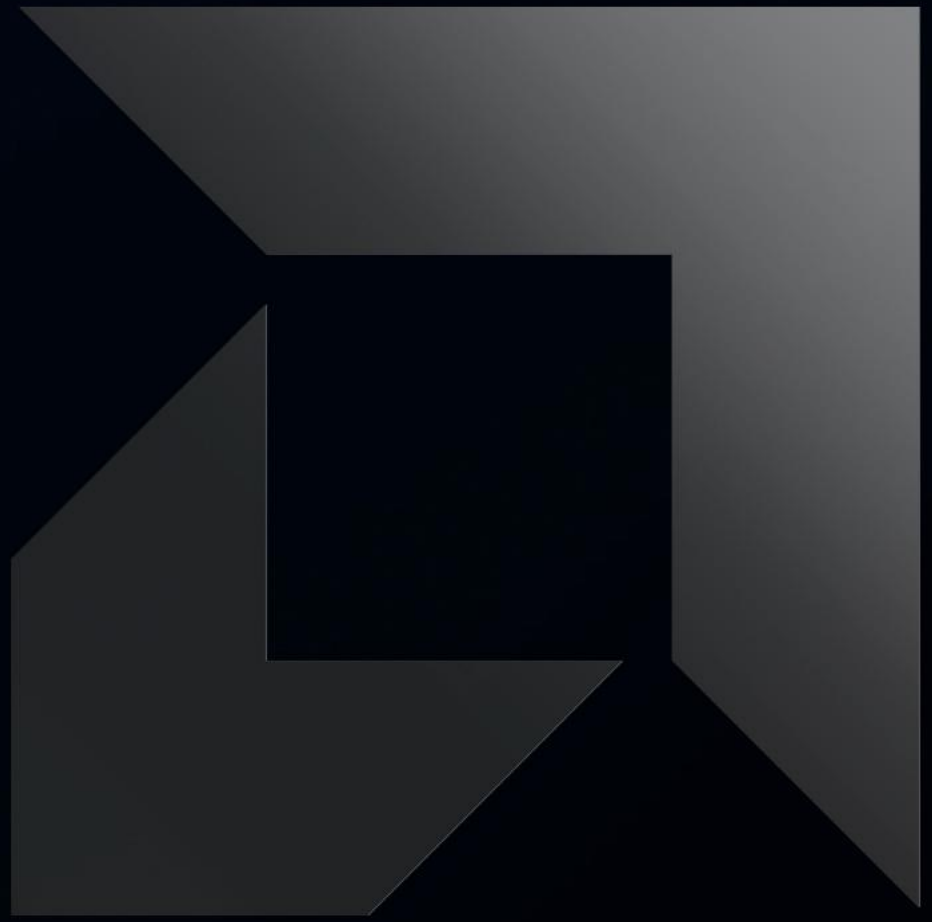


HIP: Porting from CUDA

UCL

DiRAC
TRAINING ACADEMY

AMD
together we advance_

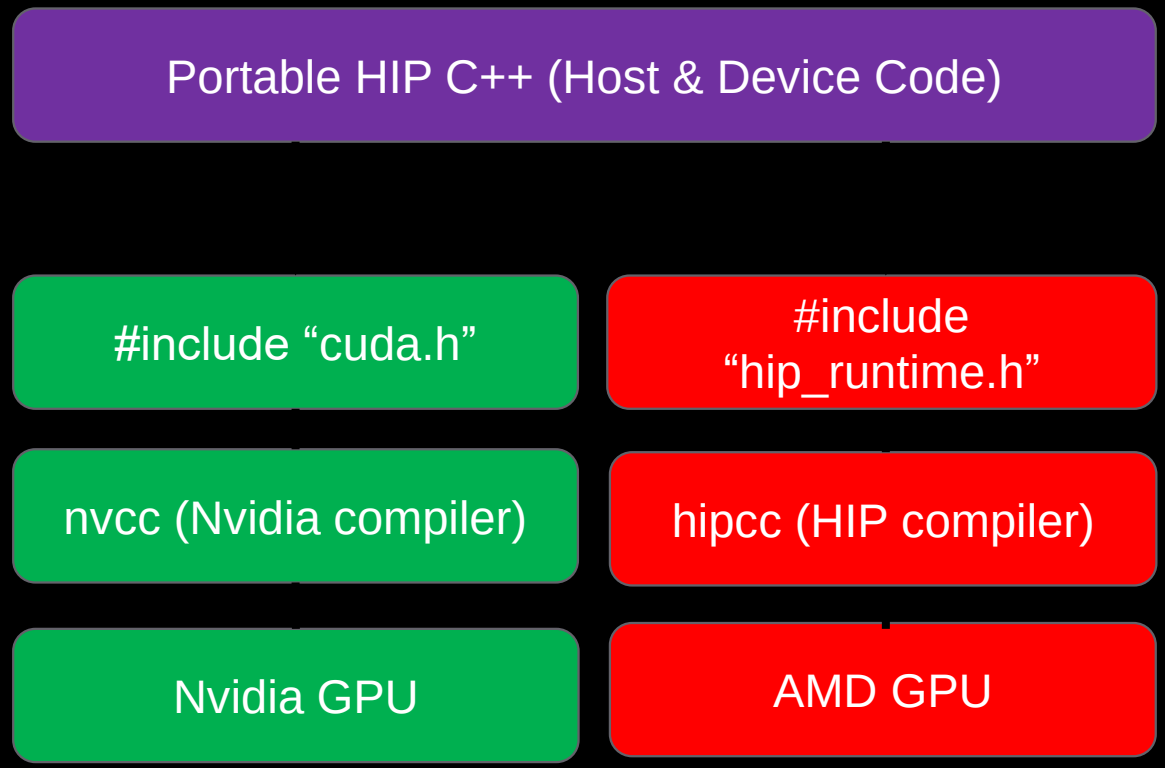
What is covered in the HIP:Porting from CUDA Presentation

