Argonne NATIONAL LABORATORY

CHIPSTAR: A HIP IMPLEMENTATION FOR AURORA

HIP ON AURORA PROJECT TEAM
Colleen Bertoni, Brice Videau

2024/08/21

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 (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 ON AURORA PROJECT OBJECTIVES

Support HIP Applications on Intel GPUs

- 2 Exascale systems will feature AMD GPUs (Frontier OLCF, El Capitan LLNL)
  - HIP is a likely target for many exascale applications
  - 3 Pre-exascale systems use Nvidia GPUs (Summit, Perlmutter, Polaris)
- What about Aurora with its Intel GPUs?
  - Bring HIP support to Aurora
- HIP on Aurora project, originally lead by Hal Finkel:
  - Implement HIP on top of Level Zero (Intel native API on Aurora) or OpenCL;
  - Restructure HIP to accept several backend plugins;
  - Create a test suite with code of interest to ECP;
  - Support selected ECP applications;
  - Implement HIP math libraries on top of oneMKL;
  - Investigate CUDA support.

HIP ON AURORA TEAM COLLABORATORS

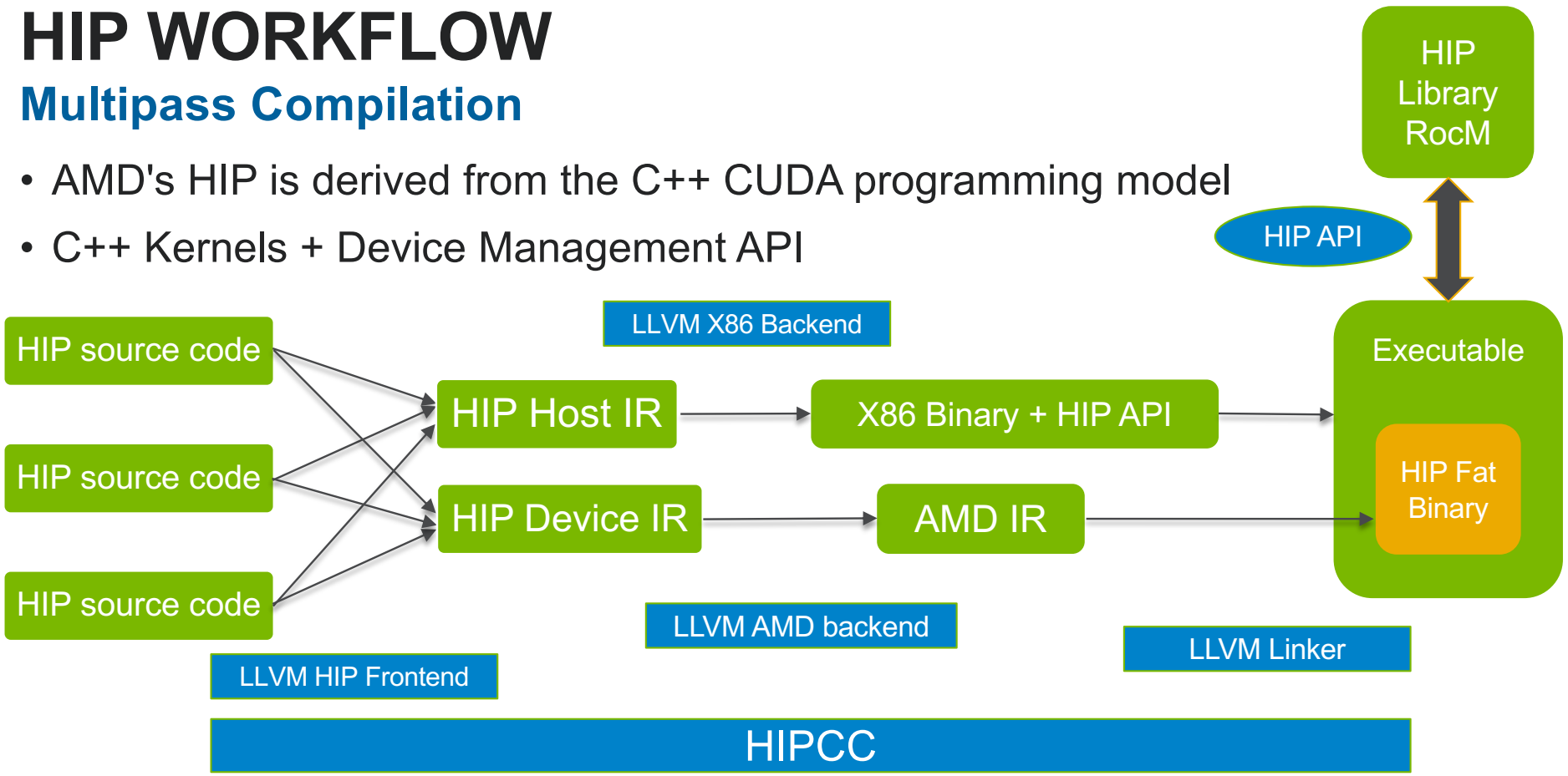
Team is made up of National Labs, Academia, Industry

