Evaluating CUDA Portability with DPCT and HIPCL

ZHEMING JIN (JINZ@ORNL.GOV)

Acknowledgment

The results presented were obtained using a donation from Intel.

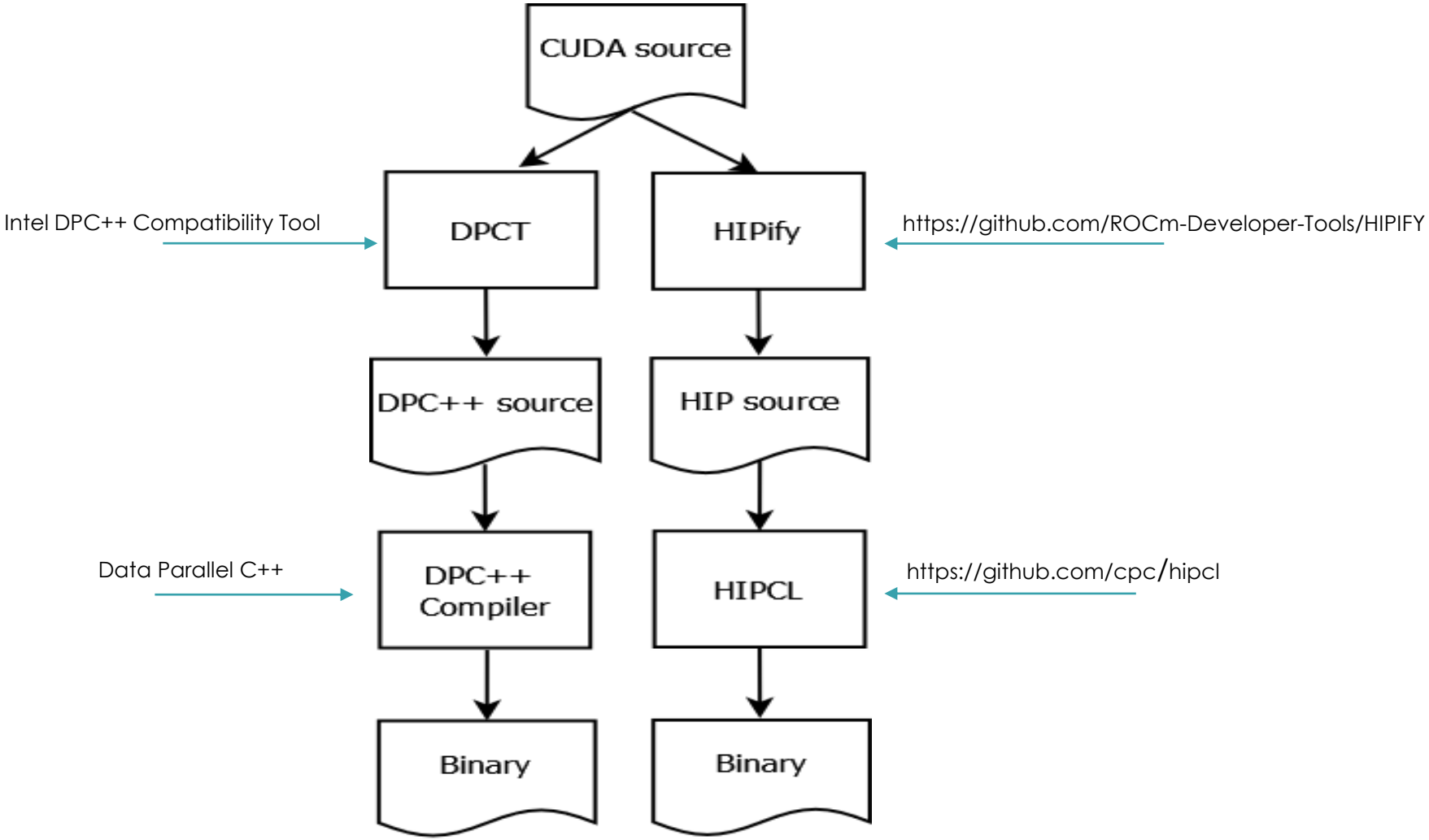
Overview

- Motivation
- Evaluation flow
- Experiments
- Conclusion

Motivation

- Significantly more CUDA programs than OpenCL/SYCL programs
  - Acknowledge CUDA's established presence in HPC
- Port CUDA programs for Intel GPUs
  - OpenCL API is a lower-level architecture compared to the commonly used CUDA API
  - OpenCL programming is tedious and error-prone
- Evaluate tools/translators that can port CUDA codes

