment variables

- By default, HIP looks for CUDA SDK in /usr/local/cuda (can be overriden by setting CUDA_PATH env variable).
- By default, HIP is installed into /opt/rocm/hip (can be overridden by setting HIP_PATH environment variable).
- Optionally, consider adding /opt/rocm/bin to your path to make it easier to use the tools.

2.6 Building HIP from Source

2.6.1 Build ROCclr

ROCclr is defined on AMD platform that HIP use Radeon Open Compute Common Language Runtime (ROCclr), which is a virtual device interface that HIP runtimes interact with different backends.

For more information, see *https://github.com/ROCm-Developer-Tools/ROCclr*

```
git clone -b rocm-4.0.x https://github.com/ROCm-Developer-Tools/ROCclr.git
export ROCclr_DIR="$(readlink -f ROCclr)"
git clone -b rocm-4.0.x https://github.com/RadeonOpenCompute/ROCm-OpenCL-Runtime.git
export OPENCL_DIR="$(readlink -f ROCm-OpenCL-Runtime)"
cd "$ROCclr_DIR"
mkdir -p build;cd build
cmake -DOPENCL_DIR="$OPENCL_DIR" -DCMAKE_INSTALL_PREFIX=/opt/rocm/rocclr ..
make -j
sudo make install
```

2.6.2 Build HIP

```
git clone -b rocm-4.0.x https://github.com/ROCm-Developer-Tools/HIP.git
export HIP_DIR="$(readlink -f HIP)"
cd "$HIP_DIR"
mkdir -p build; cd build
cmake -DCMAKE_BUILD_TYPE=Release -DHIP_COMPILER=clang -DHIP_PLATFORM=rocclr -
DCMAKE_PREFIX_PATH="$ROCclr_DIR/build;/opt/rocm/" -DCMAKE_INSTALL_PREFIX=</where/to/install/hip>
..
make -j
sudo make install
```

2.6.3 Default paths and environment variables

- By default, HIP looks for HSA in /opt/rocm/hsa (can be overridden by setting HSA_PATH environment variable).
- By default, HIP is installed into /opt/rocm/hip (can be overridden by setting HIP_PATH environment variable).
- By default, HIP looks for clang in /opt/rocm/llvm/bin (can be overridden by setting HIP_CLANG_PATH environment variable)
- By default, HIP looks for device library in /opt/rocm/lib (can be overridden by setting DEVICE_LIB_PATH environment variable).
- Optionally, consider adding /opt/rocm/bin to your PATH to make it easier to use the tools.
- Optionally, set HIPCC_VERBOSE=7 to output the command line for compilation.

After installation, ensure *HIP_PATH* is pointed to */where/to/install/hip*

2.7 Verifying HIP Installation

Run hipconfig (instructions below assume default installation path) :

```
/opt/rocm/bin/hipconfig --full
```

Compile and run the *square sample*.

Chapter 3 Programming with HIP

3.1 HIP Terminology

Term	Description
host, host cpu	Executes the HIP runtime API and is capable of initiating kernel launches to one or more devices.
default device	Each host thread maintains a "default device". Most HIP runtime APIs (including memory allocation, copy commands, kernel launches) do not use accept an explicit device argument but instead implicitly use the default device. The default device can be set with hipSetDevice.
active host thread	Thread running the HIP APIs.
HIP-Clang	Heterogeneous AMDGPU Compiler, with its capability to compile HIP programs on the AMD platform. *https://github.com/RadeonOpenCompute/llvm-project*
hipify tools	Tools to convert CUDA code to portable C++ code (https://github.com/ROCm-Developer-Tools/HIPIFY).
ROCclr	A virtual device interface that computes runtimes interact with different backends such as ROCr on Linux or PAL on Windows. The ROCclr is an abstraction layer allowing runtimes to work on both OSes without much effort. For more information, see *https://github.com/ROCm-Developer-Tools/ROCclr*
hipconfig	Tool to report various configuration properties of the target platform.
nvcc	nvcc compiler

3.2 Getting Started with HIP API

3.2.1 HIP API Overview

The HIP API includes functions such as hipMalloc, hipMemcpy, and hipFree. Programmers familiar with CUDA will also be able to quickly learn and start coding with the HIP API. Compute kernels are launched with the 'hipLaunchKernel's macro call.

For more information, refer to *Appendix A* on HIP APIs.

3.2.2 HIP API Examples

3.2.2.1 Example 1

Here is an example showing a snippet of HIP API code:

```
hipMalloc(&A_d, Nbytes));
hipMalloc(&C_d, Nbytes));
hipMemcpy(A_d, A_h, Nbytes, hipMemcpyHostToDevice);
const unsigned blocks = 512;
const unsigned threadsPerBlock = 256;
hipLaunchKernel(vector_square,   /* compute kernel*/
                dim3(blocks), dim3(threadsPerBlock), 0/*dynamic shared*/, 0/*stream*/,     /*
launch config*/
                C_d, A_d, N);  /* arguments to the compute kernel */
hipMemcpy(C_h, C_d, Nbytes, hipMemcpyDeviceToHost);
```

The HIP kernel language defines builtins for determining grid and block coordinates, math functions, short vectors, atomics, and timer functions. It also specifies additional defines and keywords for function types, address spaces, and optimization controls. For a detailed description, see

https://rocmdocs.amd.com/en/latest/Programming_Guides/Kernel_language.html#kernel-language

3.2.2.2 Example 2

Here's an example of defining a simple 'vector_square' kernel.

```
template <typename T>
__global__ void
vector_square(T *C_d, const T *A_d, size_t N)
{
    size_t offset = (hipBlockIdx_x * hipBlockDim_x + hipThreadIdx_x);
    size_t stride = hipBlockDim_x * hipGridDim_x;
    for (size_t i=offset; i<N; i+=stride) {
        C_d[i] = A_d[i] * A_d[i];
    }
}
```

The HIP Runtime API code and compute kernel definition can exist in the same source file - HIP takes care of generating host and device code appropriately.

3.3 Introduction to Memory Allocation

3.3.1 Host Memory

hipHostMalloc allocates pinned host memory which is mapped into the address space of all GPUs in the system. There are two use cases for this host memory:

- Faster HostToDevice and DeviceToHost Data Transfers: The runtime tracks the hipHostMalloc allocations and can avoid some of the setup required for regular unpinned memory. For exact measurements on a specific system, experiment with --unpinned and --pinned switches for the hipBusBandwidth tool.

- Zero-Copy GPU Access: GPU can directly access the host memory over the CPU/GPU interconnect, without need to copy the data. This avoids the need for the copy, but during the kernel access each memory access must traverse the interconnect, which can be tens of times slower than accessing the GPU's local device memory. Zero-copy memory can be a good choice when the memory accesses are infrequent (perhaps only once). Zero-copy memory is typically "Coherent" and thus not cached by the GPU but this can be overridden if desired and is explained in more detail below.

3.3.2 Memory allocation flags

hipHostMalloc always sets the hipHostMallocPortable and hipHostMallocMapped flags. Both usage models described above use the same allocation flags, and the difference is in how the surrounding code uses the host memory. See the hipHostMalloc API for more information.

3.3.3 Coherency Controls

ROCm defines two coherency options for host memory:

- Coherent memory: Supports fine-grain synchronization while the kernel is running. For example, a kernel can perform atomic operations that are visible to the host CPU or to other (peer) GPUs. Synchronization instructions include threadfence_system and C++11-style atomic operations. However, coherent memory cannot be cached by the GPU and thus may have lower performance.

- Non-coherent memory: Can be cached by GPU, but cannot support synchronization while the kernel is running. Non-coherent memory can be optionally synchronized only at command (end-of-kernel or copy command) boundaries. This memory is appropriate for high-performance access when fine-grain synchronization is not required.

HIP provides the developer with controls to select which type of memory is used via allocation flags passed to hipHostMalloc and the HIP_HOST_COHERENT environment variable:

- hipHostMallocCoherent=0, hipHostMallocNonCoherent=0: Use HIP_HOST_COHERENT environment variable:
    - If HIP_HOST_COHERENT is 1 or undefined, the host memory allocation is coherent.
    - If HIP_HOST_COHERENT is defined and 0: the host memory allocation is non-coherent.
- hipHostMallocCoherent=1, hipHostMallocNonCoherent=0: The host memory allocation will be coherent. HIP_HOST_COHERENT env variable is ignored.
- hipHostMallocCoherent=0, hipHostMallocNonCoherent=1: The host memory allocation will be non-coherent. HIP_HOST_COHERENT env variable is ignored.
- hipHostMallocCoherent=1, hipHostMallocNonCoherent=1: Illegal.

3.3.4 Visibility of Zero-Copy Host Memory

The coherent and non-coherent host memory visibility is described in the table below. Note, the coherent host memory is automatically visible at synchronization points.

HIP API	Synchronization Effect	Fence	Coherent Host Memory Visibility	Non-Coherent Host Memory Visibility
hipStreamSynchronize	host waits for all commands in the specified stream to complete	system-scope release	yes	yes
hipDeviceSynchronize	host waits for all commands in all streams on the specified device to complete	system-scope release	yes	yes
hipEventSynchronize	host waits for the specified event to complete	device-scope release	yes	depends - see the description below
hipStreamWaitEvent	stream waits for the specified event to complete	none	yes	no

3.3.4.1 hipEventSynchronize

Developers can control the release scope for hipEvents. By default, the GPU performs a device-scope acquire and release operation with each recorded event. This will make host and device memory visible to other commands executing on the same device.

A stronger system-level fence can be specified when the event is created with hipEventCreateWithFlags.

hipEventReleaseToSystem: Perform a system-scope release operation when the event is recorded. This will make both Coherent and Non-Coherent host memory visible to other agents in the system but may involve heavyweight operations such as cache flushing. Coherent memory will typically use lighter-weight in-kernel synchronization mechanisms, such as an atomic operation, and, thus, do not need to use hipEventReleaseToSystem.

Summary and Recommendations

- Coherent host memory is the default and is the easiest to use since the memory is visible to the CPU at typical synchronization points. This memory allows in-kernel synchronization commands such as threadfence_system to work transparently.
- HIP/ROCm also supports the ability to cache host memory in the GPU using the "Non-Coherent" host memory allocations. This can provide a performance benefit, but care must be taken to use the correct synchronization.

3.3.4.2 Device-Side Malloc

HIP-Clang currently does not support device-side malloc and free.

3.3.4.3 Use of Long Double Type

In HIP-Clang, long double type is 80-bit extended precision format for x86_64, which is not supported by AMDGPU. HIP-Clang treats long double type as IEEE double type for AMDGPU. Using long double type in HIP source code will not cause issue as long as data of long double type is not transferred between host and device. However, long double type should not be used as kernel argument type.

3.3.4.4 FMA and Contractions

By default, HIP-Clang assumes -ffp-contract=fast. For x86_64, FMA is off by default since the generic x86_64 target does not support FMA by default. To turn on FMA on x86_64, either use -mfma or -march=native on CPU's supporting FMA.

When contractions are enabled and the CPU has not enabled FMA instructions, the GPU can produce different numerical results than the CPU for expressions that can be contracted.

3.4 HIP Kernel Language

HIP provides a C++ syntax that is suitable for compiling most code that commonly appears in compute kernels, including classes, namespaces, operator overloading, templates and more. Additionally, it defines other language features designed specifically to target accelerators, such as the following:

- A kernel-launch syntax that uses standard C++, resembles a function call and is portable to all HIP targets
- Short-vector headers that can serve on a host or a device
- Math functions resembling those in the "math.h" header included with standard C++ compilers
- Built-in functions for accessing specific GPU hardware capabilities

This section describes the built-in variables and functions accessible from the HIP kernel. It's intended for readers who are familiar with CUDA kernel syntax and want to understand how HIP is different.

The features are marked with one of the following keywords:

- Supported - HIP supports the feature with a CUDA-equivalent function
- Not supported - HIP does not support the feature
- Under development - the feature is under development but not yet available

3.4.1 Function-Type Qualifiers

3.4.1.1 __device__

The supported __device__ functions are:

- Executed on the device
- Called from the device only

The __device__ keyword can combine with the host keyword (see host).

3.4.1.2 __global__

The supported __global__ functions are:

- Executed on the device
- Called ("launched") from the host

HIP __global__ functions must have a void return type. See the *Kernel Launch example* for more information.

HIP lacks dynamic-parallelism support, so __global__ functions cannot be called from the device.

3.4.1.3 __host__

The supported __host__ functions are:

- Executed on the host
- Called from the host

__host__ can combine with *__device__*, in which case the function compiles for both the host and device. These functions cannot use the HIP grid coordinate functions. For example, "hipThreadIdx_x". A possible workaround is to pass the necessary coordinate info as an argument to the function. *_host__* cannot combine with *__global__*.

HIP parses the *__noinline__* and *__forceinline__* keywords and converts them to the appropriate Clang attributes.

3.4.1.4 Calling __global__ Functions

__global__ functions are often referred to as kernels, and calling one is termed launching the kernel. These functions require the caller to specify an "execution configuration" that includes the grid and block dimensions. The execution configuration can also include other information for the launch, such as the amount of additional shared memory to allocate and the stream where the kernel should execute. HIP introduces a standard C++ calling convention to pass the execution configuration to the kernel in addition to the Cuda <<< >>> syntax.

- In HIP, kernels launch with either the <<< >>> syntax or the "hipLaunchKernel" function.
- The first five parameters to hipLaunchKernel are the following:
  - symbol kernelName: the name of the kernel to launch. To support template kernels which contains "," use the HIP_KERNEL_NAME macro. The hipify tools insert this automatically.
  - dim3 gridDim: 3D-grid dimensions specifying the number of blocks to launch.
  - dim3 blockDim: 3D-block dimensions specifying the number of threads in each block.
  - size_t dynamicShared: amount of additional shared memory to allocate when launching the kernel (see shared)
  - hipStream_t: stream where the kernel should execute. A value of 0 corresponds to the NULL stream (see Synchronization Functions).
- Kernel arguments must follow the five parameters

The hipLaunchKernel macro always starts with the five parameters specified above, followed by the kernel arguments. HIPIFY tools optionally convert CUDA launch syntax to hipLaunchKernel, including conversion of optional arguments in <<< >>> to the five required hipLaunchKernel parameters. The dim3 constructor accepts zero to three arguments and will by default initialize unspecified dimensions to 1. See dim3. The kernel uses the coordinate built-ins (hipThread*, hipBlock*, hipGrid*) to determine coordinate index and coordinate bounds of the work item that's currently executing. For more information, see Coordinate Built-Ins.

3.4.1.5 Kernel-Launch Example

```
// Example showing device function, __device__ __host__
// <- compile for both device and host
float PlusOne(float x)
{
    return x + 1.0;
}
__global__
void
MyKernel (const float *a, const float *b, float *c, unsigned N)
{
    unsigned gid = hipThreadIdx_x; // <- coordinate index function
    if (gid < N) {
        c[gid] = a[gid] + PlusOne(b[gid]);
    }
}
void callMyKernel()
{
    float *a, *b, *c; // initialization not shown...
    unsigned N = 1000000;
    const unsigned blockSize = 256;
    MyKernel<<<dim3(gridDim), dim3(groupDim), 0, 0>>> (a,b,c,n);
    // Alternatively, kernel can be launched by
    // hipLaunchKernel(MyKernel, dim3(N/blockSize), dim3(blockSize), 0, 0,  a,b,c,N);
}
```

3.4.2 Variable-Type Qualifiers

3.4.2.1 __constant__

The __constant__ keyword is supported. The host writes constant memory before launching the kernel; from the GPU, this memory is read-only during kernel execution. The functions for accessing constant memory (hipGetSymbolAddress(), hipGetSymbolSize(), hipMemcpyToSymbol(), hipMemcpyToSymbolAsync(), hipMemcpyFromSymbol(), hipMemcpyFromSymbolAsync()) are available.

3.4.2.2 __shared__

The __shared__ keyword is supported.

extern __shared__ allows the host to dynamically allocate shared memory and is specified as a launch parameter. HIP uses an alternate syntax based on the HIP_DYNAMIC_SHARED macro.

3.4.2.3 __managed__

Managed memory, including the __managed__ keyword, are not supported in HIP.

3.4.2.4 __restrict__

The __restrict__ keyword tells the compiler that the associated memory pointer will not alias with any other pointer in the kernel or function. This feature can help the compiler generate better code. In most cases, all pointer arguments must use this keyword to realize the benefit.

3.4.3 Built-In Variables

3.4.3.1 Coordinate Built-Ins

These built-ins determine the coordinate of the active work item in the execution grid. They are defined in hip_runtime.h (rather than being implicitly defined by the compiler).

HIP Syntax	CUDA Syntax
hipThreadIdx_x	threadIdx.x
hipThreadIdx_y	threadIdx.y
hipThreadIdx_z	threadIdx.z
hipBlockIdx_x	blockIdx.x
hipBlockIdx_y	blockIdx.y
hipBlockIdx_z	blockIdx.z
hipBlockDim_x	blockDim.x
hipBlockDim_y	blockDim.y
hipBlockDim_z	blockDim.z
hipGridDim_x	gridDim.x
hipGridDim_y	gridDim.y
hipGridDim_z	gridDim.z

3.4.3.2 warpSize

The warpSize variable is of type int and contains the warp size (in threads) for the target device. Note that all current Nvidia devices return 32 for this variable, and all current AMD devices return 64. Device code should use the warpSize built-in to develop portable wave-aware code.

3.4.4 Vector Types

Note that these types are defined in hip_runtime.h and are not automatically provided by the compiler.

3.4.4.1 Short Vector Types

Short vector types derive from the basic integer and floating-point types. They are structures defined in hip_vector_types.h. The first, second, third, and fourth components of the vector are accessible through the x, y, z, and w fields, respectively. All the short vector types support a constructor function of the form make_<type_name>(). For example, float4 make_float4(float x, float y, float z, float w) creates a vector of type float4 and value (x,y,z,w).

HIP supports the following short vector formats:

Signed Integers

- char1, char2, char3, char4
- short1, short2, short3, short4
- int1, int2, int3, int4
- long1, long2, long3, long4
- longlong1, longlong2, longlong3, longlong4

Unsigned Integers

- uchar1, uchar2, uchar3, uchar4
- ushort1, ushort2, ushort3, ushort4
- uint1, uint2, uint3, uint4
- ulong1, ulong2, ulong3, ulong4
- ulonglong1, ulonglong2, ulonglong3, ulonglong4

Floating Points

- float1, float2, float3, float4
- double1, double2, double3, double4

3.4.4.2 dim3

dim3 is a three-dimensional integer vector type commonly used to specify grid and group dimensions. Unspecified dimensions are initialized to 1.

```
typedef struct dim3 {
  uint32_t x;
  uint32_t y;
  uint32_t z;
  dim3(uint32_t _x=1, uint32_t _y=1, uint32_t _z=1) : x(_x), y(_y), z(_z) {};
};
```

3.4.5 Memory-Fence Instructions

HIP supports __threadfence() and __threadfence_block().

HIP provides a workaround for threadfence_system() under the HIP-Clang path. To enable the workaround, HIP should be built with environment variable HIP_COHERENT_HOST_ALLOC enabled.

Also, the kernels that use __threadfence_system() should be modified as follows:

- The kernel should only operate on finegrained system memory; which should be allocated with hipHostMalloc().
- Remove all memcpy for those allocated finegrained system memory regions.

3.4.6 Synchronization Functions

The __syncthreads() built-in function is supported in HIP. The __syncthreads_count(int), __syncthreads_and(int) and __syncthreads_or(int) functions are under development.

3.4.7 Math Functions

HIP-Clang supports a set of math operations callable from the device.

3.4.7.1 Single Precision Mathematical Functions

Following is the list of supported single-precision mathematical functions.

