



# Porting code to HIP

**Presenter: Giacomo Capodaglio**  
**AMD @ HLRS**  
**May 6th, 2025**

**AMD**   
together we advance\_

# No one size fits all approach

## Self contained GPU code

- ▲ Automatic conversion tools (hipify)
- ▲ Header file interception layer (hipiFLY)
  - Fast way to get running code
  - 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, `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



Velocity Magnitude  
1000 2000 3000  
1.733848 3344.522

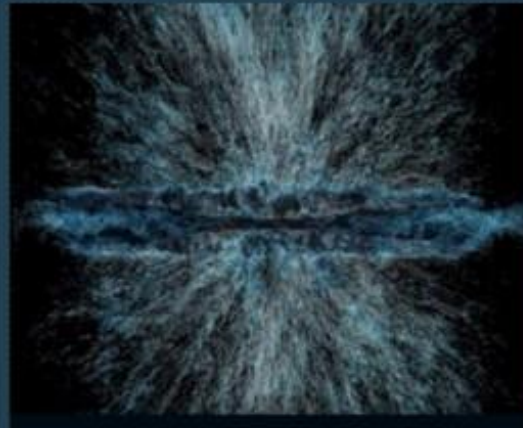
Ported in an Afternoon  
HACC  
Cosmology



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

# 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<sup>®</sup> 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).

## Source code for hipconvertinplace-perl.sh

```

1 #!/bin/bash
2
3 #usage : hipconvertinplace-perl.sh DIRNAME [hipify-perl options]
4
5 #hipify "inplace" all code files in specified directory.
6 # This can be quite handy when dealing with an existing CUDA code base since the script
7 # preserves the existing directory structure.
8
9 # For each code file, this script will:
10 # - If ".prehip" file does not exist, copy the original code to a new file with extension ".prehip". Then hipify the code file.
11 # - If ".prehip" file exists, this is used as input to hipify.
12 # (this is useful for testing improvements to the hipify-perl toolset).
13
14
15 SCRIPT_DIR=`dirname $0`
16 PRIV_SCRIPT_DIR="$SCRIPT_DIR/../../libexec/hipify"
17 SEARCH_DIR=$1
18 shift
19 $SCRIPT_DIR/hipify-perl -inplace -print-stats "$@" `"$PRIV_SCRIPT_DIR/findcode.sh $SEARCH_DIR`

```

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

```

1 #!/bin/bash
2
3 SEARCH_DIRS=$@
4
5 find $SEARCH_DIRS -name '*.cu' -o -name '*.CU'
6 find $SEARCH_DIRS -name '*.cpp' -o -name '*.cxx' -o -name '*.c' -o -name '*.cc'
7 find $SEARCH_DIRS -name '*.CPP' -o -name '*.CXX' -o -name '*.C' -o -name '*.CC'
8 find $SEARCH_DIRS -name '*.cuh' -o -name '*.CUH'
9 find $SEARCH_DIRS -name '*.h' -o -name '*.hpp' -o -name '*.inc' -o -name '*.inl' -o -name '*.hxx' -o -name '*.hdl'
10 find $SEARCH_DIRS -name '*.H' -o -name '*.HPP' -o -name '*.INC' -o -name '*.INL' -o -name '*.HXX' -o -name '*.HDL'

```

# Gotchas

- 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 functions may not have a correspondent hip version (e.g., `__shfl_down_sync` – hint: use `__shfl_down` instead)
- 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

# HIIFLY: 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



Link to the header file:

[https://github.com/amd/HPCTrainingExamples/blob/main/hipifly/vector\\_add/src/cuda\\_to\\_hip.h](https://github.com/amd/HPCTrainingExamples/blob/main/hipifly/vector_add/src/cuda_to_hip.h)

```
#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
```

# Exploiting the power of HIP: portable build systems

- One of the attractive features of HIP is that it can run on both AMD and Nvidia GPUs
- The HIP language has been developed with this in mind
  - Select ROCm and it will run on AMD GPUs
  - Select CUDA and it will run on Nvidia GPUs
- But it can be difficult to support this with a portable build system that switches between these two
- We'll demonstrate two of the most common build systems that can support portable builds
  - make
  - cmake

