Wrong Way: Lessons Learned and Possibilities for Using the "Wrong" Programming Approach on Leadership Computing Facility Systems

Philip C. Roth

National Center for Computational Sciences
Oak Ridge National Laboratory

16 February 2022

ORNL is managed by UT-Battelle LLC for the US Department of Energy

About the Presenter

Photo by Carlos Jones

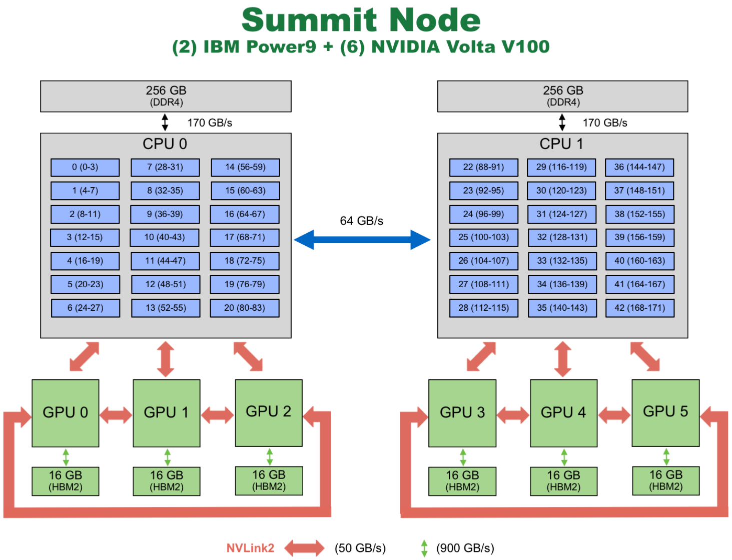
- Member of ORNL R&D Staff since 2004
  - Future Technologies group, Computer Science and Mathematics division, 2004-2018
  - Scientific Computing group, National Center for Computational Sciences (NCCS), 2018-2020
  - Leader, Algorithms and Performance Analysis group, Science Engagement section, NCCS, 2020-present
- Education focused on scalable performance tools and techniques
  - Ph.D.: University of Wisconsin—Madison, Barton Miller, Paradyn Project
  - M.S.: University of Illinois at Urbana-Champaign, Daniel Reed, Pablo project
- In between: software engineer for SuperComputers Intl./CHEN Systems Inc./MCSB Technology
  - Startup producing high performance enterprise server hardware and software (and then just software)

Early 2022 DOE Landscape

- U.S. Department of Energy (DOE) computing centers exhibit ***variety!***
  - Hardware
    - Accelerators: presence, vendors/types, CPU/GPU ratios, connectivity within node
    - Interconnect: type, topology, capabilities, connectivity within node
    - NVM: on-node vs. near-node vs. not present
  - Software, driven by hardware, community trends, and user requirements
  - Projects and users
- As a whole, DOE computing centers epitomize need for ***<u>Performance, Portability, and Productivity</u>***

The Oak Ridge Leadership Computing Facility (OLCF)

- Production
  - Summit: each node contains six NVIDIA V100 GPUs and two 22-core IBM POWER9 CPUs
  - Frontier (soon): Four AMD Radeon Instinct MI250X GPUs and one 64-core AMD EPYC 7A53 CPU
    - HPE Slingshot 11 NIC connected directly to GPUs
- Pre-Production/Training
  - Ascent: like Summit
  - Crusher (now): like Frontier
- Support
  - Andes: commodity x86_64 Linux cluster with a few GPU nodes
- ***90%+ of production systems' computational capability comes from GPUs***
- ***I will focus on GPUs and functionality (as opposed to performance) in this talk***

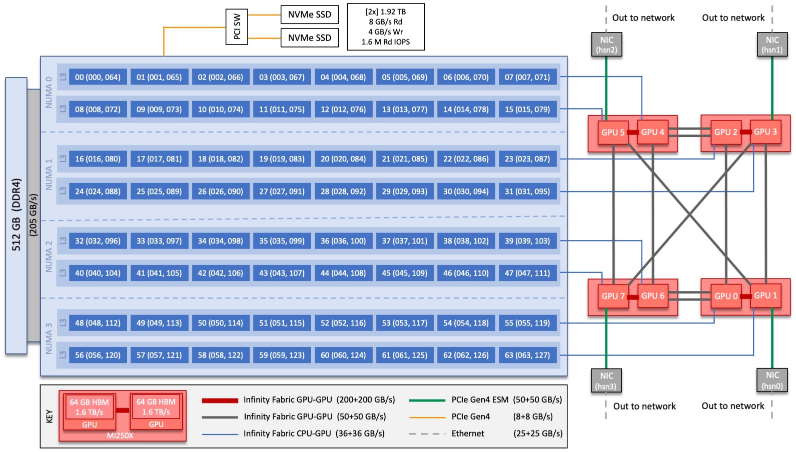


CPU 0

0 (0-3)	7 (28-31)	14 (56-59)
1 (4-7)	8 (32-35)	15 (60-63)
2 (8-11)	9 (36-39)	16 (64-67)
3 (12-15)	10 (40-43)	17 (68-71)
4 (16-19)	11 (44-47)	18 (72-75)
5 (20-23)	12 (48-51)	19 (76-79)
6 (24-27)	13 (52-55)	20 (80-83)

CPU 1

22 (88-91)	29 (116-119)	36 (144-147)
23 (92-95)	30 (120-123)	37 (148-151)
24 (96-99)	31 (124-127)	38 (152-155)
25 (100-103)	32 (128-131)	39 (156-159)
26 (104-107)	33 (132-135)	40 (160-163)
27 (108-111)	34 (136-139)	41 (164-167)
28 (112-115)	35 (140-143)	42 (168-171)