Function	Supported on Host	Supported on Device
float acosf (float x) Calculate the arc cosine of the input argument.	✓	✓
float acoshf (float x) Calculate the nonnegative arc hyperbolic cosine of the input argument.	✓	✓

Function	Supported on Host	Supported on Device
float asinf (float x) Calculate the arc sine of the input argument.	✓	✓
float asinhf (float x) Calculate the arc hyperbolic sine of the input argument.	✓	✓
float atan2f (float y, float x) Calculate the arc tangent of the ratio of first and second input arguments.	✓	✓
float atanf (float x) Calculate the arc tangent of the input argument.	✓	✓
float atanhf (float x) Calculate the arc hyperbolic tangent of the input argument.	✓	✓
float cbrtf (float x) Calculate the cube root of the input argument.	✓	✓
float ceilf (float x) Calculate ceiling of the input argument.	✓	✓
float copysignf (float x, float y) Create value with given magnitude, copying sign of second value.	✓	✓
float cosf (float x) Calculate the cosine of the input argument.	✓	✓
float coshf (float x) Calculate the hyperbolic cosine of the input argument.	✓	✓
float erfcf (float x) Calculate the complementary error function of the input argument.	✓	✓
float erff (float x) Calculate the error function of the input argument.	✓	✓

Function	Supported on Host	Supported on Device
float exp10f (float x) Calculate the base 10 exponential of the input argument.	✓	✓
float exp2f (float x) Calculate the base 2 exponential of the input argument.	✓	✓
float expf (float x) Calculate the base e exponential of the input argument.	✓	✓
float expm1f (float x) Calculate the base e exponential of the input argument, minus 1.	✓	✓
float fabsf (float x) Calculate the absolute value of its argument.	✓	✓
float fdimf (float x, float y) Compute the positive difference between x and y.	✓	✓
float floorf (float x) Calculate the largest integer less than or equal to x.	✓	✓
float fmaf (float x, float y, float z) Compute x × y + z as a single operation.	✓	✓
float fmaxf (float x, float y) Determine the maximum numeric value of the arguments.	✓	✓
float fminf (float x, float y) Determine the minimum numeric value of the arguments.	✓	✓
float fmodf (float x, float y) Calculate the floating-point remainder of x / y.	✓	✓
float frexpf (float x, int* nptr) Extract mantissa and exponent of a floating-point value.	✓	x

Function	Supported on Host	Supported on Device
float hypotf (float x, float y) Calculate the square root of the sum of squares of two arguments.	✓	✓
int ilogbf (float x) Compute the unbiased integer exponent of the argument.	✓	✓
__RETURN_TYPE1 isfinite (float a) Determine whether the argument is finite.	✓	✓
__RETURN_TYPE1 isinf (float a) Determine whether the argument is infinite.	✓	✓
__RETURN_TYPE1 isnan (float a) Determine whether the argument is a NaN.	✓	✓
float ldexpf (float x, int exp) Calculate the value of x · 2exp.	✓	✓
float log10f (float x) Calculate the base 10 logarithm of the input argument.	✓	✓
float log1pf (float x) Calculate the value of loge(1 + x).	✓	✓
float logbf (float x) Calculate the floating-point representation of the exponent of the input argument.	✓	✓
float log2f (float x) Calculate the base 2 logarithm of the input argument.	✓	✓
float logf (float x) Calculate the natural logarithm of the input argument.	✓	✓
float modff (float x, float* iptr) Break down the input argument into fractional and integral parts.	✓	x

Function	Supported on Host	Supported on Device
float nanf (const char* tagp) Returns "Not a Number" value.	x	✓
float nearbyintf (float x) Round the input argument to the nearest integer.	✓	✓
float powf (float x, float y) Calculate the value of the first argument to the power of the second argument.	✓	✓
float remainderf (float x, float y) Compute single-precision floating-point remainder.	✓	✓
float remquof (float x, float y, int* quo) Compute single-precision floating-point remainder and part of quotient.	✓	x
float roundf (float x) Round to nearest integer value in floating-point.	✓	✓
float scalbnf (float x, int n) Scale floating-point input by an integer power of two.	✓	✓
__RETURN_TYPE1 signbit (float a) Return the sign bit of the input.	✓	✓
void sincosf (float x, float* sptr, float* cptr) Calculate the sine and cosine of the first input argument.	✓	x
float sinf (float x) Calculate the sine of the input argument.	✓	✓
float sinhf (float x) Calculate the hyperbolic sine of the input argument.	✓	✓
float sqrtf (float x) Calculate the square root of the input argument.	✓	✓

Function	Supported on Host	Supported on Device
float tanf (float x) Calculate the tangent of the input argument.	✓	✓
float tanhf (float x) Calculate the hyperbolic tangent of the input argument.	✓	✓
float truncf (float x) Truncate input argument to an integral part.	✓	✓
float tgammaf (float x) Calculate the gamma function of the input argument.	✓	✓
float erfcinvf (float y) Calculate the inverse complementary function of the input argument.	✓	✓
float erfcxf (float x) Calculate the scaled complementary error function of the input argument.	✓	✓
float erfinvf (float y) Calculate the inverse error function of the input argument.	✓	✓
float fdividef (float x, float y) Divide two floating-point values.	✓	✓
float frexpf (float x, int *nptr) Extract mantissa and exponent of a floating-point value.	✓	✓
float j0f (float x) Calculate the value of the Bessel function of the first kind of order 0 for the input argument.	✓	✓
float j1f (float x) Calculate the value of the Bessel function of the first kind of order 1 for the input argument.	✓	✓
float jnf (int n, float x) Calculate the value of the Bessel function of the first kind of order n for the input argument.	✓	✓

Function	Supported on Host	Supported on Device
float lgammaf (float x) Calculate the natural logarithm of the absolute value of the gamma function of the input argument.	✓	✓
long long int llrintf (float x) Round input to nearest integer value.	✓	✓
long long int llroundf (float x) Round to nearest integer value.	✓	✓
long int lrintf (float x) Round input to the nearest integer value.	✓	✓
long int lroundf (float x) Round to nearest integer value.	✓	✓
float modff (float x, float *iptr) Break down the input argument into fractional and integral parts.	✓	✓
float nextafterf (float x, float y) Returns next representable single-precision floating-point value after an argument.	✓	✓
float norm3df (float a, float b, float c) Calculate the square root of the sum of squares of three coordinates of the argument.	✓	✓
float norm4df (float a, float b, float c, float d) Calculate the square root of the sum of squares of four coordinates of the argument.	✓	✓
float normcdff (float y) Calculate the standard normal cumulative distribution function.	✓	✓
float normcdfinvf (float y) Calculate the inverse of the standard normal cumulative distribution function.	✓	✓

Function	Supported on Host	Supported on Device
float normf (int dim, const float *a) Calculate the square root of the sum of squares of any number of coordinates.	✓	✓
float rcbrtf (float x) Calculate the reciprocal cube root function.	✓	✓
float remquof (float x, float y, int *quo) Compute single-precision floating-point remainder and part of quotient.	✓	✓
float rhypotf (float x, float y) Calculate one over the square root of the sum of squares of two arguments.	✓	✓
float rintf (float x) Round input to nearest integer value in floating-point.	✓	✓
float rnorm3df (float a, float b, float c) Calculate one over the square root of the sum of squares of three coordinates of the argument.	✓	✓
float rnorm4df (float a, float b, float c, float d) Calculate one over the square root of the sum of squares of four coordinates of the argument.	✓	✓
float rnormf (int dim, const float *a) Calculate the reciprocal of square root of the sum of squares of any number of coordinates.	✓	✓
float scalblnf (float x, long int n) Scale floating-point input by an integer power of two.	✓	✓
void sincosf (float x, float *sptr, float *cptr) Calculate the sine and cosine of the first input argument.	✓	✓
void sincospif (float x, float *sptr, float *cptr) Calculate the sine and cosine of the first input argument multiplied by PI.	✓	✓

Function	Supported on Host	Supported on Device
float y0f (float x) Calculate the value of the Bessel function of the second kind of order 0 for the input argument.	✓	✓
float y1f (float x) Calculate the value of the Bessel function of the second kind of order 1 for the input argument.	✓	✓
float ynf (int n, float x) Calculate the value of the Bessel function of the second kind of order n for the input argument.	✓	✓

3.4.7.2 Double Precision Mathematical Functions

The following table consists of supported double-precision mathematical functions.

Function	Supported on Host	Supported on Device
double acos (double x) Calculate the arc cosine of the input argument.	✓	✓
double acosh (double x) Calculate the nonnegative arc hyperbolic cosine of the input argument.	✓	✓
double asin (double x) Calculate the arc sine of the input argument.	✓	✓
double asinh (double x) Calculate the arc hyperbolic sine of the input argument.	✓	✓
double atan (double x) Calculate the arc tangent of the input argument.	✓	✓
double atan2 (double y, double x) Calculate the arc tangent of the ratio of first and second input arguments.	✓	✓

Function	Supported on Host	Supported on Device
double atanh (double x) Calculate the arc hyperbolic tangent of the input argument.	✓	✓
double cbrt (double x) Calculate the cube root of the input argument.	✓	✓
double ceil (double x) Calculate ceiling of the input argument.	✓	✓
double copysign (double x, double y) Create value with given magnitude, copying sign of second value.	✓	✓
double cos (double x) Calculate the cosine of the input argument.	✓	✓
double cosh (double x) Calculate the hyperbolic cosine of the input argument.	✓	✓
double erf (double x) Calculate the error function of the input argument.	✓	✓
double erfc (double x) Calculate the complementary error function of the input argument.	✓	✓
double exp (double x) Calculate the base e exponential of the input argument.	✓	✓
double exp10 (double x) Calculate the base 10 exponential of the input argument.	✓	✓
double exp2 (double x) Calculate the base 2 exponential of the input argument.	✓	✓
double expm1 (double x) Calculate the base e exponential of the input argument, minus 1.	✓	✓

Function	Supported on Host	Supported on Device
double fabs (double x) Calculate the absolute value of the input argument.	✓	✓
double fdim (double x, double y) Compute the positive difference between x and y.	✓	✓
double floor (double x) Calculate the largest integer less than or equal to x.	✓	✓
double fma (double x, double y, double z) Compute x × y + z as a single operation.	✓	✓
double fmax (double , double) Determine the maximum numeric value of the arguments.	✓	✓
double fmin (double x, double y) Determine the minimum numeric value of the arguments.	✓	✓
double fmod (double x, double y) Calculate the floating-point remainder of x / y.	✓	✓
double frexp (double x, int* nptr) Extract mantissa and exponent of a floating-point value.	✓	**x**
double hypot (double x, double y) Calculate the square root of the sum of squares of two arguments.	✓	✓
int ilogb (double x) Compute the unbiased integer exponent of the argument.	✓	✓
__RETURN_TYPE1 isfinite (double a) Determine whether an argument is finite.	✓	✓
__RETURN_TYPE1 isinf (double a) Determine whether an argument is infinite.	✓	✓

Function	Supported on Host	Supported on Device
__RETURN_TYPE1 isnan (double a) Determine whether an argument is a NaN.	✓	✓
double ldexp (double x, int exp) Calculate the value of x · 2exp.	✓	✓
double log (double x) Calculate the base e logarithm of the input argument.	✓	✓
double log10 (double x) Calculate the base 10 logarithm of the input argument.	✓	✓
double log1p (double x) Calculate the value of loge(1 + x).	✓	✓
double log2 (double x) Calculate the base 2 logarithm of the input argument.	✓	✓
double logb (double x) Calculate the floating-point representation of the exponent of the input argument.	✓	✓
double modf (double x, double* iptr) Break down the input argument into fractional and integral parts.	✓	**x**
double nan (const char* tagp) Returns "Not a Number" value.	**x**	✓
double nearbyint (double x) Round the input argument to the nearest integer.	✓	✓
double pow (double x, double y) Calculate the value of the first argument to the power of the second argument.	✓	✓
double remainder (double x, double y) Compute double-precision floating-point remainder.	✓	✓

Function	Supported on Host	Supported on Device
double remquo (double x, double y, int* quo) Compute double-precision floating-point remainder and part of quotient.	✓	**x**
double round (double x) Round to nearest integer value in floating-point.	✓	✓
double scalbn (double x, int n) Scale floating-point input by an integer power of two.	✓	✓
__RETURN_TYPE1 signbit (double a) Return the sign bit of the input.	✓	✓
double sin (double x) Calculate the sine of the input argument.	✓	✓
void sincos (double x, double* sptr, double* cptr) Calculate the sine and cosine of the first input argument.	✓	**x**
double sinh (double x) Calculate the hyperbolic sine of the input argument.	✓	✓
double sqrt (double x) Calculate the square root of the input argument.	✓	✓
double tan (double x) Calculate the tangent of the input argument.	✓	✓
double tanh (double x) Calculate the hyperbolic tangent of the input argument.	✓	✓
double tgamma (double x) Calculate the gamma function of the input argument.	✓	✓
double trunc (double x) Truncate input argument to an integral part.	✓	✓

Function	Supported on Host	Supported on Device
double erfcinv (double y) Calculate the inverse complementary function of the input argument.	✓	✓
double erfcx (double x) Calculate the scaled complementary error function of the input argument.	✓	✓
double erfinv (double y) Calculate the inverse error function of the input argument.	✓	✓
double frexp (float x, int *nptr) Extract mantissa and exponent of a floating-point value.	✓	✓
double j0 (double x) Calculate the value of the Bessel function of the first kind of order 0 for the input argument.	✓	✓
double j1 (double x) Calculate the value of the Bessel function of the first kind of order 1 for the input argument.	✓	✓
double jn (int n, double x) Calculate the value of the Bessel function of the first kind of order n for the input argument.	✓	✓
double lgamma (double x) Calculate the natural logarithm of the absolute value of the gamma function of the input argument.	✓	✓
long long int llrint (double x) Round input to a nearest integer value.	✓	✓
long long int llround (double x) Round to nearest integer value.	✓	✓
long int lrint (double x) Round input to a nearest integer value.	✓	✓

Function	Supported on Host	Supported on Device
long int lround (double x) Round to nearest integer value.	✓	✓
double modf (double x, double *iptr) Break down the input argument into fractional and integral parts.	✓	✓
double nextafter (double x, double y) Returns next representable single-precision floating-point value after an argument.	✓	✓
double norm3d (double a, double b, double c) Calculate the square root of the sum of squares of three coordinates of the argument.	✓	✓
float norm4d (double a, double b, double c, double d) Calculate the square root of the sum of squares of four coordinates of the argument.	✓	✓
double normcdf (double y) Calculate the standard normal cumulative distribution function.	✓	✓
double normcdfinv (double y) Calculate the inverse of the standard normal cumulative distribution function.	✓	✓
double rcbrt (double x) Calculate the reciprocal cube root function.	✓	✓
double remquo (double x, double y, int *quo) Compute single-precision floating-point remainder and part of quotient.	✓	✓
double rhypot (double x, double y) Calculate one over the square root of the sum of squares of two arguments.	✓	✓
double rint (double x) Round input to the nearest integer value in floating-point.	✓	✓

Function	Supported on Host	Supported on Device
double rnorm3d (double a, double b, double c) Calculate one over the square root of the sum of squares of three coordinates of the argument.	✓	✓
double rnorm4d (double a, double b, double c, double d) Calculate one over the square root of the sum of squares of four coordinates of the argument.	✓	✓
double rnorm (int dim, const double *a) Calculate the reciprocal of the square root of the sum of squares of any number of coordinates.	✓	✓
double scalbln (double x, long int n) Scale floating-point input by an integer power of two.	✓	✓
void sincos (double x, double *sptr, double *cptr) Calculate the sine and cosine of the first input argument.	✓	✓
void sincospi (double x, double *sptr, double *cptr) Calculate the sine and cosine of the first input argument multiplied by PI.	✓	✓
double y0f (double x) Calculate the value of the Bessel function of the second kind of order 0 for the input argument.	✓	✓
double y1 (double x) Calculate the value of the Bessel function of the second kind of order 1 for the input argument.	✓	✓
double yn (int n, double x) Calculate the value of the Bessel function of the second kind of order n for the input argument.	✓	✓

NOTE: [1] __RETURN_TYPE is dependent on the compiler. It is usually 'int' for C compilers and 'bool' for C++ compilers.

3.4.7.3 Integer Intrinsics

The following table lists supported integer intrinsics. Note, intrinsics are supported on devices only.

Function
unsigned int __brev (unsigned int x) Reverse the bit order of a 32-bit unsigned integer.
unsigned long long int __brevll (unsigned long long int x) Reverse the bit order of a 64-bit unsigned integer.
int __clz (int x) Return the number of consecutive high-order zero bits in a 32-bit integer.
unsigned int __clz(unsigned int x) Return the number of consecutive high-order zero bits in 32-bit unsigned integer.
int __clzll (long long int x) Count the number of consecutive high-order zero bits in a 64-bit integer.
unsigned int __clzll(long long int x) Return the number of consecutive high-order zero bits in 64-bit signed integer.
unsigned int __ffs(unsigned int x) Find the position of least significant bit set to 1 in a 32-bit unsigned integer.1
unsigned int __ffs(int x) Find the position of least significant bit set to 1 in a 32-bit signed integer.
unsigned int __ffsll(unsigned long long int x) Find the position of least significant bit set to 1 in a 64-bit unsigned integer.1
unsigned int __ffsll(long long int x) Find the position of least significant bit set to 1 in a 64 bit signed integer.
unsigned int __popc (unsigned int x) Count the number of bits that are set to 1 in a 32-bit integer.

Function
int __popcll (unsigned long long int x) Count the number of bits that are set to 1 in a 64-bit integer.
int __mul24 (int x, int y) Multiply two 24-bit integers.
unsigned int __umul24 (unsigned int x, unsigned int y) Multiply two 24-bit unsigned integers.

NOTE: The HIP-Clang implementation of __ffs() and __ffsll() contains code to add a constant +1 to produce the *ffs* result format. For the cases where this overhead is not acceptable and the programmer is willing to specialize for the platform, HIP-Clang provides __lastbit_u32_u32(unsigned int input) and __lastbit_u32_u64(unsigned long long int input).

3.4.7.4 Floating-point Intrinsics

The following table provides a list of supported floating-point intrinsics. Note, intrinsics are supported on devices only.

Function
float __cosf (float x) Calculate the fast approximate cosine of the input argument.
float __expf (float x) Calculate the fast approximate base e exponential of the input argument.
float __frsqrt_rn (float x) Compute 1 / √x in round-to-nearest-even mode.
float __fsqrt_rd (float x) Compute √x in round-down mode.
float __fsqrt_rn (float x) Compute √x in round-to-nearest-even mode.
float __fsqrt_ru (float x) Compute √x in round-up mode.
float __fsqrt_rz (float x) Compute √x in round-towards-zero mode.
float __log10f (float x) Calculate the fast approximate base 10 logarithm of the input argument.
float __log2f (float x) Calculate the fast approximate base 2 logarithm of the input argument.
float __logf (float x) Calculate the fast approximate base e logarithm of the input argument.
float __powf (float x, float y) Calculate the fast approximate of xy.
float __sinf (float x) Calculate the fast approximate sine of the input argument.
float __tanf (float x) Calculate the fast approximate tangent of the input argument.

Function
double __dsqrt_rd (double x) Compute √x in round-down mode.
double __dsqrt_rn (double x) Compute √x in round-to-nearest-even mode.
double __dsqrt_ru (double x) Compute √x in round-up mode.
double __dsqrt_rz (double x) Compute √x in round-towards-zero mode.

3.4.7.5 Texture Functions

Texture functions are not supported.

3.4.7.6 Surface Functions

Surface functions are not supported.

3.4.7.7 Timer Functions

HIP provides the following built-in functions for reading a high-resolution timer from the device.

```
clock_t clock()
long long int clock64()
```

Returns the value of a counter that is incremented every clock cycle on devices. The difference in values returned provides the cycles used.

3.4.7.8 Atomic Functions

Atomic functions execute as read-modify-write operations residing in global or shared memory. No other device or thread can observe or modify the memory location during an atomic operation. If multiple instructions from different devices or threads target the same memory location, the instructions are serialized in an undefined order.

HIP supports the following atomic operations:

Function	Supported in HIP	Supported in CUDA
int atomicAdd(int* address, int val)	✓	✓
unsigned int atomicAdd(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicAdd(unsigned long long int* address, unsigned long long int val)	✓	✓
float atomicAdd(float* address, float val)	✓	✓
int atomicSub(int* address, int val)	✓	✓
unsigned int atomicSub(unsigned int* address,unsigned int val)	✓	✓
int atomicExch(int* address, int val)	✓	✓
unsigned int atomicExch(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicExch(unsigned long long int* address, unsigned long long int val)	✓	✓
float atomicExch(float* address, float val)	✓	✓
int atomicMin(int* address, int val)	✓	✓
unsigned int atomicMin(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicMin(unsigned long long int* address, unsigned long long int val)	✓	✓
int atomicMax(int* address, int val)	✓	✓
unsigned int atomicMax(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicMax(unsigned long long int* address, unsigned long long int val)	✓	✓
unsigned int atomicInc(unsigned int* address)	✗	✓
unsigned int atomicDec(unsigned int* address)	✗	✓
int atomicCAS(int* address, int compare, int val)	✓	✓
unsigned int atomicCAS(unsigned int* address,unsigned int compare,unsigned int val)	✓	✓
unsigned long long int atomicCAS(unsigned long long int* address, unsigned long long int compare,unsigned long long int val)	✓	✓
int atomicAnd(int* address, int val)	✓	✓
unsigned int atomicAnd(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicAnd(unsigned long long int* address, unsigned long long int val)	✓	✓
int atomicOr(int* address, int val)	✓	✓
unsigned int atomicOr(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicOr(unsigned long long int* address, unsigned long long int val)	✓	✓
int atomicXor(int* address, int val)	✓	✓
unsigned int atomicXor(unsigned int* address,unsigned int val)	✓	✓
unsigned long long int atomicXor(unsigned long long int* address, unsigned long long int val))	✓	✓

Caveats and Features Under-Development

HIP enables atomic operations on 32-bit integers. Additionally, it supports an atomic float add. AMD hardware, however, implements the float add using a CAS loop, so this function may not perform efficiently.

3.4.7.9 Warp Cross-Lane Functions

Warp cross-lane functions operate across all lanes in a warp. The hardware guarantees that all warp lanes will execute in lockstep, so additional synchronization is unnecessary and the instructions use no shared memory.

Note that Nvidia and AMD devices have different warp sizes, so portable code should use the warpSize built-ins to query the warp size. Hipified code from the CUDA path requires careful review to ensure it doesn't assume a waveSize of 32. "Wave-aware" code that assumes a waveSize of 32 will run on a wave-64 machine, but it will utilize only half of the machine resources.

In addition to the warpSize device function, the host code can obtain the warpSize from the device properties:

```
int w = props.warpSize;
   // implement portable algorithm based on w (rather than assume 32 or 64)
```

3.4.7.10 Warp Vote and Ballot Functions

```
int __all(int predicate)
int __any(int predicate)
uint64_t __ballot(int predicate)
```

Threads in a warp are referred to as lanes and are numbered from 0 to warpSize -- 1. For these functions, each warp lane contributes 1 -- the bit value (the predicate), which is efficiently broadcast to all lanes in the warp. The 32-bit int predicate from each lane reduces to a 1-bit value: 0 (predicate = 0) or 1 (predicate != 0). __any and __all provide a summary view of the predicates that the other warp lanes contribute:

- __any() returns 1 if any warp lane contributes a nonzero predicate, or 0 otherwise
- __all() returns 1 if all other warp lanes contribute nonzero predicates, or 0 otherwise

Applications can test whether the target platform supports the any/all instruction using the hasWarpVote device property or the HIP_ARCH_HAS_WARP_VOTE compiler define.

__ballot provides a bit mask containing the 1-bit predicate value from each lane. The nth bit of the result contains the 1 bit contributed by the nth warp lane. Note that HIP's __ballot function supports a 64-bit return value (compared with 32 bits). Code ported from CUDA should support the larger warp sizes that the HIP version of this instruction supports. Applications can test whether the target platform supports the ballot instruction using the hasWarpBallot device property or the HIP_ARCH_HAS_WARP_BALLOT compiler define.

