IWOCL / SYCLCON 2020

Argonne
NATIONAL LABORATORY

EVALUATING THE PERFORMANCE OF THE HIPSYCL TOOLCHAIN FOR HPC KERNELS ON NVIDIA V100 GPUS

BRIAN HOMERDING
Argonne National Laboratory
Speaker

JOHN TRAMM
Argonne National Laboratory

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

HPC LEADERSHIP COMPUTING SYSTEMS

- Summit [1] – Oak Ridge National Laboratory
  - IBM CPUs
  - NVIDIA GPUs
- Aurora [2] – Argonne National Laboratory
  - Intel CPUs
  - Intel GPUs
- Frontier [3] – Oak Ridge National Laboratory
  - AMD CPUs
  - AMD GPUs

- Increasing in diversity

TECHNOLOGIES USED IN THIS STUDY

- CUDA [4] – supported on Summit.
  - Designed to work with C, C++ and Fortran.
  - Provides scalable programming by utilizing abstractions for the hierarch of thread groups, shared memories and barrier synchronization.
- SYCL [5] – supported on Aurora.
  - Builds on the underlying concepts of OpenCL while including the strengths of single-source C++.
  - Includes hierarchical parallelism syntax and separation of data access from data storage.
- hipSYCL [6] – SYCL compiler targeting AMD and NVIDIA GPUs.
  - Aksel Alpay - https://github.com/illuhad/hipSYCL

HIPSYCL

- Provides a SYCL 1.2.1 implementation built on top of NVIDIA CUDA / AMD HIP.

- Includes two components.
  - SYCL runtime on top of CUDA / HIP runtime.
  - Compiler plugin to compile SYCL using CUDA frontend of Clang.

- Building on top of CUDA allows us to use the NVIDIA performance analysis toolset.

OUR CONTRIBUTIONS

1. We implement a SYCL variant of the RAJA Performance Suite [7] and port two HPC mini-apps to CUDA and SYCL.
2. We collect performance data on the RAJA Performance Suite for the programming models and toolchains of interest.
3. We investigate significant performance differences found in the benchmark suite.
4. We analyze the performance of two HPC mini-apps of interest: an N-body mini-app and a Monte Carlo neutron transport mini-app.

BENCHMARKS

- RAJA Performance Suite
  - Collection of benchmark kernels of interest to the HPC community.
  - Provides many small kernels for collecting many data points.

- N-Body [8]
  - Simple simulation application for a dynamical system of particles.

- XSBench [9]
  - Computationally representative of Monte Carlo transport applications.

RAJA PERFORMANCE SUITE

Collection of performance benchmarks with RAJA and non-RAJA variants.

Checksums verified against serial execution.

- Basic (simple)

  DAXBY, IF_QUAD, INIT3, INIT_VIEW1D, INIT_VIEW1D_OFFSET, MULADDSUB, NESTED_INIT, REDUCE3_INT, TRAP_INT

- Stream (stream)

  ADD, COPY, DOT, MUL, TRIAD

- LCALS (loop optimizations)

  DIFF_PREDICT, EOS, FIRST_DIFF, HYDRO_1D, HYDRO_2D, INT_PREDICT, PLANCKIAN

- PolyBench (polyhedral optimizations)

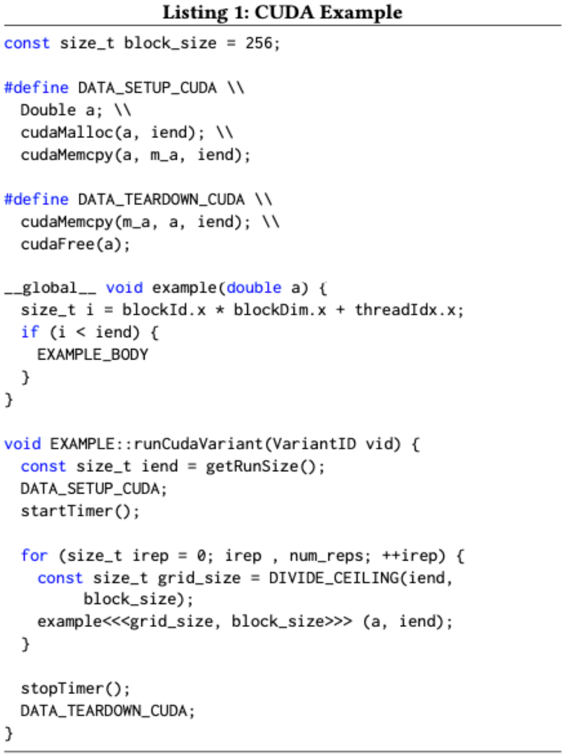
  2MM, 3MM, ADI, ATAX, FDTD_2D, FLOYD_ARSHALL, GEMM, GEMVER, GESUMMV, HEAT_3D, JACOBI_1D, JACOBI_2D, MVT