NUMA 0	00 (000, 064)	01 (001, 065)	02 (002, 066)	03 (003, 067)	04 (004, 068)	05 (005, 069)	06 (006, 070)	07 (007, 071)
	08 (008, 072)	09 (009, 073)	10 (010, 074)	11 (011, 075)	12 (012, 076)	13 (013, 077)	14 (014, 078)	15 (015, 079)
NUMA 1	16 (016, 080)	17 (017, 081)	18 (018, 082)	19 (019, 083)	20 (020, 084)	21 (021, 085)	22 (022, 086)	23 (023, 087)
	24 (024, 088)	25 (025, 089)	26 (026, 090)	27 (027, 091)	28 (028, 092)	29 (029, 093)	30 (030, 094)	31 (031, 095)
NUMA 2	32 (032, 096)	33 (033, 097)	34 (034, 098)	35 (035, 099)	36 (036, 100)	37 (037, 101)	38 (038, 102)	39 (039, 103)
	40 (040, 104)	41 (041, 105)	42 (042, 106)	43 (043, 107)	44 (044, 108)	45 (045, 109)	46 (046, 110)	47 (047, 111)
NUMA 3	48 (048, 112)	49 (049, 113)	50 (050, 114)	51 (051, 115)	52 (052, 116)	53 (053, 117)	54 (054, 118)	55 (055, 119)
	56 (056, 120)	57 (057, 121)	58 (058, 122)	59 (059, 123)	60 (060, 124)	61 (061, 125)	62 (062, 126)	63 (063, 127)

Infinity Fabric GPU-GPU	(200+200 GB/s)
Infinity Fabric GPU-GPU	(50+50 GB/s)
Infinity Fabric CPU-GPU	(36+36 GB/s)
PCIe Gen4 ESM	(50+50 GB/s)
PCIe Gen4	(8+8 GB/s)
Ethernet	(25+25 GB/s)

Crusher node diagram

Terminology

- What does “wrong” mean? ”Wrong” compared to what?
- Is “wrong” bad? Discouraged? Disallowed?

- ***For this presentation, “wrong” means:***
  - ***Not the usual or generally-accepted way to program the GPU on a given system***
  - ***Not (necessarily) supported by the system vendors or the computing facility***

- ***Focus is on what is possible, not (necessarily) what is recommended***

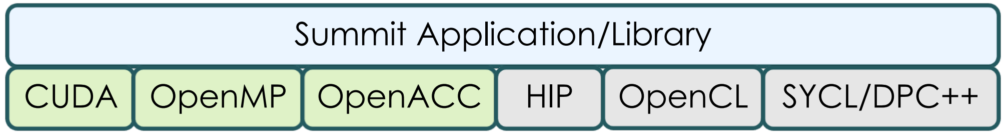
An Analogy

Dyn
inst

- Groups led by Bart Miller (U.Wisconsin-Madison) and Jeff Hollingsworth (U.Maryland) collaborate on Dyninst, an API and implementation for binary code analysis and instrumentation
- Among other things, allows a tool process to change another process' code ***while it is running,*** e.g., to insert/remove instrumentation or to patch buggy code
  - Most tools instrument at compile time or before
- Alternative model to how many people think about software - that things are fixed once compiled/linked
- But **wrong?** No – compiled code is more malleable than many think
- Requires care! (More on this later)
- Dyninst available via Spack, used by several other Spack-accessible tools like HPCToolkit, Timemory, and STAT

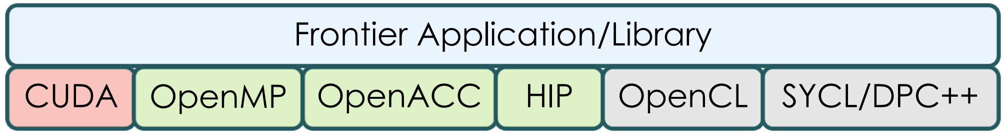
Natural and “Wrong” Ways on LCF Systems

- Determined mainly by type of GPU and source language
- Not exhaustive lists!
- Summit
  - Natural: CUDA, OpenMP target, OpenACC
  - ”Wrong”: HIP, OpenCL, SYCL/DPC++
- Frontier
  - Natural: HIP, OpenMP target offload, OpenACC
  - “Wrong”: OpenCL, SYCL/DPC++
- Aurora
  - Natural: OpenMP target offload, SYCL/DPC++
  - “Wrong”: HIP
- I consider portability layer software like Kokkos, RAJA, OCCA considered natural approach if they have backend support for at least one of a system’s natural approaches E.g.,



