ONEAPI DEVSUMMIT

Argonne NATIONAL LABORATORY

# HIP ON AURORA: BRINGING HIP TO ONEAPI

**HIP ON AURORA PROJECT TEAM**
Colleen Bertoni, **Brice Videau**



2022/05/27

# INCREASING THE PORTABILITY OF HIP APPLICATIONS

## Supporting HIP Applications on Intel GPUs

- HIP is a likely target for many large scale HPC applications
- Several recent and upcoming DOE platforms will support HIP:
  - 2 Exascale systems will feature AMD GPUs (OLCF, LLNL)
  - 3 Pre-exascale systems use NVIDIA GPUs (Summit, Perlmutter, Polaris)
- The upcoming Aurora Exascale system utilizes Intel GPUs
  - Intel is not providing support for HIP
  - CUDA/HIP codes targeting Intel GPUs typically create a SYCL implementation
- The ECP "HIP on Aurora" project is working to provide HIP on Intel GPUs
  - Unify Level Zero and OpenCL support, refactor HIP, create test suite, evaluate usages

# CODES WITH CUDA/HIP OR SYCL IMPLEMENTATIONS

## From ECP or the ALCF Early Science Program

| Codes Using SYCL and HIP |
|---|
| HACC |
| GRID |
| QUDA |
| NWChemEx |
| MetaHipMer |
| Uintah |
| NYX |
| MadGraph |
| FastCaloSim |
| NAMD |
| NekRs (via OCCA) |
| E3SM (via YALK) |
| GENE (via gtensor) |
| AMReX (AMR-Wind, MFIX, FLASH-X, WarpX, Pele) |

| Libraries Using SYCL and HIP |
|---|
| Kokkos |
| RAJA |
| SuperLU |
| Ginko |
| HYPRE |
| Sundials |
| FFTX |

| Codes with HIP only |
|---|
| Libcchem (GAMESS) |
| SHIFT |

ALCF and OLCF are also working with Codeplay on a HIP backend for Intel DPC++ Compiler

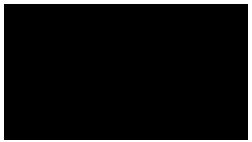
# CHIP-SPV TEAM COLLABORATORS

## Collaboration made up of National Labs, Academia, Industry










- National labs
  - Compiler and runtime researchers focused on HPC community
  - Users and developers of HPC applications
- Academia
  - Compiler and runtime researchers focused on new programming technologies
- Industry
  - Developers of AMD GPUs and HIP
  - Developers of Intel GPUs and oneAPI

