Porting Applications to HIP

Presenter: Maria Ruiz Varela, AMD

AMD
together we advance_

Acknowledgements

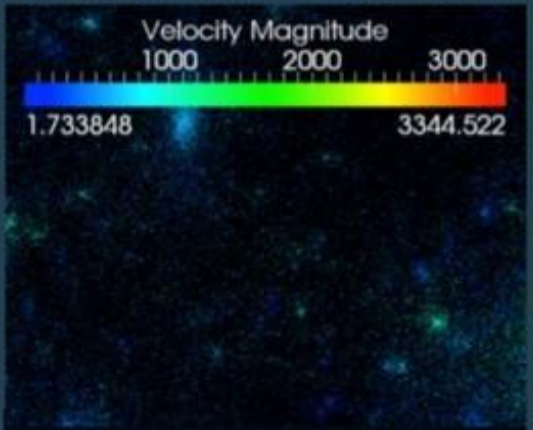
Suyash Tandon
Justin Chang
Julio Maia
Noel Chalmers
Paul T. Bauman
Nicholas Curtis
Nicholas Malaya
Alessandro Fanfarillo
Jose Noudohouenou

Chip Freitag
Damon McDougall
Noah Wolfe
Jakub Kurzak
Samuel Antao
George Markomanolis
Bob Robey
Maria Ruiz Varela

Agenda

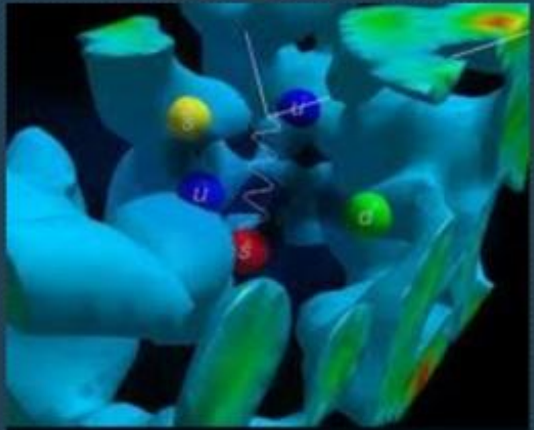
1. Porting applications to the HIP API
2. Code Conversion Tools
3. Portable HIP Build System
4. Other porting paths
5. Hipify example – Pennant mini-app
6. Questions