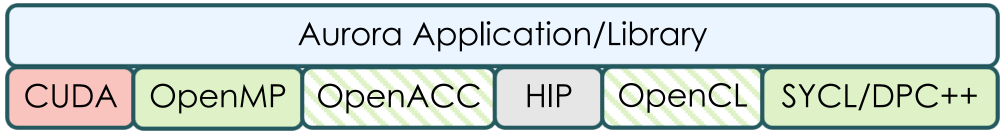
Summit Application/Library

CUDA | OpenMP | OpenACC | HIP | OpenCL | SYCL/DPC++



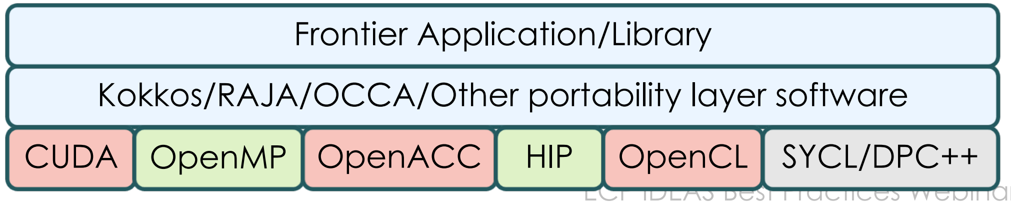
Frontier Application/Library

CUDA | OpenMP | OpenACC | HIP | OpenCL | SYCL/DPC++



Aurora Application/Library

CUDA | OpenMP | OpenACC | HIP | OpenCL | SYCL/DPC++



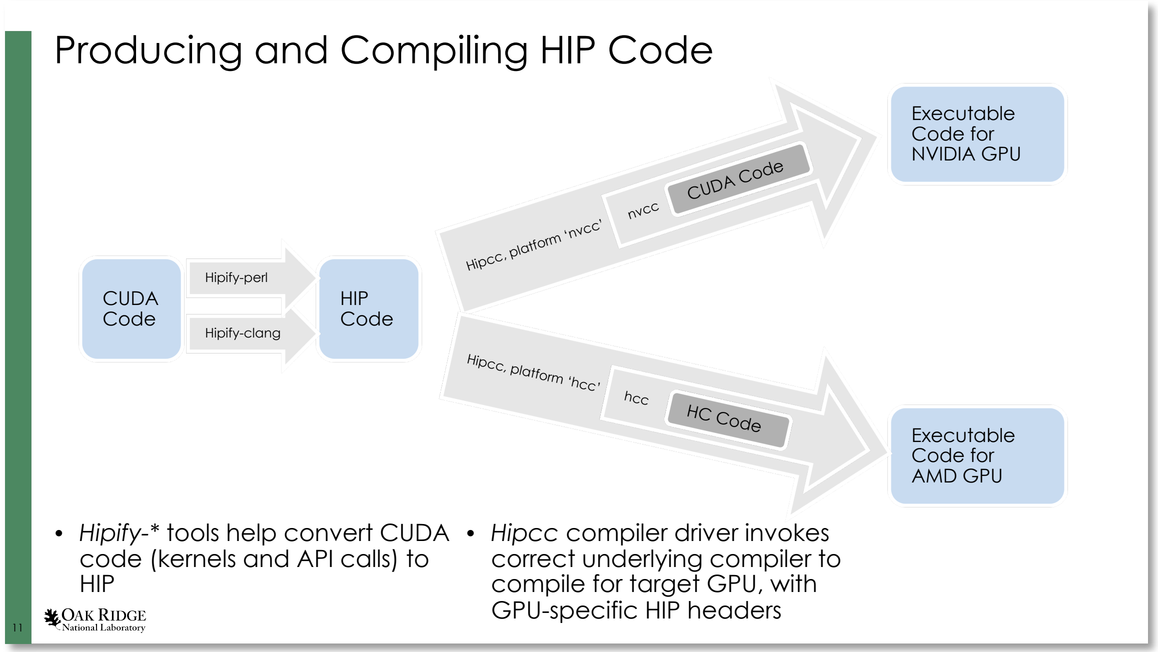
Frontier Application/Library

Kokkos/RAJA/OCCA/Other portability layer software

CUDA | OpenMP | OpenACC | HIP | OpenCL | SYCL/DPC++

HIP

- Heterogeneous-compute Interface for Portability (HIP)
- Portability layer with interface similar to CUDA, backends for AMD and NVIDIA GPUs
  - On NVIDIA systems, verify lightweight header-only library; final executable *is* a CUDA executable
- ”Hipify” tool available to ease porting from CUDA to HIP APIs
- Multiple HIP compilers (e.g., AMD within ROCm, HPE’s CCE)
- Growing ecosystem of libraries with portability interfaces and support for AMD and NVIDIA GPUs
- Open source