3.4.7.11 Warp Shuffle Functions

Half-float shuffles are not supported. The default width is warpSize---see Warp Cross-Lane Functions. Applications should not assume the warpSize is 32 or 64.

```
int   __shfl      (int var,   int srcLane, int width=warpSize);
float __shfl      (float var, int srcLane, int width=warpSize);
int   __shfl_up   (int var,   unsigned int delta, int width=warpSize);
float __shfl_up   (float var, unsigned int delta, int width=warpSize);
int   __shfl_down (int var,   unsigned int delta, int width=warpSize);
float __shfl_down (float var, unsigned int delta, int width=warpSize) ;
int   __shfl_xor  (int var,   int laneMask, int width=warpSize)
float __shfl_xor  (float var, int laneMask, int width=warpSize);
```

3.4.7.12 Cooperative Groups Functions

Cooperative groups is a mechanism for forming and communicating between groups of threads at a granularity different than the block. This feature was introduced in CUDA 9. HIP does not support any of the kernel language cooperative groups types or functions.

Function	Supported in HIP	Supported in CUDA
void thread_group.sync()		✓
unsigned thread_group.size()		✓
unsigned thread_group.thread_rank()		✓
bool thread_group.is_valid()		✓
thread_group tiled_partition(thread_group, size)		✓
thread_block_tile<N> tiled_partition<N>(thread_group)		✓
thread_block this_thread_block()		✓
T thread_block_tile.shfl()		✓
T thread_block_tile.shfl_down()		✓
T thread_block_tile.shfl_up()		✓
T thread_block_tile.shfl_xor()		✓
T thread_block_tile.any()		✓
T thread_block_tile.all()		✓
T thread_block_tile.ballot()		✓
T thread_block_tile.match_any()		✓
T thread_block_tile.match_all()		✓
coalesced_group coalesced_threads()		✓
grid_group this_grid()		✓
void grid_group.sync()		✓
unsigned grid_group.size()		✓
unsigned grid_group.thread_rank()		✓
bool grid_group.is_valid()		✓
multi_grid_group this_multi_grid()		✓
void multi_grid_group.sync()		✓
unsigned multi_grid_group.size()		✓
unsigned multi_grid_group.thread_rank()		✓
bool multi_grid_group.is_valid()		✓

3.4.7.13 Warp Matrix Functions

Warp matrix functions allow a warp to cooperatively operate on small matrices whose elements are spread over the lanes in an unspecified manner. This feature was introduced in CUDA 9.

HIP does not support any of the kernel language warp matrix types or functions.

Function	Supported in HIP	Supported in CUDA
void load_matrix_sync(fragment<...> &a, const T* mptr, unsigned lda)		✓
void load_matrix_sync(fragment<...> &a, const T* mptr, unsigned lda, layout_t layout)		✓
void store_matrix_sync(T* mptr, fragment<...> &a, unsigned lda, layout_t layout)		✓
void fill_fragment(fragment<...> &a, const T &value)		✓
void mma_sync(fragment<...> &d, const fragment<...> &a, const fragment<...> &b, const fragment<...> &c , bool sat)		✓

3.4.7.14 Independent Thread Scheduling

The hardware support for independent thread scheduling introduced in certain architectures supporting CUDA allows threads to progress independently of each other and enables intra-warp synchronizations that were previously not allowed.

HIP does not support this type of thread scheduling.

3.4.7.15 Profiler Counter Function

The Cuda __prof_trigger() instruction is not supported.

3.4.7.16 Assert

The assert function is under development. HIP does support an "abort" call which will terminate the process execution from inside the kernel.

3.4.7.17 Printf

The printf function is supported.

3.4.8 Device-Side Dynamic Global Memory Allocation

Device-side dynamic global memory allocation is under development.

3.4.9 __launch_bounds__

GPU multiprocessors have a fixed pool of resources (primarily registers and shared memory) which are shared by the actively running warps. Using more resources can increase IPC of the kernel but reduces the resources available for other warps and limits the number of warps that can be simultaneously running. Thus, GPUs have a complex relationship between resource usage and performance.

__launch_bounds__ allows the application to provide usage hints that influence the resources (primarily registers) used by the generated code. It is a function attribute that must be attached to a __global__ function:

```
__global__ void `-__launch_bounds__`-(MAX_THREADS_PER_BLOCK, MIN_WARPS_PER_EU) MyKernel(...) ...
MyKernel(...)
```

launch_bounds supports two parameters:

- MAX_THREADS_PER_BLOCK - The programmers guarantees that the kernel will be launched with threads less than MAX_THREADS_PER_BLOCK. (On NVCC this maps to the .maxntid PTX directive). If no launch_bounds is specified, MAX_THREADS_PER_BLOCK is the maximum block size supported by the device (typically 1024 or larger). Specifying MAX_THREADS_PER_BLOCK less than the maximum effectively allows the compiler to use more resources than a default unconstrained compilation that supports all possible block sizes at launch time. The threads-per-block is the product of (hipBlockDim_x * hipBlockDim_y * hipBlockDim_z).

- MIN_WARPS_PER_EU - directs the compiler to minimize resource usage so that the requested number of warps can be simultaneously active on a multi-processor. Since active warps compete for the same fixed pool of resources, the compiler must reduce resources required by each warp(primarily registers). MIN_WARPS_PER_EU is optional and defaults to 1 if not specified. Specifying a MIN_WARPS_PER_EU greater than the default 1 effectively constrains the compiler's resource usage.

3.4.9.1 Compiler Impact

The compiler uses these parameters as follows:

- The compiler uses the hints only to manage register usage and does not automatically reduce shared memory or other resources.
- Compilation fails if the compiler cannot generate a kernel that meets the requirements of the specified launch bounds.
- From MAX_THREADS_PER_BLOCK, the compiler derives the maximum number of warps/block that can be used at launch time. Values of MAX_THREADS_PER_BLOCK less than the default allows the compiler to use a larger pool of registers: each warp uses registers, and this hint contains the launch to a warps/block size that is less than maximum.
-

- From MIN_WARPS_PER_EU, the compiler derives a maximum number of registers that can be used by the kernel (to meet the required #simultaneous active blocks). If MIN_WARPS_PER_EU is 1, then the kernel can use all registers supported by the multiprocessor.
- The compiler ensures that the registers used in the kernel is less than both allowed maximums, typically by spilling registers (to shared or global memory), or by using more instructions.
- The compiler may use heuristics to increase register usage or may simply be able to avoid spilling. The MAX_THREADS_PER_BLOCK is particularly useful in this case, since it allows the compiler to use more registers and avoid situations where the compiler constrains the register usage (potentially spilling) to meet the requirements of a large block size that is never used at launch time.

3.4.9.2 CU and EU Definitions

A compute unit (CU) is responsible for executing the waves of a workgroup. It is composed of one or more execution units (EU) that are responsible for executing waves. An EU can have enough resources to maintain the state of more than one executing wave. This allows an EU to hide latency by switching between waves in a similar way to symmetric multithreading on a CPU. To allow the state for multiple waves to fit on an EU, the resources used by a single wave have to be limited. Limiting such resources can allow greater latency hiding but it can result in having to spill some register state to memory. This attribute allows an advanced developer to tune the number of waves that are capable of fitting within the resources of an EU. It can be used to ensure at least a certain number will fit to help hide latency and can also be used to ensure no more than a certain number will fit to limit cache thrashing.

3.4.9.3 Porting from CUDA __launch_bounds

CUDA defines a *__launch_bounds,* which is also designed to control occupancy:

```
__launch_bounds(MAX_THREADS_PER_BLOCK, MIN_BLOCKS_PER_MULTIPROCESSOR)
```

The second parameter *__launch_bounds* parameters must be converted to the format used *__hip_launch_bounds,* which uses warps and execution-units rather than blocks and multi-processors (this conversion is performed automatically by hipify tools).

```
MIN_WARPS_PER_EXECUTION_UNIT = (MIN_BLOCKS_PER_MULTIPROCESSOR * MAX_THREADS_PER_BLOCK) / 32
```

The key differences in the interface are:

- Warps (rather than blocks): The developer is trying to tell the compiler to control resource utilization to guarantee some amount of active Warps/EU for latency hiding. Specifying active warps in terms of blocks appears to hide the micro-architectural details of the warp size, however, makes the interface more confusing since the developer ultimately needs to compute the number of warps to obtain the desired level of control.

- Execution Units (rather than multiProcessor): The use of execution units rather than multiprocessors provides support for architectures with multiple execution units/multi-processor. For example, the AMD GCN architecture has 4 execution units per multiProcessor. The hipDeviceProps has a field executionUnitsPerMultiprocessor. Platform-specific coding techniques such as #ifdef can be used to specify different launch_bounds for NVCC and HIP-Clang platforms if desired.

3.4.9.4 Maxregcount

Unlike nvcc, HIP-Clang does not support the "--maxregcount" option. Instead, users are encouraged to use the hip_launch_bounds directive since the parameters are more intuitive and portable than micro-architecture details like registers, and also the directive allows per-kernel control rather than an entire file. hip_launch_bounds works on both HIP-Clang and nvcc targets.

3.4.10 Register Keyword

The register keyword is deprecated in C++ and is silently ignored by both nvcc and HIP-Clang. You can pass the option `-Wdeprecated-register` to the compiler warning message.

3.4.11 Pragma Unroll

Unroll with a bound that is known at compile-time is supported. For example:

```
#pragma unroll 16 /* hint to compiler to unroll next loop by 16 */
for (int i=0; i<16; i++) ...
#pragma unroll 1  /* tell compiler to never unroll the loop */
for (int i=0; i<16; i++) ...
#pragma unroll /* hint to compiler to completely unroll next loop. */
for (int i=0; i<16; i++) ...
```

3.4.12 In-Line Assembly

GCN ISA In-line assembly is supported. For example:

```
asm volatile ("v_mac_f32_e32 %0, %2, %3" : "=v" (out[i]) : "0"(out[i]), "v" (a), "v" (in[i]));
```

The HIP compiler inserts the GCN into the kernel using asm() Assembler statement. volatile keyword is used so that the optimizers must not change the number of volatile operations or change their order of execution relative to other volatile operations. v_mac_f32_e32 is the GCN instruction. For more information, refer to the AMD GCN3 ISA architecture manual Index for the respective operand in the ordered fashion is provided by % followed by a position in the list of operands "v" is the constraint code (for target-specific AMDGPU) for 32-bit VGPR register. For more information, refer to the Supported Constraint Code List for AMDGPU. Output Constraints are specified by an "=" prefix as shown above ("=v"). This indicates that assembly will write to this operand, and the operand will then be made available as a return value of the asm expression. Input constraints do not have a prefix - just the constraint code. The constraint string of "0" says to use the assigned register for output as an input as well (it being the 0'th constraint).

3.4.13 C++ Support

The following C++ features are not supported:

- Run-time-type information (RTTI)
- Virtual functions
- Try/catch

3.4.14 Kernel Compilation

hipcc now supports compiling C++/HIP kernels to binary code objects.

The file format for binary is `.co` which means Code Object. The following command builds the code object using `hipcc`.

```
`hipcc --genco --offload-arch=[TARGET GPU] [INPUT FILE] -o [OUTPUT FILE]`
[TARGET GPU] = GPU architecture
[INPUT FILE] = Name of the file containing kernels
[OUTPUT FILE] = Name of the generated code object file
```

NOTE: When using binary code objects is that the number of arguments to the kernel is different on HIP-Clang and NVCC path. Refer to the sample in samples/0_Intro/module_api for differences in the arguments to be passed to the kernel.

3.4.15 gfx-arch-specific-kernel

Clang defined '__gfx*__' macros can be used to execute gfx arch specific codes inside the kernel. Refer to the sample 14_gpu_arch in samples/2_Cookbook.

3.5 HIP Logging

HIP provides a logging mechanism, which is a convenient way of printing important information to trace HIP API and runtime codes during the execution of a HIP application. It assists the HIP development team in the development of HIP runtime and is useful for HIP application developers as well. Depending on the setting of logging level and logging mask, HIP logging will print different kinds of information, for different types of functionalities such as HIP APIs, executed kernels, queue commands, and queue contents, etc.

3.5.1 HIP Logging Level

By default, HIP logging is disabled, it can be enabled via environment setting,

AMD_LOG_LEVEL

The value of the setting controls different logging level.

```
enum LogLevel {
LOG_NONE = 0,
LOG_ERROR = 1,
LOG_WARNING = 2,
LOG_INFO = 3,
LOG_DEBUG = 4
};
```

3.5.2 HIP Logging Mask

Logging mask is designed to print types of functionalities during the execution of HIP application. It can be set as one of the following values:

```
enum LogMask {
  LOG_API       = 0x00000001, //!< API call
  LOG_CMD       = 0x00000002, //!< Kernel and Copy Commands and Barriers
  LOG_WAIT      = 0x00000004, //!< Synchronization and waiting for commands to finish
  LOG_AQL       = 0x00000008, //!< Decode and display AQL packets
  LOG_QUEUE     = 0x00000010, //!< Queue commands and queue contents
  LOG_SIG       = 0x00000020, //!< Signal creation, allocation, pool
  LOG_LOCK      = 0x00000040, //!< Locks and thread-safety code.
  LOG_KERN      = 0x00000080, //!< kernel creations and arguments, etc.
  LOG_COPY      = 0x00000100, //!< Copy debug
  LOG_COPY2     = 0x00000200, //!< Detailed copy debug
  LOG_RESOURCE  = 0x00000400, //!< Resource allocation, performance-impacting events.
  LOG_INIT      = 0x00000800, //!< Initialization and shutdown
  LOG_MISC      = 0x00001000, //!< misc debug, not yet classified
  LOG_AQL2      = 0x00002000, //!< Show raw bytes of AQL packet
  LOG_CODE      = 0x00004000, //!< Show code creation debug
  LOG_CMD2      = 0x00008000, //!< More detailed command info, including barrier commands
  LOG_LOCATION  = 0x00010000, //!< Log message location
  LOG_ALWAYS    = 0xFFFFFFFF, //!< Log always even mask flag is zero
};
```

Once AMD_LOG_LEVEL is set, the logging mask is set as default with the value 0x7FFFFFFF. However, for different purpose of logging functionalities, logging mask can be defined as well via an environment variable,

AMD_LOG_MASK

3.5.3 HIP Logging Command

To pring HIP logging information, the function is defined as

```
#define ClPrint(level, mask, format, ...)
  do {
    if (AMD_LOG_LEVEL >= level) {
      if (AMD_LOG_MASK & mask || mask == amd::LOG_ALWAYS) {
        if (AMD_LOG_MASK & amd::LOG_LOCATION) {
          amd::log_printf(level, __FILENAME__, __LINE__, format, ##__VA_ARGS__);
        } else {
          amd::log_printf(level, "", 0, format, ##__VA_ARGS__);
        }
      }
    }
  } while (false)
```

In the HIP code, call ClPrint() function with proper input varibles as needed, for example,

```
ClPrint(amd::LOG_INFO, amd::LOG_INIT, "Initializing HSA stack.");
```

3.5.4 HIP Logging Example

Below is an example to enable HIP logging and get logging information during execution of hipinfo,

```
user@user-test:~/hip/bin$ export AMD_LOG_LEVEL=4
user@user-test:~/hip/bin$ ./hipinfo
:3:rocdevice.cpp            :453 : 23647210092: Initializing HSA stack.
:3:comgrctx.cpp             :33  : 23647639336: Loading COMGR library.
:3:rocdevice.cpp            :203 : 23647687108: Numa select cpu
agent[0]=0x13407c0(fine=0x13409a0,coarse=0x1340ad0) for gpu agent=0x1346150
:4:runtime.cpp              :82  : 23647698669: init
:3:hip_device_runtime.cpp   :473 : 23647698869: 5617 : [7fad295dd840] hipGetDeviceCount:
Returned hipSuccess
:3:hip_device_runtime.cpp   :502 : 23647698990: 5617 : [7fad295dd840] hipSetDevice ( 0 )
:3:hip_device_runtime.cpp   :507 : 23647699042: 5617 : [7fad295dd840] hipSetDevice: Returned
hipSuccess
--------------------------------------------------------------------------------
device#                           0
:3:hip_device.cpp           :150 : 23647699276: 5617 : [7fad295dd840] hipGetDeviceProperties (
0x7ffdbe7db730, 0 )
:3:hip_device.cpp           :237 : 23647699335: 5617 : [7fad295dd840] hipGetDeviceProperties:
Returned hipSuccess
Name:                             Device 7341
pciBusID:                         3
pciDeviceID:                      0
pciDomainID:                      0
multiProcessorCount:              11
maxThreadsPerMultiProcessor:      2560
isMultiGpuBoard:                  0
clockRate:                        1900 Mhz
memoryClockRate:                  875 Mhz
```

```
memoryBusWidth:                   0
clockInstructionRate:             1000 Mhz
totalGlobalMem:                   7.98 GB
maxSharedMemoryPerMultiProcessor: 64.00 KB
totalConstMem:                    8573157376
sharedMemPerBlock:                64.00 KB
canMapHostMemory:                 1
regsPerBlock:                     0
warpSize:                         32
l2CacheSize:                      0
computeMode:                      0
maxThreadsPerBlock:               1024
maxThreadsDim.x:                  1024
maxThreadsDim.y:                  1024
maxThreadsDim.z:                  1024
maxGridSize.x:                    2147483647
maxGridSize.y:                    2147483647
maxGridSize.z:                    2147483647
major:                            10
minor:                            12
concurrentKernels:                1
cooperativeLaunch:                0
cooperativeMultiDeviceLaunch:     0
arch.hasGlobalInt32Atomics:       1
arch.hasGlobalFloatAtomicExch:    1
arch.hasSharedInt32Atomics:       1
arch.hasSharedFloatAtomicExch:    1
arch.hasFloatAtomicAdd:           1
arch.hasGlobalInt64Atomics:       1
arch.hasSharedInt64Atomics:       1
arch.hasDoubles:                  1
arch.hasWarpVote:                 1
arch.hasWarpBallot:               1
arch.hasWarpShuffle:              1
arch.hasFunnelShift:              0
arch.hasThreadFenceSystem:        1
arch.hasSyncThreadsExt:           0
arch.hasSurfaceFuncs:             0
arch.has3dGrid:                   1
arch.hasDynamicParallelism:       0
gcnArch:                          1012
isIntegrated:                     0
maxTexture1D:                     65536
maxTexture2D.width:               16384
maxTexture2D.height:              16384
maxTexture3D.width:               2048
maxTexture3D.height:              2048
maxTexture3D.depth:               2048
isLargeBar:                       0
:3:hip_device_runtime.cpp   :471 : 23647701557: 5617 : [7fad295dd840] hipGetDeviceCount (
0x7ffdbe7db714 )
:3:hip_device_runtime.cpp   :473 : 23647701608: 5617 : [7fad295dd840] hipGetDeviceCount:
Returned hipSuccess
```

```
:3:hip_peer.cpp             :76  : 23647701731: 5617 : [7fad295dd840] hipDeviceCanAccessPeer (
0x7ffdbe7db728, 0, 0 )
:3:hip_peer.cpp             :60  : 23647701784: 5617 : [7fad295dd840] canAccessPeer: Returned
hipSuccess
:3:hip_peer.cpp             :77  : 23647701831: 5617 : [7fad295dd840] hipDeviceCanAccessPeer:
Returned hipSuccess
peers:
:3:hip_peer.cpp             :76  : 23647701921: 5617 : [7fad295dd840] hipDeviceCanAccessPeer (
0x7ffdbe7db728, 0, 0 )
:3:hip_peer.cpp             :60  : 23647701965: 5617 : [7fad295dd840] canAccessPeer: Returned
hipSuccess
:3:hip_peer.cpp             :77  : 23647701998: 5617 : [7fad295dd840] hipDeviceCanAccessPeer:
Returned hipSuccess
non-peers:                       device#0

:3:hip_memory.cpp           :345 : 23647702191: 5617 : [7fad295dd840] hipMemGetInfo (
0x7ffdbe7db718, 0x7ffdbe7db720 )
:3:hip_memory.cpp           :360 : 23647702243: 5617 : [7fad295dd840] hipMemGetInfo: Returned
hipSuccess
memInfo.total:                   7.98 GB
memInfo.free:                    7.98 GB (100%)
```

3.5.5 HIP Logging Tips

- HIP logging works for both release and debug version of HIP application.
- Logging function with different logging level can be called in the code as needed.
- Information with a logging level less than AMD_LOG_LEVEL will be printed.
- If need to save the HIP logging output information in a file, just define the file at the command when running the application at the terminal, for example,

```
user@user-test:~/hip/bin$ ./hipinfo > ~/hip_log.txt
```

Chapter 4 Transiting from CUDA to HIP

4.1 Transition Tool: HIPIFY

4.1.1 Sample and Practice

Here is a simple test, which shows how to use hipify-Perl to port CUDA code to HIP. See a related *blog* that explains the example. Now, it is even simpler and requires no manual modification to the hipified source code - just hipify and compile:

1. Add hip/bin path to the PATH.

```
$ export PATH=$PATH:[MYHIP]/bin
```

2. Define the environment variable.

```
$ export HIP_PATH=[MYHIP]
```

3. Build an executable file.

```
$ cd ~/hip/samples/0_Intro/square
$ make
/home/user/hip/bin/hipify-perl square.cu > square.cpp
/home/user/hip/bin/hipcc  square.cpp -o square.out
/home/user/hip/bin/hipcc -use-staticlib  square.cpp -o square.out.static
```

4. Execute the file.

```
$ ./square.out
info: running on device Vega20 [Radeon Pro W5500]
info: allocate host mem (  7.63 MB)
info: allocate device mem (  7.63 MB)
info: copy Host2Device
info: launch 'vector_square' kernel
info: copy Device2Host
info: check result
PASSED!
```

4.2 HIP Porting Process

4.2.1 Porting a New CUDA Project

4.2.1.1 General Tips

- Starting the port on a CUDA machine is often the easiest approach since you can incrementally port pieces of the code to HIP while leaving the rest in CUDA. (Recall that on CUDA machines HIP is just a thin layer over CUDA, so the two code types can interoperate on nvcc platforms.) Also, the HIP port can be compared with the original CUDA code for function and performance.