Porting applications to the HIP API

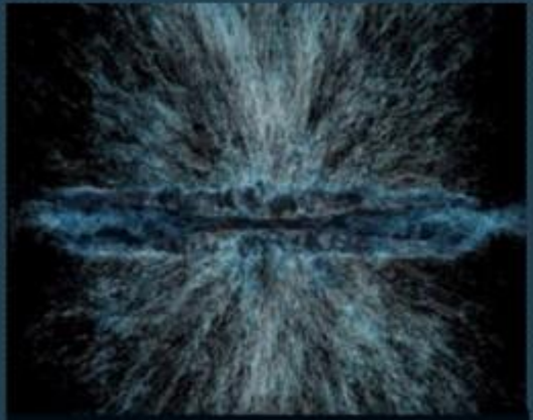


Ported in an Afternoon
HACC
Cosmology

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

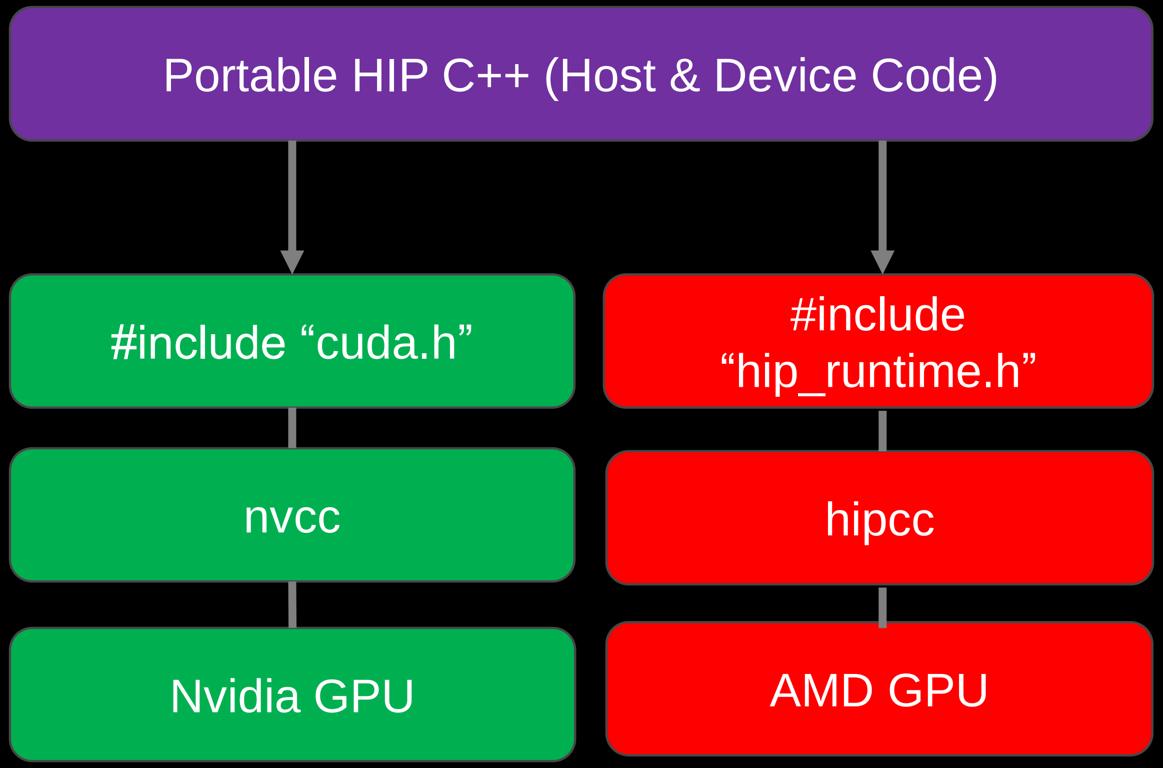
PYTORCH · GridTools · ALTAIR · SIRIUS · AMBER · PIConGPU · CP2K · LSMS

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's accelerators as well as CUDA devices

HIP:

- Is open-source
- Provides an API for an application to leverage GPU acceleration for both AMD and CUDA devices
- Syntactically similar to CUDA. Most CUDA API calls can be converted in place: cuda -> hip
- Supports a strong subset of CUDA runtime functionality



Code Conversion Tools

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
- Does not check for correctness

Hipify-clang

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

Hipify tools

Individual file tools

- hipify-perl
- hipify-clang

Recursive directory tools

- 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

Hipify-perl

- It is located in $HIP/bin/ (**export PATH=$PATH:[MYHIP]/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

- Build from source
- 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

- https://github.com/ROCm-Developer-Tools/HIPIFY/blob/master/README.md

Recursive directory-based tools

hipifyexamine.sh and hipifyexamine-perl.sh

- 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).

hipifyconvertinplace.sh and hipifyconvertinplace-perl.sh

- hipifyexamine-perl.sh recursively runs hipify-perl with the -inplace -print-stats options.

Let's show the convert script to understand what they do.

Source code for hipconvertinplace-perl.sh

```bash
#!/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.

```bash
#!/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'
```

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
- 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
  - Can either use inline GCN ISA, or convert it to HIP
- Hipify-perl and hipify-clang can both convert library calls

- 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.

What to look for when porting:

- Inline PTX assembly
- CUDA Intrinsics
- Hardcoded dependencies on warp size, or shared memory size
  - Grep for "32" *just in case*
  - Do not hardcode the warpsize! Rely on warpSize device definition, #define WARPSIZE *size,* or props.warpSize from host
- Code geared toward limiting size of register file on NVIDIA hardware
- Unsupported functions

Portable HIP Build System

1. Portable Makefiles
2. Portable Cmake
3. Library Equivalents
4. Specifying HIP Target
5. Identifying Compiler
6. Compiling for Host or Device
7. Compiler Defines

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 without 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

- There have been changes with each ROCm version which may require some adjustments

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
HIP_PATH ?= $(shell hipconfig --path)

ifeq ($(HIP_PLATFORM), nvidia)
  HIPCC_FLAGS += -x cu -I${HIP_PATH}/include/
  LDFLAGS = -lcudadevrt -lcudart_static -lrt -lpthread –ldl
endif
ifeq ($(HIP_PLATFORM), amd)
  HIPCC_FLAGS += -x hip -munsafe-fp-atomics
  LDFLAGS = -L${ROCM_PATH}/hip/lib -lamdhip64
endif
```