- What is HIP.
- Ways to convert CUDA code to HIP code (process that will be referred to as *hipification*), to obtain a code that runs on AMD and NVIDIA accelerators.
- Hipification tools that come with the ROCm stack (`hipify-perl` and `hipify-clang`). These produce a new source code that has CUDA calls replaced by HIP calls.
- Some common issues that are associated with the porting of code from CUDA to HIP
- Header approach that allows to replace CUDA calls with HIP calls on the fly (hence called *hipifly*). This approach does not require users to change their CUDA code which ideally can be kept as is

What is HIP?

AMD's **H**eterogeneous-compute **I**nterface for **P**ortability, or **HIP**, is a C++ runtime API and kernel language that allows developers to create portable applications that can run on AMD accelerators as well as CUDA accelerators

- **Open-source**
- Syntactically similar to CUDA. Most CUDA API calls can be converted in place from CUDA to HIP.
- Supports a strong subset of CUDA runtime functionality



Porting from CUDA to HIP: no one size fits all approach

The porting experience from CUDA to HIP depends on various factors, including the complexity of the code and the amount of specific hardware optimization that is included.

Self contained GPU code

- Automatic conversion tools (hipify)
  - Targeting the AMD HIP Interoperability layer
  - Alternatively, converting to ROCm native AMD language
- Header file interception layer (hipiFLY)
  - Fast way to get running code
  - Leaves your code as CUDA source
  - May break on creative use of previous device code

More complex accelerator layers

- Need combination of automatic conversion and manual rewrite
  - Abstraction layer needs to be adapted to use HIP API
  - Can later use HIP to target both ROCm and CUDA
  - Longer time to running code

Code Conversion Tools

EXTEND YOUR APPLICATION PLATFORM SUPPORT BY CONVERTING CUDA® CODE

Single source

Maintain portability

Maintain performance

Hipify-perl

- Easiest to use; point at a directory and it will hipify CUDA code
- Very simple string replacement technique; may require manual post-processing
- It replaces cuda with hip, conceptually similar to `sed -e 's/cuda/hip/g'`, (e.g., cudaMemcpy becomes hipMemcpy)
- Recommended for quick scans of projects
- It will not translate if it does not recognize a CUDA call and it will report it

Hipify-clang

- More robust translation of the code
- Generates warnings and assistance for additional analysis
- High quality translation, particularly for cases where the user is familiar with the make system

Hipify-perl

It is located in `/opt/rocm/bin`

- Command line tool: `hipify-perl foo.cu > new_foo.cpp`

- Compile: **`hipcc new_foo.cpp`**

How does this this work in practice?

- Hipify source code
- Check it in to your favorite version control
- Try to build
- Manually work on the rest

Hipify-clang

It is located in `/opt/rocm/bin`

Build from source (needs clang compiler)

- hipify-clang has unit tests using LLVM™ lit/FileCheck (44 tests)

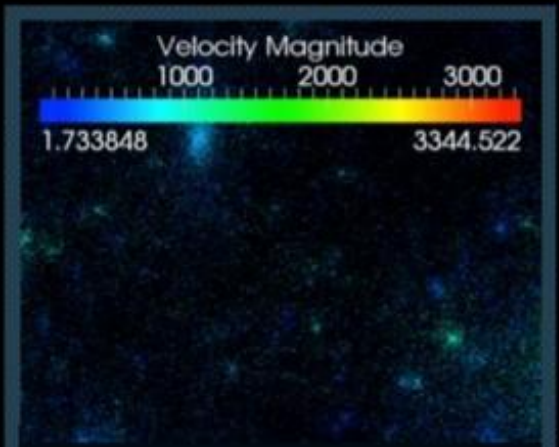
Hipification requires same headers that would be **needed to compile it with clang**:

- `./hipify-clang foo.cu -I /usr/local/cuda-8.0/samples/common/inc`

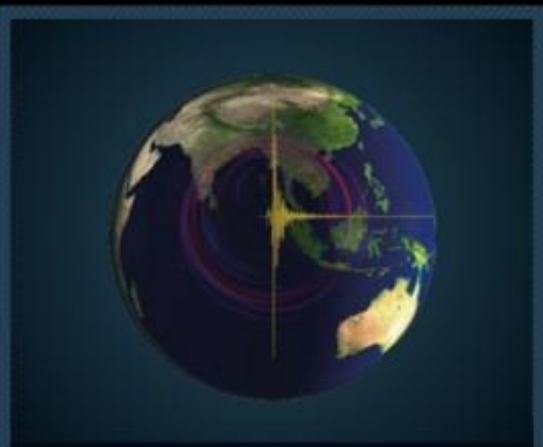
`More info at: https://github.com/ROCm/HIPIFY/blob/master/README.md`

Can be used to perform build-time conversion if project can be automatically converted

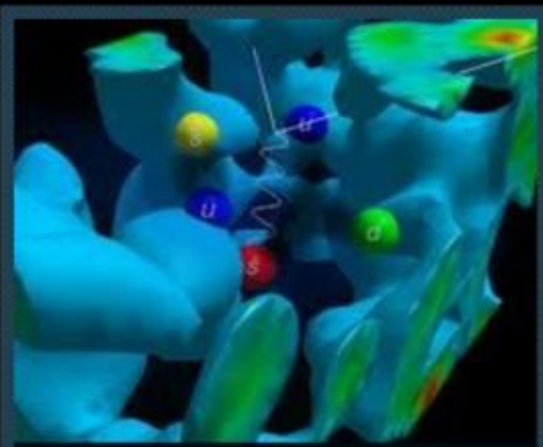
CUDA to HIP experiences: porting duration depends on the code



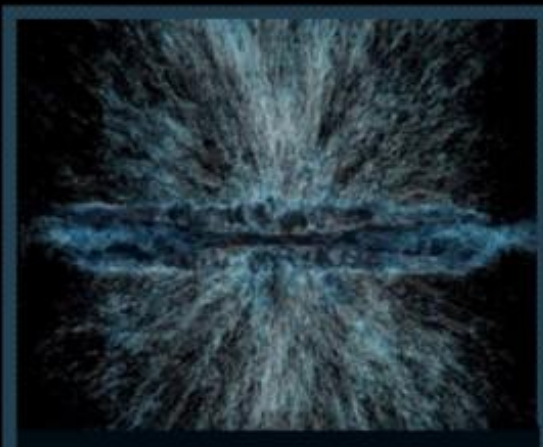
Ported in an Afternoon
HACC
Cosmology



Ported in a Single Day
SPECFEM3D
Seismology



Ported in 21 Days
QUDA
Quantum Physics



Ported in a Couple of Days
CHOLLA
Astrophysics

NAMD Scalable Molecular Dynamics · LAMMPS · kokkos · Nekbone · GROMACS FAST. FLEXIBLE. FREE. · MILC · Chroma · TensorFlow

PYTORCH · GridTools · ALTAIR · SIRIUS · AMBER · PIConGPU · CP2K · LSMS

UCL · DiRAC TRAINING ACADEMY · AMD together we advance_

Other Hipify tools

Individual file tools (already discussed)

- hipify-perl
- hipify-clang

Recursive directory tools (also in `/opt/rocm/bin`)

- hipconvertinplace.sh
- hipconvertinplace-perl.sh
- hipexamine.sh
- hipexamine-perl.sh

The perl® scripts are a set and the shell/clang tools are a set. The directory-based tools basically call the base tools, hipify-perl and hipify-clang, respectively.

For example:
hipifyexamine-perl.sh recursively runs hipify-perl with the -no-output -print-stats options (-examine option is a shorthand for -no-output -print-stats options).