- 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 HIP
  - Developers of Intel GPUs

HIP WORKFLOW

Multipass Compilation

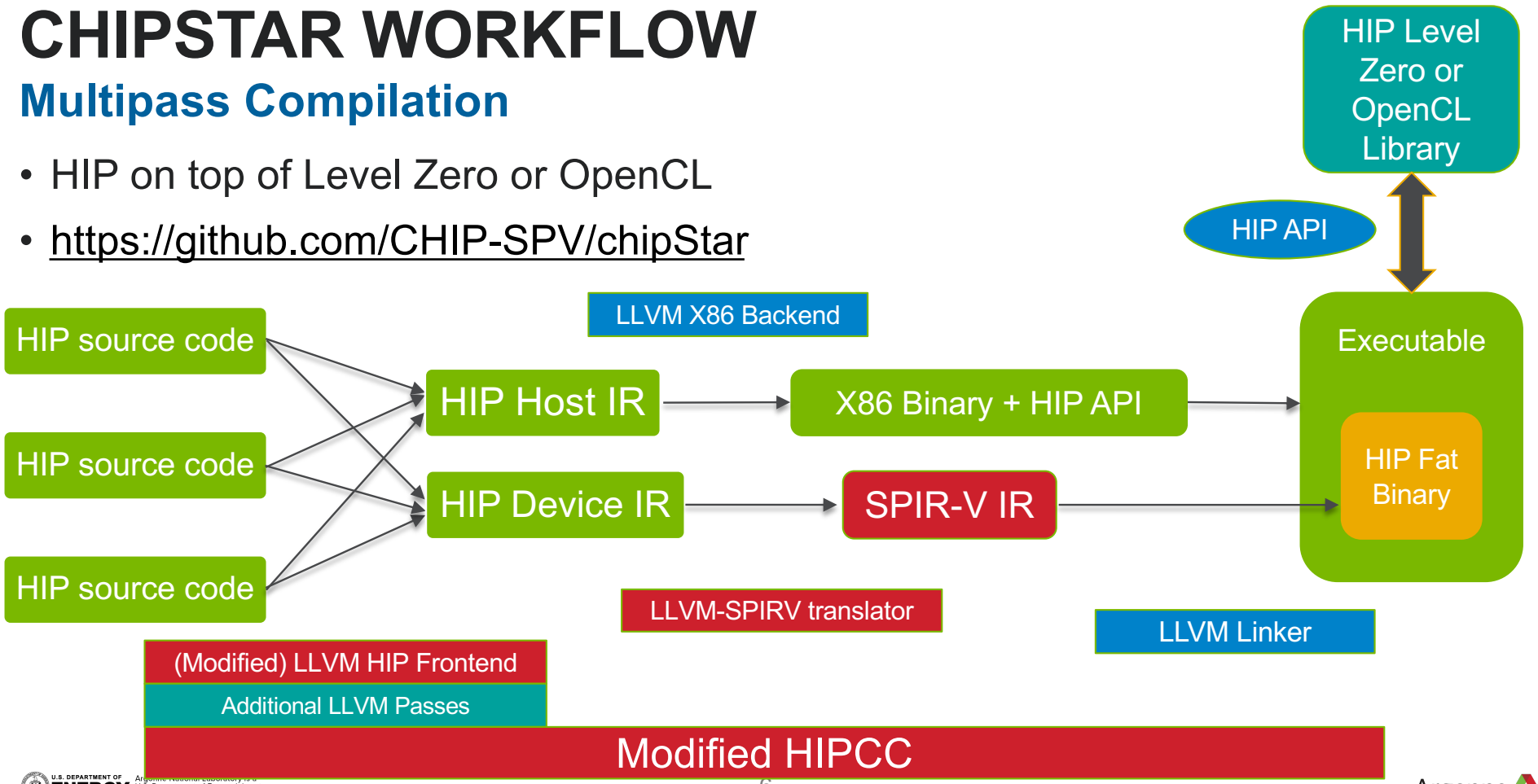
- AMD's HIP is derived from the C++ CUDA programming model
- C++ Kernels + Device Management API