Setting default device compiler (HIP_PLATFORM ?= amd)

Setting compile flags for different GPUs (HIPCC_FLAGS for nvidia and amd)

```
%.o: %.hip
	hipcc $(HIPCC_FLAGS) -c $^ -o $@

$(EXECUTABLE): $(OBJECTS)
	hipcc $< $(LDFLAGS) -o $@

test: $(EXECUTABLE)
	$(EXECUTABLE)

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

Pattern rule for HIP source (%.o: %.hip)

AMD together we advance_

Using a portable Makefile

- For ROCm

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

- To build and run

```
make vectoradd
./srun
```

- For CUDA

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

We still need HIP for the portability layer

- To build and run

```
HIP_PLATFORM=nvidia make vectoradd
./srun
```

Overriding default to compile with Nvidia

For Frontier

- For AMD programming environment

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

- To build and run

```
make vectoradd
srun ./vectoradd
```

- For Cray programming environment
  - `module load PrgEnv-cray`
  - `module load amd-mixed`
  - `module load cmake`

- `To build and run`
  - `CXX=CC CRAY_CPU_TARGET=x86-64 make vectoradd`
  - `srun ./vectoradd`

For Perlmutter

- For Perlmutter

```
module load PrgEnv-gnu/8.3.3
Module load hip/5.4.3
module load PrgEnv-nvidia/8.3.3
module load cmake
```

We still need HIP for the portability layer

Overriding default to compile with Nvidia

- To build and run

```
HIP_PLATFORM=nvidia make vectoradd
srun ./vectoradd
```

Portable Build Systems – CMakeLists.text

```
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 (NOT CMAKE_GPU_RUNTIME)
   set(GPU_RUNTIME "${CMAKE_GPU_RUNTIME}" CACHE STRING "Switches between ROCM and CUDA")
endif (NOT CMAKE_GPU_RUNTIME)
# Really should only be ROCM or CUDA, but allowing HIP because it is the currently built-in option
set(GPU_RUNTIMES "ROCM" "CUDA" "HIP")
if(NOT "${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 ${ERROR_MESSAGE})endif()# GPU_RUNTIME for AMD GPUs should really be ROCM, if selecting AMD
GPUs
# so manually resetting to HIP if ROCM is selected
if (${GPU_RUNTIME} MATCHES "ROCM")
   set(GPU_RUNTIME "HIP")
endif (${GPU_RUNTIME} MATCHES "ROCM")
set_property(CACHE GPU_RUNTIME PROPERTY STRINGS ${GPU_RUNTIMES})
```

Setting GPU_RUNTIME

Defining GPU_RUNTIME will select ROCM or CUDA
(e.g. -DGPU_RUNTIME=ROCM)

AMD together we advance_

Portable Build Systems – CMakeLists.text

```
enable_language(${GPU_RUNTIME})
set(CMAKE_${GPU_RUNTIME}_EXTENSIONS OFF)
set(CMAKE_${GPU_RUNTIME}_STANDARD_REQUIRED ON)

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-if (${GPU_RUNTIME} MATCHES “CUDA")
   set(HIPCC_FLAGS "${CUDACC_FLAGS}")
else (throw and error)
endif

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

install(TARGETS vectoradd)
```

Enabling either CUDA or HIP(ROCM)

Setting different flags for each GPU type

Setting language type for HIP source files

Setting device compile flags

Using a portable CMakeLists.txt

- For ROCm

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

- To Build

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

- For CUDA

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

- To Build

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

Overrides default GPU runtime to specify CUDA

Frontier and Perlmutter

- For Frontier

```
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 Perlmutter

```
module load PrgEnv-gnu/8.3.3
Module load hip/5.4.3
module load PrgEnv-nvidia/8.3.3
module load cmake
```

- To build and run

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

HIP build tools

- hipconfig
  - `hip-clang-cxxflags :  -isystem "/opt/rocm-5.6.0/include" -O3`
  - `hip-clang-ldflags  :  -O3 --hip-link --rtlib=compiler-rt -unwindlib=libgcc`