For example, source code for `hipconvertinplace-perl.sh`

```
#!/bin/bash

#usage : hipconvertinplace-perl.sh DIRNAME [hipify-perl options]

#hipify "inplace" all code files in specified directory.
# This can be quite handy when dealing with an existing CUDA code base since the script
# preserves the existing directory structure.

#  For each code file, this script will:
#    - If ".prehip file does not exist, copy the original code to a new file with extension ".prehip". Then hipify the code file.
#    - If ".prehip" file exists, this is used as input to hipify.
# (this is useful for testing improvements to the hipify-perl toolset).


SCRIPT_DIR=`dirname $0`
PRIV_SCRIPT_DIR="$SCRIPT_DIR/../libexec/hipify"
SEARCH_DIR=$1
shift
$SCRIPT_DIR/hipify-perl -inplace -print-stats "$@" `$PRIV_SCRIPT_DIR/findcode.sh $SEARCH_DIR`
```

Calls the findcode.sh script which recursively looks for files with the extensions seen below.

```
#!/bin/bash

SEARCH_DIRS=$@

find $SEARCH_DIRS -name '*.cu' -o -name '*.CU'
find $SEARCH_DIRS -name '*.cpp' -o -name '*.cxx' -o -name '*.c' -o -name '*.cc'
find $SEARCH_DIRS -name '*.CPP' -o -name '*.CXX' -o -name '*.C' -o -name '*.CC'
find $SEARCH_DIRS -name '*.cuh' -o -name '*.CUH'
find $SEARCH_DIRS -name '*.h' -o -name '*.hpp' -o -name '*.inc' -o -name '*.inl' -o -name '*.hxx' -o -name '*.hdl'
find $SEARCH_DIRS -name '*.H' -o -name '*.HPP' -o -name '*.INC' -o -name '*.INL' -o -name '*.HXX' -o -name '*.HDL'
```

Things to look out for

- Hipify tools are not running your application, or checking correctness
- Code relying on specific Nvidia hardware aspects (e.g., warp size == 32) may need attention after conversion (grep for "32" *just in case*). Use `#define WARPSIZE size`.
- Certain CUDA functions may not have a correspondent HIP function (e.g., __shfl_down_sync – hint: use __shfl_down instead)
- Hipify-clang modifies the code in place, so better to make a copy of the code before running the tool
- Hipifying can't handle inline PTX assembly or CUDA intrinsics
  - Can either use inline GCN ISA, or convert it to HIP
- None of the tools convert your build system script such as CMAKE or whatever else you use. The user is responsible to find the appropriate flags and paths to build the new converted HIP code.

Notes

- Hipify-perl and hipify-clang can both convert library calls (i.e. cuBLAS becomes hipBLAS)
- CMake from version 3.21 can be used to automatically set up basic compilation flags by using enable_language(HIP), supports CMAKE_HIP_ARCHITECTURES for setting devices to build for

HIPIFLY: Intercept API method to choose GPU backend

- Enable running existing code on different backends with single header
- Can change between targeting CUDA and ROCm in one place
- Only works if no difference between API calls
- Existing code cannot use any CUDA specific hard coded values
- Performance needs to be evaluated on a case-by-case basis

```
#ifndef CUDA_TO_HIP_H
#define CUDA_TO_HIP_H

#include <hip/hip_runtime.h>

#define WARPSIZE 64
static constexpr int maxWarpsPerBlock = 1024/WARPSIZE;

#define CUFFT_D2Z HIPFFT_D2Z
#define CUFFT_FORWARD HIPFFT_FORWARD
#define CUFFT_INVERSE HIPFFT_BACKWARD
#define CUFFT_Z2D HIPFFT_Z2D
#define CUFFT_Z2Z HIPFFT_Z2Z

#define cudaDeviceSynchronize hipDeviceSynchronize
#define cudaError hipError_t
#define cudaError_t hipError_t
#define cudaErrorInsufficientDriver hipErrorInsufficientDriver
#define cudaErrorNoDevice hipErrorNoDevice
#define cudaEvent_t hipEvent_t
#define cudaEventCreate hipEventCreate
#define cudaEventElapsedTime hipEventElapsedTime
#define cudaEventRecord hipEventRecord
#define cudaEventSynchronize hipEventSynchronize
#define cudaFree hipFree
#define cudaFreeHost hipHostFree
#define cudaGetDevice hipGetDevice
#define cudaGetDeviceCount hipGetDeviceCount
#define cudaGetErrorString hipGetErrorString
#define cudaGetLastError hipGetLastError
#define cudaHostAlloc hipHostMalloc
#define cudaHostAllocDefault hipHostMallocDefault
#define cudaMalloc hipMalloc
#define cudaMemcpy hipMemcpy
#define cudaMemcpyAsync hipMemcpyAsync
#define cudaMemcpyDeviceToHost hipMemcpyDeviceToHost
#define cudaMemcpyDeviceToDevice hipMemcpyDeviceToDevice
#define cudaMemcpyHostToDevice hipMemcpyHostToDevice
#define cudaMemGetInfo hipMemGetInfo
#define cudaMemset hipMemset
#define cudaReadModeElementType hipReadModeElementType
```

Link to the header file:
https://github.com/amd/HPCTrainingExamples/blob/main/hipifly/vector_add/src/cuda_to_hip.h

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.