# HIPLZ: HIP ON TOP OF LEVEL ZERO

# WHAT IS HIP?

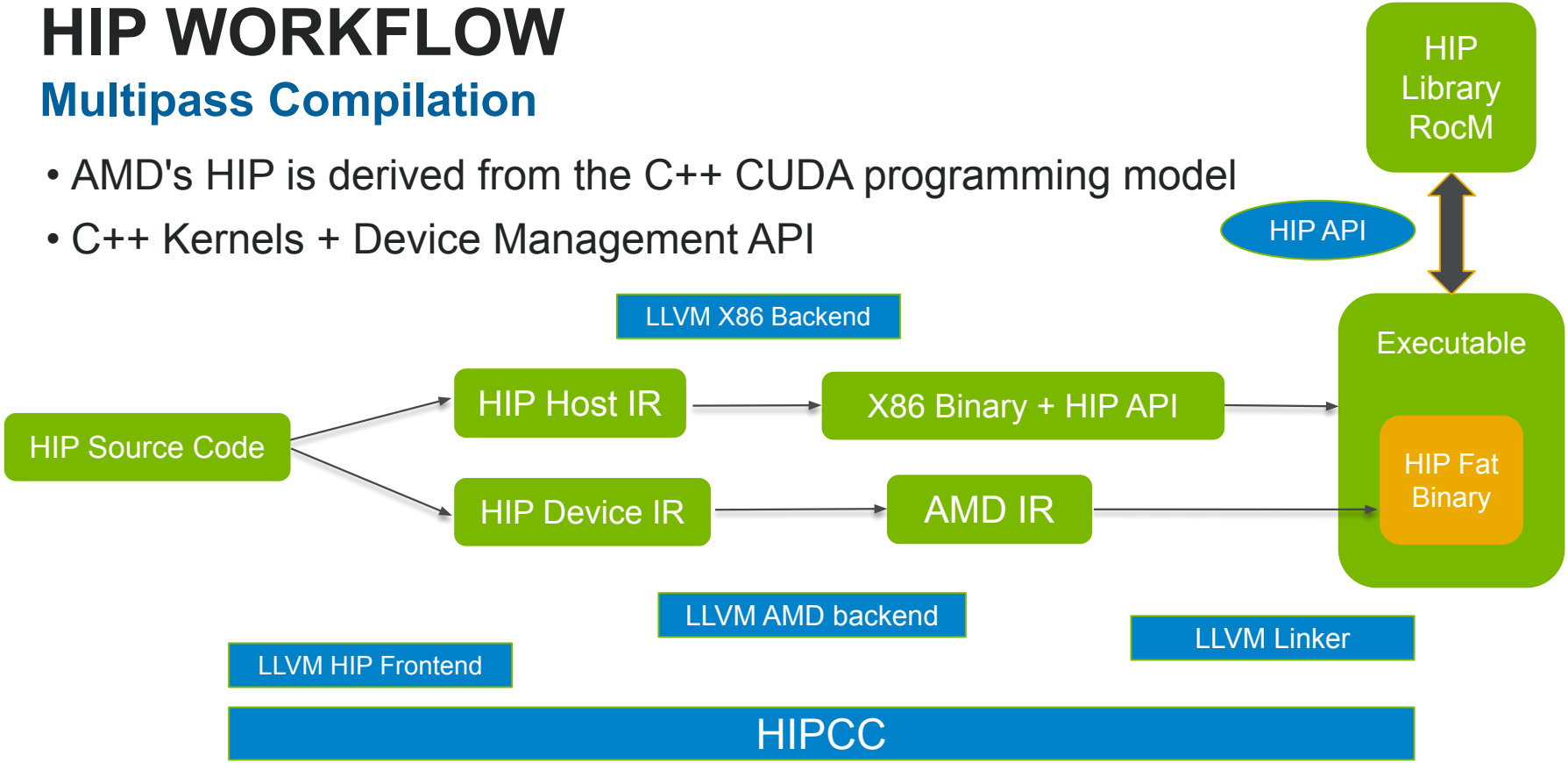
## Heterogeneous-Computing Interface for Portability

- HIP is AMD's portable GPU programming model
- Very similar to CUDA
  - CUDA programs can be automatically translated to HIP (via HIPIFY)
  - HIP programs run natively on AMD GPUs using ROCm + LLVM/Clang
  - HIP programs run natively on NVIDIA GPUs using CUDA + nvcc
- AMD provides GPU accelerated math libraries
  - hipBLAS, hipFFT, hipSPARSE, …
  - Wrappers around AMD and Nvidia GPU compute libraries
    - Example: hipBLAS wraps rocBLAS and cuBLAS
- High performance on both AMD and Nvidia GPUs by design