- We can use the output from this command to set compiler options in the regular makefile system
  - `--hip-link` is only for clang++, so we use the more portable `-L${ROCM_PATH}/hip/lib -lamdhip64` that will work with other compilers
  - clang++ -x hip is roughly equivalent to using hipcc

We can also get these variables and use them directly in a Makefile
  - `CXXFLAGS += $(shell $(HIP_PATH)/bin/hipconfig --cxx_config)`
  - `CPPFLAGS += $(shell $(HIP_PATH)/bin/hipconfig --cpp_config)`

- For both make and cmake, the .cu extension can be used as a quick workaround to renaming to .hip

Important CMake variables

- CMAKE_HIP_ARCHITECTURES
  - `CMAKE_HIP_ARCHITECTURES="gfx90a;gfx908"`
  - `GPU_TARGETS="gfx90a;gfx908"`
- `CMAKE_CXX_COMPILER:PATH=/opt/rocm/bin/amdclang++`
- CMAKE_HIP_COMPILER_ROCM_ROOT:PATH=/opt/rocm-5.6.0 – to help cmake find the cmake config files
- `CMAKE_PREFIX_PATH=/opt/rocm-5.6.0`
- Specifying HIP language – two possible ways
  - `project(MyProj LANGUAGES HIP)`
  - `set_source_files_properties(MyLib.cu PROPERTIES LANGUAGE HIP)`
  - `??? Enable_language(HIP) Available in Cmake 3.21 and newer`
- 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)`

AMD together we advance_

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 – HIP port	C++ template library for linear algebra: matrices, vectors, numerical solvers,
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents

ROCm CMake Packages

Component	Package	Targets
HIP	hip	hip::host, hip::device
rocPRIM	rocprim	roc::rocprim
rocThrust	rocthrust	roc::rocthrust
hipCUB	hipcub	hip::hipcub
rocRAND	rocrand	roc::rocrand
rocBLAS	rocblas	roc::rocblas
rocSOLVER	rocsolver	roc::rocsolver
hipBLAS	hipblas	roc::hipblas
rocFFT	rocfft	roc::rocfft
hipFFT	hipfft	hip::hipfft
rocSPARSE	rocsparse	roc::rocsparse
hipSPARSE	hipsparse	roc::hipsparse
rocALUTION	rocalution	roc::rocalution
RCCL	rccl	rccl
MIOpen	miopen	MIOpen
MIGraphX	migraphx	migraphx::migraphx, migraphx::migraphx_c, migraphx::migraphx_cpu, migraphx::migraphx_gpu, migraphx::migraphx_onnx, migraphx::migraphx_tf

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_AMD is defined if the HIP platform targets AMD. Note, HIP_PLATFORM_HCC was previously defined if the HIP platform targeted AMD, it is deprecated.
- HIP_PLATFORM_NVDIA is defined if the HIP platform targets NVIDIA. Note, HIP_PLATFORM_NVCC was previously defined if the HIP platform targeted NVIDIA, it is deprecated.

Identifying the Compiler: hip-clang or nvcc

- Often, it's 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_AMD__
// Compiled with HIP-Clang
#endif

#ifdef __HIP_PLATFORM_NVIDIA__
// Compiled with nvcc
//  Could be compiling either CUDA file or a host compile