- Apps (applications)

  DEL_DOT_VEC_2D, ENERGY, FIR, LTIMES, LTIMES_NOVIEW, PRESSURE, VOL3D

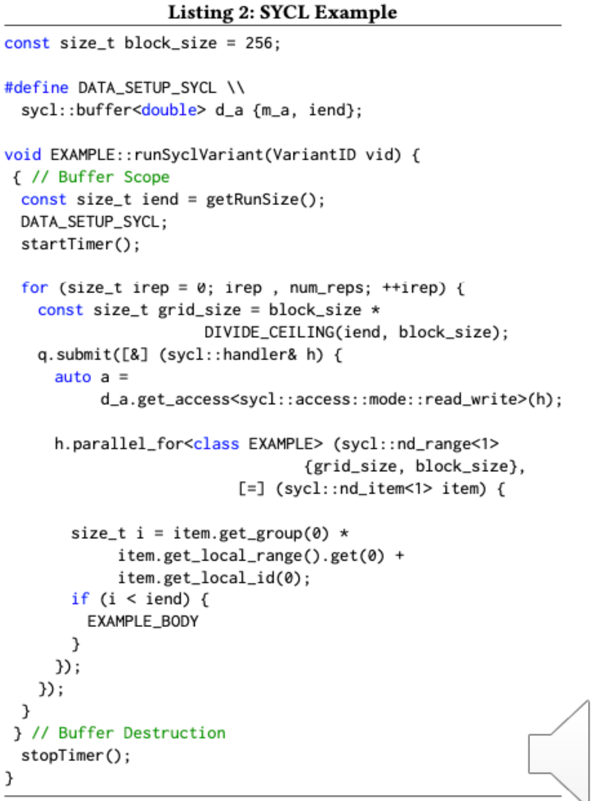
PORTING FOR COMPARABILITY

- Block size and grid size
- Indexing
- Memory management



Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \\
  Double a; \\
  cudaMalloc(a, iend); \\
  cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \\
  cudaMemcpy(m_a, a, iend); \\
  cudaFree(a);

__global__ void example(double a) {
  size_t i = blockId.x * blockDim.x + threadIdx.x;
  if (i < iend) {
    EXAMPLE_BODY
  }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
  const size_t iend = getRunSize();
  DATA_SETUP_CUDA;
  startTimer();

  for (size_t irep = 0; irep , num_reps; ++irep) {
    const size_t grid_size = DIVIDE_CEILING(iend,
          block_size);
    example<<<grid_size, block_size>>> (a, iend);
  }