# HIP WORKFLOW

## Multipass Compilation

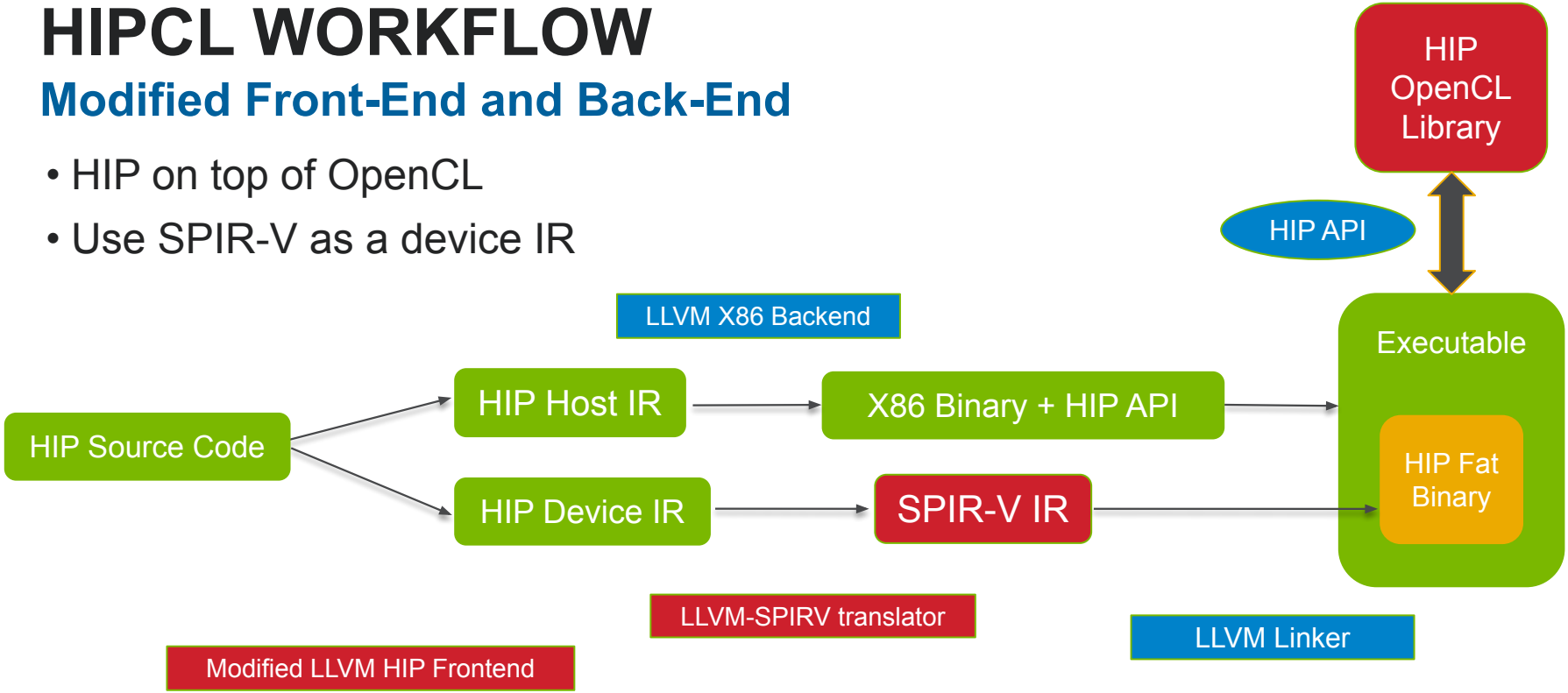
- AMD's HIP is derived from the C++ CUDA programming model
- C++ Kernels + Device Management API