#ifdef __CUDACC__
// Compiled with nvcc (CUDA language extensions enabled)
Compiler directly generates the host code (using the Clang x86 target) and passes the
code to another host compiler. Thus, they have no equivalent of the __CUDA_ACC define.
```

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 doesn't represent the feature capability of the target device.

Compiler Defines

Define	HIP-Clang	nvcc	Other (GCC, ICC, Clang, etc.)
HIP-related defines:			
__HIP_PLATFORM_AMD__	Defined	Undefined	Defined if targeting AMD platform; undefined otherwise
__HIP_PLATFORM_NVIDIA__	Undefined	Defined	Defined if targeting NVIDIA 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 rocm docs)	0 or 1 depending on feature support (see rocm docs)	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

Other porting paths

Fortran

- First Scenario: Fortran + CUDA C/C++
  - Assuming there is no CUDA code in the Fortran files.
  - Hipify CUDA
  - Compile and link with hipcc
- Second Scenario: CUDA Fortran
  - There is no hipify equivalent but there is another approach…
  - HIP functions are callable from C, using `extern C`
  - See hipfort

CUDA Fortran -> Fortran + HIP C/C++

- There is no HIP equivalent to CUDA Fortran
- But HIP functions are callable from C, using `extern C`, so they can be called directly from Fortran
- The strategy here is:
  - **Manually port** CUDA Fortran code to HIP kernels in C-like syntax
  - Wrap the kernel launch in a C function
  - Call the C function from Fortran through Fortran's ISO_C_binding. It requires Fortran 2008 because of the pointers utilization.
- 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/ROCmSoftwarePlatform/hipfort

Alternatives to HIP

- Can also target AMD GPUs through OpenMP® 5.0 target offload
  - ROCm provides OpenMP® support
  - AMD OpenMP® compiler (AOMP) could integrate updated improvements regarding OpenMP® offloading performance, sometimes experimental stuff to validate before ROCm integration (https://github.com/ROCm-Developer-Tools/aomp)
  - GCC provides OpenMP® offload support.
- GCC will provide OpenACC
- Clacc from ORNL: https://github.com/llvm-doe-org/llvm-project/tree/clacc/main OpenACC from LLVM™ only for C (Fortran and C++ in the future)
  - Translate OpenACC to OpenMP® Offloading

OpenMP® Offload GPU Support

- ROCm and AOMP
  - ROCm supports both HIP and OpenMP®
  - AOMP: the AMD OpenMP® research compiler, it is used to prototype the new OpenMP® features for ROCm
- HPE Compilers
  - Provides offloading support to AMD GPUs, through OpenMP, HIP, and OpenACC (only for Fortran)
- GNU compilers:
  - Provide OpenMP® and OpenACC offloading support for AMD GPUs
  - GCC 11: Supports AMD GCN gfx908
  - GCC 13: Supports AMD GCN gfx90a

Understanding the hardware options

- ***rocminfo***
  - 110 CUs
  - Wavefront of size 64
  - 4 SIMDs per CU

```
Node:                    11
Device Type:             GPU
Cache Info:
  L1:                      16(0x10) KB
  L2:                      8192(0x2000) KB
Chip ID:                 29704(0x7408)
Cacheline Size:          64(0x40)
Max Clock Freq. (MHz):   1700
BDFID:                   56832
Internal Node ID:        11
Compute Unit:            110
SIMDs per CU:            4
Shader Engines:          8
Shader Arrs. per Eng.:   1
WatchPts on Addr. Ranges:4
Features:                KERNEL_DISPATCH
Fast F16 Operation:      TRUE
Wavefront Size:          64(0x40)
Workgroup Max Size:      1024(0x400)
Workgroup Max Size per Dimension:
  x                        1024(0x400)
  y                        1024(0x400)
  z                        1024(0x400)
Max Waves Per CU:        32(0x20)
Max Work-item Per CU:    2048(0x800)
```

`#pragma omp target teams distribute parallel for` *`simd`*

Options for pragma omp teams target:

- num_teams(220): Multiple number of workgroups with regards the compute units
- thread_limit(256): Threads per workgroup

- Thread limit is multiple of 64
- Teams*thread_limit should be multiple or a divisor of the trip count of a loop

AMD together we advance_

Hipify example
Pennant mini-app

What about a real example of converting a CUDA code to HIP

Pennant is a mini-app for unstructured Lagrangian hydrodynamics

Download the Pennant implementation for CUDA

- https://asc.llnl.gov/sites/asc/files/2020-09/pennant-singlenode-cude.tgz
- tar -xzvf pennant-singlenode-cude.tgz
- cd PENNANT

Use the hipify command for converting a whole directory tree

- ./hipconvertinplace-perl.sh .
- mv src/HydroGPU.cu src/HydroGPU.hip

Additional source modifications

- most are related to the double2 type (HIP_vector_type <double,2)
- HIP has support for operations on the HIP_vector_type

Changes to the Makefile

- All compiles use the hipcc compiler (not split between device and host)

Additional source modifications

- Change all occurrences of __CUDACC__ to __HIPCC__ in src/Vec2.hh (double2 definition)
- Comment out all HIP_vector_type operations with an #ifdef __CUDACC__ in src/Vec2.hh
- Comment out atomicMin operation with #ifdef __CUDACC__ in src/HydroGPU.hip
- Move #include <hip/hip_runtime.h> (2 occurrences) in src/HydroGPU.hip into a #ifdef __HIPCC__ block in src/Vec2.hh

Changes to Vec2.hh – double2 type and hip include file

```
+-- 15 lines: Vec2.hh-----------------------------------------------------------------
#include <cmath>
#include <string>
#include <sstream>
#include <ostream>


#ifdef __CUDACC__
---------------------------------------------------------------------------------------
#define FNQUALIFIERS __host__ __device__
#else
#define FNQUALIFIERS
#endif

// This class is defined with nearly all functions inline,
// to give the compiler maximum opportunity to optimize.
// Only the functions involving strings and I/O are
// out-of-line.