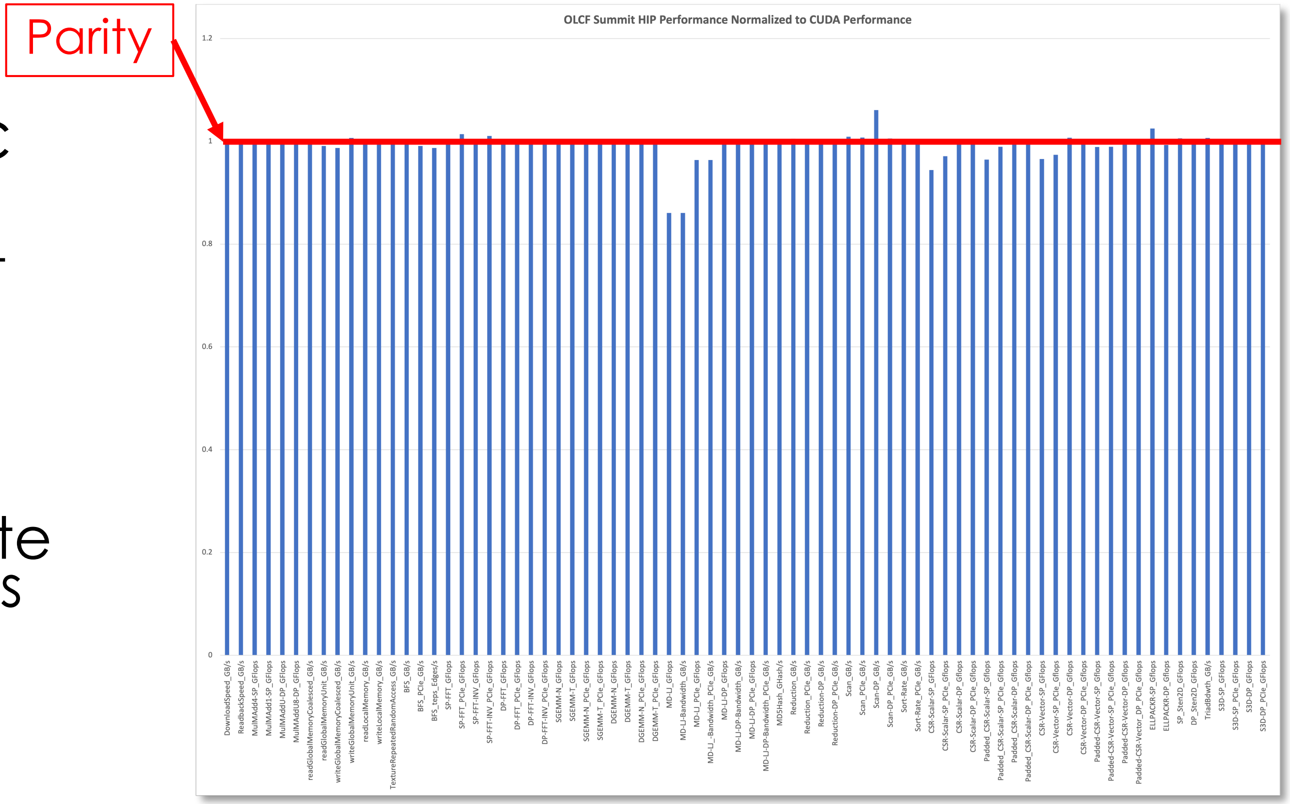
Producing and Compiling HIP Code

- *Hipify-** tools help convert CUDA code (kernels and API calls) to HIP
- *Hipcc* compiler driver invokes correct underlying compiler to compile for target GPU, with GPU-specific HIP headers

How HIP code is translated to NVIDIA executable or AMD executable with AMD tools. Information is slightly dated: no longer a separate hipify-clang executable, nor does hipcc invoke a compiler called hcc when compiling for AMD GPUs

HIP on Summit

- OLCF now provides a rocm-hip module to enable access hipcc and the HIP headers
  - In earlier days, using HIP on Summit only involved cloning the HIP repository and adding its bin directory to the PATH
  - Current implementation requires a little bit of installation, e.g., to create a header containing the software's version information
- HIP performance is very, very close to direct use of CUDA
  - On Summit, HIP is a header-only library

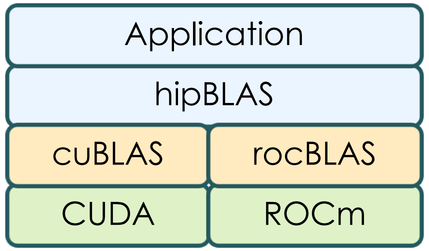


Performance of HIP SHOC benchmarks normalized to CUDA SHOC benchmarks. Only benchmarks whose results are reported by the SHOC benchmark suite driver script are included. CUDA 11.0.3, rocm-hip/4.3.0, gcc 9.1. Average HIP performance including data transfers is 99.4% of CUDA performance.

The Importance of the Ecosystem

- Almost any real application targeting GPUs needs more than just a HIP (or CUDA, or OpenMP, or OpenCL, or SYCL, or DPC++) compiler – they rely on one or more GPU-accelerated ***libraries***

- HIP ecosystem includes variety of HIP* libraries that can use either the AMD ROCm or NVIDIA CUDA library as a back-end
  - E.g., hipBLAS -> cuBLAS, rocBLAS

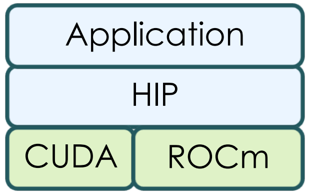


- OLCF does not (yet?) provide system-wide installations of the hip* libraries on Summit

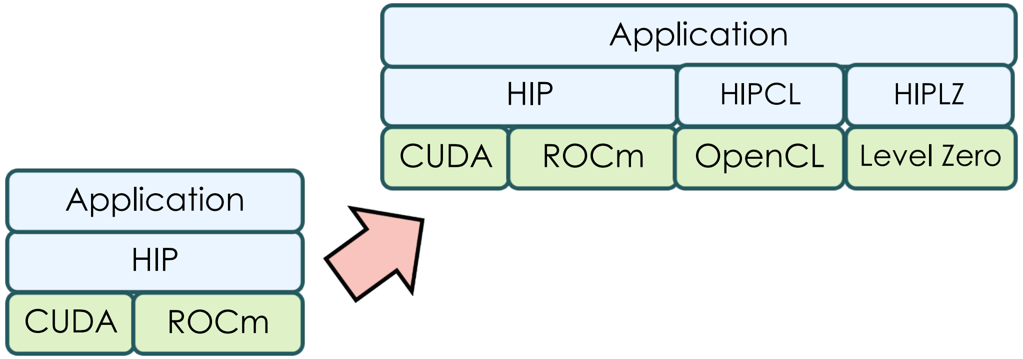
- HIP* libraries are open source, relatively easy to build and install in one's home directory
  - Many (most?) involve compiling some shim code

HIP on Aurora?

- Recall HIP is a portability layer

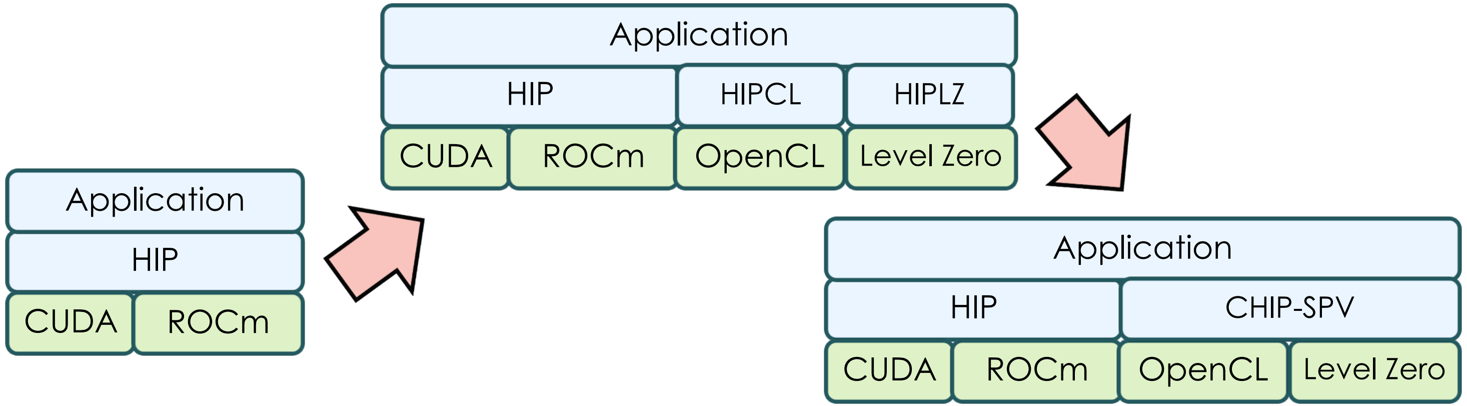


HIP on Aurora?