- Once the CUDA code is ported to HIP and is running on the CUDA machine, compile the HIP code using the HIP compiler on an AMD machine.

- HIP ports can replace CUDA versions: HIP can deliver the same performance as a native CUDA implementation, with the benefit of portability to both Nvidia and AMD architectures as well as a path to future C++ standard support. You can handle platform-specific features through the conditional compilation or by adding them to the open-source HIP infrastructure.

- Use *bin/hipconvertinplace-perl.sh* to hipify all code files in the CUDA source directory.

4.2.1.2 Scanning existing CUDA code to scope the porting effort

The *hipexamine-perl.sh* tool will scan a source directory to determine which files contain CUDA code and how much of that code can be automatically hipified.

```
> cd examples/rodinia_3.0/cuda/kmeans
> $HIP_DIR/bin/hipexamine-perl.sh.
info: hipify ./kmeans.h =====>
info: hipify ./unistd.h =====>
info: hipify ./kmeans.c =====>
info: hipify ./kmeans_cuda_kernel.cu =====>
  info: converted 40 CUDA->HIP refs( dev:0 mem:0 kern:0 builtin:37 math:0 stream:0 event:0 err:0
def:0 tex:3 other:0 ) warn:0 LOC:185
info: hipify ./getopt.h =====>
info: hipify ./kmeans_cuda.cu =====>
  info: converted 49 CUDA->HIP refs( dev:3 mem:32 kern:2 builtin:0 math:0 stream:0 event:0 err:0
def:0 tex:12 other:0 ) warn:0 LOC:311
info: hipify ./rmse.c =====>
info: hipify ./cluster.c =====>
info: hipify ./getopt.c =====>
info: hipify ./kmeans_clustering.c =====>
info: TOTAL-converted 89 CUDA->HIP refs( dev:3 mem:32 kern:2 builtin:37 math:0 stream:0 event:0
err:0 def:0 tex:15 other:0 ) warn:0 LOC:3607
  kernels (1 total) :   kmeansPoint(1)
```

hipexamine-perl scans each code file (cpp, c, h, hpp, etc.) found in the specified directory:

- Files with no CUDA code (kmeans.h) print a one-line summary just listing the source file name.

- Files with CUDA code print a summary of what was found - for example, the kmeans_cuda_kernel.cu file:

```
info: hipify ./kmeans_cuda_kernel.cu =====>
info: converted 40 CUDA->HIP refs( dev:0 mem:0 kern:0 builtin:37 math:0 stream:0 event:0
```

- Information in kmeans_cuda_kernel.cu :
  - How many CUDA calls were converted to HIP (40)
  - Breakdown of the CUDA functionality used (dev:0 mem:0 etc). This file uses many CUDA builtins (37) and texture functions (3).
  - Warning for code that looks like CUDA API but was not converted (0 in this file).
  - Count Lines-of-Code (LOC) - 185 for this file.

- hipexamine-perl also presents a summary at the end of the process for the statistics collected across all files. This has a similar format to the per-file reporting, and also includes a list of all kernels which have been called. An example from above:

```
info: TOTAL-converted 89 CUDA->HIP refs( dev:3 mem:32 kern:2 builtin:37 math:0 stream:0 event:0
err:0 def:0 tex:15 other:0 ) warn:0 LOC:3607
kernels (1 total) :   kmeansPoint(1)
```

4.2.1.3 Converting a project in-place

```
> hipify-perl --inplace
```

For each input file FILE, this script will: - If FILE.prehip file does not exist, copy the original code to a new file with extension.prehip. Then hipify the code file. If “FILE.prehip” file exists, hipify FILE.prehip and save to FILE.

This is useful for testing improvements to the hipify toolset.

The hipconvertinplace-perl.sh script will perform an in-place conversion for all code files in the specified directory. This can be quite handy when dealing with an existing CUDA code base since the script preserves the existing directory structure and filenames - and includes work. After converting in-place, you can review the code to add additional parameters to directory names.

```
> hipconvertinplace-perl.sh MY_SRC_DIR
```

4.2.1.4 Library Equivalents

CUDA Library	ROCm Library	Comment
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transfer Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSOLVER	Lapack library
AMG-X	rocALUTION	Sparse iterative solvers and preconditioners with Geometric and Algebraic MultiGrid
Thrust	rocThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
EIGEN	EIGEN	C++ template library for linear algebra: matrices, vectors, numerical solvers,
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents

4.2.2 Distinguishing Compiler Modes

4.2.2.1 Identifying HIP Target Platform

All HIP projects target either AMD or NVIDIA platform. The platform affects which headers are included and which libraries are used for linking.

- HIP_PLATFORM_HCC is defined if the HIP platform targets AMD
- HIP_PLATFORM_NVCC is defined if the HIP platform targets NVIDIA

4.2.2.2 Identifying the Compiler: HIP-Clang or NVCC

Often, it is useful to know whether the underlying compiler is HIP-Clang or nvcc. This knowledge can guard platform-specific code or aid in platform-specific performance tuning.

```
#ifdef __HIP_PLATFORM_HCC__
// Compiled with HIP-Clang
#endif
#ifdef __NVCC__
// Compiled with nvcc
//  Could be compiling with CUDA language extensions enabled (for example, a ".cu file)
//  Could be in pass-through mode to an underlying host compile OR (for example, a .cpp file)
#ifdef __CUDACC__
// Compiled with nvcc (CUDA language extensions enabled)
```

HIP-Clang directly generates the host code (using the Clang x86 target) without passing the code to another host compiler. Thus, they have no equivalent of the __CUDACC__ define.

4.2.2.3 Identifying Current Compilation Pass: Host or Device

NVCC makes two passes over the code: one for host code and one for device code. HIP-Clang will have multiple passes over the code: one for the host code, and one for each architecture on the device code. __HIP_DEVICE_COMPILE__ is set to a nonzero value when the compiler (HIP-Clang or nvcc) is compiling code for a device inside a __global__ kernel or for a device function. __HIP_DEVICE_COMPILE__ can replace #ifdef checks on the __CUDA_ARCH__ define.

```
// #ifdef __CUDA_ARCH__
#if __HIP_DEVICE_COMPILE__
```

Unlike __CUDA_ARCH__, the __HIP_DEVICE_COMPILE__ value is 1 or undefined, and it does not represent the feature capability of the target device.

4.2.3 Compiler Defines: Summary

Define	HIP-Clang	nvcc	Other (GCC, ICC, Clang, etc.)
HIP-related defines:			
__HIP_PLATFORM_HCC__	Defined	Undefined	Defined if targeting AMD platform; undefined otherwise
__HIP_PLATFORM_NVCC__	Undefined	Defined	Defined if targeting nvcc platform; undefined otherwise
__HIP_DEVICE_COMPILE__	1 if compiling for device; undefined if compiling for host	1 if compiling for device; undefined if compiling for host	Undefined
__HIPCC__	Defined	Defined	Undefined
__HIP_ARCH_*	0 or 1 depending on feature support (see below)	0 or 1 depending on feature support (see below)	0
nvcc-related defines:			
__CUDACC__	Defined if source code is compiled by nvcc; undefined otherwise	Undefined	
__NVCC__	Undefined	Defined	Undefined
__CUDA_ARCH__	Undefined	Unsigned representing compute capability (e.g., "130") if in device code; 0 if in host code	Undefined
hip-clang-related defines:			
__HIP__	Defined	Undefined	Undefined
HIP-Clang common defines:			
__clang__	Defined	Defined	Undefined

4.3 Identifying Architecture Features

4.3.1 HIP_ARCH Defines

Some CUDA code tests __CUDA_ARCH__ for a specific value to determine whether the machine supports a certain architectural feature. For instance,

```
#if (__CUDA_ARCH__ >= 130)
// doubles are supported
```

This type of code requires special attention since AMD and CUDA devices have different architectural capabilities. Moreover, you can't determine the presence of a feature using a simple comparison against an architecture's version number. HIP provides a set of defines and device properties to query whether a specific architectural feature is supported.

The __HIP_ARCH_* defines can replace comparisons of __CUDA_ARCH__ values:

```
//#if (__CUDA_ARCH__ >= 130)   // non-portable
if __HIP_ARCH_HAS_DOUBLES__ {  // portable HIP feature query
   // doubles are supported
}
```

For host code, the __HIP_ARCH__* defines are set to 0. You should only use the HIP_ARCH fields in the device code.

4.3.2 Device-Architecture Properties

The host code should query the architecture feature flags in the device properties that hipGetDeviceProperties returns, rather than testing the "major" and "minor" fields directly:

```
hipGetDeviceProperties(&deviceProp, device);
//if ((deviceProp.major == 1 && deviceProp.minor < 2))  // non-portable
if (deviceProp.arch.hasSharedInt32Atomics) {            // portable HIP feature query
    // has shared int32 atomic operations ...
}
```

4.3.3 Table of Architecture Properties

The table below shows the full set of architectural properties that HIP supports.

Define (use only in device code)	Device Property (run-time query)	Comment
32-bit atomics:		
__HIP_ARCH_HAS_GLOBAL_INT32_ATOMICS__	hasGlobalInt32Atomics	32-bit integer atomics for global memory
__HIP_ARCH_HAS_GLOBAL_FLOAT_ATOMIC_EXCH__	hasGlobalFloatAtomicExch	32-bit float atomic exchange for global memory
__HIP_ARCH_HAS_SHARED_INT32_ATOMICS__	hasSharedInt32Atomics	32-bit integer atomics for shared memory
__HIP_ARCH_HAS_SHARED_FLOAT_ATOMIC_EXCH__	hasSharedFloatAtomicExch	32-bit float atomic exchange for shared memory
__HIP_ARCH_HAS_FLOAT_ATOMIC_ADD__	hasFloatAtomicAdd	32-bit float atomic add in global and shared memory
64-bit atomics		
__HIP_ARCH_HAS_GLOBAL_INT64_ATOMICS__	hasGlobalInt64Atomics	64-bit integer atomics for global memory
__HIP_ARCH_HAS_SHARED_INT64_ATOMICS__	hasSharedInt64Atomics	64-bit integer atomics for shared memory
Doubles		
__HIP_ARCH_HAS_DOUBLES__	hasDoubles	Double-precision floating point
Warp cross-lane operations:		
__HIP_ARCH_HAS_WARP_VOTE__	hasWarpVote	Warp vote instructions (any, all)
__HIP_ARCH_HAS_WARP_BALLOT__	hasWarpBallot	Warp ballot instructions
__HIP_ARCH_HAS_WARP_SHUFFLE__	hasWarpShuffle	Warp shuffle operations (shfl_*)
__HIP_ARCH_HAS_WARP_FUNNEL_SHIFT__	hasFunnelShift	Funnel shift two input words into one
Sync		
__HIP_ARCH_HAS_THREAD_FENCE_SYSTEM__	hasThreadFenceSystem	threadfence_system
__HIP_ARCH_HAS_SYNC_THREAD_EXT__	hasSyncThreadsExt	syncthreads_count, syncthreads_and, syncthreads_or
Miscellaneous		
__HIP_ARCH_HAS_SURFACE_FUNCS__	hasSurfaceFuncs	

Define (use only in device code)	Device Property (run-time query)	Comment
__HIP_ARCH_HAS_3DGRID__	has3dGrid	Grids and groups are 3D
__HIP_ARCH_HAS_DYNAMIC_PARALLEL__	hasDynamicParallelism	

4.3.4 Finding HIP

Makefiles can use the following syntax to conditionally provide a default HIP_PATH if one does not exist:

```
HIP_PATH ?= $(shell hipconfig --path)
```

4.3.5 Identifying HIP Runtime

HIP can depend on ROCclr, or NVCC as runtime.

AMD platform HIP_ROCclr is defined on AMD platform that HIP use Radeon Open Compute Common Language Runtime, called ROCclr.

ROCclr is a virtual device interface that HIP runtimes interact with different backends, which allows runtimes to work on Linux and Windows without much effort.

On the Nvidia platform, HIP is just a thin layer on top of CUDA. On a non-AMD platform, HIP runtime determines if nvcc is available and can be used. If available, HIP_PLATFORM is set to nvcc and underneath CUDA path is used.

4.3.6 hipLaunchKernel

hipLaunchKernel is a variadic macro that accepts as parameters the launch configurations (grid dims, group dims, stream, dynamic shared size) followed by a variable number of kernel arguments. This sequence is then expanded into the appropriate kernel launch syntax depending on the platform. While this can be a convenient single-line kernel launch syntax, the macro implementation can cause issues when nested inside other macros. For example, consider the following:

```
// Will cause compile error:
#define MY_LAUNCH(command, doTrace) \
{\
    if (doTrace) printf ("TRACE: %s\n", #command); \
    (command);   /* The nested ( ) will cause compile error */\
}

MY_LAUNCH (hipLaunchKernel(vAdd, dim3(1024), dim3(1), 0, 0, Ad), true, "firstCall");
```

NOTE: Avoid nesting macro parameters inside parenthesis - here's an alternative that will work:

```
#define MY_LAUNCH(command, doTrace) \
{\
    if (doTrace) printf ("TRACE: %s\n", #command); \
    command;\
}

MY_LAUNCH (hipLaunchKernel(vAdd, dim3(1024), dim3(1), 0, 0, Ad), true, "firstCall");
```

4.3.7 Compiler Options

HIPcc is a portable compiler driver that calls nvcc or HIP-Clang (depending on the target system) and attach all required include and library options. It passes options through to the target compiler. Tools that call hipcc must ensure the compiler options are appropriate for the target compiler. The hipconfig script may help in identifying the target platform, compiler, and runtime. It can also help set options appropriately.

4.3.7.1 Compiler Options Supported on AMD Platforms

Option	Description
--amdgpu-target=<gpu_arch>	[DEPRECATED] This option is replaced by `--offload-arch=<target>`. Generate code for the given GPU target. Supported targets are gfx701, gfx801, gfx802, gfx803, gfx900, gfx906, gfx908, gfx1010, gfx1011, gfx1012, gfx1030, gfx1031. This option could appear multiple times on the same command line to generate a fat binary for multiple targets.
--fgpu-rdc	Generate relocatable device code, which allows kernels or device functions calling device functions in different translation units.
-ggdb	Equivalent to `-g` plus tuning for GDB. This is recommended when using ROCm's GDB to debug GPU code.
--gpu-max-threads-per-block=<num>	Generate code to support up to the specified number of threads per block.
-O<n>	Specify the optimization level.
-offload-arch=<target>	Specify the AMD GPU [target ID] *https://clang.llvm.org/docs/ClangOffloadBundlerFileFormat.html#target-id*
-save-temps	Save the compiler-generated intermediate files.
-v	Show the compilation steps.

4.3.7.2 Option for specifying GPU processor

To specify target ID, use

--offload-arch=X

NOTE: For backward compatibility, hipcc also accepts *--amdgpu-target=X* for specifying target ID. However, it will be deprecated in future releases.

4.3.8 Linking Issues

4.3.8.1 Linking with hipcc

hipcc adds the necessary libraries for HIP as well as for the accelerator compiler (nvcc or AMD compiler). It is recommended to link with hipcc since it automatically links the binary to the necessary HIP runtime libraries. It also enables linking and managing GPU objects.

-lm Option

NOTE: hipcc adds *-lm* by default to the link command.

4.4 Linking Code with Other Compilers

CUDA code often uses nvcc for accelerator code (defining and launching kernels, typically defined in .cu or .cuh files). It also uses a standard compiler (g++) for the rest of the application. nvcc is a preprocessor that employs a standard host compiler (gcc) to generate the host code. The code compiled using this tool can employ only the intersection of language features supported by both nvcc and the host compiler. In some cases, you must take care to ensure the data types and alignment of the host compiler are identical to those of the device compiler. Only some host compilers are supported---for example, recent nvcc versions lack Clang host-compiler capability.

HIP-Clang generates both device and host code using the same Clang-based compiler. The code uses the same API as gcc, which allows code generated by different gcc-compatible compilers to be linked together. For example, code compiled using HIP-Clang can link with code compiled using "standard" compilers (such as gcc, ICC, and Clang). Take care to ensure all compilers use the same standard C++ header and library formats.

4.4.1 libc++ and libstdc++

hipcc links to libstdc++ by default. This provides better compatibility between g++ and HIP.

If you pass "--stdlib=libc++" to hipcc, hipcc will use the libc++ library. Generally, libc++ provides a broader set of C++ features while libstdc++ is the standard for more compilers (notably including g++).

When cross-linking C++ code, any C++ functions that use types from the C++ standard library (including std::string, std::vector and other containers) must use the same standard-library implementation. They include the following:

- Functions or kernels defined in HIP-Clang that are called from a standard compiler
- Functions defined in a standard compiler are called from HIP-Clang.
- Applications with these interfaces should use the default libstdc++ linking.

Applications that are compiled entirely with hipcc, and which benefit from advanced C++ features not supported in libstdc++, and which do not require portability to nvcc, may choose to use libc++.

4.4.2 HIP Headers (hip_runtime.h, hip_runtime_api.h)

The hip_runtime.h and hip_runtime_api.h files define the types, functions and enumerations needed to compile a HIP program:

- hip_runtime_api.h: defines all the HIP runtime APIs (e.g., hipMalloc) and the types required to call them. A source file that is only calling HIP APIs but neither defines nor launches any kernels can include hip_runtime_api.h. hip_runtime_api.h uses no custom hc language features and can be compiled using a standard C++ compiler.

- hip_runtime.h: included in hip_runtime_api.h. It additionally provides the types and defines required to create and launch kernels. hip_runtime.h does use custom hc language features, but they are guarded by ifdef checks. It can be compiled using a standard C++ compiler but will expose a subset of the available functions.

CUDA has slightly different content for these two files. In some cases, you may need to convert hipified code to include the richer hip_runtime.h instead of hip_runtime_api.h.

4.4.3 Using a Standard C++ Compiler

You can compile hip_runtime_api.h using a standard C or C++ compiler (e.g., gcc or ICC). The HIP include paths and defines (__HIP_PLATFORM_HCC__ or __HIP_PLATFORM_NVCC__) must pass to the standard compiler; hipconfig then returns the necessary options:

```
> hipconfig --cxx_config
 -D__HIP_PLATFORM_HCC__ -I/home/user1/hip/include
```

You can capture the hipconfig output and passed it to the standard compiler; below is a sample makefile syntax:

```
CPPFLAGS += $(shell $(HIP_PATH)/bin/hipconfig --cpp_config)
```

Nvcc includes some headers by default. However, HIP does not include default headers, and instead, all required files must be explicitly included. Specifically, files that call HIP run-time APIs or define HIP kernels must explicitly include the appropriate HIP headers. If the compilation process reports that it cannot find necessary APIs (for example, "error: identifier ‘hipSetDevice’ is undefined"), ensure that the file includes hip_runtime.h (or hip_runtime_api.h, if appropriate). The hipify-perl script automatically converts "cuda_runtime.h" to "hip_runtime.h," and it converts "cuda_runtime_api.h" to "hip_runtime_api.h", but it may miss nested headers or macros.

4.4.3.1 cuda.h

The HIP-Clang path provides an empty cuda.h file. Some existing CUDA programs include this file but do not require any of the functions.

4.4.4 Choosing HIP File Extensions

Many existing CUDA projects use the ".cu" and ".cuh" file extensions to indicate code that should be run through the nvcc compiler. For quick HIP ports, leaving these file extensions unchanged is often easier, as it minimizes the work required to change file names in the directory and #include statements in the files.

For new projects or ports which can be re-factored, we recommend the use of the extension ".hip.cpp" for source files, and ".hip.h" or ".hip.hpp" for header files. This indicates that the code is standard C++ code, but also provides a unique indication for *make* tools to run hipcc when appropriate.

4.5 Workarounds

4.5.1 memcpyToSymbol

HIP support for hipMemcpyToSymbol is complete. This feature allows a kernel to define a device-side data symbol that can be accessed on the host side. The symbol can be in __constant or device space.

Note that the symbol name needs to be encased in the HIP_SYMBOL macro, as shown in the code example below. This also applies to hipMemcpyFromSymbol, hipGetSymbolAddress, and hipGetSymbolSize.

For example:

Device Code:

```
#include<hip/hip_runtime.h>
#include<hip/hip_runtime_api.h>
#include<iostream>
#define HIP_ASSERT(status) \
    assert(status == hipSuccess)
#define LEN 512
#define SIZE 2048
__constant__ int Value[LEN];
__global__ void Get(hipLaunchParm lp, int *Ad)
{
    int tid = hipThreadIdx_x + hipBlockIdx_x * hipBlockDim_x;
    Ad[tid] = Value[tid];
}

int main()
{
    int *A, *B, *Ad;
    A = new int[LEN];
    B = new int[LEN];
    for(unsigned i=0;i<LEN;i++)
    {
        A[i] = -1*i;
        B[i] = 0;
    }
    HIP_ASSERT(hipMalloc((void**)&Ad, SIZE));
    HIP_ASSERT(hipMemcpyToSymbol(HIP_SYMBOL(Value), A, SIZE, 0, hipMemcpyHostToDevice));
    hipLaunchKernel(Get, dim3(1,1,1), dim3(LEN,1,1), 0, 0, Ad);
    HIP_ASSERT(hipMemcpy(B, Ad, SIZE, hipMemcpyDeviceToHost));
    for(unsigned i=0;i<LEN;i++)
    {
        assert(A[i] == B[i]);
    }
    std::cout<<"Passed"<<std::endl;
}
```

4.5.2 CU_POINTER_ATTRIBUTE_MEMORY_TYPE

To get pointer's memory type in HIP/HIP-Clang one should use hipPointerGetAttributes API. The first parameter of the API is hipPointerAttribute_t which has 'memoryType' as a member variable. 'memoryType' indicates the input pointer is allocated on device or host.