#ifndef __CUDACC__
// we are not in CUDA, so need to define our own double2 struct
struct double2
{
    typedef double value_type;
    double x, y;
    inline double2() {}
+-- 39 lines: inline double2(const double& x_, const double& y_) : x(x_), y(y_) {}--------------
}; // double2

inline double2 make_double2(const double& x_, const double& y_) {
    return(double2(x_, y_));
}

#else
// we are in CUDA; double2 is defined but needs op= operators
FNQUALIFIERS
inline double2& operator+=(double2& v, const double2& v2)
{
    v.x += v2.x;
    v.y += v2.y;
+--182 lines: return(v);--------------------------------------------------------------
```

```
+-- 15 lines: Vec2.hh-----------------------------------------------------------------
#include <cmath>
#include <string>
#include <sstream>
#include <ostream>


#ifdef __HIPCC__
#include <hip/hip_runtime.h>
#define FNQUALIFIERS __host__ __device__
#else
#define FNQUALIFIERS
#endif

// This class is defined with nearly all functions inline,
// to give the compiler maximum opportunity to optimize.
// Only the functions involving strings and I/O are
// out-of-line.

#ifndef __HIPCC__
// we are not in CUDA, so need to define our own double2 struct
struct double2
{
    typedef double value_type;
    double x, y;
    inline double2() {}
+-- 39 lines: inline double2(const double& x_, const double& y_) : x(x_), y(y_) {}--------------
}; // double2

inline double2 make_double2(const double& x_, const double& y_) {
    return(double2(x_, y_));
}

#elif defined(__CUDACC__)
// we are in CUDA; double2 is defined but needs op= operators
FNQUALIFIERS
inline double2& operator+=(double2& v, const double2& v2)
{
    v.x += v2.x;
    v.y += v2.y;
+--182 lines: return(v);--------------------------------------------------------------
```

Additional changes to HydroGPU.hip

```
------------------------------------------------------------------------------------------------
/*
 * HydroGPU.cu
 *
 *  Created on: Aug 2, 2012
 *      Author: cferenba
 *
+-- 5 lines: * Copyright (c) 2012, Los Alamos National Security, LLC.--------------------------

#include "HydroGPU.hh"

#include <cmath>
#include <cstdio>
#include <algorithm>
------------------------------------------------------------------------------------------------
#include <thrust/copy.h>
#include <thrust/sequence.h>
#include <thrust/sort.h>
#include <thrust/device_ptr.h>

#include "Memory.hh"
+--693 lines: #include "Vec2.hh"-----------------------------------------------------------------
        idtz = (z << 1) | 1;
    }

}


#ifdef __CUDACC__
static __device__ double atomicMin(double* address, double val)
{
    unsigned long long int* address_as_ull =
            (unsigned long long int*)address;
    unsigned long long int old = *address_as_ull, assumed;
    do {
+-- 2 lines: assumed = old;----------------------------------------------------------------------
                __double_as_longlong(min(val,
                __longlong_as_double(assumed))));
    } while (assumed != old);
    return __longlong_as_double(old);
}

#endif

static __device__ void hydroFindMinDt(
        const int z,
        const int z0,
        const int zlength,
        const double dtz,
+--564 lines: const int idtz,-----------------------------------------------------------------------
```

```
#include "hip/hip_runtime.h"
/*
 * HydroGPU.cu
 *
 *  Created on: Aug 2, 2012
 *      Author: cferenba
 *
+-- 5 lines: * Copyright (c) 2012, Los Alamos National Security, LLC.--------------------------

#include "HydroGPU.hh"

#include <cmath>
#include <cstdio>
#include <algorithm>
#include <hip/hip_runtime.h>
#include <thrust/copy.h>
#include <thrust/sequence.h>
#include <thrust/sort.h>
#include <thrust/device_ptr.h>