- Recall HIP is a portability layer
- The HIP on Level Zero (HIPLZ) ECP project is working to enable HIP applications to run on Intel GPUs
  - Level Zero is an Intel runtime that targets GPUs
- Original approach was to adapt the HIPCL (HIP over an OpenCL runtime) implementation to use Level Zero instead of the OpenCL runtime
- Have demonstrated several benchmarks and mini/proxy apps running via HIPLZ on Intel integrated GPUs
  - Recall importance of the ecosystem! Sparkler proxy app involved implementing a stub hipBLAS library that used MKL as a backend for the two BLAS routines it uses

HIP on Aurora?



Application
HIP
CUDA | ROCm

Application
HIP | HIPCL | HIPLZ
CUDA | ROCm | OpenCL | Level Zero

Application
HIP | CHIP-SPV
CUDA | ROCm | OpenCL | Level Zero

- Recall HIP is a portability layer
- The HIP on Level Zero (HIPLZ) ECP project is working to enable HIP applications to run on Intel GPUs
  - Level Zero is an Intel runtime that targets GPUs
- Original approach was to adapt the HIPCL (HIP over an OpenCL runtime) implementation to use Level Zero instead of the OpenCL runtime
- Have demonstrated several benchmarks and mini/proxy apps running via HIPLZ on Intel integrated GPUs
  - Recall importance of the ecosystem! Sparkler proxy app involved implementing a stub hipBLAS library that used MKL as a backend for the two BLAS routines it uses
- More recently, HIPCL and HIPLZ projects have combined efforts in the CHIP-SPV project that can target Level Zero or OpenCL backends
- Depends on SPIR-V, so Summit and Crusher/Frontier are also potential targets for POCL with SPIR-V support

OpenCL

- Khronos standard
- By some definitions, very good performance portability due to widespread availability of mature implementations
- Originally, C-based kernel language but more recent versions have supported C++ features (OpenCL C++ and C++ for OpenCL)
- Allows creation of kernels by:
  - Compiling source dynamically
  - Ingesting pre-compiled code, either device-native or portable intermediate representation
  - Not all implementations support all options
- Historically close ties with SPIR-V, a Khronos standard portable binary intermediate representation

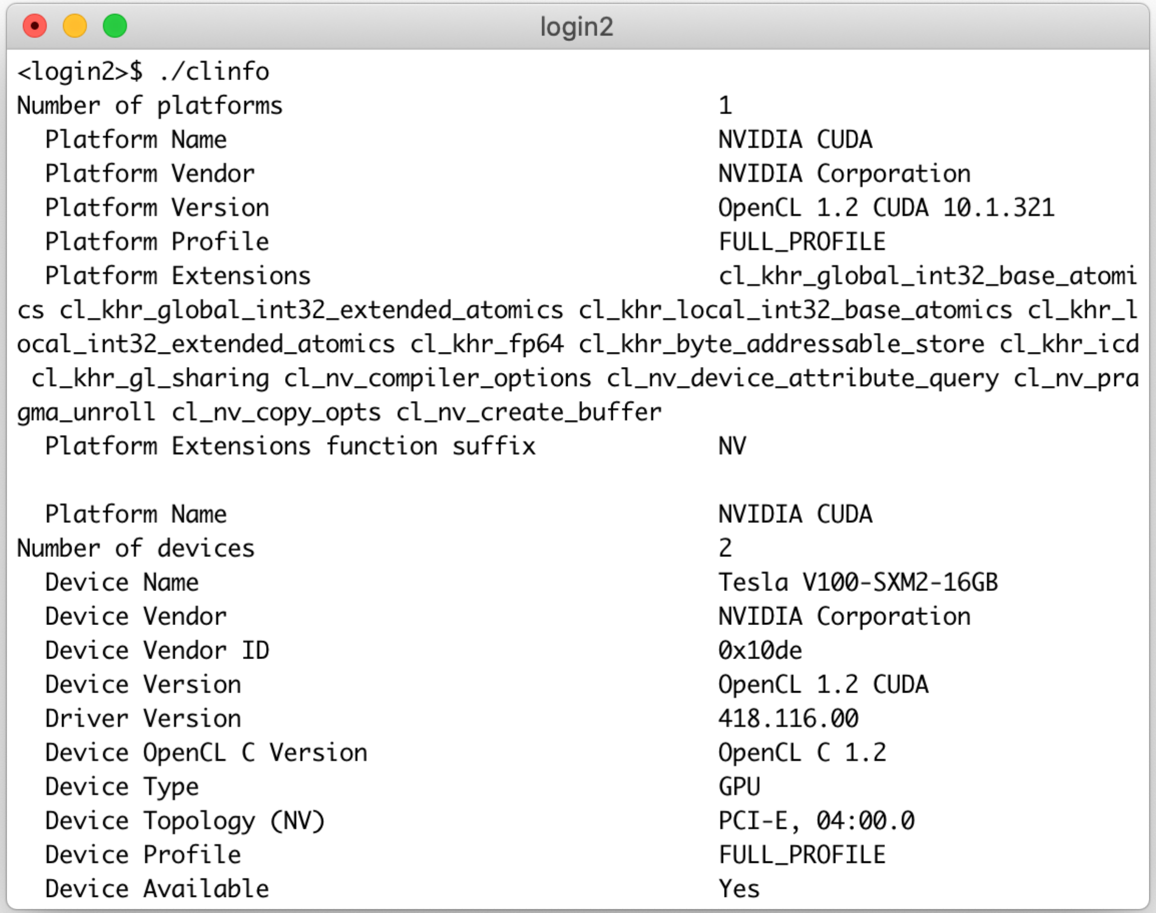
OpenCL on Summit

- Summit has incomplete OpenCL support
  - From RHEL distribution, have device-independent runtime library but not development headers or libraries
  - NVIDIA driver installation adds GPU-specific OpenCL driver
  - CUDA installations include a device-independent runtime library but it is for the wrong CPU architecture for Summit