Evaluation flow



Experimental setup – List of kernels

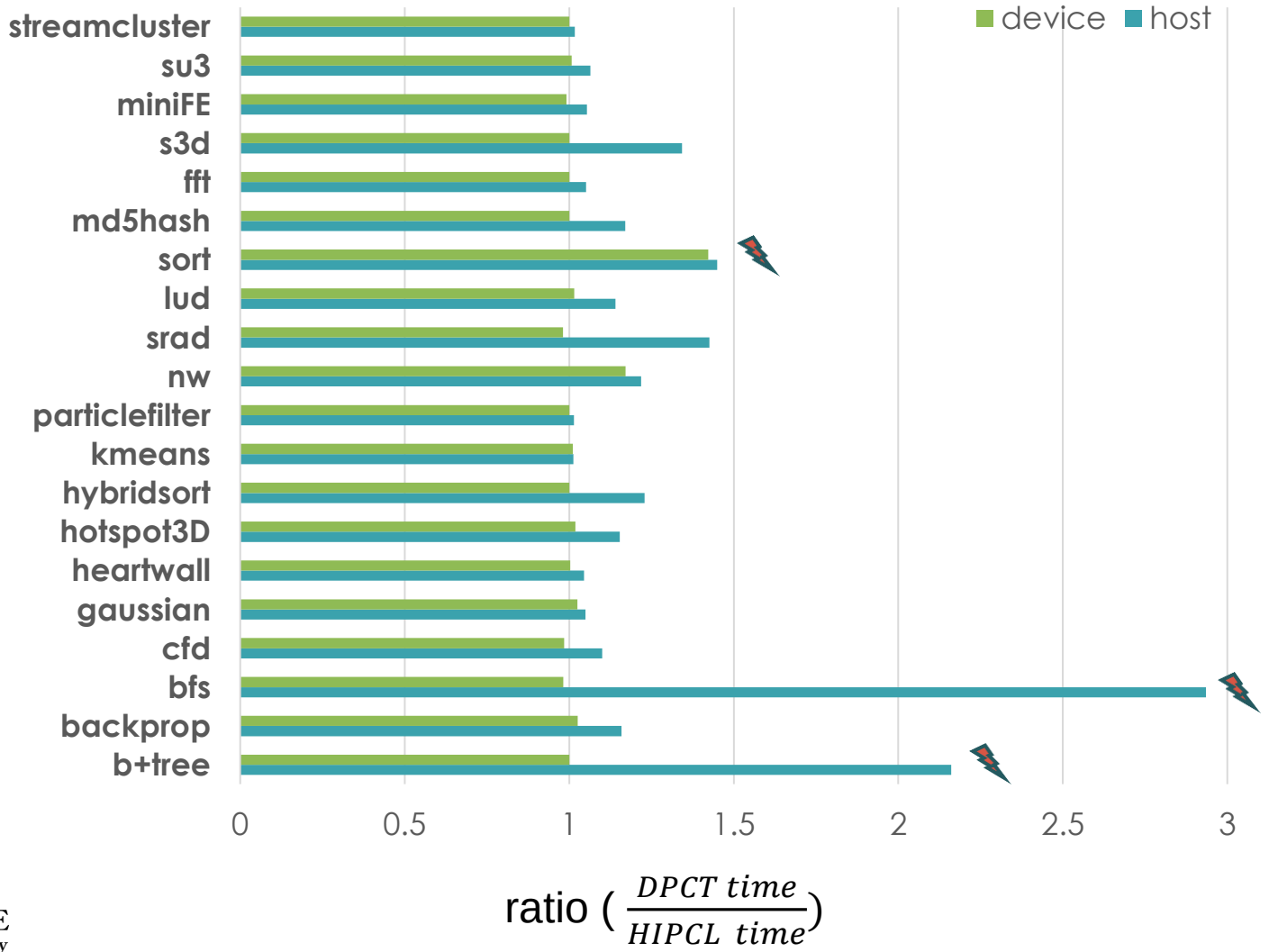
Name	Domain	#Kernels	Problem size
b+tree	Database search	2	1 million keys
backprop	Pattern recognition	2	65536 keys
bfs	Graph traversal	1	1 million vertices
cfd	Fluid dynamics	5	97047 elements
gaussian	Linear algebra	2	4096×4096 matrix
heartwall	Medical imaging	1	104 frames
hotspot3D	Physics simulation	1	512×512 points
hybridsort	Sorting	7	50 million numbers
kmeans	Data mining	2	494020 points and 34 features per point
particlefilter	Medical imaging	4	400000 points
nw	Bioinformatics	2	2048×2048 data points
srad	Image processing	6	512×512 data points
lud	Linear algebra	3	8192×8192 points
sort	Sorting	3	16M numbers
md5hash	Cryptography	1	10M keyspace
fft	Linear algebra	2	16M complex numbers
s3d	Combustion simulation	27	16×16×16 grid
miniFE	Unstructured grids	9	128×128×128 grid
su3	Physics simulation	1	32×32×32×32 sites
streamcluster	Data mining	1	65536 points