# HIPCL WORKFLOW

## Modified Front-End and Back-End

- HIP on top of OpenCL
- Use SPIR-V as a device IR



# HIPLZ: A PROTOTYPE FOR CHIP-SPV

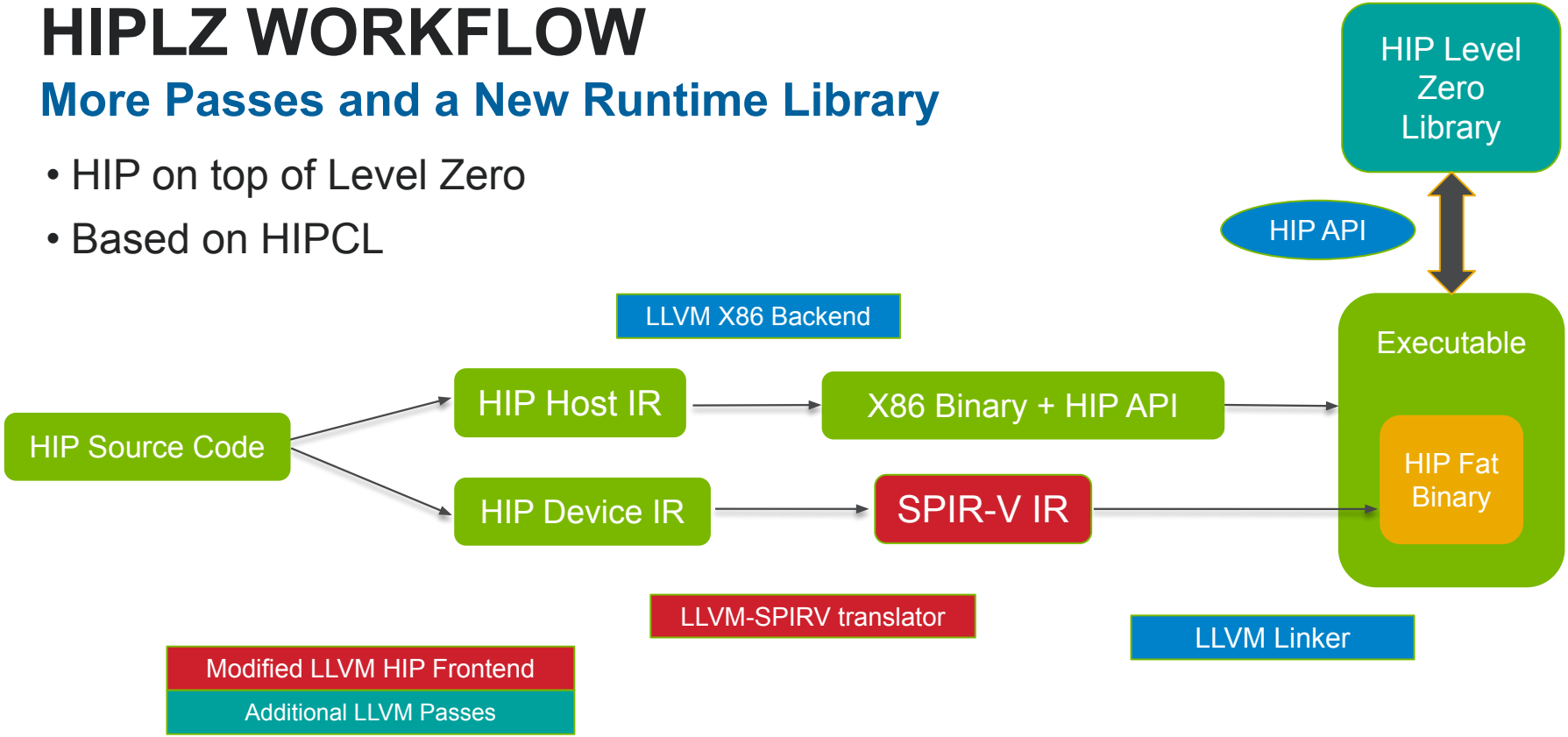
## HIPCL: OpenCL ==> HIPLZ: Level Zero

- Replace the OpenCL backend of HIPCL by Level Zero (L0):
  - L0: native offload API on Intel architectures,
  - Mostly one to one mapping between OpenCL API and Level-Zero API,
  - Supported on Aurora.
- Add support for new functionalities (and backport to HIPCL):
  - Textures,
  - Device linking, global variables,
  - Host callbacks,
  - Unified memory,
  - Interoperability with SYCL + hipBLAS prototype on top of oneMKL
  - Add missing API and math functions.
- Work happened on off-the-shelf hardware and the open source Intel oneAPI toolchain
- Heavy refactoring of the original HIPCL code
- Deprecated prototype available here: https://github.com/jz10/anl-gt-gpu

