Porting Applications to HIP

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

CASTIEL
May 2nd, 2023

AMD
together we advance_

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

Perspective for Code Porting Project Managers

- Conversion of CUDA to HIP (Hipifying) has been described as a "half-days' work"
  - Perhaps a little exaggerated – it often takes a bit longer. But it is pretty fast.
  - Actual code conversions have reported 80 to 90% of work is done by hipify tools
    - HACC – 95% converted automatically
    - CAFFE – 99.6% auto-converted
  - Extreme or lower-level CUDA code can require more manual work

- Adapting the build system can be more difficult
  - More complex build systems can take days

- HIP is designed to be a portable layer on top of ROCm and CUDA
  - Maintaining this portability takes extra effort
  - But it has great benefits

Portability and open-source are core values for AMD GPU software

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

- Can also convert to ROCm code with –roc option

Hipify-perl help page

```
hipify-perl is a tool to translate CUDA source code into portable HIP C++

USAGE: hipify-perl [OPTIONS] INPUT_FILE

OPTIONS:
  -cuda-kernel-execution-syntax - Keep CUDA kernel launch syntax (default)
  -examine                      - Combines -no-output and -print-stats options
  -exclude-dirs=s               - Exclude directories
  -exclude-files=s              - Exclude files
  -experimental                 - HIPIFY experimentally supported APIs
  -help                         - Display available options
  -hip-kernel-execution-syntax  - Transform CUDA kernel launch syntax to a regular HIP function call
                                  (overrides "--cuda-kernel-execution-syntax")
  -inplace                      - Backup the input file in .prehip file, modify the input file inplace
  -no-output                    - Don't write any translated output to stdout
  -o=s                          - Output filename
  -print-stats                  - Print translation statistics
  -quiet-warnings               - Don't print warnings on unknown CUDA identifiers
  -roc                          - Translate to roc instead of hip where it is possible
  -version                      - The supported HIP version
  -whitelist=s                  - Whitelist of identifiers
```

Hipify-clang

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

CUDA installation (headers) required

- 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

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)
- 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

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

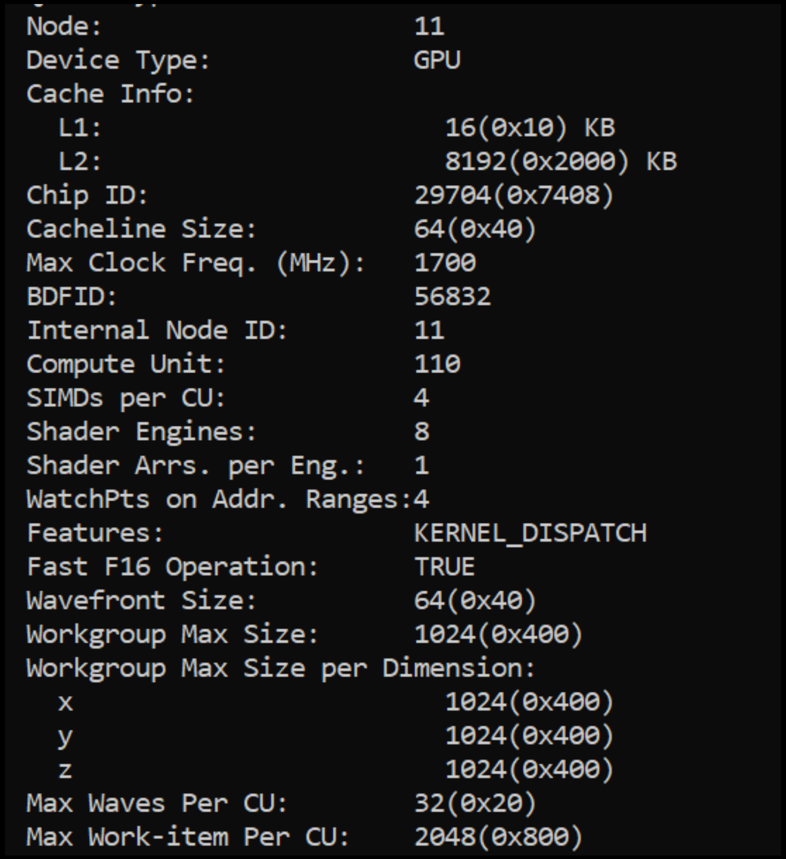
- Can also target AMD GPUs through OpenMP® 5.0 target offload
  - ROCm provides OpenMP® support
  - AMD OpenMP® compiler (AOMP) could integrate updated improvements regarding OpenMP® offloading performance, sometimes experimental stuff to validate before ROCm integration (https://github.com/ROCm-Developer-Tools/aomp)
  - 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_

Disclaimer

Questions?