Contents

Porting Applications to HIP

Hipify Examples

First, get the examples for this lecture

```
git clone git@github.com:olcf/hip-training-series.git
```

Frontier instructions

For the first interactive example, get an slurm interactive session on Frontier (see further below for NERSC Perlmutter):

```
salloc -N 1 -p batch --reservation=hip_training_2023_8_28 --gpus=1 -t 10:00 -A <project>
```

Outside the reservation window or if you're not on the reservation list, you can do: `salloc -N 1 -p batch --gpus=1 -t 10:00 -A <project>`

Use your project id in the project field. If you do not remember it, run the command without the -A option and it should report your valid projects.

The environment needs to be set up for the rocm software such as hipify-perl and hipcc. Here are the commands for Frontier.

```
module load PrgEnv-amd
module load amd
module load cmake
```

Perlumtter instructions

During the training session node reservation hours, get a slurm interactive session with

```
salloc -N 1 -q shared -C gpu -c 32 -G 1 -t 30:00 -A ntrain8 --reservation=hip_aug28
```

Outside the reservation hours, use

```
salloc -N 1 -q interactive -C gpu -c 32 -G 1 -t 30:00 -A <a project>
```

use your own project instead of ntrain8 if you have a NERSC regular project.

The modules needed for Perlmutter are slightly different than Frontier. Use these instead.

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

Exercise 1: Manual code conversion from CUDA to HIP (10 min)

Choose one or more of the CUDA samples in `hip-training-series/Lecture2/HIPIFY/mini-nbody/cuda` directory. Manually convert it to HIP. Tip: for example, the cudaMalloc will be called hipMalloc. You can choose from `nbody-block.cu, nbody-orig.cu, or nbody-soa.cu`

You'll want to compile on the node you've been allocated so that hipcc will choose the correct GPU architecture.

Exercise 2: Code conversion from CUDA to HIP using HIPify tools (10 min)

Use the `hipify-perl` script to "hipify" the CUDA samples you used to manually convert to HIP in Exercise 1. hipify-perl is in `$ROCM_PATH/hip/bin` directory and should be in your path.

First test the conversion to see what will be converted

```
hipify-perl -examine nbody-orig.cu
```

You'll see the statistics of HIP APIs that will be generated. The output might be different depending on the ROCm version.

```
[HIPIFY] info: file 'nbody-orig.cu' statistics:
  CONVERTED refs count: 7
  TOTAL lines of code: 91
  WARNINGS: 0
[HIPIFY] info: CONVERTED refs by names:
  cudaFree => hipFree: 1
  cudaMalloc => hipMalloc: 1
  cudaMemcpyDeviceToHost => hipMemcpyDeviceToHost: 1
  cudaMemcpyHostToDevice => hipMemcpyHostToDevice: 1
```

`hipify-perl` is in `$ROCM_PATH/hip/bin` directory and should be in your path. In some versions of ROCm, the script is called `hipify-perl`.

Now let's actually do the conversion.

```
hipify-perl nbody-orig.cu > nbody-orig.cpp
```

Compile the HIP programs.

```
hipcc -DSHMOO -I ../ nbody-orig.cpp -o nbody-orig`
```

The `#define SHMOO` fixes some timer printouts. Add `--offload-arch=<gpu_type>` to specify the GPU type and avoid the autodetection issues when running on a single GPU on a node.

- Fix any compiler issues, for example, if there was something that didn't hipify correctly.
- Be on the lookout for hard-coded Nvidia specific things like warp sizes and PTX.