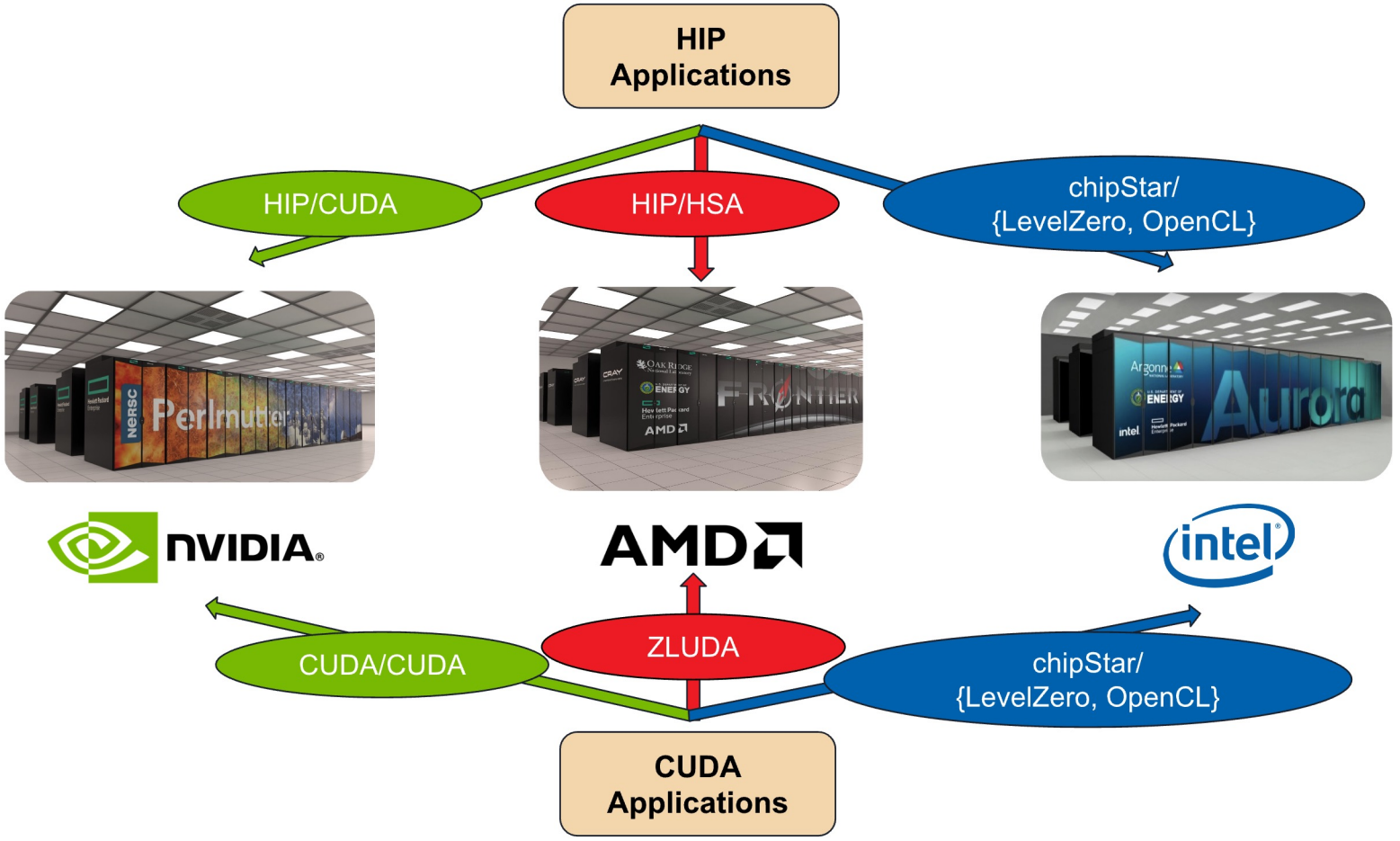


CHIPSTAR WORKFLOW

Multipass Compilation

- HIP on top of Level Zero or OpenCL
- https://github.com/CHIP-SPV/chipStar




HIP
Applications
HIP/CUDA
HIP/HSA
chipStar/
{LevelZero, OpenCL}
Perlmutter
FRONTIER
Aurora
NVIDIA
AMD
intel
CUDA/CUDA
ZLUDA
chipStar/
{LevelZero, OpenCL}
CUDA
Applications

SUPPORTED SOFTWARE

CHIPSTAR

A HIP and CUDA Implementation on top of Level Zero and OpenCL

- Available online: https://github.com/CHIP-SPV/chipStar
- Built from open source components:
  - HIP Common
  - HIP Tests
  - ROCm-Device-Libs
  - LLVM (15, 16, 17)
  - SPIRV LLVM Translator
- GitHub action CI leverages AMD unit tests
- Version 1.1 released early 2024, focus on performance improvements
- Currently supported by ALCF (through PaganLC) and Intel.
- CUDA is experimentally supported by converting CUDA calls to chipStar calls

SUPPORTED LIBRARIES

Provide HIP Math Libraries for Applications

- Actively supported and built on top of oneMKL, thank to oneAPI SYCL interoperability
- Unified architecture to interface with SYCL oneMKL
- Delivered to ECP
  - hipRTC
  - hipBLAS
  - hipSOLVER
- Work in progress
  - hipFFT
  - hipCUB
- Prospective
  - hipRand

SUPPORTED APPLICATIONS

Help ECP, ESP, and others to use HIP on Aurora

- Actively engaged with application teams
- Functional, optimizing performance
  - CP2K (several CP2K apps are supported, more are coming)
  - LibCEED (some quirks exist for some patterns)
  - GAMESS
- In progress
  - ExaBiome
- Prospective, always looking for more
  - LS_DALTON (TAL_SH)
  - DIRAC (TAL_SH)
  - TRITON
  - OpenMM