  stopTimer();
  DATA_TEARDOWN_CUDA;
}
```

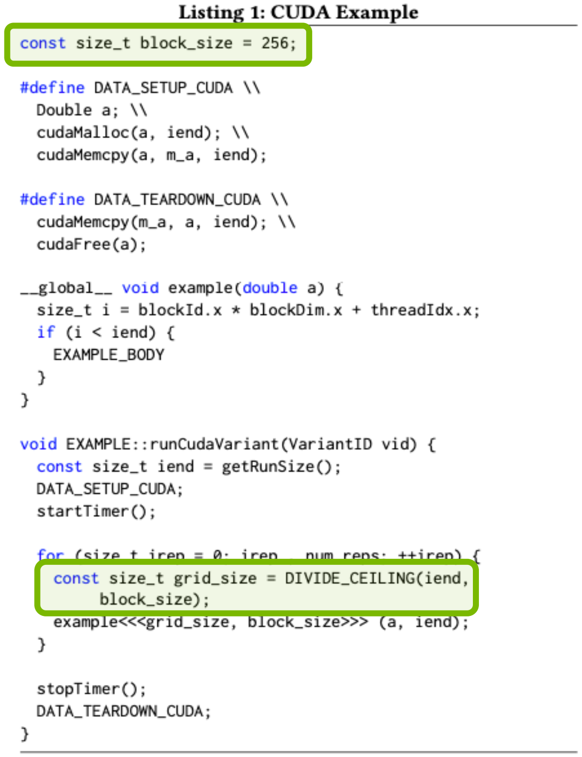


Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \\
  sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
 { // Buffer Scope
  const size_t iend = getRunSize();
  DATA_SETUP_SYCL;
  startTimer();

  for (size_t irep = 0; irep , num_reps; ++irep) {
    const size_t grid_size = block_size *
                      DIVIDE_CEILING(iend, block_size);
    q.submit([&] (sycl::handler& h) {
      auto a =
           d_a.get_access<sycl::access::mode::read_write>(h);

      h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                                 {grid_size, block_size},
                         [=] (sycl::nd_item<1> item) {

        size_t i = item.get_group(0) *
             item.get_local_range().get(0) +
             item.get_local_id(0);
        if (i < iend) {
          EXAMPLE_BODY
        }
      });
    });
  }
 } // Buffer Destruction
  stopTimer();
}
```

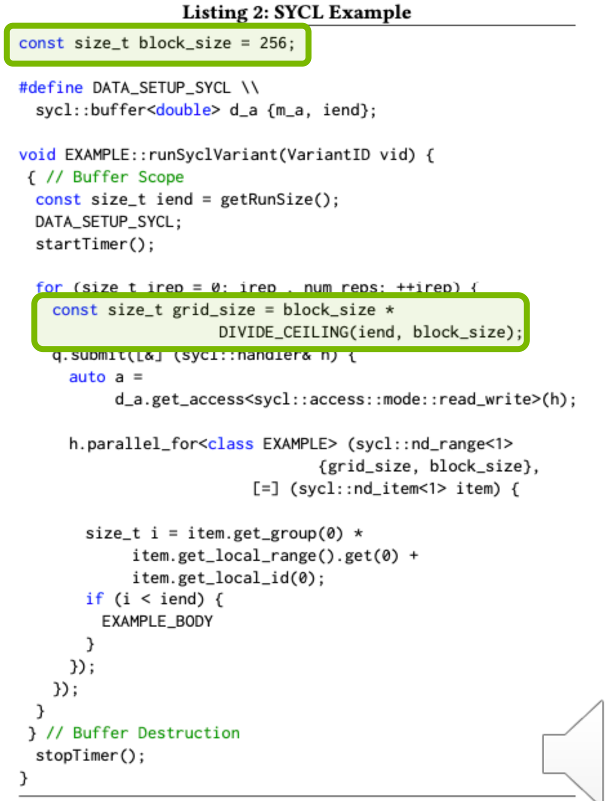
PORTING FOR COMPARABILITY

- Block size and grid size
- Indexing
- Memory management



Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \\
  Double a; \\
  cudaMalloc(a, iend); \\
  cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \\
  cudaMemcpy(m_a, a, iend); \\
  cudaFree(a);

__global__ void example(double a) {
  size_t i = blockId.x * blockDim.x + threadIdx.x;
  if (i < iend) {
    EXAMPLE_BODY
  }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
  const size_t iend = getRunSize();
  DATA_SETUP_CUDA;
  startTimer();

  for (size_t irep = 0; irep < num_reps; ++irep) {
    const size_t grid_size = DIVIDE_CEILING(iend,
         block_size);
    example<<<grid_size, block_size>>> (a, iend);
  }