Run the program

```
srun ./nbody-orig
```

A batch version of Exercise 2 for Frontier is given below. The batch scripts are also located in the mini-nbody directory. Please check than and modify them for your project and the reservation.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --ntasks=1
#SBATCH --gpus=1
#SBATCH -p batch
#SBATCH -A <project id>
#SBATCH --reservation=<reservation_name>
#SBATCH -t 00:10:00
```

```
module load PrgEnv-amd
module load amd
module load cmake

cd $HOME/hip-training-series/Lecture2/HIPIFY/mini-nbody/cuda
hipify-perl -print-stats nbody-orig.cu > nbody-orig.cpp
hipcc -DSHMOO -I ../ nbody-orig.cpp -o nbody-orig
srun ./nbody-orig
cd ../../..
```

Notes:

- Hipify tools do not check correctness
- `hipconvertinplace-perl` is a convenience script that does `hipify-perl -inplace -print-stats` command

Mini-App conversion example

Load the proper environment

```
module load PrgEnv-amd
module load amd
```

Get the CUDA version of the Pennant mini-app.

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

cd PENNANT

./hipexamine-perl.sh
```

And review the output

Now do the actual conversion. We want to do the conversion for the whole directory tree, so we'll use hipconvertinplace-sh

```
./hipconvertinplace-perl.sh
```

We want to use `.hip` extensions rather than `.cu`, so change all files with `.cu` to `.hip`

```
mv src/HydroGPU.cu src/HydroGPU.hip
```

Now we have two options to convert the build system to work with both ROCm and CUDA

Makefile option

First cut at converting the Makefile. Testing with `make` can help identify the next step.

- Change all occurances of CUDA to HIP
- Change the CXX variable to `clang++` located in `${ROCM_PATH}/llvm/bin/clang++`
- Change all the HIPC variables to HIPCC
- Change HIPCC to point to hipcc
- Change HIPCCFLAGS with CUDA options to HIPCCFLAGS_CUDA
- Remove `-fast` and `-fno-alias` from the CXXFLAGS_OPT
- Change all `.cu` to `.hip` in the Makefile

Now we are just getting compile errors from the source files. We will have to do fixes there. We'll tackle them one-by-one.

The first errors are related to the double2 type.

```
compiling src/HydroGPU.hip
(CPATH=;hipcc -O3 -I.  -c -o build/HydroGPU.o src/HydroGPU.hip)
In file included from src/HydroGPU.hip:14:
In file included from src/HydroGPU.hh:16:
src/Vec2.hh:35:8: error: definition of type 'double2' conflicts with type alias of the same name
struct double2
       ^

/opt/rocm-5.6.0/include/hip/amd_detail/amd_hip_vector_types.h:1098:1: note: 'double2' declared here
__MAKE_VECTOR_TYPE__(double, double);
^

/opt/rocm-5.6.0/include/hip/amd_detail/amd_hip_vector_types.h:1062:15: note: expanded from macro '__MAK
        using CUDA_name##2 = HIP_vector_type<T, 2>;\
              ^

<scratch space>:316:1: note: expanded from here
double2
```

HIP defines double2. Let's look at Vec2.hh. At line 33 where the first error occurs. We see an `#ifdef __CUDACC__` around a block of code there. We also need the #ifdef to include HIP as well. Let's check the available compiler defines from the presentation to see what is available. It looks like we can use `__HIP_DEVICE_COMPILE__` or maybe `__HIPCC__`.

Change line 33 in Vec2.hh to #ifndef `__HIPCC__`

The next error is about function attributes that are incorrect for device code.

```
compiling src/HydroGPU.hip
(CPATH=;hipcc -O3 -I.  -c -o build/HydroGPU.o src/HydroGPU.hip
src/HydroGPU.hip:168:23: error: no matching function for call to 'cross
    double sa = 0.5 * cross(px[p2] - px[p1],  zx[z] - px[p1]);
                      ^~~~
src/Vec2.hh:206:15: note: candidate function not viable: call to __host__ function from __device__ func
```

The FNQUALIFIER macro is what handles the attributes in the code. We find that defined at line 22 and again we see a `#ifdef __CUDACC__`. It is another `#ifdef __CUDACC__`. We can see that we need to pay attention to all the CUDA ifdef statements.