LIMITATIONS

Most Common Encountered Issues

- Kernel limitations (apply to SYCL as well)
  - Some masked group functions are not supported
    - 3 argument shuffles
    - HW limitation, working on emulation
  - Early exit from thread will cause issues
    - HW limitation
    - Working on compiler pass to alleviate simple cases
- Missing libraries
  - Supporting new libraries can be done, some are in progress
  - Intel oneMKL and other libraries usually map one to one with CUDA libraries
- Bugs, they should be reported and the project will address them

BENCHMARKING WITH HECBENCH

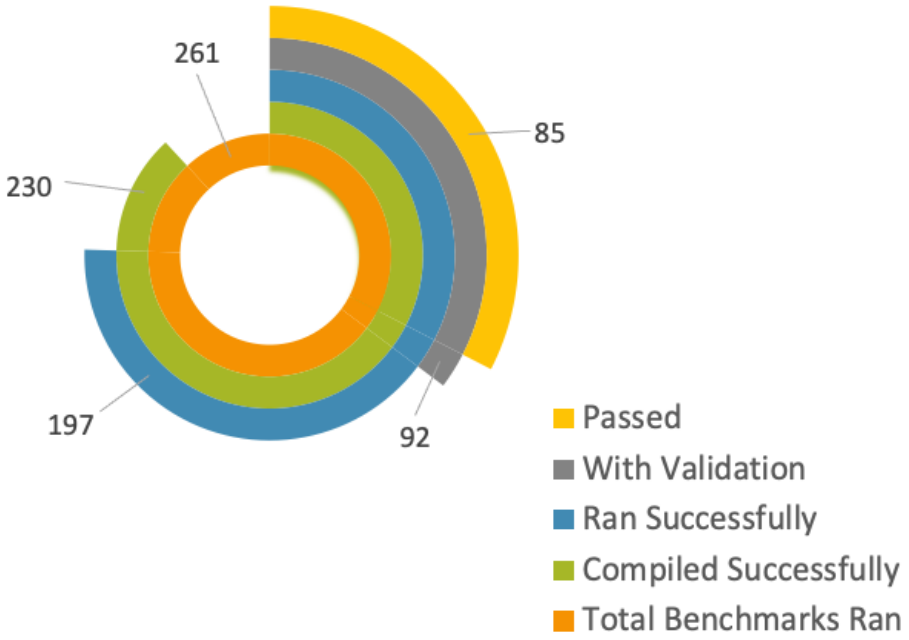
EVALUATING CHIPSTAR WITH THE HECBENCH BENCHMARK SET

- Goals:
  - Assess the functionality of chipStar for HIP and CUDA APIs by looking at the pass rate of the benchmark set
  - Report any bugs found
  - Assess the performance of chipStar by comparing the performance of HIP codes running with chipStar to SYCL codes running with Intel oneAPI on Intel GPUs
- We chose to use HeCBench (https://github.com/zjin-lcf/HeCBench) since it is extensive and has the same code written in multiple different languages