For example:

```
double * ptr;
hipMalloc(reinterpret_cast<void**>(&ptr), sizeof(double));
hipPointerAttribute_t attr;
hipPointerGetAttributes(&attr, ptr); /*attr.memoryType will have value as hipMemoryTypeDevice*/

double* ptrHost;
hipHostMalloc(&ptrHost, sizeof(double));
hipPointerAttribute_t attr;
hipPointerGetAttributes(&attr, ptrHost); /*attr.memoryType will have value as
hipMemoryTypeHost*/
```

4.5.3 threadfence_system

Threadfence_system makes all device memory writes, all writes to mapped host memory, and all writes to peer memory visible to CPU and other GPU devices. Some implementations can provide this behavior by flushing the GPU L2 cache. HIP/HIP-Clang does not provide this functionality. As a workaround, users can set the environment variable HSA_DISABLE_CACHE=1 to disable the GPU L2 cache. This will affect all accesses and for all kernels and so may have a performance impact.

4.5.4 Textures and Cache Control

Compute programs sometimes use textures either to access dedicated texture caches or to use the texture-sampling hardware for interpolation and clamping. The former approach uses simple point samplers with linear interpolation, essentially only reading a single point. The latter approach uses the sampler hardware to interpolate and combine multiple samples. AMD hardware, as well as recent competing hardware, has a unified texture/L1 cache, so it no longer has a dedicated texture cache. But the nvcc path often caches global loads in the L2 cache, and some programs may benefit from explicit control of the L1 cache contents. We recommend the __ldg instruction for this purpose.

AMD compilers currently load all data into both the L1 and L2 caches, so __ldg is treated as a no-op.

We recommend the following for functional portability:

- For programs that use textures only to benefit from improved caching, use the *__ldg* instruction
- Programs that use texture object and reference APIs work well on HIP

4.6 More Tips

4.6.1 HIP Logging

On an AMD platform, set the AMD_LOG_LEVEL environment variable to log HIP application execution information.

For more information about HIP Logging refer to *section 3.5* in this document.

4.6.2 Debugging hipcc

To see the detailed commands that hipcc issues, set the environment variable HIPCC_VERBOSE to 1. Doing so will print to stderr the HIP-clang (or nvcc) commands that hipcc generates.

4.6.3 Editor Highlighting

See the utils/vim or utils/gedit directories to add handy highlighting to hip files.

4.7 HIP Porting Driver API

4.7.1 Porting CUDA Driver API

CUDA provides a separate CUDA Driver and Runtime APIs. The two APIs have significant overlap in functionality:

- Both APIs support events, streams, memory management, memory copy, and error handling.
- Both APIs deliver similar performance.
- Driver APIs calls begin with the prefix cu while Runtime APIs begin with the prefix cuda. For example, the Driver API API contains cuEventCreate while the Runtime API contains cudaEventCreate, with similar functionality.
- The Driver API defines a different but largely overlapping error code space than the Runtime API uses a different coding convention. For example, Driver API defines CUDA_ERROR_INVALID_VALUE while the Runtime API defines cudaErrorInvalidValue

NOTE: The Driver API offers two additional pieces of functionality not provided by the Runtime API: cuModule and cuCtx APIs.

4.7.2 cuModule API

The Module section of the Driver API provides additional control over how and when accelerator code objects are loaded. For example, the driver API allows code objects to be loaded from files or memory pointers. Symbols for kernels or global data can be extracted from the loaded code objects. In contrast, the Runtime API automatically loads and (if necessary) compiles all of the kernels from an executable binary when run. In this mode, NVCC must be used to compile kernel code so the automatic loading can function correctly.

Both Driver and Runtime APIs define a function for launching kernels (called cuLaunchKernel or cudaLaunchKernel. The kernel arguments and the execution configuration (grid dimensions, group dimensions, dynamic shared memory, and stream) are passed as arguments to the launch function. The Runtime additionally provides the <<< >>> syntax for launching kernels, which resembles a special function call and is easier to use than explicit launch API (in particular the handling of kernel arguments). However, this syntax is not standard C++ and is available only when NVCC is used to compile the host code.

The Module features are useful in an environment that generates the code objects directly, such as a new accelerator language front-end. Here, NVCC is not used. Instead, the environment may have a different kernel language or a different compilation flow. Other environments have many kernels and do not want them to be all loaded automatically. The Module functions can be used to load the generated code objects and launch kernels. As we will see below, HIP defines a Module API which provides similar explicit control over code object management.

4.7.3 cuCtx API

The Driver API defines "Context" and "Devices" as separate entities. Contexts contain a single device, and a device can theoretically have multiple contexts. Each context contains a set of streams and events specific to the context. Historically contexts also defined a unique address space for the GPU, though this may no longer be the case in Unified Memory platforms (since the CPU and all the devices in the same process share a single unified address space). The Context APIs also provide a mechanism to switch between devices, which allowed a single CPU thread to send commands to different GPUs. HIP as well as a recent version of CUDA Runtime provide other mechanisms to accomplish this feat - for example using streams or cudaSetDevice.

The CUDA Runtime API unifies the Context API with the Device API. This simplifies the APIs and has little loss of functionality since each Context can contain a single device, and the benefits of multiple contexts have been replaced with other interfaces. HIP provides a context API to facilitate easy porting from existing Driver codes. In HIP, the Ctx functions largely provide an alternate syntax for changing the active device. Most new applications will prefer to use hipSetDevice or the stream APIs , therefore HIP has marked hipCtx APIs as deprecated. Support for these APIs may not be available in future releases. For more details on deprecated APIs, refer to HIP deprecated APIs at:

https://github.com/ROCm-Developer-Tools/HIP/blob/master/docs/markdown/hip_deprecated_api_list.md

4.7.4 HIP Module and Ctx APIs

Rather than present two separate APIs, HIP extends the HIP API with new APIs for Modules and Ctx control.

4.7.4.1 hipModule API

Like the CUDA Driver API, the Module API provides additional control over how code is loaded, including options to load code from files or in-memory pointers. NVCC and HIP-Clang target different architectures and use different code object formats: NVCC is `cubin` or `ptx` files, while the HIP-Clang path is the `hsaco` format. The external compilers which generate these code objects are responsible for generating and loading the correct code object for each platform. Notably, there is no fat binary format that can contain code for both NVCC and HIP-Clang platforms. The following table summarizes the formats used on each platform:

Format	APIs	NVCC	HIP-CLANG
Code Object	hipModuleLoad, hipModuleLoadData	.cubin or PTX text	.hsaco
Fat Binary	hipModuleLoadFatBin	.fatbin	.hip_fatbin

`hipcc` uses HIP-Clang or NVCC to compile host codes. Both may embed code objects into the final executable, and these code objects will be automatically loaded when the application starts. The hipModule API can be used to load additional code objects, and in this way provides an extended capability to the automatically loaded code objects. HIP-Clang allows both capabilities to be used together if desired. It is possible to create a program with no kernels and thus no automatic loading.

4.7.5 hipCtx API

HIP provides a Ctx API as a thin layer over the existing Device functions. This Ctx API can be used to set the current context or to query properties of the device associated with the context. The current context is implicitly used by other APIs such as *hipStreamCreate*.

4.7.6 hipify translation of CUDA Driver API

The HIPIFY tools convert CUDA Driver APIs for streams, events, modules, devices, memory management, context, profiler to the equivalent HIP driver calls. For example, cuEventCreate will be translated into hipEventCreate. HIPIFY tools also convert error codes from the Driver namespace and coding convention to the equivalent HIP error code. Thus, HIP unifies the APIs for these common functions. The memory copy API requires additional explanation. The CUDA driver includes the memory direction in the name of the API (ie cuMemcpyH2D) while the CUDA driver API provides a single memory copy API with a parameter that specifies the direction and additionally supports a "default" direction where the runtime determines the direction automatically. HIP provides APIs with both styles: for example, hipMemcpyH2D as well as hipMemcpy. The first flavor may be faster in some cases since they avoid host overhead to detect different memory directions.

HIP defines a single error space and uses camel-case for all errors (i.e. hipErrorInvalidValue)

4.8 HIP-Clang Implementation Notes

4.8.1 .hip_fatbin

hip-clang links device code from different translation units together. For each device target, a code object is generated. Code objects for different device targets are bundled by clang-offload-bundler as one fatbinary, which is embedded as a global symbol __hip_fatbin in the .hip_fatbin section of the ELF file of the executable or shared object.

4.8.2 Initialization and Termination Functions

HIP-Clang generates initialization and termination functions for each translation unit for the host code compilation. The initialization functions call *__hipRegisterFatBinary* to register the fatbinary embedded in the ELF file. They also call *__hipRegisterFunction* and *__hipRegisterVar* to register kernel functions and device-side global variables. The termination functions call *__hipUnregisterFatBinar*y. HIP-Clang emits a global variable *__hip_gpubin_handle* of void** type with linkonce linkage and initial value 0 for each host translation unit. Each initialization function checks *__hip_gpubin_handle* and register the fatbinary only if *__hip_gpubin_handle* is 0 and saves the return value of *__hip_gpubin_handle* to *__hip_gpubin_handle*. This is to guarantee that the fatbinary is only registered once. A similar check is done in the termination functions.

4.8.3 Kernel Launching

HIP-Clang supports kernel launching by CUDA <<<>>> syntax, hipLaunchKernel, and hipLaunchKernelGGL. The latter two are macros that expand to CUDA <<<>>> syntax.

When the executable or shared library is loaded by the dynamic linker, the initialization functions are called. In the initialization functions, when __hipRegisterFatBinary is called, the code objects containing all kernels are loaded; when __hipRegisterFunction is called, the stub functions are associated with the corresponding kernels in code objects. HIP-Clang implements two sets of kernels launching APIs.

By default, in the host code, for the <<<>>> statement, hip-clang first emits call of hipConfigureCall to set up the threads and grids, then emits call of the stub function with the given arguments. In the stub function, hipSetupArgument is called for each kernel argument, then hipLaunchByPtr is called with a function pointer to the stub function. In *hipLaunchByPtr*, the real kernel associated with the stub function is launched.

If HIP program is compiled with -fhip-new-launch-api, in the host code, for the <<<>>> statement, hip-clang first emits call of __hipPushCallConfiguration to save the grid dimension, block dimension, shared memory usage and stream to a stack, then emits call of the stub function with the given arguments. In the stub function, __hipPopCallConfiguration is called to get the saved grid dimension, block dimension, shared memory usage and stream, then hipLaunchKernel is called with a function pointer to the stub function. In hipLaunchKernel, the real kernel associated with the stub function is launched.

4.8.4 Address Spaces

HIP-Clang defines a process-wide address space where the CPU and all devices allocate addresses from a single unified pool. Thus, addresses may be shared between contexts, and unlike the original CUDA definition, a new context does not create a new address space for the device.

4.8.5 Using hipModuleLaunchKernel

`hipModuleLaunchKernel` is `cuLaunchKernel` in HIP world. It takes the same arguments as `cuLaunchKernel`.

4.8.6 Additional Information

HIP-Clang creates a primary context when the HIP API is called. In a pure driver API code, HIP-Clang will create a primary context while HIP/NVCC will have an empty context stack. HIP-Clang will push the primary context to the context stack when it is empty. This can have subtle differences in applications that mix the runtime and driver APIs.

4.9 NVCC Implementation Notes

4.9.1 Interoperation between HIP and CUDA Driver

CUDA applications may want to mix CUDA driver code with HIP code. This table shows the type equivalence to enable this interaction.

HIP Type	CU Driver Type	CUDA Runtime Type
hipModule_t	CUmodule	
hipFunction_t	CUfunction	
hipCtx_t	CUcontext	
hipDevice_t	CUdevice	
hipStream_t	CUstream	cudaStream_t
hipEvent_t	CUevent	cudaEvent_t
hipArray	CUarray	cudaArray

4.9.2 Compilation Options

The hipModule_t interface does not support cuModuleLoadDataEx function, which is used to control PTX compilation options. HIP-Clang does not use PTX and does not support these compilation options. HIP-Clang code objects always contain fully compiled ISA and do not require additional compilation as a part of the load step.

The corresponding HIP function `hipModuleLoadDataEx` behaves as `hipModuleLoadData` on HIP-Clang path (compilation options are not used) and as `cuModuleLoadDataEx` on NVCC path.

For example,

CUDA

```
CUmodule module;
void *imagePtr = ...;  // Somehow populate data pointer with code object
const int numOptions = 1;
CUJit_option options[numOptions];
void * optionValues[numOptions];
options[0] = CU_JIT_MAX_REGISTERS;
unsigned maxRegs = 15;
optionValues[0] = (void*)(&maxRegs);

cuModuleLoadDataEx(module, imagePtr, numOptions, options, optionValues);
CUfunction k;
cuModuleGetFunction(&k, module, "myKernel");
```

HIP

```
hipModule_t module;
void *imagePtr = ...;  // Somehow populate data pointer with code object
const int numOptions = 1;
hipJitOption options[numOptions];
void * optionValues[numOptions];
options[0] = hipJitOptionMaxRegisters;
unsigned maxRegs = 15;
optionValues[0] = (void*)(&maxRegs);
// hipModuleLoadData(module, imagePtr) will be called on HIP-Clang path, JIT options will not be
used, and
// cupModuleLoadDataEx(module, imagePtr, numOptions, options, optionValues) will be called on
NVCC path
hipModuleLoadDataEx(module, imagePtr, numOptions, options, optionValues);
hipFunction_t k;
hipModuleGetFunction(&k, module, "myKernel");
```

The sample below shows how to use hipModuleGetFunction:

```
#include<hip_runtime.h>
#include<hip_runtime_api.h>
#include<iostream>
#include<fstream>
#include<vector>
#define LEN 64
#define SIZE LEN<<2
#ifdef __HIP_PLATFORM_HCC__
#define fileName "vcpy_isa.co"
#endif
#ifdef __HIP_PLATFORM_NVCC__
#define fileName "vcpy_isa.ptx"
#endif
#define kernel_name "hello_world"
int main(){
    float *A, *B;
    hipDeviceptr_t Ad, Bd;
    A = new float[LEN];
    B = new float[LEN];
    for(uint32_t i=0;i<LEN;i++){
        A[i] = i*1.0f;
        B[i] = 0.0f;
        std::cout<<A[i] << " "<<B[i]<<std::endl;
    }

#ifdef __HIP_PLATFORM_NVCC__
          hipInit(0);
          hipDevice_t device;
          hipCtx_t context;
          hipDeviceGet(&device, 0);
          hipCtxCreate(&context, 0, device);
#endif
```

```
    hipMalloc((void**)&Ad, SIZE);
    hipMalloc((void**)&Bd, SIZE);
    hipMemcpyHtoD(Ad, A, SIZE);
    hipMemcpyHtoD(Bd, B, SIZE);
    hipModule_t Module;
    hipFunction_t Function;
    hipModuleLoad(&Module, fileName);
    hipModuleGetFunction(&Function, Module, kernel_name);
    std::vector<void*>argBuffer(2);
    memcpy(&argBuffer[0], &Ad, sizeof(void*));
    memcpy(&argBuffer[1], &Bd, sizeof(void*));
    size_t size = argBuffer.size()*sizeof(void*);
    void *config[] = {
      HIP_LAUNCH_PARAM_BUFFER_POINTER, &argBuffer[0],
      HIP_LAUNCH_PARAM_BUFFER_SIZE, &size,
      HIP_LAUNCH_PARAM_END
    };
    hipModuleLaunchKernel(Function, 1, 1, 1, LEN, 1, 1, 0, 0, NULL, (void**)&config);
    hipMemcpyDtoH(B, Bd, SIZE);
    for(uint32_t i=0;i<LEN;i++){
        std::cout<<A[i]<<" - "<<B[i]<<std::endl;
    }

#ifdef __HIP_PLATFORM_NVCC__
          hipCtxDetach(context);
#endif

    return 0;
}
```

4.9.3 HIP Module and Texture Driver API

HIP supports texture driver APIs however texture reference should be declared in host scope. The following code explains the use of texture reference for the HIP_PLATFORM_HCC platform.

```
// Code to generate code object
#include "hip/hip_runtime.h"
extern texture<float, 2, hipReadModeElementType> tex;
__global__ void tex2dKernel(hipLaunchParm lp, float* outputData,
                            int width,
                            int height)
{
    int x = hipBlockIdx_x*hipBlockDim_x + hipThreadIdx_x;
    int y = hipBlockIdx_y*hipBlockDim_y + hipThreadIdx_y;
    outputData[y*width + x] = tex2D(tex, x, y);
}
// Host code:
texture<float, 2, hipReadModeElementType> tex;
void myFunc ()
{
    // ...
    textureReference* texref;
```

```
    hipModuleGetTexRef(&texref, Module1, "tex");
    hipTexRefSetAddressMode(texref, 0, hipAddressModeWrap);
    hipTexRefSetAddressMode(texref, 1, hipAddressModeWrap);
    hipTexRefSetFilterMode(texref, hipFilterModePoint);
    hipTexRefSetFlags(texref, 0);
    hipTexRefSetFormat(texref, HIP_AD_FORMAT_FLOAT, 1);
    hipTexRefSetArray(texref, array, HIP_TRSA_OVERRIDE_FORMAT);
   // ...
}
```

Chapter 5 Appendix A – HIP API

The following appendices are available on the AMD ROCm documentation website at:

http://rocmdocs.amd.com

5.1 HIP API Guide

You can access the Doxygen-generated HIP API Guide at the following location:

https://github.com/RadeonOpenCompute/ROCm/blob/master/HIP-API_Guide_v4.0.pdf

5.2 Supported CUDA APIs

To access the following supported CUDA APIs, see

https://rocmdocs.amd.com/en/latest/Programming_Guides/Programming-Guides.html#hip-faq-porting-guide-and-programming-guide

- Runtime API
- Driver API
- cuComplex API
- cuBLAS
- cuRAND
- cuDNN
- cuFFT
- cuSPARSE

5.3 Deprecated HIP APIs

5.3.1 HIP Context Management APIs

CUDA supports cuCtx API, the Driver API that defines "Context" and "Devices" as separate entities. Contexts contain a single device, and a device can theoretically have multiple contexts. HIP initially added limited support for APIs to facilitate easy porting from existing driver codes. The APIs are marked as deprecated now as there is a better alternate interface (such as hipSetDevice or the stream API) to achieve the required functions.

- hipCtxPopCurrent
- hipCtxPushCurrent
- hipCtxSetCurrent
- hipCtxGetCurrent
- hipCtxGetDevice
- hipCtxGetApiVersion

- hipCtxGetCacheConfig
- hipCtxSetCacheConfig
- hipCtxSetSharedMemConfig
- hipCtxGetSharedMemConfig
- hipCtxSynchronize
- hipCtxGetFlags
- hipCtxEnablePeerAccess
- hipCtxDisablePeerAccess

5.3.2 HIP Memory Management APIs

5.3.2.1 hipMallocHost

Use "hipHostMalloc" instead.

5.3.2.2 hipMemAllocHost

Use "hipHostMalloc" instead.

5.3.2.3 hipHostAlloc

Use "hipHostMalloc" instead.

5.3.2.4 hipFreeHost

Use "hipHostFree" instead.

5.4 Supported HIP Math APIs

You can access the supported HIP Math APIs at:

https://rocmdocs.amd.com/en/latest/ROCm_API_References/HIP-MATH.html#hip-math

Chapter 6 Appendix B – Supported Clang Options

6.1 Supported Clang Options

Clang version: clang version 12.0.0 927e2776dc0e4bb0119efbc5ea405b7425d7f4ac

Option	Support	Description
-###	Supported	Print (but do not run) the commands to run for this compilation
--analyzer-output <value>	Supported	Static analyzer report output format (html|plist|plist-multi-file|plist-html|sarif|text).
--analyze	Supported	Run the static analyzer
-arcmt-migrate-emit-errors	Unsupported	Emit ARC errors even if the migrator can fix them
-arcmt-migrate-report-output <value>	Unsupported	Output path for the plist report
-byteswapio	Supported	Swap byte-order for unformatted input/output
-B <dir>	Supported	Add <dir> to search path for binaries and object files used implicitly
-CC	Supported	Include comments from within macros in preprocessed output
-cl-denorms-are-zero	Supported	OpenCL only. Allow denormals to be flushed to zero.
-cl-fast-relaxed-math	Supported	OpenCL only. Sets -cl-finite-math-only and -cl-unsafe-math-optimizations, and defines __FAST_RELAXED_MATH__.
-cl-finite-math-only	Supported	OpenCL only. Allow floating-point optimizations that assume arguments and results are not NaNs or +-Inf.
-cl-fp32-correctly-rounded-divide-sqrt	Supported	OpenCL only. Specify that single-precision floating-point divide and sqrt used in the program source are correctly rounded.
-cl-kernel-arg-info	Supported	OpenCL only. Generate kernel argument metadata.
-cl-mad-enable	Supported	OpenCL only. Allow use of less precise MAD computations in the generated binary.
-cl-no-signed-zeros	Supported	OpenCL only. Allow use of less precise no signed zeros computations in the generated binary.
-cl-opt-disable	Supported	OpenCL only. This option disables all optimizations. By default optimizations are enabled.
-cl-single-precision-constant	Supported	OpenCL only. Treat double-precision floating-point constant as single precision constant.
-cl-std=<value>	Supported	OpenCL language standard to compile for.
-cl-strict-aliasing	Supported	OpenCL only. This option is added for compatibility with OpenCL 1.0.
-cl-uniform-work-group-size	Supported	OpenCL only. Defines that the global work-size be a multiple of the work-group size specified to clEnqueueNDRangeKernel
-cl-unsafe-math-optimizations	Supported	OpenCL only. Allow unsafe floating-point optimizations. Also implies -cl-no-signed-zeros and -cl-mad-enable.
--config <value>	Supported	Specifies configuration file
--cuda-compile-host-device	Supported	Compile CUDA code for both host and device (default). Has no effect on non-CUDA compilations.
--cuda-device-only	Supported	Compile CUDA code for device only
--cuda-host-only	Supported	Compile CUDA code for host only. Has no effect on non-CUDA compilations.