# HIPLZ WORKFLOW

## More Passes and a New Runtime Library

- HIP on top of Level Zero
- Based on HIPCL



# NEW SUPPORTED FEATURES

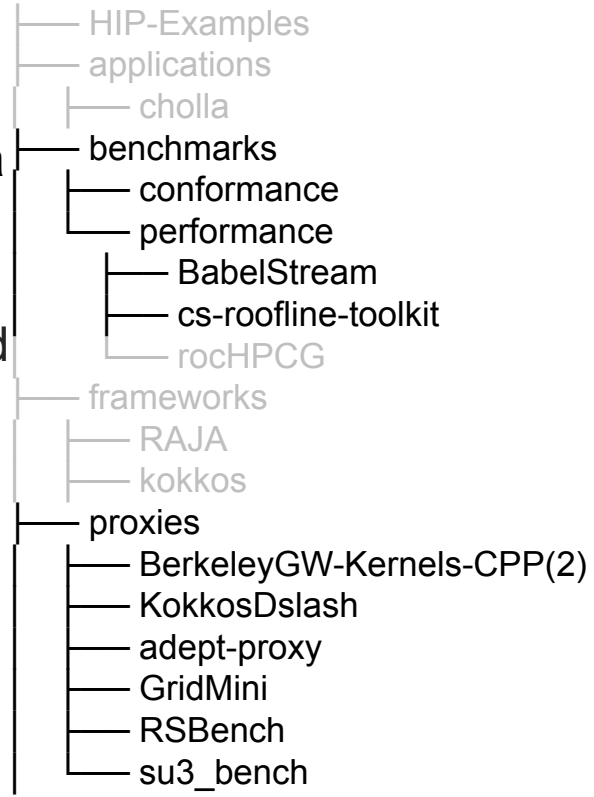
## Techniques used

- Textures:
  - hipTextureObject_t encapsulates textures and samplers in an opaque handle,
  - Technically not possible with SPIRV, but in practice works if calls are correctly inline by the device compiler,
  - On the host side textures and samplers are passed as individual kernel arguments,
  - No support for per dimension sampling type (unsupported in HIP as well).
- Device linking, global variables:
  - Compiler side is (almost) straightforward,
  - Accessing device global variables on the runtime side requires use of Intel extensions.
- Host callbacks:
  - Complex in Level-Zero, requires lot of synchronization and helper threads.
- Unified Memory:
  - Implemented on top of USM in Level-Zero and Intel USM extension in OpenCL.

# HIPLZ EVALUATION

## Test Suite

- Created a test suite with code of interest for Aurora
- Currently working well
  - SU3_bench (MILC microbenchmark)
  - ERT and BabelStream (Memory Bandwidth and Floating point benchmarks)
  - Sparkler (Miniapp for CoMet)
- Issues faced with the other proxies
  - Variables hard-coded for Nvidia architecture (like 32 for warp size)
  - Some HIP functions are not yet implemented (for example 3 argument shuffles)
- Deprecated repository:
  - https://github.com/jz10/hip-test_suite



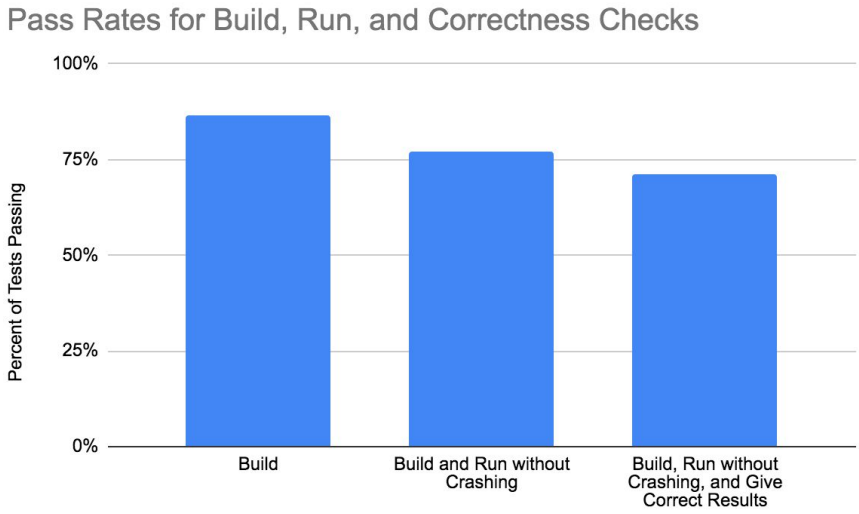
# PROGRESS REPORT

## Test Suite

- Assess the functionalities of HIPCL/HIPLZ
- Out of 52 tests,
  - 46 (88%) compile without errors
  - 40 (77%) compile and run without crashing
  - 37 (71%) compile, run without crashing, and give the correct answer.
- Main causes of failure:
  - cmake integration being out of date with AMD HIP's cmake
  - not all functions implemented yet, such as __shfl (3 argument version)
- Kokkos now compiles, thanks to new hipMemcpyToSymbol, etc. implementations!

```
$ git clone \
https://github.com/jz10/hip-test_suite.git

$ cd hip-test_suite
$ ./run_tests.sh
```



# TEST SUITE RESULTS

| Test | Build | Run | Correct Answer |
|---|---|---|---|
| BabelStream | Y | Y | Y |
| cs-roofline-toolkit | Y | Y | Y |
| cholla | N | | |
| KokkosDslash | N | | |
| su3_bench | Y | Y | Y |
| BerkeleyGW-FF | N | | |
| BerkeleyGW-GPP | Y | N | |
| add4 | Y | Y | Y |
| cuda-stream | Y | Y | Y |
| gpu-burn | Y | Y | |
| mini-nbody | Y | Y | Y |
| reduction | Y | Y | Y |
| rodinia_3.0 (18 tests) | N | N | N |
| rtm8 | Y | Y | Y |
| strided-access | Y | Y | Y |
| vectorAdd | Y | Y | Y |
| GPU-STREAM | Y | Y | |