Change line 22 to `#ifdef __HIPCC__`

Finally we get an error about already defined operators on double2 types. These appear to be defined in HIP, but not in CUDA. So we change line 84

```
compiling src/HydroGPU.hip
(CPATH=;hipcc -O3 -I.  -c -o build/HydroGPU.o src/HydroGPU.hip)
src/HydroGPU.hip:149:15: error: use of overloaded operator '+=' is ambiguous (with operand types 'doubl
        zxtot += ctemp2[sn];
        ~~~~~ ^  ~~~~~~~~~~
/opt/rocm-5.6.0/include/hip/amd_detail/amd_hip_vector_types.h:510:26: note: candidate function
        HIP_vector_type& operator+=(const HIP_vector_type& x) noexcept
                         ^
src/Vec2.hh:88:17: note: candidate function
inline double2& operator+=(double2& v, const double2& v2)
```

Change line 85 to `#elif defined(__CUDACC__)`

Now we start getting errors for HydroGPU.hip. The first is for the atomicMin function. It is already defined in HIP, so we need to add an ifdef for CUDA around the code.

```
compiling src/HydroGPU.hip
(CPATH=;hipcc -O3 -I.  -c -o build/HydroGPU.o src/HydroGPU.hip)
```

```
src/HydroGPU.hip:725:26: error: static declaration of 'atomicMin' follows non-static declaration
static __device__ double atomicMin(double* address, double val)
                         ^
/opt/rocm-5.6.0/include/hip/amd_detail/amd_hip_atomic.h:478:8: note: previous definition is here
double atomicMin(double* addr, double val) {
       ^
1 error generated when compiling for gfx90a.
```

Add `#ifndef __CUDACC__/endif` to the block of code in `HydroGPU.hip` from line 725 to 737

We finally got through the compiler errors and move on to link errors

```
linking build/pennant
/opt/rocm-5.6.0//llvm/bin/clang++ -o build/pennant build/ExportGold.o build/ImportGMV.o build/Parallel.o
ld.lld: error: unable to find library -lcudart
```

In the Makefile, change the LDFLAGS while keeping the old settings for when we set up the switch between GPU platforms.

```
LDFLAGS_CUDA := -L$(HIP_INSTALL_PATH)/lib64 -lcudart
LDFLAGS := -L${ROCM_PATH}/hip/lib -lamdhip64
```

We then get the link error

```
linking build/pennant
/opt/rocm-5.6.0//llvm/bin/clang++ -o build/pennant build/ExportGold.o build/ImportGMV.o build/Parallel.o
ld.lld: error: undefined symbol: hydroInit(int, int, int, int, int, double, double, double, double, dou
>>> referenced by Hydro.cc
>>>               build/Hydro.o:(Hydro::Hydro(InputFile const*, Mesh*))

ld.lld: error: undefined symbol: hydroGetData(int, int, double2*, double*, double*, double*)
>>> referenced by Hydro.cc
>>>               build/Hydro.o:(Hydro::getData())
```

This one is a little harder. We can get more information by using `nm build/Hydro.o |grep hydroGetData` and `nm build/HydroGPU.o |grep hydroGetData`. We can see that the subroutine signatures are slightly different due to the double2 type on the host and GPU. You can also switch the compiler from clang++ to g++ to get a slightly more informative error. We are in a tough spot here because we need the hipmemcpy in the body of the subroutine, but the types for double2 are for the device instead of the host. One solution is to just compile and link everything with hipcc, but we really don't want to do that if only one routine needs to use the device compiler. So we cheat by declaring the prototype arguments as `void *` and casting the type in the call with `(void *)`. The types are really the same and it is just arguing with the compiler.

```
nm build/Hydro.o |grep hydroGetData
                 U _Z12hydroGetDataiiP7double2PdS1_S1_
nm build/HydroGPU.o |grep hydroGetData
0000000000003750 T _Z12hydroGetDataiiP15HIP_vector_typeIdLj2EEPdS2_S2_
```

In HydroGPU.hh

- Change line 38 and 39 to from `const double2*` to `const void*`
- Change line 62 from `double2*` to `void*`

In HydroGPU.hip

- Change line 1031 and 1032 to `const void*`
- Change line 1284 to `const void*`

In Hydro.cc

- Add `(void *)` before the arguments on lines 59, 60, and 145

Now it compiles and we can test the run with

```
build/pennant test/sedovbig/sedovbig.pnt
```

So we have the code converted to HIP and fixed the build system for it. But we haven't accomplished our original goal of running with both ROCm and CUDA.