Option	Support	Description
--cuda-include-ptx=<value>	Unsupported	Include PTX for the following GPU architecture (e.g. sm_35) or 'all'. May be specified more than once.
--cuda-noopt-device-debug	Unsupported	Enable device-side debug info generation. Disables ptxas optimizations.
--cuda-path-ignore-env	Unsupported	Ignore environment variables to detect CUDA installation
--cuda-path=<value>	Unsupported	CUDA installation path
-cxx-isystem <directory>	Supported	Add a directory to the C++ SYSTEM include search path
-C	Supported	Include comments in preprocessed output
-c	Supported	Only run preprocess, compile, and assemble steps
-dD	Supported	Print macro definitions in -E mode in addition to normal output
-dependency-dot <value>	Supported	Filename to write DOT-formatted header dependencies to
-dependency-file <value>	Supported	Filename (or -) to write dependency output to
-dI	Supported	Print include directives in -E mode in addition to normal output
-dM	Supported	Print macro definitions in -E mode instead of normal output
-dsym-dir <dir>	Unsupported	Directory to output dSYM's (if any) to
-D <macro>	Supported	=<value> Define <macro> to <value> (or 1 if <value> omitted)
-emit-ast	Supported	Emit Clang AST files for source inputs
-emit-interface-stubs	Supported	Generate Interface Stub Files.
-emit-llvm	Supported	Use the LLVM representation for assembler and object files
-emit-merged-ifs	Supported	Generate Interface Stub Files, emit merged text not binary.
--emit-static-lib	Supported	Enable linker job to emit a static library.
-enable-trivial-auto-var-init-zero-knowing-it-will-be-removed-from-clang	Supported	Trivial automatic variable initialization to zero is only here for benchmarks, it'll eventually be removed, and I'm OK with that because I'm only using it to benchmark
-E	Supported	Only run the preprocessor
-fAAPCSBitfieldLoad	Unsupported	Follows the AAPCS standard that all volatile bit-field write generates at least one load. (ARM only).
-faddrsig	Supported	Emit an address-significance table
-faligned-allocation	Supported	Enable C++17 aligned allocation functions
-fallow-editor-placeholders	Supported	Treat editor placeholders as valid source code
-fallow-fortran-gnu-ext	Supported	Allow Fortran GNU extensions
-fansi-escape-codes	Supported	Use ANSI escape codes for diagnostics
-fapple-kext	Unsupported	Use Apple's kernel extensions ABI
-fapple-link-rtlib	Unsupported	Force linking the clang builtins runtime library
-fapple-pragma-pack	Unsupported	Enable Apple gcc-compatible #pragma pack handling
-fapplication-extension	Unsupported	Restrict code to those available for App Extensions
-fbackslash	Supported	Treat backslash as C-style escape character
-fbasic-block-sections=<value>	Supported	Place each function's basic blocks in unique sections (ELF Only) : all | labels | none | list=<file>
-fblocks	Supported	Enable the 'blocks' language feature
-fborland-extensions	Unsupported	Accept non-standard constructs supported by the Borland compiler
-fbuild-session-file=<file>	Supported	Use the last modification time of <file> as the build session timestamp
-fbuild-session-timestamp=<time since Epoch in seconds>	Supported	Time when the current build session started
-fbuiltin-module-map	Unsupported	Load the clang builtins module map file.
-fcall-saved-x10	Unsupported	Make the x10 register call-saved (AArch64 only)
-fcall-saved-x11	Unsupported	Make the x11 register call-saved (AArch64 only)
-fcall-saved-x12	Unsupported	Make the x12 register call-saved (AArch64 only)
-fcall-saved-x13	Unsupported	Make the x13 register call-saved (AArch64 only)

Option	Support	Description
-fcall-saved-x14	Unsupported	Make the x14 register call-saved (AArch64 only)
-fcall-saved-x15	Unsupported	Make the x15 register call-saved (AArch64 only)
-fcall-saved-x18	Unsupported	Make the x18 register call-saved (AArch64 only)
-fcall-saved-x8	Unsupported	Make the x8 register call-saved (AArch64 only)
-fcall-saved-x9	Unsupported	Make the x9 register call-saved (AArch64 only)
-fcf-protection=<value>	Unsupported	Instrument control-flow architecture protection. Options: return, branch, full, none.
-fcf-protection	Unsupported	Enable cf-protection in 'full' mode
-fchar8_t	Supported	Enable C++ builtin type char8_t
-fclang-abi-compat=<version>	Supported	Attempt to match the ABI of Clang <version>
-fcolor-diagnostics	Supported	Enable colors in diagnostics
-fcomment-block-commands=<arg>	Supported	Treat each comma separated argument in <arg> as a documentation comment block command
-fcommon	Supported	Place uninitialized global variables in a common block
-fcomplete-member-pointers	Supported	Require member pointer base types to be complete if they would be significant under the Microsoft ABI
-fconvergent-functions	Supported	Assume functions may be convergent
-fcoroutines-ts	Supported	Enable support for the C++ Coroutines TS
-fcoverage-mapping	Unsupported	Generate coverage mapping to enable code coverage analysis
-fcs-profile-generate=<directory>	Unsupported	Generate instrumented code to collect context sensitive execution counts into <directory>/default.profraw (overridden by LLVM_PROFILE_FILE env var)
-fcs-profile-generate	Unsupported	Generate instrumented code to collect context-sensitive execution counts into default.profraw (overridden by LLVM_PROFILE_FILE env var)
-fcuda-approx-transcendentals	Unsupported	Use approximate transcendental functions
-fcuda-flush-denormals-to-zero	Supported	Flush denormal floating-point values to zero in CUDA device mode.
-fcuda-short-ptr	Unsupported	Use 32-bit pointers for accessing const/local/shared address spaces
-fcxx-exceptions	Supported	Enable C++ exceptions
-fdata-sections	Supported	Place each data in its section
-fdebug-compilation-dir <value>	Supported	The compilation directory to embed in the debug info.
-fdebug-default-version=<value>	Supported	Default DWARF version to use, if a -g option caused DWARF debug info to be produced
-fdebug-info-for-profiling	Supported	Emit extra debug info to make the sample profile more accurate
-fdebug-macro	Supported	Emit macro debug information
-fdebug-prefix-map=<value>	Supported	remap file source paths in debug info
-fdebug-ranges-base-address	Supported	Use DWARF base address selection entries in .debug_ranges
-fdebug-types-section	Supported	Place debug types in their section (ELF Only)
-fdeclspec	Supported	Allow __declspec as a keyword
-fdelayed-template-parsing	Supported	Parse templated function definitions at the end of the translation unit
-fdelete-null-pointer-checks	Supported	Treat usage of null pointers as undefined behavior (default)
-fdiagnostics-absolute-paths	Supported	Print absolute paths in diagnostics
-fdiagnostics-hotness-threshold=<number>	Unsupported	Prevent optimization remarks from being output if they do not have at least this profile count

Option	Support	Description
-fdiagnostics-parseable-fixits	Supported	Print fix-its in machine parseable form
-fdiagnostics-print-source-range-info	Supported	Print source range spans in numeric form
-fdiagnostics-show-hotness	Unsupported	Enable profile hotness information in diagnostic line
-fdiagnostics-show-note-include-stack	Supported	Display include stacks for diagnostic notes
-fdiagnostics-show-option	Supported	Print option name with mappable diagnostics
-fdiagnostics-show-template-tree	Supported	Print a template comparison tree for differing templates
-fdigraphs	Supported	Enable alternative token representations '<:', ':>', '<%', '%>', '%:', '%:%:' (default)
-fdiscard-value-names	Supported	Discard value names in LLVM IR
-fdollars-in-identifiers	Supported	Allow '$' in identifiers
-fdouble-square-bracket-attributes	Supported	Enable '[[]]' attributes in all C and C++ language modes
-fdwarf-exceptions	Unsupported	Use DWARF style exceptions
-feliminate-unused-debug-types	Supported	Do not emit debug info for defined but unused types
-fembed-bitcode-marker	Supported	Embed placeholder LLVM IR data as a marker
-fembed-bitcode=<option>	Supported	Embed LLVM bitcode (option: off, all, bitcode, marker)
-fembed-bitcode	Supported	Embed LLVM IR bitcode as data
-femit-all-decls	Supported	Emit all declarations, even if unused
-femulated-tls	Supported	Use emutls functions to access thread_local variables
-fenable-matrix	Supported	Enable matrix data type and related builtin functions
-fexceptions	Supported	Enable support for exception handling
-fexperimental-new-constant-interpreter	Supported	Enable the experimental new constant interpreter
-fexperimental-new-pass-manager	Supported	Enables an experimental new pass manager in LLVM.
-fexperimental-relative-c++-abi-vtables	Supported	Use the experimental C++ class ABI for classes with virtual tables
-fexperimental-strict-floating-point	Supported	Enables experimental strict floating point in LLVM.
-ffast-math	Supported	Allow aggressive, lossy floating-point optimizations
-ffile-prefix-map=<value>	Supported	remap file source paths in debug info and predefined preprocessor macros
-ffine-grained-bitfield-accesses	Supported	Use separate accesses for consecutive bitfield runs with legal widths and alignments.
-ffixed-form	Supported	Enable fixed-form format for Fortran
-ffixed-point	Supported	Enable fixed point types
-ffixed-r19	Unsupported	Reserve register r19 (Hexagon only)
-ffixed-r9	Unsupported	Reserve the r9 register (ARM only)
-ffixed-x10	Unsupported	Reserve the x10 register (AArch64/RISC-V only)
-ffixed-x11	Unsupported	Reserve the x11 register (AArch64/RISC-V only)
-ffixed-x12	Unsupported	Reserve the x12 register (AArch64/RISC-V only)
-ffixed-x13	Unsupported	Reserve the x13 register (AArch64/RISC-V only)
-ffixed-x14	Unsupported	Reserve the x14 register (AArch64/RISC-V only)
-ffixed-x15	Unsupported	Reserve the x15 register (AArch64/RISC-V only)
-ffixed-x16	Unsupported	Reserve the x16 register (AArch64/RISC-V only)
-ffixed-x17	Unsupported	Reserve the x17 register (AArch64/RISC-V only)
-ffixed-x18	Unsupported	Reserve the x18 register (AArch64/RISC-V only)

Option	Support	Description
-ffixed-x19	Unsupported	Reserve the x19 register (AArch64/RISC-V only)
-ffixed-x1	Unsupported	Reserve the x1 register (AArch64/RISC-V only)
-ffixed-x20	Unsupported	Reserve the x20 register (AArch64/RISC-V only)
-ffixed-x21	Unsupported	Reserve the x21 register (AArch64/RISC-V only)
-ffixed-x22	Unsupported	Reserve the x22 register (AArch64/RISC-V only)
-ffixed-x23	Unsupported	Reserve the x23 register (AArch64/RISC-V only)
-ffixed-x24	Unsupported	Reserve the x24 register (AArch64/RISC-V only)
-ffixed-x25	Unsupported	Reserve the x25 register (AArch64/RISC-V only)
-ffixed-x26	Unsupported	Reserve the x26 register (AArch64/RISC-V only)
-ffixed-x27	Unsupported	Reserve the x27 register (AArch64/RISC-V only)
-ffixed-x28	Unsupported	Reserve the x28 register (AArch64/RISC-V only)
-ffixed-x29	Unsupported	Reserve the x29 register (AArch64/RISC-V only)
-ffixed-x2	Unsupported	Reserve the x2 register (AArch64/RISC-V only)
-ffixed-x30	Unsupported	Reserve the x30 register (AArch64/RISC-V only)
-ffixed-x31	Unsupported	Reserve the x31 register (AArch64/RISC-V only)
-ffixed-x3	Unsupported	Reserve the x3 register (AArch64/RISC-V only)
-ffixed-x4	Unsupported	Reserve the x4 register (AArch64/RISC-V only)
-ffixed-x5	Unsupported	Reserve the x5 register (AArch64/RISC-V only)
-ffixed-x6	Unsupported	Reserve the x6 register (AArch64/RISC-V only)
-ffixed-x7	Unsupported	Reserve the x7 register (AArch64/RISC-V only)
-ffixed-x8	Unsupported	Reserve the x8 register (AArch64/RISC-V only)
-ffixed-x9	Unsupported	Reserve the x9 register (AArch64/RISC-V only)
-fforce-dwarf-frame	Supported	Always emit a debug frame section
-fforce-emit-vtables	Supported	Emits more virtual tables to improve devirtualization
-fforce-enable-int128	Supported	Enable support for int128_t type
-ffp-contract=<value>	Supported	Form fused FP ops (e.g. FMAs): fast (everywhere) | on (according to FP_CONTRACT pragma) | off (never fuse). Default is 'fast' for CUDA/HIP and 'on' otherwise.
-ffp-exception-behavior=<value>	Supported	Specifies the exception behavior of floating-point operations.
-ffp-model=<value>	Supported	Controls the semantics of floating-point calculations.
-ffree-form	Supported	Enable free-form format for Fortran
-ffreestanding	Supported	Assert that the compilation takes place in a freestanding environment
-ffunc-args-alias	Supported	Function argument may alias (equivalent to ansi alias)
-ffunction-sections	Supported	Place each function in its section
-fglobal-isel	Supported	Enables the global instruction selector
-fgnu-keywords	Supported	Allow GNU-extension keywords regardless of a language standard
-fgnu-runtime	Unsupported	Generate output compatible with the standard GNU Objective-C runtime
-fgnu89-inline	Unsupported	Use the gnu89 inline semantics
-fgnuc-version=<value>	Supported	Sets various macros to claim compatibility with the given GCC version (default is 4.2.1)
-fgpu-allow-device-init	Supported	Allow device-side init function in HIP
-fgpu-rdc	Supported	Generate relocatable device code, also known as separate compilation mode
-fhip-new-launch-api	Supported	Use new kernel launching API for HIP
-fignore-exceptions	Supported	Enable support for ignoring exception handling constructs
-fimplicit-module-maps	Unsupported	Implicitly search the file system for module map files.
-finline-functions	Supported	Inline suitable functions
-finline-hint-functions	Supported	Inline functions that are (explicitly or implicitly) marked inline

Option	Support	Description
-finstrument-function-entry-bare	Unsupported	Instrument function entry only, after inlining, without arguments to the instrumentation call
-finstrument-functions-after-inlining	Unsupported	Like -finstrument-functions, but insert the calls after inlining
-finstrument-functions	Unsupported	Generate calls to instrument function entry and exit
-fintegrated-as	Supported	Enable the integrated assembler
-fintegrated-cc1	Supported	Run cc1 in-process
-fjump-tables	Supported	Use jump tables for lowering switches
-fkeep-static-consts	Supported	Keep static const variables if unused
-flax-vector-conversions=<value>	Supported	Enable implicit vector bit-casts
-flto-jobs=<value>	Unsupported	Controls the backend parallelism of -flto=thin (default of 0 means the number of threads will be derived from the number of CPUs detected)
-flto=<value>	Unsupported	Set LTO mode to either 'full' or 'thin'
-flto	Unsupported	Enable LTO in 'full' mode
-fmacro-prefix-map=<value>	Supported	remap file source paths in predefined preprocessor macros
-fmath-errno	Supported	Require math functions to indicate errors by setting errno
-fmax-tokens=<value>	Supported	Max total number of preprocessed tokens for -Wmax-tokens.
-fmax-type-align=<value>	Supported	Specify the maximum alignment to enforce on pointers lacking an explicit alignment
-fmemory-profile	Supported	Enable heap memory profiling
-fmerge-all-constants	Supported	Allow merging of constants
-fmessage-length=<value>	Supported	Format message diagnostics so that they fit within N columns
-fmodule-file=[<name>=]<file>	Unsupported	Specify the mapping of module name to precompiled module file, or load a module file if name is omitted.
-fmodule-map-file=<file>	Unsupported	Load this module map file
-fmodule-name=<name>	Unsupported	Specify the name of the module to build
-fmodules-cache-path=<directory>	Unsupported	Specify the module cache path
-fmodules-decluse	Unsupported	Require declaration of modules used within a module
-fmodules-disable-diagnostic-validation	Unsupported	Disable validation of the diagnostic options when loading the module
-fmodules-ignore-macro=<value>	Unsupported	Ignore the definition of the given macro when building and loading modules
-fmodules-prune-after=<seconds>	Unsupported	Specify the interval (in seconds) after which a module file will be considered unused
-fmodules-prune-interval=<seconds>	Unsupported	Specify the interval (in seconds) between attempts to prune the module cache
-fmodules-search-all	Unsupported	Search even non-imported modules to resolve references
-fmodules-strict-decluse	Unsupported	Like -fmodules-decluse but requires all headers to be in modules
-fmodules-ts	Unsupported	Enable support for the C++ Modules TS
-fmodules-user-build-path <directory>	Unsupported	Specify the module user build path
-fmodules-validate-input-files-content	Supported	Validate PCM input files based on content if mtime differs
-fmodules-validate-once-per-build-session	Unsupported	Don't verify input files for the modules if the module has been successfully validated or loaded during this build session
-fmodules-validate-system-headers	Supported	Validate the system headers that a module depends on when loading the module
-fmodules	Unsupported	Enable the 'modules' language feature

Option	Support	Description
-fms-compatibility-version=<value>	Supported	Dot-separated value representing the Microsoft compiler version number to report in _MSC_VER (0 = don't define it (default))
-fms-compatibility	Supported	Enable full Microsoft Visual C++ compatibility
-fms-extensions	Supported	Accept some non-standard constructs supported by the Microsoft compiler
-fmsc-version=<value>	Supported	Microsoft compiler version number to report in _MSC_VER (0 = don't define it (default))
-fnew-alignment=<align>	Supported	Specifies the largest alignment guaranteed by '::operator new(size_t)'
-fno-addrsig	Supported	Don't emit an address-significance table
-fno-allow-fortran-gnu-ext	Supported	Allow Fortran GNU extensions
-fno-assume-sane-operator-new	Supported	Don't assume that C++'s global operator new can't alias any pointer
-fno-autolink	Supported	Disable generation of linker directives for automatic library linking
-fno-backslash	Supported	Treat backslash like any other character in character strings
-fno-builtin-<value>	Supported	Disable implicit builtin knowledge of a specific function
-fno-builtin	Supported	Disable implicit builtin knowledge of functions
-fno-c++-static-destructors	Supported	Disable C++ static destructor registration
-fno-char8_t	Supported	Disable C++ builtin type char8_t
-fno-color-diagnostics	Supported	Disable colors in diagnostics
-fno-common	Supported	Compile common globals like normal definitions
-fno-complete-member-pointers	Supported	Do not require member pointer base types to be complete if they would be significant under the Microsoft ABI
-fno-constant-cfstrings	Supported	Disable creation of CodeFoundation-type constant strings
-fno-coverage-mapping	Supported	Disable code coverage analysis
-fno-crash-diagnostics	Supported	Disable auto-generation of preprocessed source files and a script for reproduction during a clang crash
-fno-cuda-approx-transcendentals	Unsupported	Don't use approximate transcendental functions
-fno-debug-macro	Supported	Do not emit macro debug information
-fno-declspec	Unsupported	Disallow __declspec as a keyword
-fno-delayed-template-parsing	Supported	Disable delayed template parsing
-fno-delete-null-pointer-checks	Supported	Do not treat usage of null pointers as undefined behavior
-fno-diagnostics-fixit-info	Supported	Do not include fixit information in diagnostics
-fno-digraphs	Supported	Disallow alternative token representations '<:', ':>', '<%', '%>', '%:', '%:%:'
-fno-discard-value-names	Supported	Do not discard value names in LLVM IR
-fno-dollars-in-identifiers	Supported	Disallow '$' in identifiers
-fno-double-square-bracket-attributes	Supported	Disable '[[]]' attributes in all C and C++ language modes
-fno-elide-constructors	Supported	Disable C++ copy constructor elision
-fno-elide-type	Supported	Do not elide types when printing diagnostics
-fno-eliminate-unused-debug-types	Supported	Emit debug info for defined but unused types
-fno-exceptions	Supported	Disable support for exception handling
-fno-experimental-new-pass-manager	Supported	Disables an experimental new pass manager in LLVM.
-fno-experimental-relative-c++-abi-vtables	Supported	Do not use the experimental C++ class ABI for classes with virtual tables

Option	Support	Description
-fno-fine-grained-bitfield-accesses	Supported	Use large-integer access for consecutive bitfield runs.
-fno-fixed-form	Supported	Disable fixed-form format for Fortran
-fno-fixed-point	Supported	Disable fixed point types
-fno-force-enable-int128	Supported	Disable support for int128_t type
-fno-fortran-main	Supported	Don't link in Fortran main
-fno-free-form	Supported	Disable free-form format for Fortran
-fno-func-args-alias	Supported	Function argument may alias (equivalent to ansi alias)
-fno-global-isel	Supported	Disables the global instruction selector
-fno-gnu-inline-asm	Supported	Disable GNU style inline asm
-fno-gpu-allow-device-init	Supported	Don't allow device-side init function in HIP
-fno-hip-new-launch-api	Supported	Don't use new kernel launching API for HIP
-fno-integrated-as	Supported	Disable the integrated assembler
-fno-integrated-cc1	Supported	Spawn a separate process for each cc1
-fno-jump-tables	Supported	Do not use jump tables for lowering switches
-fno-keep-static-consts	Supported	Don't keep static const variables if unused
-fno-lto	Supported	Disable LTO mode (default)
-fno-memory-profile	Supported	Disable heap memory profiling
-fno-merge-all-constants	Supported	Disallow merging of constants
-fno-no-access-control	Supported	Disable C++ access control
-fno-objc-infer-related-result-type	Supported	do not infer Objective-C related result type based on method family
-fno-operator-names	Supported	Do not treat C++ operator name keywords as synonyms for operators
-fno-pch-codegen	Supported	Do not generate code for uses of this PCH that assumes an explicit object file will be built for the PCH
-fno-pch-debuginfo	Supported	Do not generate debug info for types in an object file built from this PCH and do not generate them elsewhere
-fno-plt	Supported	Use GOT indirection instead of PLT to make external function calls (x86 only)
-fno-preserve-as-comments	Supported	Do not preserve comments in inline assembly
-fno-profile-generate	Supported	Disable generation of profile instrumentation.
-fno-profile-instr-generate	Supported	Disable generation of profile instrumentation.
-fno-profile-instr-use	Supported	Disable using instrumentation data for profile-guided optimization
-fno-register-global-dtors-with-atexit	Supported	Don't use atexit or __cxa_atexit to register global destructors
-fno-rtlib-add-rpath	Supported	Do not add -rpath with architecture-specific resource directory to the linker flags
-fno-rtti-data	Supported	Disable generation of RTTI data
-fno-rtti	Supported	Disable generation of rtti information
-fno-sanitize-address-poison-custom-array-cookie	Supported on Host only	Disable poisoning array cookies when using custom operator new[] in AddressSanitizer
-fno-sanitize-address-use-after-scope	Supported on Host only	Disable use-after-scope detection in AddressSanitizer
-fno-sanitize-address-use-odr-indicator	Supported on Host only	Disable ODR indicator globals
-fno-sanitize-blacklist	Supported on Host only	Don't use blacklist file for sanitizers
-fno-sanitize-cfi-canonical-jump-tables	Supported on Host only	Do not make the jump table addresses canonical in the symbol table
-fno-sanitize-cfi-cross-dso	Supported on Host only	Disable control flow integrity (CFI) checks for cross-DSO calls.