- NVIDIA, OLCF never claimed to support OpenCL on POWER9
  - I am **not** saying they should! But exploring alternatives is the point of the investigation

OpenCL on Summit: Experiences I

- Easy to build Khronos' reference implementation of device-independent runtime library
  - Spack's builtin repository has **ocl-icd** package with headers variant
- Platform/device queries (e.g., with CLInfo utility) and data transfer are functional
- Unable to compile source code dynamically
  - (Though device queries report compiler and linker are available)



```
<login2>$ ./clinfo
Number of platforms                               1
  Platform Name                                   NVIDIA CUDA
  Platform Vendor                                 NVIDIA Corporation
  Platform Version                                OpenCL 1.2 CUDA 10.1.321
  Platform Profile                                FULL_PROFILE
  Platform Extensions                             cl_khr_global_int32_base_atomi
cs cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_l
ocal_int32_extended_atomics cl_khr_fp64 cl_khr_byte_addressable_store cl_khr_icd
 cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pra
gma_unroll cl_nv_copy_opts cl_nv_create_buffer
  Platform Extensions function suffix             NV

  Platform Name                                   NVIDIA CUDA
Number of devices                                 2
  Device Name                                     Tesla V100-SXM2-16GB
  Device Vendor                                   NVIDIA Corporation
  Device Vendor ID                                0x10de
  Device Version                                  OpenCL 1.2 CUDA
  Driver Version                                  418.116.00
  Device OpenCL C Version                         OpenCL C 1.2
  Device Type                                     GPU
  Device Topology (NV)                            PCI-E, 04:00.0
  Device Profile                                  FULL_PROFILE
  Device Available                                Yes
```

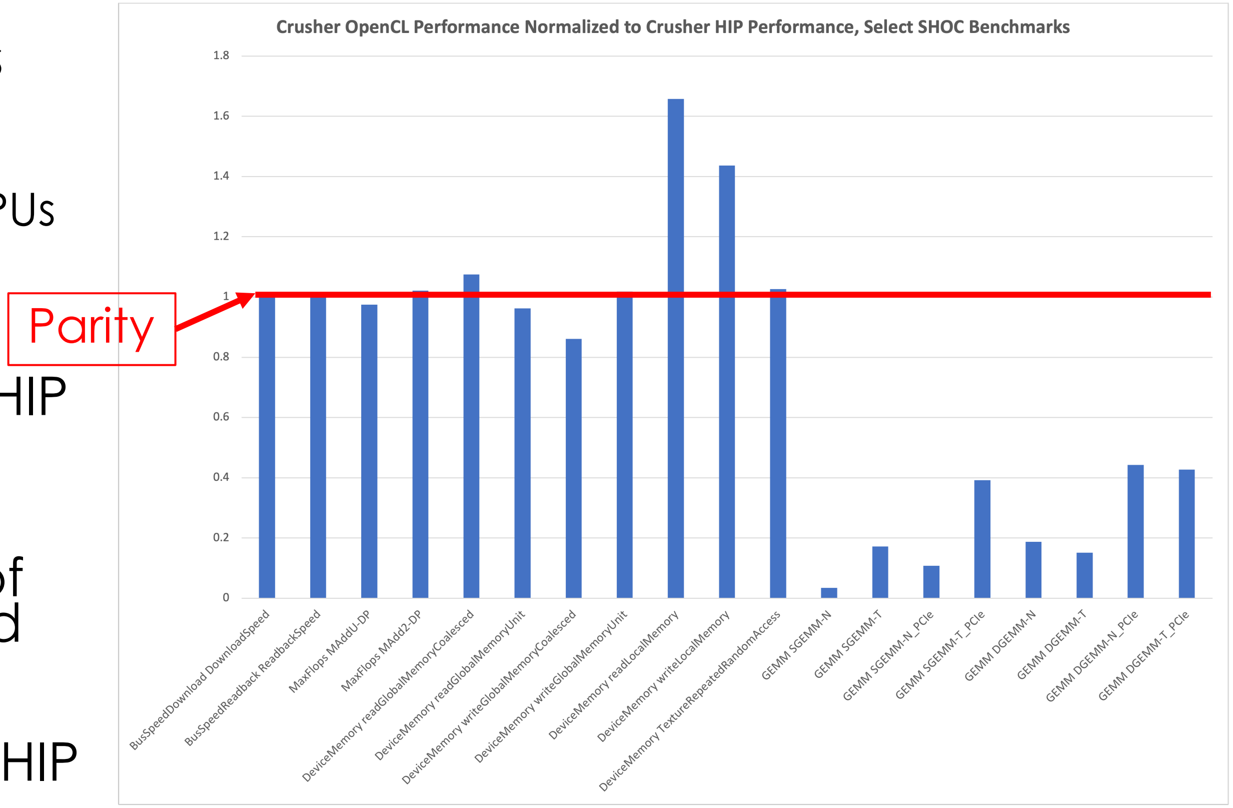
CLInfo output with NVIDIA-specific OpenCL driver on Summit

OpenCL on Summit: Experiences II

- Portable Computing Language (POCL) provides alternative to NVIDIA’s ICD
  - Uses CUDA runtime
  - In theory, can ingest SPIR-V
- *Have not yet demonstrated working build on Summit, but I think others have*
  - *I may be tripping up on the SPIR-V support*