We can copy a sample portable Makefile from `hip-training-series/Lecture1/HIP/saxpy/Makefile` and modify it for this application.

```
EXECUTABLE = pennant
BUILDDIR := build
SRCDIR = src
all: $(BUILDDIR)/$(EXECUTABLE) test

.PHONY: test

OBJECTS =  $(BUILDDIR)/Driver.o $(BUILDDIR)/GenMesh.o $(BUILDDIR)/HydroBC.o
OBJECTS += $(BUILDDIR)/ImportGMV.o $(BUILDDIR)/Mesh.o $(BUILDDIR)/PolyGas.o
OBJECTS += $(BUILDDIR)/TTS.o $(BUILDDIR)/main.o $(BUILDDIR)/ExportGold.o
OBJECTS += $(BUILDDIR)/Hydro.o $(BUILDDIR)/HydroGPU.o $(BUILDDIR)/InputFile.o
OBJECTS += $(BUILDDIR)/Parallel.o $(BUILDDIR)/QCS.o $(BUILDDIR)/WriteXY.o

CXXFLAGS = -g -O3 -DNDEBUG -fPIC
HIPCC_FLAGS = -O3 -g -DNDEBUG

HIP_PLATFORM ?= amd

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

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

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

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

$(BUILDDIR)/%.o : $(SRCDIR)/%.hip
    @echo compiling $<
    $(maketargetdir)
    hipcc $(HIPCC_FLAGS) -c $^ -o $@
```

```
$(BUILDDIR)/$(EXECUTABLE) : $(OBJECTS)
    @echo linking $@
    $(maketargetdir)
    hipcc $(OBJECTS) $(LDFLAGS) -o $@

test : $(BUILDDIR)/$(EXECUTABLE)
    $(BUILDDIR)/$(EXECUTABLE) test/sedovbig/sedovbig.pnt

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

clean :
    rm -rf $(BUILDDIR)
```

To test the makefile,

```
make build/pennant
make test
```

or just `make` to both build and run the test

To test the makefile build system with CUDA

```
module load cuda
HIP_PLATFORM=nvdia CXX=g++ make
```

To create a cmake build system, we can copy a sample portable Makefile from `hip-training-series/HIP/saxpy/CMakeLists.t` and modify it for this application.

```
cmake_minimum_required(VERSION 3.21 FATAL_ERROR)
project(Pennant LANGUAGES CXX)
include(CTest)

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,
    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})

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

set(PENNANT_CXX_SRCS src/Driver.cc src/ExportGold.cc src/GenMesh.cc src/Hydro.cc src/HydroBC.cc
                     src/ImportGMV.cc src/InputFile.cc src/Mesh.cc src/Parallel.cc src/PolyGas.cc
                     src/QCS.cc src/TTS.cc src/WriteXY.cc src/main.cc)

set(PENNANT_HIP_SRCS src/HydroGPU.hip)

add_executable(pennant ${PENNANT_CXX_SRCS} ${PENNANT_HIP_SRCS} )

# Make example runnable using ctest
add_test(NAME Pennant COMMAND pennant ../test/sedovbig/sedovbig.pnt )
set_property(TEST Pennant PROPERTY PASS_REGULAR_EXPRESSION "End cycle   3800, time = 9.64621e-01")

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

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

set_source_files_properties(${PENNANT_HIP_SRCS} PROPERTIES LANGUAGE ${GPU_RUNTIME})
set_source_files_properties(HydroGPU.hip PROPERTIES COMPILE_FLAGS ${HIPCC_FLAGS})

install(TARGETS pennant)
```

To test the cmake build system, do the following

```
mkdir build && cd build
cmake ..
make VERBOSE=1
ctest
```

Now testing for CUDA

```
module load cuda

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