BENCHMARKING WITH HECBENCH: HIP API

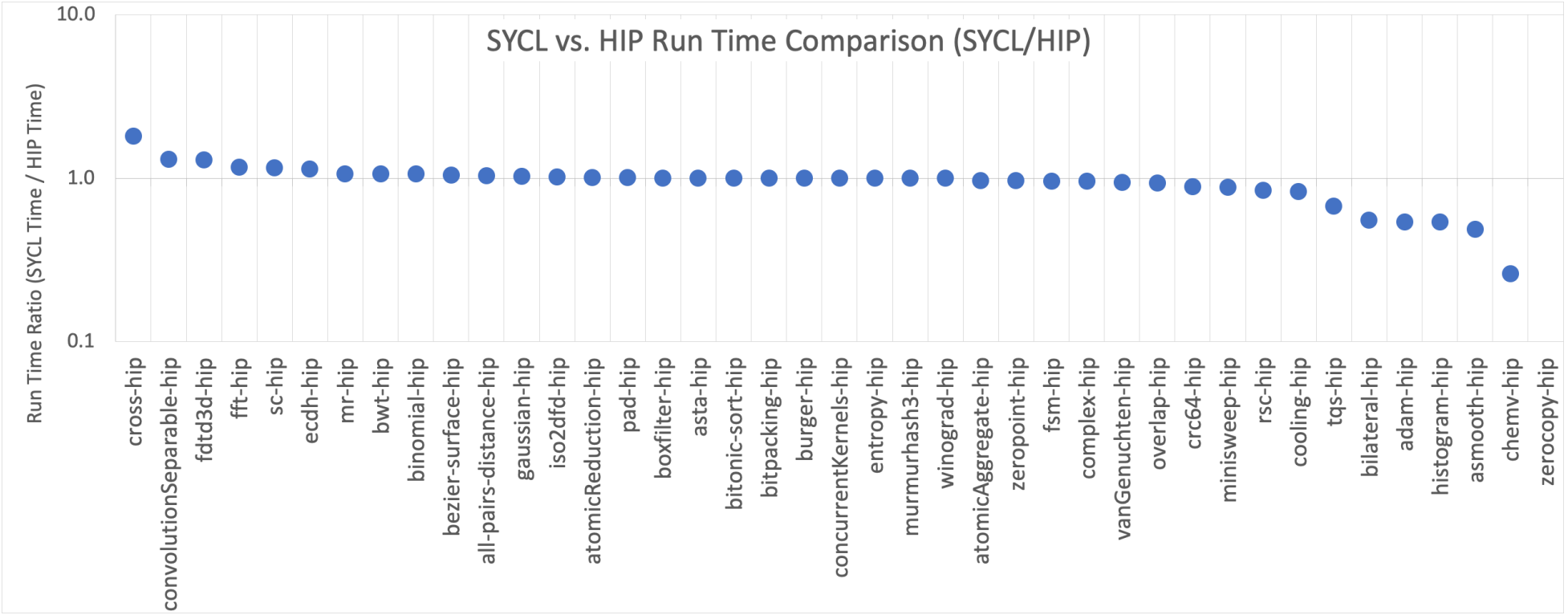
HECBENCHMARK: HIP FUNCTIONALITY

- Running chipStar on PVC, 88.12% of (230/261) HIP benchmarks compiled successfully, and within the compiled ones, 85.65% (197/230) ran successfully. For those ran and had result verification, 92.39% (85/92) passed.
- The errors are a variety of Intel runtime errors, hangs, incorrect answers, and chipStar errors



Credit to Kali Balleda (Intel)