OpenCL on Crusher/Frontier

- AMD supports OpenCL on x86_64 systems with MI250X GPUs (Crusher/Frontier) and MI100 (Spock) GPUs
  - ICDs installed for GPUs, apparently not for CPUs
  - Support dynamic compilation of OpenCL source code to AMDGCN
- OpenCL performance comparable with HIP performance for low-level SHOC benchmarks (MaxFlops, data transfers)
- GEMM performance gap shows benefit of using optimized hipBLAS library vs untuned OpenCL implementation
- Too many extreme differences (both pro-HIP and pro-OpenCL) to show results from higher-level benchmarks
  - No platform-specific optimizations or problem diagnosis yet attempted



Crusher OpenCL Performance Normalized to Crusher HIP Performance, Select SHOC Benchmarks

OpenCL on Aurora: Promising

- Intel traditionally has supported OpenCL and has championed SPIR-V
- Intel's oneAPI includes interoperability support between SYCL and OpenCL
- ALCF includes OpenCL in public lists of programming models planned to be available on Aurora

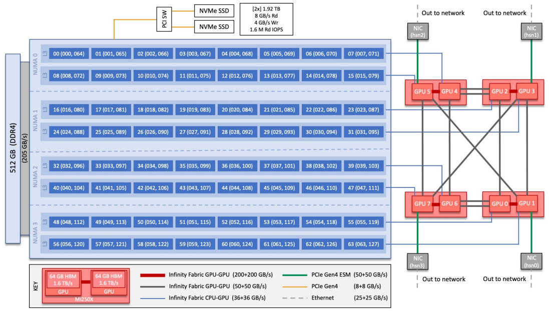
SYCL and DPC++

- SYCL: Khronos standard, C++-based, OpenCL’s spiritual successor
  - Originally had strong connection to SPIR/SPIR-V
- DPC++: part of Intel’s oneAPI, SYCL 1.2 plus useful extensions intended to improve productivity
  - Some extensions appear in SYCL 2020 standard

SYCL on Summit

- Some options for this “wrong way”:
  - hipSYCL: a SYCL 1.2 implementation built on HIP
    - CUDA for GPU, OpenMP for CPU
    - Have demonstrated this running on Summit with simple examples, e.g., matrix aX+Y
  - Intel’s Github LLVM staging repository includes DPC++ compiler sources
  - Found small number of build problems, e.g., reliance on CPUID instruction that isn’t supported on POWER9
  - Others have reported some success in working around for other non-x86_64 platforms, so may be possible soon on Summit
  - Also tried using CodePlay’s Community Edition to compile kernels to PTX code on spare x86_64+NVIDIA GPU system, transferring to Summit, and loading kernels via POCL – not successful

Frontier: SYCL/DPC++



00 (000, 064)	01 (001, 065)	02 (002, 066)	03 (003, 067)	04 (004, 068)	05 (005, 069)	06 (006, 070)	07 (007, 071)
08 (008, 072)	09 (009, 073)	10 (010, 074)	11 (011, 075)	12 (012, 076)	13 (013, 077)	14 (014, 078)	15 (015, 079)
16 (016, 080)	17 (017, 081)	18 (018, 082)	19 (019, 083)	20 (020, 084)	21 (021, 085)	22 (022, 086)	23 (023, 087)
24 (024, 088)	25 (025, 089)	26 (026, 090)	27 (027, 091)	28 (028, 092)	29 (029, 093)	30 (030, 094)	31 (031, 095)
32 (032, 096)	33 (033, 097)	34 (034, 098)	35 (035, 099)	36 (036, 100)	37 (037, 101)	38 (038, 102)	39 (039, 103)
40 (040, 104)	41 (041, 105)	42 (042, 106)	43 (043, 107)	44 (044, 108)	45 (045, 109)	46 (046, 110)	47 (047, 111)
48 (048, 112)	49 (049, 113)	50 (050, 114)	51 (051, 115)	52 (052, 116)	53 (053, 117)	54 (054, 118)	55 (055, 119)
56 (056, 120)	57 (057, 121)	58 (058, 122)	59 (059, 123)	60 (060, 124)	61 (061, 125)	62 (062, 126)	63 (063, 127)

- Have less experience trying these "wrong way" approaches on pre-Frontier systems so far
- AMD has traditionally supported OpenCL
  - But SPIR/SPIR-V support varies by product line - not supported on MI25/MI60
  - Options: POCL, "manual" conversion of SPIR-V to AMDGCN
- SYCL and DPC++
  - CodePlay funded to implement basic SYCL and DPC++ functionality for AMD GPUs in pre-Frontier systems
  - Intel LLVM repository
    - CPUID not an issue here
    - Reliant on SPIR-V tools/translator to convert to AMDGCN, perhaps via LLVM IR?
- Recall the importance of the ecosystem!

Back to General “Wrong”-ness

Dyn inst

- Recall Dyninst and the “Requires care” teaser? Lots of effort involved in portability and maintenance
  - Many combinations possible between CPU ISAs, operating systems, compilers, optimization levels
  - Must also be maintained over time as things change (e.g., new compiler versions)