#include "Memory.hh"
+--693 lines: #include "Vec2.hh"-----------------------------------------------------------------
        idtz = (z << 1) | 1;
    }

}


------------------------------------------------------------------------------------------------
static __device__ double atomicMin(double* address, double val)
{
    unsigned long long int* address_as_ull =
            (unsigned long long int*)address;
    unsigned long long int old = *address_as_ull, assumed;
    do {
+-- 2 lines: assumed = old;----------------------------------------------------------------------
                __double_as_longlong(min(val,
                __longlong_as_double(assumed))));
    } while (assumed != old);
    return __longlong_as_double(old);
}

------------------------------------------------------------------------------------------------

static __device__ void hydroFindMinDt(
        const int z,
        const int z0,
        const int zlength,
        const double dtz,
+--564 lines: const int idtz,-----------------------------------------------------------------------
```

Makefile changes

- Change all CUDAC occurrences to CXX
- Comment out first CXX definition block so second one takes effect
  - Comment out the CXXFLAGS := $(CXXFLAGS_OPT) $(CPPFLAGS) line so next line takes effect
- Change nvcc to hipcc
- Change CXXFLAGS to add -std=c++14 --offload-arch=gfx90a
- Change LDFLAGS to –offload-arch=gfx90a instead of CUDA libraries
- Comment out all build rules for .cu files

- We'll do a more thorough code conversion in the exercises with a portable build system.

Makefile diffs

```
+-- 22 lines: BUILDDIR := build--------------------------------------------------------------
#CXX := g++
#CXXFLAGS_DEBUG := -g
#CXXFLAGS_OPT := -O3
#CXXFLAGS_OPENMP := -fopenmp

# intel flags:
#CXX := icpc
#CXXFLAGS_DEBUG := -g
#CXXFLAGS_OPT := -O3 -fast -fno-alias
#CXXFLAGS_OPENMP := -openmp

# pgi flags:
#CXX := pgCC
#CXXFLAGS_DEBUG := -g
#CXXFLAGS_OPT := -O3 -fastsse
#CXXFLAGS_OPENMP := -mp

# end compiler-dependent flags

CXX := hipcc
CXXFLAGS := -std=c++14
CXXFLAGS_DEBUG := -G -lineinfo
CXXFLAGS_OPT := -O3

LD := $(CXX)
LDFLAGS :=

# select optimized or debug
#CXXFLAGS := $(CXXFLAGS_OPT) $(CPPFLAGS)
CXXFLAGS += $(CXXFLAGS_OPT) $(CPPFLAGS)
#CXXFLAGS := $(CXXFLAGS_DEBUG) $(CPPFLAGS)
#CXXFLAGS += $(CXXFLAGS_DEBUG) $(CPPFLAGS)

# add openmp flags (comment out for serial build)
#CXXFLAGS += $(CXXFLAGS_OPENMP)
#LDFLAGS += $(CXXFLAGS_OPENMP)

all : $(BINARY)
+-- 7 lines: -include $(DEPS)---------------------------------------------------------------

$(BUILDDIR)/%.o : $(SRCDIR)/%.cc
        @echo compiling $<
        $(maketargetdir)
        $(CXX) $(CXXFLAGS) $(CXXINCLUDES) -c -o $@ $<

#$(BUILDDIR)/%.o : $(SRCDIR)/%.cu
#       @echo compiling $<
#       $(maketargetdir)
#       @# unsetting of CPATH is needed to make hipcc and icpc
#       @# play nicely together
#       (CPATH=;$(CXX) $(CXXFLAGS) $(CXXINCLUDES) -c -o $@ $<)

$(BUILDDIR)/%.d : $(SRCDIR)/%.cc
        @echo making depends for $<
        $(maketargetdir)
        @$(CXX) $(CXXFLAGS) $(CXXINCLUDES) -M $< | sed "1s![^ \t]\+\.o!$(@:.d=.o) $@!" >$@