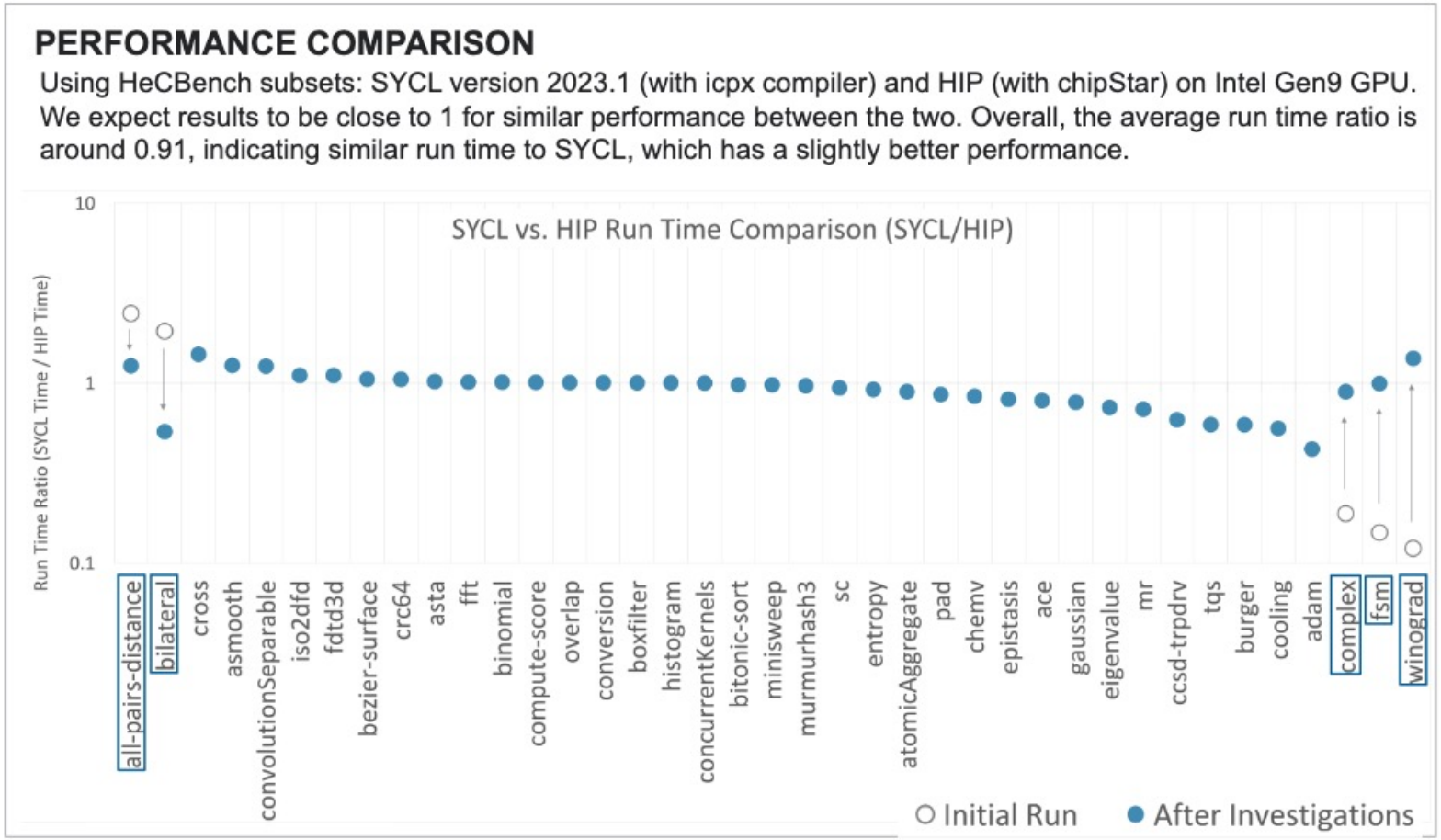
HECBENCHMARK: HIP PERFORMANCE ON PVC



Note:
- > 1.0 means HIP is better
- Close to 1.0 means HIP and SYCL are similar
- < 1.0 means HIP is worse

Credit to Kali Balleda (Intel)

HECBENCHMARK: PERFORMANCE ON GEN9



Credit to Jenny Chen (ALCF summer intern)

EVALUATING CUDA SUPPORT WITH HECBENCH

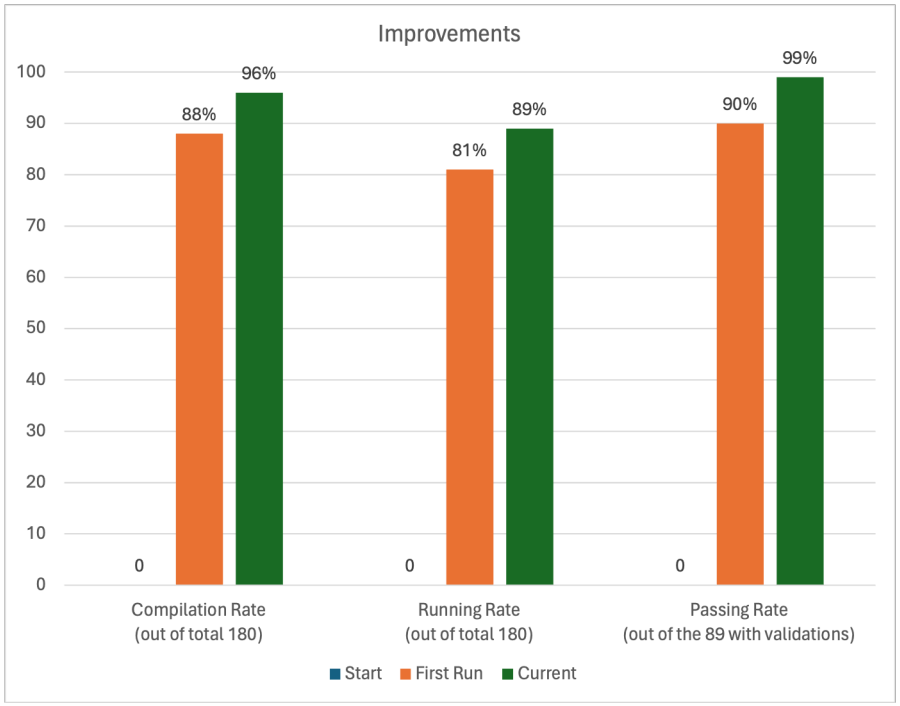
HECBENCHMARK: CUDA FUNCTIONALITY

Start
- Benchmarks couldn't be run because of the nvcc wrapper script naming
- Modified benchmarking script to adapt and successfully ran HeCBench with chipStar

First Run
- Start running benchmarks to identify bugs
- A total of 180 CUDA benchmarks were run

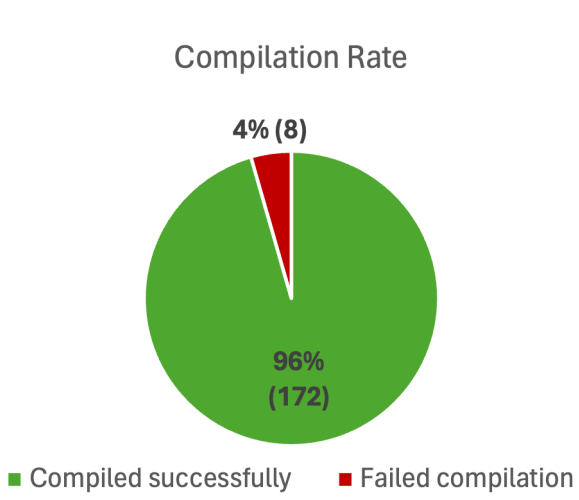
Fixing
- Testing and digging into errors
- Fixing, proposing solutions, and reporting issues