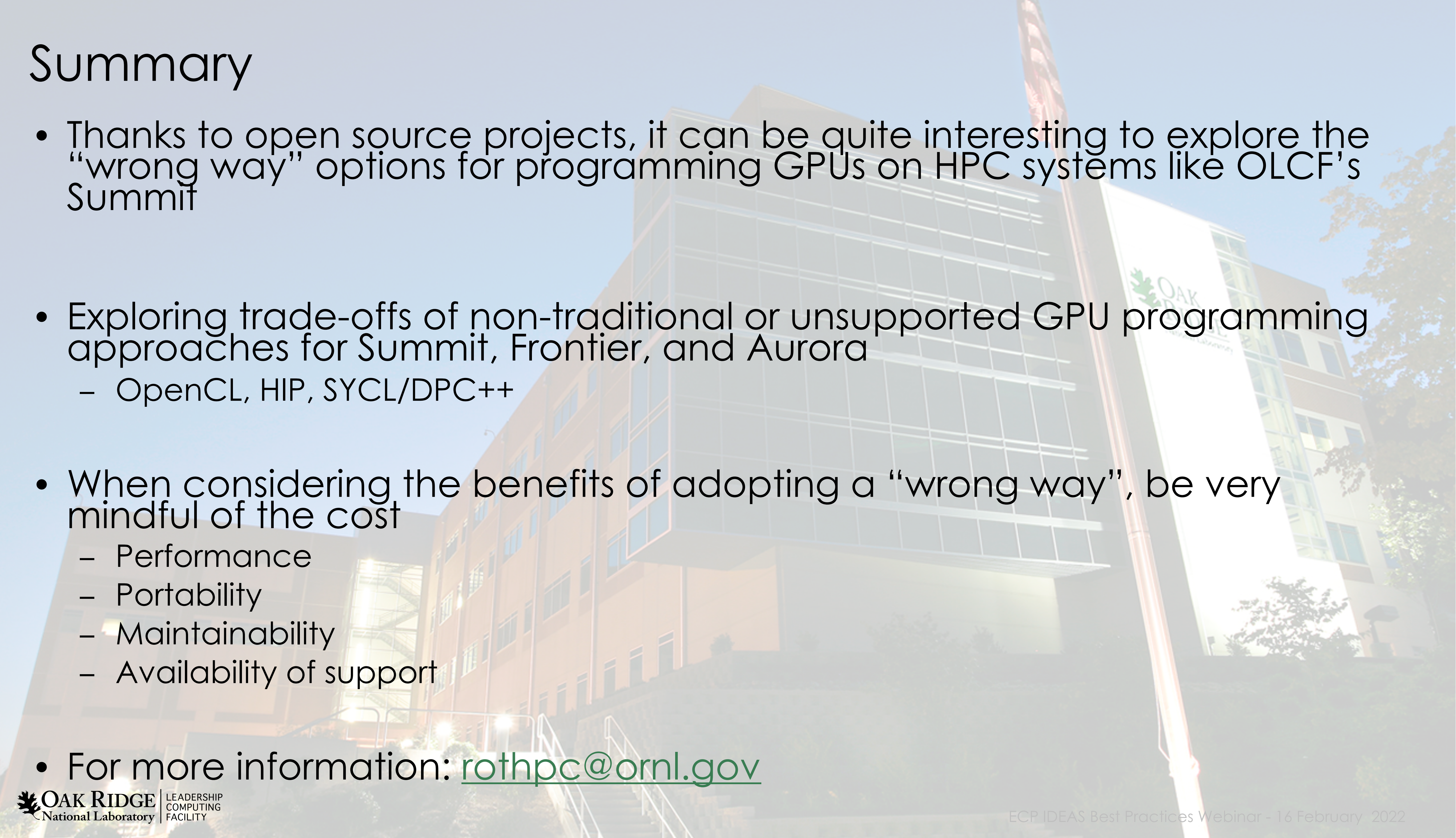
- ***Adopting a “wrong” way comes at a cost!***

Thoughtfully Compare Benefits and Costs

- How much are you and your project willing to devote, now and over time, to obtain the benefits of adopting a “wrong” way?
- Evaluate several factors when deciding
  - Performance impact
  - Portability (current and future)
  - Maintainability
  - Cost of conversion, including whether intermediate steps are possible
- My opinion: in general, the HPC community does not yet have tools or discipline to make strong quantitative benefit/cost comparisons
  - Followup question: how much do we care? How much do we actually ***have*** to care?
- Don’t let lack of small number of near-ubiquitous standards (a la MPI for communications) cause paralysis
  - Consider whether chosen approach is close enough to others to make it easy to transition if it just isn’t turning out as desired

Acknowledgements

- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.



Summary

- Thanks to open source projects, it can be quite interesting to explore the “wrong way” options for programming GPUs on HPC systems like OLCF’s Summit

- Exploring trade-offs of non-traditional or unsupported GPU programming approaches for Summit, Frontier, and Aurora
  - OpenCL, HIP, SYCL/DPC++

- When considering the benefits of adopting a “wrong way”, be very mindful of the cost
  - Performance
  - Portability
  - Maintainability
  - Availability of support

- For more information: rothpc@ornl.gov

References I

- CUDA Toolkit for NVIDIA GPUs: https://developer.nvidia.com/cuda-toolkit
- DPC++
  - Programming Guide: https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/oneapi-programming-model/data-parallel-c-dpc.html
  - Intel LLVM staging repository (including DPC++ implementation): https://github.com/intel/llvm
- Dyninst dynamic instrumentation project: https://www.dyninst.org. Also available via Spack.
- HIP C++ performance portability software
  - ROCm HIP implementation: https://github.com/ROCm-Developer-Tools/HIP
  - CHIP-SPV HIP implementation over OpenCL/Intel Level Zero runtime (subsumes HIPCL and HIPLZ): https://github.com/CHIP-SPV/chip-spv
  - HIP over Intel's Level Zero runtime (HIPLZ) on Argonne's JLSE systems: https://github.com/jz10/hip-training

References II

- Khronos Group standards consortium: https://www.khronos.org
- Kokkos C++ performance portability software: https://kokkos.org
- OLCF System User Guides/Quick-Start Guides: https://docs-internal.apps.granite.ccs.ornl.gov/systems/index.html
- OpenCL application programming interface and kernel language
  - Standard specification, links to software like SDK: https://www.khronos.org/opencl/
  - OpenCL CLInfo utility: https://github.com/Oblomov/clinfo
  - Portable Computing Language (POCL): http://portablecl.org
- OCCA C++ performance portability software: https://libocca.org

References III

- RAJA C++ performance portability software: https://github.com/LLNL/RAJA
- Spack package manager: https://spack.readthedocs.io
- SPIR-V portable binary intermediate language: https://www.khronos.org/spir/
- SYCL specification and resources: https://www.khronos.org/sycl/