Option	Support	Description
-fno-sanitize-coverage=<value>	Supported on Host only	Disable specified features of coverage instrumentation for Sanitizers
-fno-sanitize-memory-track-origins	Supported on Host only	Disable origins tracking in MemorySanitizer
-fno-sanitize-memory-use-after-dtor	Supported on Host only	Disable use-after-destroy detection in MemorySanitizer
-fno-sanitize-recover=<value>	Supported on Host only	Disable recovery for specified sanitizers
-fno-sanitize-stats	Supported on Host only	Disable sanitizer statistics gathering.
-fno-sanitize-thread-atomics	Supported on Host only	Disable atomic operations instrumentation in ThreadSanitizer
-fno-sanitize-thread-func-entry-exit	Supported on Host only	Disable function entry/exit instrumentation in ThreadSanitizer
-fno-sanitize-thread-memory-access	Supported on Host only	Disable memory access instrumentation in ThreadSanitizer
-fno-sanitize-trap=<value>	Supported on Host only	Disable trapping for specified sanitizers
-fno-sanitize-trap	Supported on Host only	Disable trapping for all sanitizers
-fno-short-wchar	Supported	Force wchar_t to be an unsigned int
-fno-show-column	Supported	Do not include column number on diagnostics
-fno-show-source-location	Supported	Do not include source location information with diagnostics
-fno-signed-char	Supported	char is unsigned
-fno-signed-zeros	Supported	Allow optimizations that ignore the sign of floating point zeros
-fno-spell-checking	Supported	Disable spell-checking
-fno-split-machine-functions	Supported	Disable late function splitting using profile information (x86 ELF)
-fno-stack-clash-protection	Supported	Disable stack clash protection
-fno-stack-protector	Supported	Disable the use of stack protectors
-fno-standalone-debug	Supported	Limit debug information produced to reduce size of debug binary
-fno-strict-float-cast-overflow	Supported	Relax language rules and try to match the behavior of the target's native float-to-int conversion instructions
-fno-strict-return	Supported	Don't treat control flow paths that fall off the end of a non-void function as unreachable
-fno-sycl	Unsupported	Disable SYCL kernels compilation for device
-fno-temp-file	Supported	Directly create compilation output files. This may lead to incorrect incremental builds if the compiler crashes
-fno-threadsafe-statics	Supported	Do not emit code to make initialization of local statics thread safe
-fno-trigraphs	Supported	Do not process trigraph sequences
-fno-unique-section-names	Supported	Don't use unique names for text and data sections
-fno-unroll-loops	Supported	Turn off loop unroller
-fno-use-cxa-atexit	Supported	Don't use __cxa_atexit for calling destructors
-fno-use-flang-math-libs	Supported	Use Flang internal runtime math library instead of LLVM math intrinsics.
-fno-use-init-array	Supported	Use .ctors/.dtors instead of .init_array/.fini_array
-fno-visibility-inlines-hidden-static-local-var	Supported	Disables -fvisibility-inlines-hidden-static-local-var (this is the default on non-darwin targets)
-fno-xray-function-index	Unsupported	Omit function index section at the expense of single-function patching performance
-fno-zero-initialized-in-bss	Supported	Don't place zero initialized data in BSS

Option	Support	Description
-fobjc-arc-exceptions	Unsupported	Use EH-safe code when synthesizing retains and releases in -fobjc-arc
-fobjc-arc	Unsupported	Synthesize retain and release calls for Objective-C pointers
-fobjc-exceptions	Unsupported	Enable Objective-C exceptions
-fobjc-runtime=<value>	Unsupported	Specify the target Objective-C runtime kind and version
-fobjc-weak	Unsupported	Enable ARC-style weak references in Objective-C
-fopenmp-simd	Unsupported	Emit OpenMP code only for SIMD-based constructs.
-fopenmp-targets=<value>	Unsupported	Specify a comma-separated list of triples OpenMP offloading targets to be supported
-fopenmp	Unsupported	Parse OpenMP pragmas and generate parallel code.
-foptimization-record-file=<file>	Supported	Specify the output name of the file containing the optimization remarks. Implies -fsave-optimization-record. On Darwin platforms, this cannot be used with multiple -arch <arch> options.
-foptimization-record-passes=<regex>	Supported	Only include passes that match a specified regular expression in the generated optimization record (by default, include all passes)
-forder-file-instrumentation	Supported	Generate instrumented code to collect order file into default.profraw file (overridden by '=' form of option or LLVM_PROFILE_FILE env var)
-fpack-struct=<value>	Unsupported	Specify the default maximum struct packing alignment
-fpascal-strings	Supported	Recognize and construct Pascal-style string literals
-fpass-plugin=<dsopath>	Supported	Load pass plugin from a dynamic shared object file (only with new pass manager).
-fpatchable-function-entry=<N,M>	Supported	Generate M NOPs before function entry and N-M NOPs after function entry
-fpcc-struct-return	Unsupported	Override the default ABI to return all structs on the stack
-fpch-codegen	Supported	Generate code for uses of this PCH that assumes an explicit object file will be built for the PCH
-fpch-debuginfo	Supported	Generate debug info for types in an object file built from this PCH and do not generate them elsewhere
-fpch-instantiate-templates	Supported	Instantiate templates already while building a PCH
-fpch-validate-input-files-content	Supported	Validate PCH input files based on content if mtime differs
-fplugin=<dsopath>	Supported	Load the named plugin (dynamic shared object)
-fprebuilt-module-path=<directory>	Unsupported	Specify the prebuilt module path
-fprofile-exclude-files=<value>	Unsupported	Instrument only functions from files where names don't match all the regexes separated by a semi-colon
-fprofile-filter-files=<value>	Unsupported	Instrument only functions from files where names match any regex separated by a semi-colon
-fprofile-generate=<directory>	Unsupported	Generate instrumented code to collect execution counts into <directory>/default.profraw (overridden by LLVM_PROFILE_FILE env var)
-fprofile-generate	Unsupported	Generate instrumented code to collect execution counts into default.profraw (overridden by LLVM_PROFILE_FILE env var)
-fprofile-instr-generate=<file>	Unsupported	Generate instrumented code to collect execution counts into <file> (overridden by LLVM_PROFILE_FILE env var)
-fprofile-instr-generate	Unsupported	Generate instrumented code to collect execution counts into default.profraw file (overridden by '=' form of option or LLVM_PROFILE_FILE env var)
-fprofile-instr-use=<value>	Unsupported	Use instrumentation data for profile-guided optimization
-fprofile-remapping-file=<file>	Unsupported	Use the remappings described in <file> to match the profile data against names in the program
-fprofile-sample-accurate	Unsupported	Specifies that the sample profile is accurate

Option	Support	Description
-fprofile-sample-use=<value>	Unsupported	Enable sample-based profile guided optimizations
-fprofile-use=<pathname>	Unsupported	Use instrumentation data for profile-guided optimization. If pathname is a directory, it reads from <pathname>/default.profdata. Otherwise, it reads from file <pathname>.
-freciprocal-math	Supported	Allow division operations to be reassociated
-freg-struct-return	Unsupported	Override the default ABI to return small structs in registers
-fregister-global-dtors-with-atexit	Supported	Use atexit or __cxa_atexit to register global destructors
-frelaxed-template-template-args	Supported	Enable C++17 relaxed template argument matching
-freroll-loops	Supported	Turn on loop reroller
-fropi	Unsupported	Generate read-only position independent code (ARM only)
-frtlib-add-rpath	Supported	Add -rpath with architecture-specific resource directory to the linker flags
-frwpi	Unsupported	Generate read-write position independent code (ARM only)
-fsanitize-address-field-padding=<value>	Supported on Host only	Level of field padding for AddressSanitizer
-fsanitize-address-globals-dead-stripping	Supported on Host only	Enable linker dead stripping of globals in AddressSanitizer
-fsanitize-address-poison-custom-array-cookie	Supported on Host only	Enable poisoning array cookies when using custom operator new[] in AddressSanitizer
-fsanitize-address-use-after-scope	Supported on Host only	Enable use-after-scope detection in AddressSanitizer
-fsanitize-address-use-odr-indicator	Supported on Host only	Enable ODR indicator globals to avoid false ODR violation reports in partially sanitized programs at the cost of an increase in binary size
-fsanitize-blacklist=<value>	Supported on Host only	Path to blacklist file for sanitizers
-fsanitize-cfi-canonical-jump-tables	Supported on Host only	Make the jump table addresses canonical in the symbol table
-fsanitize-cfi-cross-dso	Supported on Host only	Enable control flow integrity (CFI) checks for cross-DSO calls.
-fsanitize-cfi-icall-generalize-pointers	Supported on Host only	Generalize pointers in CFI indirect call type signature checks
-fsanitize-coverage-allowlist=<value>	Supported on Host only	Restrict sanitizer coverage instrumentation exclusively to modules and functions that match the provided special case list, except the blocked ones
-fsanitize-coverage-blacklist=<value>	Supported on Host only	Deprecated, use -fsanitize-coverage-blocklist= instead
-fsanitize-coverage-blocklist=<value>	Supported on Host only	Disable sanitizer coverage instrumentation for modules and functions that match the provided special case list, even the allowed ones
-fsanitize-coverage-whitelist=<value>	Supported on Host only	Deprecated, use -fsanitize-coverage-allowlist= instead
-fsanitize-coverage=<value>	Supported on Host only	Specify the type of coverage instrumentation for Sanitizers
-fsanitize-hwaddress-abi=<value>	Supported on Host only	Select the HWAddressSanitizer ABI to target (interceptor or platform, default interceptor). This option is currently unused.
-fsanitize-memory-track-origins=<value>	Supported on Host only	Enable origins tracking in MemorySanitizer

Option	Support	Description
-fsanitize-memory-track-origins	Supported on Host only	Enable origins tracking in MemorySanitizer
-fsanitize-memory-use-after-dtor	Supported on Host only	Enable use-after-destroy detection in MemorySanitizer
-fsanitize-recover=<value>	Supported on Host only	Enable recovery for specified sanitizers
-fsanitize-stats	Supported on Host only	Enable sanitizer statistics gathering.
-fsanitize-system-blacklist=<value>	Supported on Host only	Path to system blacklist file for sanitizers
-fsanitize-thread-atomics	Supported on Host only	Enable atomic operations instrumentation in ThreadSanitizer (default)
-fsanitize-thread-func-entry-exit	Supported on Host only	Enable function entry/exit instrumentation in ThreadSanitizer (default)
-fsanitize-thread-memory-access	Supported on Host only	Enable memory access instrumentation in ThreadSanitizer (default)
-fsanitize-trap=<value>	Supported on Host only	Enable trapping for specified sanitizers
-fsanitize-trap	Supported on Host only	Enable trapping for all sanitizers
-fsanitize-undefined-strip-path-components=<number>	Supported on Host only	Strip (or keep only, if negative) a given number of path components when emitting check metadata.
-fsanitize=<check>	Supported on Host only	Turn on runtime checks for various forms of undefined or suspicious behavior. See user manual for available checks
-fsave-optimization-record=<format>	Supported	Generate an optimization record file in a specific format
-fsave-optimization-record	Supported	Generate a YAML optimization record file
-fseh-exceptions	Supported	Use SEH style exceptions
-fshort-enums	Supported	Allocate to an enum type only as many bytes as it needs for the declared range of possible values
-fshort-wchar	Unsupported	Force wchar_t to be a short unsigned int
-fshow-overloads=<value>	Supported	Which overload candidates to show when overload resolution fails: best|all; defaults to all
-fsigned-char	Supported	char is signed
-fsized-deallocation	Supported	Enable C++14 sized global deallocation functions
-fsjlj-exceptions	Supported	Use SjLj style exceptions
-fslp-vectorize	Supported	Enable the superword-level parallelism vectorization passes
-fsplit-dwarf-inlining	Unsupported	Provide minimal debug info in the object/executable to facilitate online symbolication/stack traces in the absence of .dwo/.dwp files when using Split DWARF
-fsplit-lto-unit	Unsupported	Enables splitting of the LTO unit
-fsplit-machine-functions	Supported	Enable late function splitting using profile information (x86 ELF)
-fstack-clash-protection	Supported	Enable stack clash protection
-fstack-protector-all	Unsupported	Enable stack protectors for all functions
-fstack-protector-strong	Unsupported	Enable stack protectors for some functions vulnerable to stack smashing. Compared to -fstack-protector, this uses a stronger heuristic that includes functions containing arrays of any size (and any type), as well as any calls to alloca or the taking of an address from a local variable
-fstack-protector	Unsupported	Enable stack protectors for some functions vulnerable to stack smashing. This uses a loose heuristic that considers functions vulnerable if they contain a char (or 8bit integer) array or constant

Option	Support	Description
		sized calls to alloca , which are of greater size than ssp-buffer-size (default: 8 bytes). All variable sized calls to alloca are considered vulnerable. A function with a stack protector has a guard value added to the stack frame that is checked on function exit. The guard value must be positioned in the stack frame such that a buffer overflow from a vulnerable variable will overwrite the guard value before overwriting the function's return address. The reference stack guard value is stored in a global variable.
-fstack-size-section	Supported	Emit section containing metadata on function stack sizes
-fstandalone-debug	Supported	Emit full debug info for all types used by the program
-fstrict-enums	Supported	Enable optimizations based on the strict definition of an enum's value range
-fstrict-float-cast-overflow	Supported	Assume that overflowing float-to-int casts are undefined (default)
-fstrict-vtable-pointers	Supported	Enable optimizations based on the strict rules for overwriting polymorphic C++ objects
-fsycl	Unsupported	Enable SYCL kernels compilation for device
-fsystem-module	u	Build this module as a system module. Only used with -emit-module
-fthin-link-bitcode=<value>	Supported	Write minimized bitcode to <file> for the ThinLTO thin link only
-fthinlto-index=<value>	Unsupported	Perform ThinLTO importing using the provided function summary index
-ftime-trace-granularity=<value>	Supported	Minimum time granularity (in microseconds) traced by time profiler
-ftime-trace	Supported	Turn on time profiler. Generates JSON file based on output filename.
-ftrap-function=<value>	Unsupported	Issue call to specified function rather than a trap instruction
-ftrapv-handler=<function name>	Unsupported	Specify the function to be called on overflow
-ftrapv	Unsupported	Trap on integer overflow
-ftrigraphs	Supported	Process trigraph sequences
-ftrivial-auto-var-init-stop-after=<value>	Supported	Stop initializing trivial automatic stack variables after the specified number of instances
-ftrivial-auto-var-init=<value>	Supported	Initialize trivial automatic stack variables: uninitialized (default) | pattern
-funique-basic-block-section-names	Supported	Use unique names for basic block sections (ELF Only)
-funique-internal-linkage-names	Supported	Uniqueify Internal Linkage Symbol Names by appending the MD5 hash of the module path
-funroll-loops	Supported	Turn on loop unroller
-fuse-flang-math-libs	Supported	Use Flang internal runtime math library instead of LLVM math intrinsics.
-fuse-line-directives	Supported	Use #line in preprocessed output
-fvalidate-ast-input-files-content	Supported	Compute and store the hash of input files used to build an AST. Files with mismatching mtime's are considered valid if both contents is identical
-fveclib=<value>	Unsupported	Use the given vector functions library
-fvectorize	Unsupported	Enable the loop vectorization passes
-fverbose-asm	Supported	Generate verbose assembly output
-fvirtual-function-elimination	Supported	Enables dead virtual function elimination optimization. Requires -flto=full
-fvisibility-global-new-delete-hidden	Supported	Give global C++ operator new and delete declarations hidden visibility

Option	Support	Description
-fvisibility-inlines-hidden-static-local-var	Supported	When -fvisibility-inlines-hidden is enabled, static variables in inline C++ member functions will also be given hidden visibility by default
-fvisibility-inlines-hidden	Supported	Give inline C++ member functions hidden visibility by default
-fvisibility-ms-compat	Supported	Give global types 'default' visibility and global functions and variables 'hidden' visibility by default
-fvisibility=<value>	Supported	Set the default symbol visibility for all global declarations
-fwasm-exceptions	Unsupported	Use WebAssembly style exceptions
-fwhole-program-vtables	Unsupported	Enables whole-program vtable optimization. Requires -flto
-fwrapv	Supported	Treat signed integer overflow as two's complement
-fwritable-strings	Supported	Store string literals as writable data
-fxray-always-emit-customevents	Unsupported	Always emit __xray_customevent(...) calls even if the containing function is not always instrumented
-fxray-always-emit-typedevents	Unsupported	Always emit __xray_typedevent(...) calls even if the containing function is not always instrumented
-fxray-always-instrument=<value>	Unsupported	DEPRECATED: Filename defining the whitelist for imbuing the 'always instrument' XRay attribute.
-fxray-attr-list= <value>	Unsupported	Filename defining the list of functions/types for imbuing XRay attributes.
-fxray-ignore-loops	Unsupported	Don't instrument functions with loops unless they also meet the minimum function size
-fxray-instruction-threshold= <value>	Unsupported	Sets the minimum function size to instrument with XRay
-fxray-instrumentation-bundle= <value>	Unsupported	Select which XRay instrumentation points to emit. Options: all, none, function-entry, function-exit, function, custom. Default is 'all'. 'function' includes both 'function-entry' and 'function-exit'.
-fxray-instrument	Unsupported	Generate XRay instrumentation sleds on function entry and exit
-fxray-link-deps	Unsupported	Tells clang to add the link dependencies for XRay.
-fxray-modes= <value>	Unsupported	List of modes to link in by default into XRay instrumented binaries.
-fxray-never-instrument=<value>	Unsupported	DEPRECATED: Filename defining the whitelist for imbuing the 'never instrument' XRay attribute.
-fzvector	Supported	Enable System z vector language extension
-F <value>	Unsupported	Add directory to framework include search path
--gcc-toolchain=<value>	Supported	Use the gcc toolchain at the given directory
-gcodeview-ghash	Supported	Emit type record hashes in a .debug$H section
-gcodeview	Supported	Generate CodeView debug information
-gdwarf-2	Supported	Generate source-level debug information with dwarf version 2
-gdwarf-3	Supported	Generate source-level debug information with dwarf version 3
-gdwarf-4	Supported	Generate source-level debug information with dwarf version 4
-gdwarf-5	Supported	Generate source-level debug information with dwarf version 5
-gdwarf	Supported	Generate source-level debug information with the default dwarf version
-gembed-source	Supported	Embed source text in DWARF debug sections
-gline-directives-only	Supported	Emit debug line info directives only
-gline-tables-only	Supported	Emit debug line number tables only
-gmodules	Supported	Generate debug info with external references to clang modules or precompiled headers
-gno-embed-source	Supported	Restore the default behavior of not embedding source text in DWARF debug sections
-gno-inline-line-tables	Supported	Don't emit inline line tables

Option	Support	Description
--gpu-max-threads-per-block=<value>	Supported	Default max threads per block for kernel launch bounds for HIP
-gsplit-dwarf=<value>	Supported	Set DWARF fission mode to either 'split' or 'single'
-gz=<value>	Supported	DWARF debug sections compression type
-gz	Supported	DWARF debug sections compression type
-G <size>	Unsupported	Put objects of at most <size> bytes into small data section (MIPS / Hexagon)
-g	Supported	Generate source-level debug information
--help-hidden	Supported	Display help for hidden options
-help	Supported	Display available options
--hip-device-lib=<value>	Supported	HIP device library
--hip-link	Supported	Link clang-offload-bundler bundles for HIP
--hip-version=<value>	Supported	HIP version in the format of major.minor.patch
-H	Supported	Show header includes and nesting depth
-I-	Supported	Restrict all prior -I flags to double-quoted inclusion and remove the current directory from include path
-ibuiltininc	Supported	Enable builtin #include directories even when -nostdinc is used before or after -ibuiltininc. Using -nobuiltininc after the option disables it
-idirafter <value>	Supported	Add directory to AFTER include search path
-iframeworkwithsysroot <directory>	Unsupported	Add directory to SYSTEM framework search path, absolute paths are relative to -isysroot
-iframework <value>	Unsupported	Add directory to SYSTEM framework search path
-imacros <file>	Supported	Include macros from file before parsing
-include-pch <file>	Supported	Include precompiled header file
-include <file>	Supported	Include file before parsing
-index-header-map	Supported	Make the next included directory (-I or -F) an indexer header map
-iprefix <dir>	Supported	Set the -iwithprefix/-iwithprefixbefore prefix
-iquote <directory>	Supported	Add directory to QUOTE include search path
-isysroot <dir>	Supported	Set the system root directory (usually /)
-isystem-after <directory>	Supported	Add directory to end of the SYSTEM include search path
-isystem <directory>	Supported	Add directory to SYSTEM include search path
-ivfsoverlay <value>	Supported	Overlay the virtual filesystem described by file over the real file system
-iwithprefixbefore <dir>	Supported	Set directory to include search path with prefix
-iwithprefix <dir>	Supported	Set directory to SYSTEM include search path with prefix
-iwithsysroot <directory>	Supported	Add directory to SYSTEM include search path, absolute paths are relative to -isysroot
-I <dir>	Supported	Add directory to include search path. If there are multiple -I options, these directories are searched in the order they are given before the standard system directories are searched. If the same directory is in the SYSTEM include search paths, for example, if also specified with -isystem, the -I option will be ignored
--libomptarget-nvptx-path=<value>	Unsupported	Path to libomptarget-nvptx libraries
-L <dir>	Supported	Add directory to library search path
-mabicalls	Unsupported	Enable SVR4-style position-independent code (Mips only)
-maix-struct-return	Unsupported	Return all structs in memory (PPC32 only)
-malign-branch-boundary=<value>	Supported	Specify the boundary's size to align branches
-malign-branch=<value>	Supported	Specify types of branches to align
-malign-double	Supported	Align doubles to two words in structs (x86 only)