https://github.com/zjin-lcf/oneAPI-DirectProgramming

Experimental setup - Software/Hardware

- Intel Iris Xe Graphics (Mobile)
  - 96 execution units
  - Emulate double-precision floating-point operations
- Intel oneAPI Base Toolkit 2021.2.0 on Ubuntu 20.04
  - DPCT converts CUDA codes
  - DPC++ builds converted codes
- Build HIPCL from source (https://github.com/cpc/hipcl)
- Timing measured with the Intel OpenCL intercept layer
  - The host timing: total elapsed time of executing OpenCL API functions on a host
  - The device timing: total elapsed time of executing OpenCL API functions on a GPU device.
  - The Plugin interface is OpenCL

Comparison of host and device execution time



Looking into the “sort” in the DPCT code

- Performance bottleneck
  - The fence space of a work-group barrier is global rather than local
  - Global fence stalls the execution of a GPU device for global memory synchronization
  - A local space reduces the execution time from 3.58 s to 2.44 s on the host, and from 3.2 s to 2.06 s on the device

Looking into the “bfs” and “b+tree” in the DPCT code

- Performance bottleneck
  - clCreateContext: An OpenCL context is created with one or more devices.

bfs (DPCT)

OpenCL API	DPCT time	Percentage
clBuildProgram	67 ms	22%
clCreateContext	183 ms	61%
clGetPlatformID	30 ms	10%

bfs (HIPCL)

OpenCL API	HIPCL time	Percentage
clLinkProgram	70 ms	77%
clEnqueueSVMMemcpy	10 ms	12%
clFinish	6.7 ms	7%

b+tree (DPCT)

OpenCL API	DPCT time	Percentage
clBuildProgram	135 ms	42%
clCreateContext	145 ms	45%
clGetPlatformID	29 ms	9%

b+tree (HIPCL)

OpenCL API	HIPCL time	Percentage
clLinkProgram	136 ms	92%
clEnqueueSVMMemcpy	9 ms	6%
clFinish	1.6 ms	1%

Double-precision floating-point emulation

- Paradox
  - No double-precision operations in the "float" mode
  - babelStream, fft, s3d, black-scholes …
- Suggestion
  - "-cl-single-precision-constant" is OpenCL-only
  - Treat double-precision floating-point constant as single-precision constant in the DPC++ compiler
  - Tedious to cast legacy applications (e.g., s3d) that contain hundreds of double-precision floating-point constants

Related work

- *MCUDA*
  - broaden the applicability of a previously accelerator-specific programming model to a CPU architecture
- *Swan*
  - a high-level library for an application to call Swan API which is then mapped to the CUDA or OpenCL API
- *Coriander*
  - a compiler and runtime for running CUDA applications on OpenCL 1.2 devices
- *CU2CL*
  - a source-to-source translator built upon the Clang compiler for converting a CUDA program to an OpenCL program

Conclusion and Future Work

- DPCT may significantly reduce porting effort
- Developers may manually change DPCT programs
- Comments in automatically generated DPCT codes are useful
- No tools are perfect in translating a CUDA application
- Evaluate HIPCL and DPCT using more applications in our future work

Thanks to

The DPCT and HIPCL teams