# Portable build systems – Makefile

```
EXECUTABLE = ./vectoradd
all: $(EXECUTABLE) test
.PHONY: test
```

```
OBJECTS = vectoradd.o
```

```
CXXFLAGS = -g -O2 -DNDEBUG -fPIC
```

```
HIPCC_FLAGS = -O2 -g -DNDEBUG
```

```
HIP_PLATFORM ?= amd
```

← Setting default device compiler

```
ifeq ($(HIP_PLATFORM), nvidia)
```

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

```
    HIPCC_FLAGS += -x cu -I${HIP_PATH}/include/
```

```
endif
```

```
ifeq ($(HIP_PLATFORM), amd)
```

```
    HIPCC_FLAGS += -x hip -munsafe-fp-atomics
```

```
endif
```

← Setting compile flags for different GPUs

← Pattern rule for HIP source

```
%.o: %.hip
```

```
    hipcc $(HIPCC_FLAGS) -c $^ -o $@
```

```
$(EXECUTABLE): $(OBJECTS)
```

```
    hipcc $< $(LDFLAGS) -o $@
```

```
test: $(EXECUTABLE)
```

```
    $(EXECUTABLE)
```

```
clean:
```

```
    rm -f $(EXECUTABLE) $(OBJECTS) build
```

# Using a portable Makefile

- For ROCm

```
module load rocm  
export CXX=${ROCM_PATH}/llvm/bin/clang++
```

To build and run:

```
make vectoradd  
./vectoradd
```

- For CUDA

```
module load rocm ← We still need HIP for the portability layer  
module load cuda
```

To build and run:

```
HIP_PLATFORM=nvidia make vectoradd ← Overriding default to compile with nvidia  
./vectoradd
```

# Portable Build Systems – CMakeLists.txt (1 of 3)

```
cmake_minimum_required(VERSION 3.21 FATAL_ERROR)
project(Vectoradd LANGUAGES CXX)

set (CMAKE_CXX_STANDARD 14)

if (NOT CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE RelWithDebInfo)
endif(NOT CMAKE_BUILD_TYPE)

string(REPLACE -O2 -O3 CMAKE_CXX_FLAGS_RELWITHDEBINFO ${CMAKE_CXX_FLAGS_RELWITHDEBINFO})

if (NOT CMAKE_GPU_RUNTIME)
    set(GPU_RUNTIME "ROCM" CACHE STRING "Switches between ROCM and CUDA")
else (CMAKE_GPU_RUNTIME)
    set(GPU_RUNTIME "${CMAKE_GPU_RUNTIME}" CACHE STRING "Switches between ROCM and CUDA")
endif (NOT CMAKE_GPU_RUNTIME)
```

} Setting  
GPU\_RUNTIME

# Portable Build Systems – CMakeLists.txt (2 of 3)

```
# Should only be ROCM or CUDA, but allowing HIP because it is the currently built-in option
# Select with e.g., -DGPU_RUNTIME=ROCM
set(GPU_RUNTIMES "ROCM" "CUDA" "HIP")
if(NOT "${GPU_RUNTIME}" IN_LIST GPU_RUNTIMES)
    set(ERROR_MESSAGE
        "GPU_RUNTIME is set to \"${GPU_RUNTIME}\".\nGPU_RUNTIME must be either HIP, ROCM, or CUDA.")
    message(FATAL_ERROR ${ERROR_MESSAGE})
endif()

# GPU_RUNTIME should really be ROCM for AMD GPUs, so manually resetting to HIP if ROCM is selected
if (${GPU_RUNTIME} MATCHES "ROCM")
    set(GPU_RUNTIME "HIP")
endif (${GPU_RUNTIME} MATCHES "ROCM")
set_property(CACHE GPU_RUNTIME PROPERTY STRINGS ${GPU_RUNTIMES})

enable_language(${GPU_RUNTIME}) ← Enabling either CUDA or HIP (ROCM)
set(CMAKE_${GPU_RUNTIME}_EXTENSIONS OFF)
set(CMAKE_${GPU_RUNTIME}_STANDARD_REQUIRED ON)
```