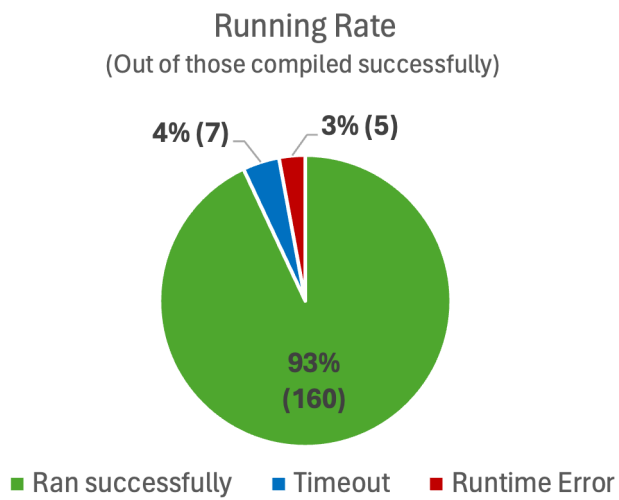
Current Result
- Compilation, running, and passing rates are greatly improved

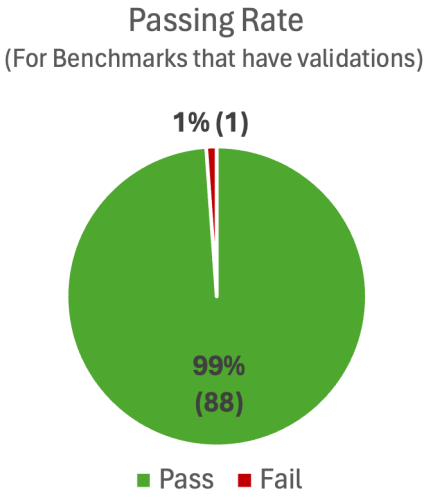


RESULTS

Overall Results – Compilation, Running, Passing Rates







Credit to Jenny Chen (ALCF summer intern)

APPLICATION HIGHLIGHT: GAMESS

GAMESS: INTRODUCTION

- General Atomic and Molecular Electronic Structure System (GAMESS) is a quantum chemistry software package
  - Contains a CUDA/HIP library targeting Summit and Frontier
- To port to chipStar, we focused on the Hartee-Fock (HF) method.
  - Over **20,000 lines of HIP/CUDA kernel code** and multiple hipBLAS and hipSOVLER routines (**hipblasDscal, hipblasDgemm, hipblasDcopy, hipblasDaxpy, hipblasDdot, hipblasDgemv, hipblasDgeam, hipsolverDsyevd**.)
- The kernels use **shared memory with __syncthreads()** calls to ensure copying values from global memory to shared memory completed for the threadblock before using it.

GAMESS: TIMING COMPARISON

- Porting effort was relatively low
  - Build files are very similar
    - We could use the same cmake for HIP on Nvidia, AMD, and Intel HW
  - One issue was due to kernel thread synchronization: In CUDA group barriers don't count exited threads but in LevelZero and OpenCL this is a undefined behavior and can deadlock

Hardware	SCF time (s)
Nvidia A100	1.998
AMD MI250 (one GCD)	26.09*
Intel PVC (one tile)	5.31

* This is due to the eigensolve in hipBLAS being relatively slow.

TRY IT OUT ON JLSE OR SUNSPOT!

QUICK START: CHIPSTAR ON JLSE

```
> # get a node
> qsub -q iris -A ${project} -n 1 -t 120 -I

> # put the appropriate modules in your path
> module use /soft/modulefiles/

> # puts chipStar 1.2 Preview in your environment
> module load chipStar/1.2-preview

> # clone training repo
> git clone https://github.com/jz10/hip-training.git
> cd hip-training/simple
```

QUICK START: CHIPSTAR ON JLSE

```
> # compile and run CUDA and HIP examples with chipStar
> make
hipcc -o saxpy_hip saxpy_hip.cpp
nvcc -o saxpy_cuda saxpy_cuda.cu
warning: cucc is a work-in-progress. It is incomplete and may behave incorrectly.

> ./saxpy_hip
Max error: 0.000000
> ./saxpy_cuda
Max error: 0.000000
```

QUICK START: CHIPSTAR ON SUNSPOT

```
> # get a node
> qsub -q workq -A ${project} -l walltime=0:120:00 -l select=1 -I

> # put the appropriate modules in your path
> module use /home/bertoni/modulefiles/

> # puts chipStar 1.2 Preview in your environment
> module load chipStar/1.2-preview

> # clone training repo
> git clone https://github.com/jz10/hip-training.git
> cd hip-training/simple
```

QUICK START: CHIPSTAR ON SUNSPOT

```
> # compile and run CUDA and HIP examples with chipStar
> make
hipcc -o saxpy_hip saxpy_hip.cpp
nvcc -o saxpy_cuda saxpy_cuda.cu
warning: cucc is a work-in-progress. It is incomplete and may behave incorrectly.

> ./saxpy_hip
Max error: 0.000000
> ./saxpy_cuda
Max error: 0.000000
```

QUICK START: PROFILING AND DEBUGGING

- For profiling and debugging, on Sunspot you can use iprof with "module load thapi" just like with Intel OneAPI SYCL and OpenMP!