| Test | Build | Run | Correct Answer |
|---|---|---|---|
| mixbench | Y | Y | Y |
| BinomialOption | Y | Y | Y |
| BitonicSort | Y | Y | Y |
| FastWalshTransform | Y | Y | Y |
| FloydWarshall | Y | Y | Y |
| HelloWorld | Y | Y | |
| Histogram | Y | Y | Y |
| MatrixMultiplication | Y | Y | Y |
| PrefixSum | Y | Y | Y |
| RecursiveGaussian | Y | Y | Y |
| SimpleConvolution | Y | Y | Y |
| dct | Y | Y | Y |
| dwtHaar1D | Y | Y | Y |
| kokkos | N | | |
| raja | N | | |
| adept-proxy | N | | |
| occa | Y | N | |

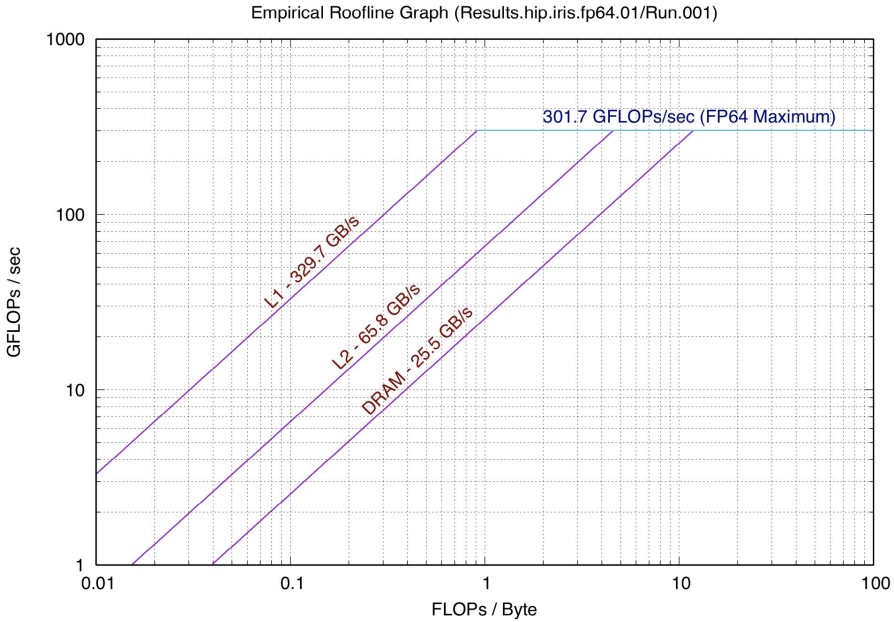
# HIGHLIGHTS: HIPCL AND HIPLZ ON INTEL GEN9

- **Empirical roofline toolkit**
  - Empirically measures bandwidth and peak performance

| | DRAM Bandwidth (GB/s) | FP64 peak (Gflop/s) | FP32 peak (Gflop/s) |
|---|---|---|---|
| **HIPCL** | 25.48 | 301.66 | 1235.39 |
| **OpenCL** | 25.77 | 299.12 | 1184.91 |
| **HIPLZ** | 25.84 | 303.22 | 1240.69 |

- **Su3_bench microbenchmark**
  - Kernel based on the SU(3) routine in the MILC Lattice Quantum Chromodynamics code

| | Total execution time (s) | Total GFLOP/s | Total GByte/s (GPU) |
|---|---|---|---|
| **HIPCL** | 30.26 | 29.93 | 22.17 |
| **OpenCL** | 31.59 | 28.67 | 21.24 |
| **HIPLZ** | 31.46 | 28.79 | 21.32 |




Argonne NATIONAL LABORATORY

# HIPCL + HIPLZ => CHIP-SPV

# HIP RESTRUCTURING BY AMD

- HIP Restructuring Objectives:
  - Allow independent backends in HIP,
  - Allow new backends to leverage CI tests,
  - Create a test suite derived from the HIPLZ test suite.
- HIP Restructuring:
  - HIP has been split into a common repository, and an AMD specific backend repository
  - CI tests have been reworked to allow running them in different environement
  - HIP test suit available on GitHub
  - hipcc and hipConfig have been ported to C++ and provide a plugin mechanism to extend to new backends