#$(BUILDDIR)/%.d : $(SRCDIR)/%.cu
#       @echo making depends for $<
#       $(maketargetdir)
#       @$(CXX) $(CXXFLAGS) $(CXXINCLUDES) -M $< | sed "1s![^ \t]\+\.o!$(@:.d=.o) $@!" >$@
#
define maketargetdir
        -@mkdir -p $(dir $@) > /dev/null 2>&1
endef

clean :
        rm -f $(BINARY) $(OBJS) $(DEPS)
```

```
+-- 22 lines: BUILDDIR := build--------------------------------------------------------------
#CXX := g++
#CXXFLAGS_DEBUG := -g
#CXXFLAGS_OPT := -O3
#CXXFLAGS_OPENMP := -fopenmp

# intel flags:
CXX := icpc
CXXFLAGS_DEBUG := -g
CXXFLAGS_OPT := -O3 -fast -fno-alias
CXXFLAGS_OPENMP := -openmp

# pgi flags:
#CXX := pgCC
#CXXFLAGS_DEBUG := -g
#CXXFLAGS_OPT := -O3 -fastsse
#CXXFLAGS_OPENMP := -mp

# end compiler-dependent flags

CUDAC := nvcc
CUDACFLAGS := -arch=sm_21 --ptxas-options=-v
CUDACFLAGS_DEBUG := -G -lineinfo
CUDACFLAGS_OPT := -O3

LD := $(CXX)
LDFLAGS := -L$(CUDA_INSTALL_PATH)/lib64 -lcudart

# select optimized or debug
CXXFLAGS := $(CXXFLAGS_OPT) $(CPPFLAGS)
CUDACFLAGS += $(CUDACFLAGS_OPT) $(CPPFLAGS)
#CXXFLAGS := $(CXXFLAGS_DEBUG) $(CPPFLAGS)
#CUDACFLAGS += $(CUDACFLAGS_DEBUG) $(CPPFLAGS)

# add openmp flags (comment out for serial build)
#CXXFLAGS += $(CXXFLAGS_OPENMP)
#LDFLAGS += $(CXXFLAGS_OPENMP)

all : $(BINARY)
+-- 7 lines: -include $(DEPS)---------------------------------------------------------------

$(BUILDDIR)/%.o : $(SRCDIR)/%.cc
        @echo compiling $<
        $(maketargetdir)
        $(CXX) $(CXXFLAGS) $(CXXINCLUDES) -c -o $@ $<

$(BUILDDIR)/%.o : $(SRCDIR)/%.cu
        @echo compiling $<
        $(maketargetdir)
        @# unsetting of CPATH is needed to make nvcc and icpc
        @# play nicely together
        (CPATH=;$(CUDAC) $(CUDACFLAGS) $(CUDACINCLUDES) -c -o $@ $<)

$(BUILDDIR)/%.d : $(SRCDIR)/%.cc
        @echo making depends for $<
        $(maketargetdir)
        @$(CXX) $(CXXFLAGS) $(CXXINCLUDES) -M $< | sed "1s![^ \t]\+\.o!$(@:.d=.o) $@!" >$@

$(BUILDDIR)/%.d : $(SRCDIR)/%.cu
        @echo making depends for $<
        $(maketargetdir)
        @$(CUDAC) $(CUDACFLAGS) $(CUDACINCLUDES) -M $< | sed "1s![^ \t]\+\.o!$(@:.d=.o) $@!" >$@

define maketargetdir
        -@mkdir -p $(dir $@) > /dev/null 2>&1
endef

clean :
        rm -f $(BINARY) $(OBJS) $(DEPS)
```

AMD GPU programming resources

- ROCm platform: https://github.com/RadeonOpenCompute/ROCm/
  - With instructions for installing from Debian/CentOS/RHEL binary repositories
  - Has links to source repositories for all components, including HIP
- HIP porting guide: https://github.com/ROCm-Developer-Tools/HIP/blob/master/docs/markdown/hip_porting_guide.md
- ROCm/HIP libraries: https://github.com/ROCmSoftwarePlatform
- ROC-profiler: https://github.com/ROCm-Developer-Tools/rocprofiler
  - Collects application traces and performance counters
  - Trace timeline can be visualized with https://ui.perfetto.dev/
- AMD GPU ISA docs and more: https://developer.amd.com/resources/developer-guides-manuals/

Summary

- HIP has an extensive API similar to CUDA to enable portability
- Most of the changes are automatic
- The more specialized use of vector types on the GPU required some manual work
- Watch out for #ifdefs. They usually haven't considered all the cases.
- The makefile required more changes than the source
- This is a simple makefile. More complex build systems may require more work.

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.

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.

Questions?

AMD