# Portable Build Systems – CMakeLists.txt (3 of 3)

```
set(VECTORADD_CXX_SRCS "")
set(VECTORADD_HIP_SRCS vectoradd.hip)

add_executable(vectoradd ${VECTORADD_CXX_SRCS} ${VECTORADD_HIP_SRCS} )
```

```
set(ROCMCC_FLAGS "${ROCMCC_FLAGS} -munsafe-fp-atomics")
set(CUDACC_FLAGS "${CUDACC_FLAGS} ")
```

```
if (${GPU_RUNTIME} MATCHES "HIP")
    set(HIPCC_FLAGS "${ROCMCC_FLAGS}")
else (${GPU_RUNTIME} MATCHES "CUDA")
    set(HIPCC_FLAGS "${CUDACC_FLAGS}")
endif (${GPU_RUNTIME} MATCHES "HIP")
```

Setting different flags for each GPU type

Setting language type for HIP source files

```
set_source_files_properties(${VECTORADD_HIP_SRCS} PROPERTIES LANGUAGE ${GPU_RUNTIME})
set_source_files_properties(vectoradd.hip PROPERTIES COMPILE_FLAGS ${HIPCC_FLAGS})
```

Setting device compile flags

```
install(TARGETS vectoradd)
```

# Using a portable CMakeLists.txt

- For ROCm

```
module load rocm
module load cmake
export CXX=${ROCM_PATH}/llvm/bin/clang++
```

To build and run:

```
mkdir build && cd build
cmake ..
make VERBOSE=1
./vectoradd
```

- For CUDA

```
module load rocm
module load cuda
module load cmake
```

To build and run:

```
mkdir build && cd build
cmake -DCMAKE_GPU_RUNTIME=CUDA ..
make VERBOSE=1
./vectoradd
```

← Overrides default GPU runtime to specify CUDA

# Important CMake variables

- CMAKE\_HIP\_ARCHITECTURES
  - CMAKE\_HIP\_ARCHITECTURES="gfx90a;gfx908"
  - GPU\_TARGETS="gfx90a;gfx908"

List of gfx models: <https://llvm.org/docs/AMDGPUUsage.html>

Find the gfx model with rocinfo: rocinfo | grep gfx | sed -e 's/Name:/' | head -1 | sed 's/ //g'

- CMAKE\_CXX\_COMPILER:PATH=/opt/rocm/bin/amdclang++
- CMAKE\_HIP\_COMPILER\_ROCM\_ROOT:PATH=/opt/rocm – to help cmake find the cmake config files
- CMAKE\_PREFIX\_PATH=/opt/rocm
- Finding HIP packages and use results
  - find\_package(rocprim)
  - target\_link\_libraries(MyLib PUBLIC roc::rocprim)
- Using host and device from find\_package(hip)
  - target\_link\_libraries(MyLib PRIVATE hip::device)
  - target\_link\_libraries(MyApp PRIVATE hip::host)

# CUDA Fortran Fortran + HIP C/C++

- There is no HIP equivalent to CUDA Fortran
- HIP functions are callable from C, using extern C, so they can be called directly from Fortran
- The strategy here is:
  - 1) **Manually port** CUDA Fortran code to HIP kernels in C-like syntax
  - 2) Wrap the kernel launch in a C function
  - 3) Call the C function from Fortran through Fortran's ISO\_C\_binding. It requires Fortran 2008 because of using pointers to share data.
- This strategy should be usable by Fortran users since it is standard conforming Fortran
- ROCm has an interface layer, hipFort, which provides the wrapped bindings for use in Fortran
  - <https://github.com/ROCm/hipfort>

# Hands-on exercises

Located in our HPC Training Examples repo:

<https://github.com/amd/HPCTrainingExamples>

A table of contents for the READMEs if available at the top-level README in the repo

Relevant exercises for this presentation located in:

- [HIPIFY](#) directory
- [hipify](#) directory
- [HIPFort](#) directory

Link to instructions on how to run the HIPIFY tests: [HIPIFY/README.md](#)

Log into the AAC node and clone the repo:

```
ssh <username>@aac6.amd.com -p 7000 -i <path_to_ssh_key>  
git clone https://github.com/amd/HPCTrainingExamples.git
```

# 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.

© 2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm, Radeon™, CDNA, Instinct, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

The OpenMP® name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board.

HPE is a registered trademark of Hewlett Packard Enterprise Company and/or its affiliates.

LLVM™ is a trademark of LLVM Foundation

Perl® is a trademark of Perl Foundation.

**AMD** 