# THE CHIP-SPV PROJECT

## Compiler and Back-End

- Two distinct aspect
  - Upstream necessary front-end compiler modifications into Clang/LLVM
  - Create unified back-end supporting OpenCL and Level-Zero
- Front-end upstreaming:
  - Necessary patches are live in Clang14, adding a SPIR-V target to the LLVM/Clang HIP frontend
  - Relies on the *spirv-llvm* translator, but is ready to use the SPIR-V LLVM backend once it arrives
- CHIP-SPV backend (https://github.com/CHIP-SPV):
  - New passes added to diminish reliance on extensions and provide new functionalities
    - Global variables
    - Textures
    - printf, abort
  - Unified OpenCL and Level-Zero behind a common abstraction layer
  - Switching between Level-Zero and OpenCL can be done at runtime
  - 80 % pass rate on HIP unit tests

# CHIP-SPV COMPONENTS

| Name | Description | Open Source | Origin | Maintainer |
|---|---|---|---|---|
| HIP Frontend | LLVM HIP Frontend | Yes (GitHub) | AMD | LLVM/Clang + AMD |
| HIPSPV | HIP LLVM/Clang SPIR-V compiler backend | Yes (GitHub) | HIPCL/HIPLZ | LLVM/Clang |
| LLVM/SPIR-V Translator | Translation layer | Yes (GitHub) | Khronos | Khronos |
| HIP Common | HIP Frontend + CI + Tools | Yes (GitHub) | AMD/HIPLZ | AMD |
| HIP Test Suite | Test suite of HIP applications | Yes (GitHub) | AMD/HIPLZ | AMD |
| IGC | Intel Graphics Compiler | Yes (GitHub) | Intel | Intel |
| Level Zero | Intel new Runtime API | Yes (GitHub) | Intel | Intel |
| CHIP-SPV | New HIP backend for Intel GPUs | Yes (GitHub) | HIPCL/HIPLZ | ALCF |

# NEXT STEPS

## Real Application Support

- Continue the upstreaming work
  - Implement new HIP features as they are adopted
  - Continue pushing patches to LLVM (add required passes as they mature)
  - Continue the test suite work and HIP refactoring
- Find full-fledged HIP applications, identify gaps, and close them:
  - Implement missing runtime feature for applications
  - Solve compiler related gaps
  - Work on enabling required math libraries on top of oneAPI
- Investigate CHIP-SPV as a CUDA backend

# CONCLUSION AND PERSPECTIVES

# CONCLUSION

## oneAPI with SPIRV Support is an Awesome Portability Platform

- oneAPI and Level-Zero allowed:
  - Porting most features of HIP to Intel GPUs,
  - Leveraging existing oneAPI libraries,
  - While maintaining great performance.
- SPIR-V support enabled:
  - Leveraging Khronos tools,
  - The work to be upstreamed in LLVM,
  - Interchangeable Level-Zero and OpenCL backends,
  - Interest from other OpenCL vendors.

# PERSPECTIVES

## Convergence of Heterogeneous Programming Models

- Continues removing portability obstacles:
  - New SPIR-V and OpenCL extensions are being written (printf, abort),
  - Working with Intel to ease writing runtimes on top of Level-Zero,
  - Identify and close gaps in oneAPI math libraries functionalities and performances vs HIP equivalents.
- Investigate real applications:
  - ECP applications,
  - Anyone interested in running specific HIP applications on Intel GPUs?
- SPIR-V LLVM backend in Clang15
- Investigate CUDA on SPIR-V backend using Clang toolc-chain

# QUESTIONS?

# ACKNOWLEDGMENTS

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY

# ACKNOWLEDGMENT


- The HIP on Aurora team is composed of: Colleen Bertoni (ANL), Kevin Harms (ANL), Scott Parker (ANL), Jisheng Zaho (GaTech), Jeffrey Young (GaTech), Wael Elwasif (ORNL), Phillip Roth (ORNL), Rahulkumar Gayatri (NERSC), Pekka Jääskeläinen (Parmance), Henry Linjamäki (Parmance), Paulius Velesko (PaganLC), Diwakar Das (AMD), Rahul Garg (AMD), Angela Wang (AMD),  Peng Tu (Intel), and all those I forgot, apologies.
- This work was supported by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration). We also gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.