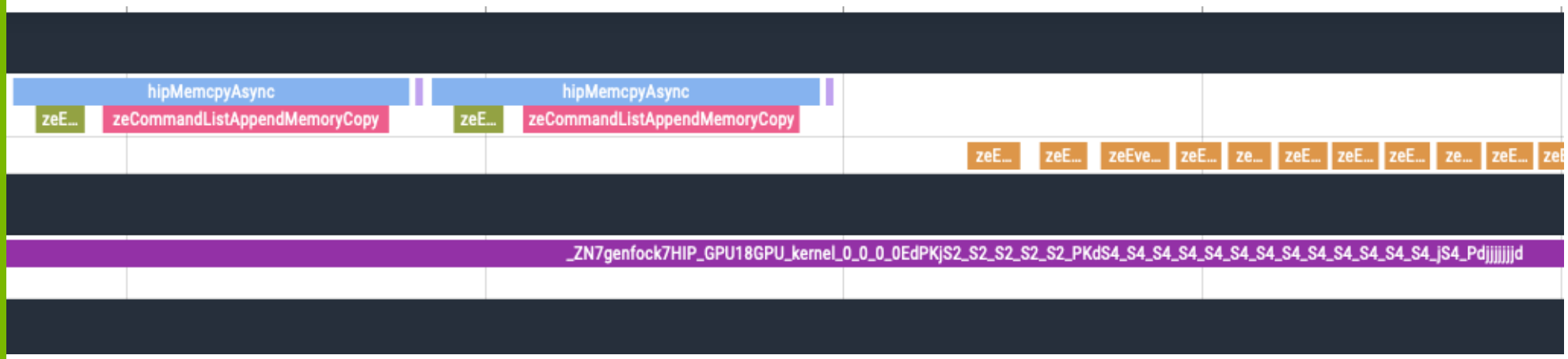
```
> LTTNG_UST_HIP_LIBAMDHIP64=libCHIP.so iprof ./saxpy_hip
Max error: 0.000000
...
BACKEND_HIP,BACKEND_ZE | 1 Hostnames | 1 Processes | 2 Threads |

                                Name |     Time | Time(%) | Calls |  Average |
          __hipUnregisterFatBinary | 542.69ms |  53.85% |     1 | 542.69ms |
            __hipRegisterFatBinary | 306.79ms |  30.44% |     1 | 306.79ms |
                   hipLaunchKernel |  69.53ms |   6.90% |     1 |  69.53ms |
                    zeModuleCreate |  69.09ms |   6.86% |     1 |  69.09ms |
                         hipMalloc |   6.61ms |   0.66% |     2 |   3.31ms |
zeCommandQueueExecuteCommandLists |   3.75ms |   0.37% |    12 | 312.17us |
                         hipMemcpy |   2.81ms |   0.28% |     3 | 936.32us |
    zeCommandListAppendMemoryCopy |   1.34ms |   0.13% |     5 | 267.81us |
   ...
```

QUICK START: PROFILING AND DEBUGGING 2

Timeline with iprof from libcchem (GAMESS)

```
LTTNG_UST_HIP_LIBAMDHIP64=libCHIP.so iprof -l - mpriun -n 2 …
```



CHIPSTAR INSTALLATION

CHIPSTAR INSTALLATION

- Full instructions here:
  - https://github.com/CHIP-SPV/chipStar?tab=readme-ov-file#prerequisites
- Prerequisites:
  - Development files for OpenCL and/or Level-Zero
  - A working OpenCL or Level-Zero device driver
    - POCL or Intel OpenCL CPU work if you want to try on CPU
  - Optionally oneAPI + oneMKL for SYCL interoperability and libraries
- Overview:
  - Build and install LLVM/Clang with SPIRV-LLVM-Translator support
  - Build and install chipStar
  - Optionally Build and Install additional math libraries

CONCLUSION

CONCLUSIONS

chipStar is HIP for Aurora

- chipStar 1.0 (delivered almost 50 ECP MS)
  - Working HIP implementation
  - Supporting selected applications and libraries
  - Competitive performances with SYCL
- chipStar 1.1, focusing on performance improvements
- Upcoming chipStar 1.2, experimental CUDA, improve performance and stability
- Deployed on our platforms, can leverage Level Zero or OpenCL
  - Sunspot (PVC)
  - JLSE (PVC, Iris)
- Works on your laptop if it uses an Intel GPU
- But can also be used on CPU using OpenCL
  - POCL, Intel OpenCL CPU driver

PERSPECTIVES

Exciting Possibilities

- Native CUDA support
  - Experimental but promising
- Support for more
  - Applications: collaborating with new ECP and ESP teams
  - Features: HIP is constantly evolving, focusing on Applications requirements
  - Libraries as required by applications
  - Platforms
    - Extensions to OpenCL to better support HIP and CUDA
    - ARM is under investigation
    - Rusticl
- Looking for Additional funding through the S4PST SSO project

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

ACKNOWLEDGMENTS

- This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.
- 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