Option	Support	Description
-Mallocatable=<value>	Unsupported	Select semantics for assignments to allocatables (F03 or F95)
-mbackchain	Unsupported	Link stack frames through backchain on System Z
-mbranch-protection=<value>	Unsupported	Enforce targets of indirect branches and function returns
-mbranches-within-32B-boundaries	Supported	Align selected branches (fused, jcc, jmp) within 32-byte boundary
-mcmodel=medany	Unsupported	Equivalent to -mcmodel=medium, compatible with RISC-V gcc.
-mcmodel=medlow	Unsupported	Equivalent to -mcmodel=small, compatible with RISC-V gcc.
-mcmse	Unsupported	Allow use of CMSE (Armv8-M Security Extensions)
-mcode-object-v3	Supported	Legacy option to specify code object ABI V2 (-mnocode-object-v3) or V3 (-mcode-object-v3) (AMDGPU only)
-mcode-object-version=<version>	Supported	Specify code object ABI version. Defaults to 4. (AMDGPU only)
-mcrc	Unsupported	Allow use of CRC instructions (ARM/Mips only)
-mcumode	Supported	Specify CU (-mcumode) or WGP (-mno-cumode) wavefront execution mode (AMDGPU only)
-mdouble=<value>	Supported	Force double to be 32 bits or 64 bits
-MD	Supported	Write a depfile containing user and system headers
-meabi <value>	Supported	Set EABI type, e.g. 4, 5 or gnu (default depends on triple)
-membedded-data	Unsupported	Place constants in the .rodata section instead of the .sdata section even if they meet the -G <size> threshold (MIPS)
-menable-experimental-extensions	Unsupported	Enable use of experimental RISC-V extensions.
-mexec-model=<value>	Unsupported	Execution model (WebAssembly only)
-mexecute-only	Unsupported	Disallow generation of data access to code sections (ARM only)
-mextern-sdata	Unsupported	Assume that externally defined data is in the small data if it meets the -G <size> threshold (MIPS)
-mfentry	Unsupported	Insert calls to fentry at function entry (x86/SystemZ only)
-mfix-cortex-a53-835769	Unsupported	Workaround Cortex-A53 erratum 835769 (AArch64 only)
-mfp32	Unsupported	Use 32-bit floating point registers (MIPS only)
-mfp64	Unsupported	Use 64-bit floating point registers (MIPS only)
-MF <file>	Supported	Write depfile output from -MMD, -MD, -MM, or -M to <file>
-mgeneral-regs-only	Unsupported	Generate code which only uses the general purpose registers (AArch64 only)
-mglobal-merge	Supported	Enable merging of globals
-mgpopt	Unsupported	Use GP relative accesses for symbols known to be in a small data section (MIPS)
-MG	Supported	Add missing headers to depfile
-mharden-sls=<value>	Unsupported	Select straight-line speculation hardening scope
-mhvx-length=<value>	Unsupported	Set Hexagon Vector Length
-mhvx=<value>	Unsupported	Enable Hexagon Vector eXtensions
-mhvx	Unsupported	Enable Hexagon Vector eXtensions
-miamcu	Unsupported	Use Intel MCU ABI
--migrate	Unsupported	Run the migrator
-mincremental-linker-compatible	Supported	(integrated-as) Emit an object file that can be used with an incremental linker
-mindirect-jump=<value>	Unsupported	Change indirect jump instructions to inhibit speculation
-Minform=<value>	Supported	Set error level of messages to display
-mios-version-min=<value>	Unsupported	Set iOS deployment target
-MJ <value>	Unsupported	Write a compilation database entry per input
-mllvm <value>	Supported	Additional arguments to forward to LLVM's option processing
-mlocal-sdata	Unsupported	Extend the -G behavior to object local data (MIPS)

Option	Support	Description
-mlong-calls	Supported	Generate branches with extended addressability, usually via indirect jumps.
-mlong-double-128	Supported on Host only	Force long double to be 128 bits
-mlong-double-64	Supported	Force long double to be 64 bits
-mlong-double-80	Supported on Host only	Force long double to be 80 bits, padded to 128 bits for storage
-mlvi-cfi	Supported on Host only	Enable only control-flow mitigations for Load Value Injection (LVI)
-mlvi-hardening	Supported on Host only	Enable all mitigations for Load Value Injection (LVI)
-mmacosx-version-min=<value>	Unsupported	Set Mac OS X deployment target
-mmadd4	Supported	Enable the generation of 4-operand madd.s, madd.d and related instructions.
-mmark-bti-property	Unsupported	Add .note.gnu.property with BTI to assembly files (AArch64 only)
-MMD	Supported	Write a depfile containing user headers
-mmemops	Supported	Enable generation of memop instructions
-mms-bitfields	Unsupported	Set the default structure layout to be compatible with the Microsoft compiler standard
-mmsa	Unsupported	Enable MSA ASE (MIPS only)
-mmt	Unsupported	Enable MT ASE (MIPS only)
-MM	Supported	Like -MMD, but also implies -E and writes to stdout by default
-mno-abicalls	Unsupported	Disable SVR4-style position-independent code (Mips only)
-mno-crc	Unsupported	Disallow use of CRC instructions (Mips only)
-mno-embedded-data	Unsupported	Do not place constants in the .rodata section instead of the .sdata if they meet the -G <size> threshold (MIPS)
-mno-execute-only	Unsupported	Allow generation of data access to code sections (ARM only)
-mno-extern-sdata	Unsupported	Do not assume that externally defined data is in the small data if it meets the -G <size> threshold (MIPS)
-mno-fix-cortex-a53-835769	Unsupported	Don't workaround Cortex-A53 erratum 835769 (AArch64 only)
-mno-global-merge	Supported	Disable merging of globals
-mno-gpopt	Unsupported	Do not use GP relative accesses for symbols known to be in a small data section (MIPS)
-mno-hvx	Unsupported	Disable Hexagon Vector eXtensions
-mno-implicit-float	Supported	Don't generate implicit floating point instructions
-mno-incremental-linker-compatible	Supported	(integrated-as) Emit an object file which cannot be used with an incremental linker
-mno-local-sdata	Unsupported	Do not extend the -G behaviour to object local data (MIPS)
-mno-long-calls	Supported	Restore the default behaviour of not generating long calls
-mno-lvi-cfi	Supported on Host only	Disable control-flow mitigations for Load Value Injection (LVI)
-mno-lvi-hardening	Supported on Host only	Disable mitigations for Load Value Injection (LVI)
-mno-madd4	Supported	Disable the generation of 4-operand madd.s, madd.d and related instructions.
-mno-memops	Supported	Disable generation of memop instructions
-mno-movt	Supported	Disallow use of movt/movw pairs (ARM only)
-mno-ms-bitfields	Supported	Do not set the default structure layout to be compatible with the Microsoft compiler standard
-mno-msa	Unsupported	Disable MSA ASE (MIPS only)

Option	Support	Description
-mno-mt	Unsupported	Disable MT ASE (MIPS only)
-mno-neg-immediates	Supported	Disallow converting instructions with negative immediates to their negation or inversion.
-mno-nvj	Supported	Disable generation of new-value jumps
-mno-nvs	Supported	Disable generation of new-value stores
-mno-outline	Unsupported	Disable function outlining (AArch64 only)
-mno-packets	Supported	Disable generation of instruction packets
-mno-relax	Supported	Disable linker relaxation
-mno-restrict-it	Unsupported	Allow generation of deprecated IT blocks for ARMv8. It is off by default for ARMv8 Thumb mode
-mno-save-restore	Unsupported	Disable using library calls for save and restore
-mno-seses	Unsupported	Disable speculative execution side effect suppression (SESES)
-mno-stack-arg-probe	Supported	Disable stack probes which are enabled by default
-mno-tls-direct-seg-refs	Supported	Disable direct TLS access through segment registers
-mno-unaligned-access	Unsupported	Force all memory accesses to be aligned (AArch32/AArch64 only)
-mno-wavefrontsize64	Supported	Specify wavefront size 32 mode (AMDGPU only)
-mnocrc	Unsupported	Disallow use of CRC instructions (ARM only)
-mnop-mcount	Supported	Generate mcount/__fentry__ calls as nops. To activate they need to be patched in.
-mnvj	Supported	Enable generation of new-value jumps
-mnvs	Supported	Enable generation of new-value stores
-module-dependency-dir <value>	Unsupported	Directory to dump module dependencies to
-module-file-info	Unsupported	Provide information about a particular module file
-momit-leaf-frame-pointer	Supported	Omit frame pointer setup for leaf functions
-moutline	Unsupported	Enable function outlining (AArch64 only)
-mpacked-stack	Unsupported	Use packed stack layout (SystemZ only).
-mpackets	Supported	Enable generation of instruction packets
-mpad-max-prefix-size=<value>	Supported	Specify maximum number of prefixes to use for padding
-mpie-copy-relocations	Supported	Use copy relocations support for PIE builds
-mprefer-vector-width=<value>	Unsupported	Specifies preferred vector width for auto-vectorization. Defaults to 'none' which allows target specific decisions.
-MP	Supported	Create phony target for each dependency (other than main file)
-mqdsp6-compat	Unsupported	Enable hexagon-qdsp6 backward compatibility
-MQ <value>	Supported	Specify name of main file output to quote in depfile
-mrecord-mcount	Supported	Generate a __mcount_loc section entry for each __fentry__ call.
-mrelax-all	Supported	(integrated-as) Relax all machine instructions
-mrelax	Supported	Enable linker relaxation
-mrestrict-it	Unsupported	Disallow generation of deprecated IT blocks for ARMv8. It is on by default for ARMv8 Thumb mode.
-mrtd	Unsupported	Make StdCall calling convention the default
-msave-restore	Unsupported	Enable using library calls for save and restore
-mseses	Unsupported	Enable speculative execution side effect suppression (SESES). Includes LVI control flow integrity mitigations
-msign-return-address=<value>	Unsupported	Select return address signing scope
-msmall-data-limit=<value>	Supported	Put global and static data smaller than the limit into a special section
-msoft-float	Supported	Use software floating point

Option	Support	Description
-msram-ecc	Supported	Legacy option to specify SRAM ECC mode (AMDGPU only). Should use --offload-arch with :sramecc+ instead
-mstack-alignment=<value>	Unsupported	Set the stack alignment
-mstack-arg-probe	Unsupported	Enable stack probes
-mstack-probe-size=<value>	Unsupported	Set the stack probe size
-mstackrealign	Unsupported	Force realign the stack at entry to every function
-msve-vector-bits=<value>	Unsupported	Specify the size in bits of an SVE vector register. Defaults to the vector length agnostic value of "scalable". (AArch64 only)
-msvr4-struct-return	Unsupported	Return small structs in registers (PPC32 only)
-mthread-model <value>	Supported	The thread model to use, e.g. posix, single (posix by default)
-mtls-direct-seg-refs	Supported	Enable direct TLS access through segment registers (default)
-mtls-size=<value>	Unsupported	Specify bit size of immediate TLS offsets (AArch64 ELF only): 12 (for 4KB) | 24 (for 16MB, default) | 32 (for 4GB) | 48 (for 256TB, needs -mcmodel=large)
-mtp=<value>	Unsupported	Thread pointer access method (AArch32/AArch64 only)
-mtune=<value>	Supported on Host only	Only supported on X86. Otherwise accepted for compatibility with GCC.
-MT <value>	Unsupported	Specify name of main file output in depfile
-munaligned-access	Unsupported	Allow memory accesses to be unaligned (AArch32/AArch64 only)
-MV	Supported	Use NMake/Jom format for the depfile
-mwavefrontsize64	Supported	Specify wavefront size 64 mode (AMDGPU only)
-mxnack	Supported	Legacy option to specify XNACK mode (AMDGPU only). Should use --offload-arch with :xnack+ instead
-M	Supported	Like -MD, but also implies -E and writes to stdout by default
--no-cuda-include-ptx=<value>	Supported	Do not include PTX for the following GPU architecture (e.g. sm_35) or 'all'. May be specified more than once.
--no-cuda-version-check	Supported	Don't error out if the detected version of the CUDA install is too low for the requested CUDA gpu architecture.
-no-flang-libs	Supported	Do not link against Flang libraries
--no-offload-arch=<value>	Supported	Remove CUDA/HIP offloading device architecture (e.g. sm_35, gfx906) from the list of devices to compile for. 'all' resets the list to its default value.
--no-system-header-prefix=<prefix>	Supported	Treat all #include paths starting with <prefix> as not including a system header.
-nobuiltininc	Supported	Disable builtin #include directories
-nogpuinc	Supported	Do not add CUDA/HIP include paths and include default CUDA/HIP wrapper header files
-nogpulib	Supported	Do not link device library for CUDA/HIP device compilation
-nostdinc++	Unsupported	Disable standard #include directories for the C++ standard library
-ObjC++	Unsupported	Treat source input files as Objective-C++ inputs
-objcmt-atomic-property	Unsupported	Make migration to 'atomic' properties
-objcmt-migrate-all	Unsupported	Enable migration to modern ObjC
-objcmt-migrate-annotation	Unsupported	Enable migration to property and method annotations
-objcmt-migrate-designated-init	Unsupported	Enable migration to infer NS_DESIGNATED_INITIALIZER for initializer methods
-objcmt-migrate-instancetype	Unsupported	Enable migration to infer instancetype for method result type
-objcmt-migrate-literals	Unsupported	Enable migration to modern ObjC literals
-objcmt-migrate-ns-macros	Unsupported	Enable migration to NS_ENUM/NS_OPTIONS macros

Option	Support	Description
-objcmt-migrate-property-dot-syntax	Unsupported	Enable migration of setter/getter messages to property-dot syntax
-objcmt-migrate-property	Unsupported	Enable migration to modern ObjC property
-objcmt-migrate-protocol-conformance	Unsupported	Enable migration to add protocol conformance on classes
-objcmt-migrate-readonly-property	Unsupported	Enable migration to modern ObjC readonly property
-objcmt-migrate-readwrite-property	Unsupported	Enable migration to modern ObjC readwrite property
-objcmt-migrate-subscripting	Unsupported	Enable migration to modern ObjC subscripting
-objcmt-ns-nonatomic-iosonly	Unsupported	Enable migration to use NS_NONATOMIC_IOSONLY macro for setting property's 'atomic' attribute
-objcmt-returns-innerpointer-property	Unsupported	Enable migration to annotate property with NS_RETURNS_INNER_POINTER
-objcmt-whitelist-dir-path=<value>	Unsupported	Only modify files with a filename contained in the provided directory path
-ObjC	Unsupported	Treat source input files as Objective-C inputs
--offload-arch=<value>	Supported	CUDA offloading device architecture (e.g. sm_35), or HIP offloading target ID in the form of a device architecture followed by target ID features delimited by a colon. Each target ID feature is a pre-defined string followed by a plus or minus sign (e.g. gfx908:xnack+:sramecc-). May be specified more than once.
-o <file>	Supported	Write output to <file>
-parallel-jobs=<value>	Supported	Number of parallel jobs
-pg	Supported	Enable mcount instrumentation
-pipe	Supported	Use pipes between commands, when possible
--precompile	Supported	Only precompile the input
-print-effective-triple	Supported	Print the effective target triple
-print-file-name=<file>	Supported	Print the full library path of <file>
-print-ivar-layout	Unsupported	Enable Objective-C Ivar layout bitmap print trace
-print-libgcc-file-name	Supported	Print the library path for the currently used compiler runtime library ("libgcc.a" or "libclang_rt.builtins.*.a")
-print-prog-name=<name>	Supported	Print the full program path of <name>
-print-resource-dir	Supported	Print the resource directory pathname
-print-search-dirs	Supported	Print the paths used for finding libraries and programs
-print-supported-cpus	Supported	Print supported cpu models for the given target (if target is not specified, it will print the supported cpus for the default target)
-print-target-triple	Supported	Print the normalized target triple
-print-targets	Supported	Print the registered targets
-pthread	Supported	Support POSIX threads in generated code
--ptxas-path=<value>	Unsupported	Path to ptxas (used for compiling CUDA code)
-P	Supported	Disable linemarker output in -E mode
-Qn	Supported	Do not emit metadata containing compiler name and version
-Qunused-arguments	Supported	Don't emit warning for unused driver arguments
-Qy	Supported	Emit metadata containing compiler name and version
-relocatable-pch	Supported	Whether to build a relocatable precompiled header
-rewrite-legacy-objc	Unsupported	Rewrite Legacy Objective-C source to C++
-rewrite-objc	Unsupported	Rewrite Objective-C source to C++
--rocm-device-lib-path=<value>	Supported	ROCm device library path. Alternative to rocm-path.

Option	Support	Description
--rocm-path=<value>	Supported	ROCm installation path, used for finding and automatically linking required bitcode libraries.
-Rpass-analysis=<value>	Supported	Report transformation analysis from optimization passes whose name matches the given POSIX regular expression
-Rpass-missed=<value>	Supported	Report missed transformations by optimization passes whose name matches the given POSIX regular expression
-Rpass=<value>	Supported	Report transformations performed by optimization passes whose name matches the given POSIX regular expression
-rtlib=<value>	Unsupported	Compiler runtime library to use
-R<remark>	Unsupported	Enable the specified remark
-save-stats=<value>	Supported	Save llvm statistics.
-save-stats	Supported	Save llvm statistics.
-save-temps=<value>	Supported	Save intermediate compilation results.
-save-temps	Supported	Save intermediate compilation results
-serialize-diagnostics <value>	Supported	Serialize compiler diagnostics to a file
-shared-libsan	Unsupported	Dynamically link the sanitizer runtime
-static-flang-libs	Supported	Link using static Flang libraries
-static-libsan	Unsupported	Statically link the sanitizer runtime
-static-openmp	Supported	Use the static host OpenMP runtime while linking.
-std=<value>	Supported	Language standard to compile for
-stdlib++-isystem <directory>	Supported	Use directory as the C++ standard library include path
-stdlib=<value>	Supported	C++ standard library to use
-sycl-std=<value>	Unsupported	SYCL language standard to compile for.
--system-header-prefix=<prefix>	Supported	Treat all #include paths starting with <prefix> as including a system header.
-S	Supported	Only run preprocess and compilation steps
--target=<value>	Supported	Generate code for the given target
-Tbss <addr>	Supported	Set starting address of BSS to <addr>
-Tdata <addr>	Supported	Set starting address of DATA to <addr>
-time	Supported	Time individual commands
-traditional-cpp	Unsupported	Enable some traditional CPP emulation
-trigraphs	Supported	Process trigraph sequences
-Ttext <addr>	Supported	Set starting address of TEXT to <addr>
-T <script>	Unsupported	Specify <script> as linker script
-undef	Supported	undef all system defines
-unwindlib=<value>	Supported	Unwind library to use
-U <macro>	Supported	Undefine macro <macro>
--verify-debug-info	Supported	Verify the binary representation of debug output
-verify-pch	Unsupported	Load and verify that a pre-compiled header file is not stale
--version	Supported	Print version information
-v	Supported	Show commands to run and use verbose output
-Wa,<arg>	Supported	Pass the comma separated arguments in <arg> to the assembler
-Wdeprecated	Supported	Enable warnings for deprecated constructs and define __DEPRECATED
-Wl,<arg>	Supported	Pass the comma separated arguments in <arg> to the linker
-working-directory <value>	Supported	Resolve file paths relative to the specified directory
-Wp,<arg>	Supported	Pass the comma separated arguments in <arg> to the preprocessor
-W<warning>	Supported	Enable the specified warning
-w	Supported	Suppress all warnings
-Xanalyzer <arg>	Supported	Pass <arg> to the static analyzer

Option	Support	Description
-Xarch_device <arg>	Supported	Pass <arg> to the CUDA/HIP device compilation
-Xarch_host <arg>	Supported	Pass <arg> to the CUDA/HIP host compilation
-Xassembler <arg>	Supported	Pass <arg> to the assembler
-Xclang <arg>	Supported	Pass <arg> to the clang compiler
-Xcuda-fatbinary <arg>	Supported	Pass <arg> to fatbinary invocation
-Xcuda-ptxas <arg>	Supported	Pass <arg> to the ptxas assembler
-Xlinker <arg>	Supported	Pass <arg> to the linker
-Xopenmp-target=<triple> <arg>	Supported	Pass <arg> to the target offloading toolchain identified by <triple>.
-Xopenmp-target <arg>	Supported	Pass <arg> to the target offloading toolchain.
-Xpreprocessor <arg>	Supported	Pass <arg> to the preprocessor
-x <language>	Supported	Treat subsequent input files as having type <language>
-z <arg>	Supported	Pass -z <arg> to the linker

Chapter 7 Appendix C

7.1 HIP FAQ

You can access the HIP FAQ at:

https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-FAQ.html#hip-faq