  stopTimer();
  DATA_TEARDOWN_CUDA;
}
```



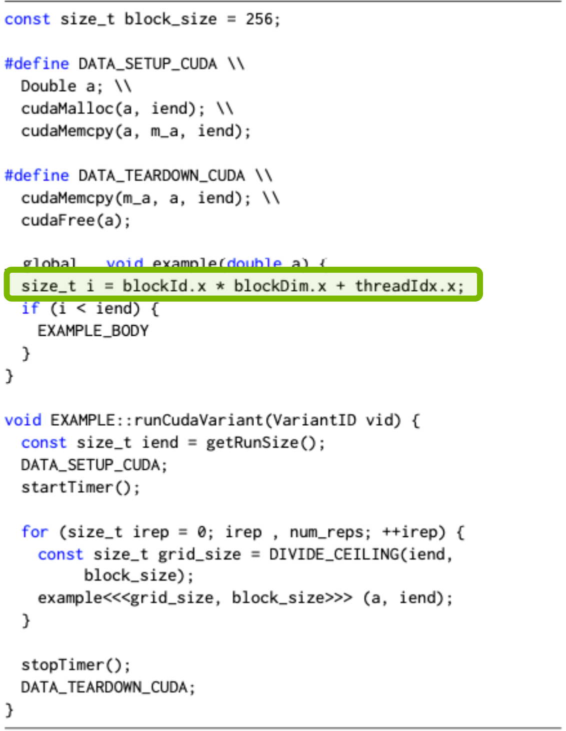
Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \\
  sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
 { // Buffer Scope
  const size_t iend = getRunSize();
  DATA_SETUP_SYCL;
  startTimer();

  for (size_t irep = 0; irep < num_reps; ++irep) {
    const size_t grid_size = block_size *
                       DIVIDE_CEILING(iend, block_size);
    q.submit([&] (sycl::handler& h) {
      auto a =
           d_a.get_access<sycl::access::mode::read_write>(h);

      h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                           {grid_size, block_size},
                       [=] (sycl::nd_item<1> item) {

        size_t i = item.get_group(0) *
             item.get_local_range().get(0) +
             item.get_local_id(0);
        if (i < iend) {
          EXAMPLE_BODY
        }
      });
    });
  }
 } // Buffer Destruction
  stopTimer();
}
```

PORTING FOR COMPARABILITY

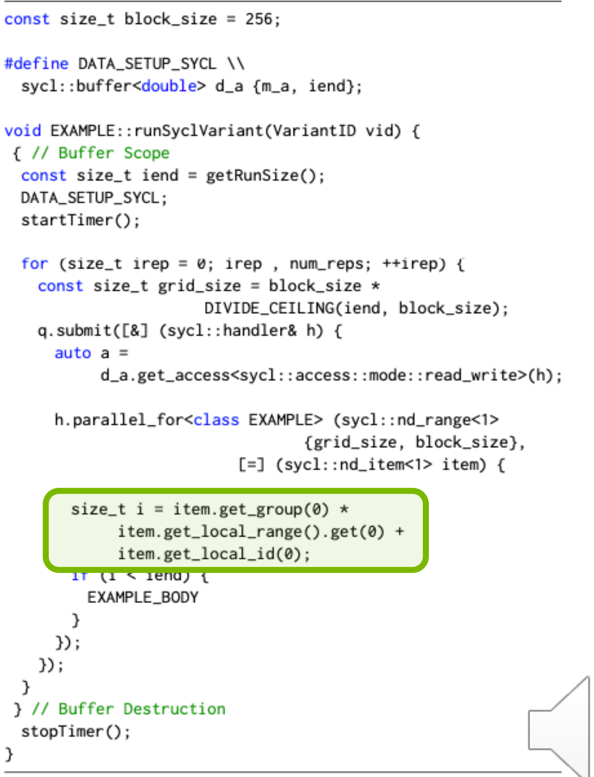
- Block size and grid size
- Indexing
- Memory management

Listing 1: CUDA Example



```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \\
  Double a; \\
  cudaMalloc(a, iend); \\
  cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \\
  cudaMemcpy(m_a, a, iend); \\
  cudaFree(a);

__global__ void example(double a) {
  size_t i = blockId.x * blockDim.x + threadIdx.x;
  if (i < iend) {
    EXAMPLE_BODY
  }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
  const size_t iend = getRunSize();
  DATA_SETUP_CUDA;
  startTimer();

  for (size_t irep = 0; irep , num_reps; ++irep) {
    const size_t grid_size = DIVIDE_CEILING(iend,
          block_size);
    example<<<grid_size, block_size>>> (a, iend);
  }

  stopTimer();
  DATA_TEARDOWN_CUDA;
}
```

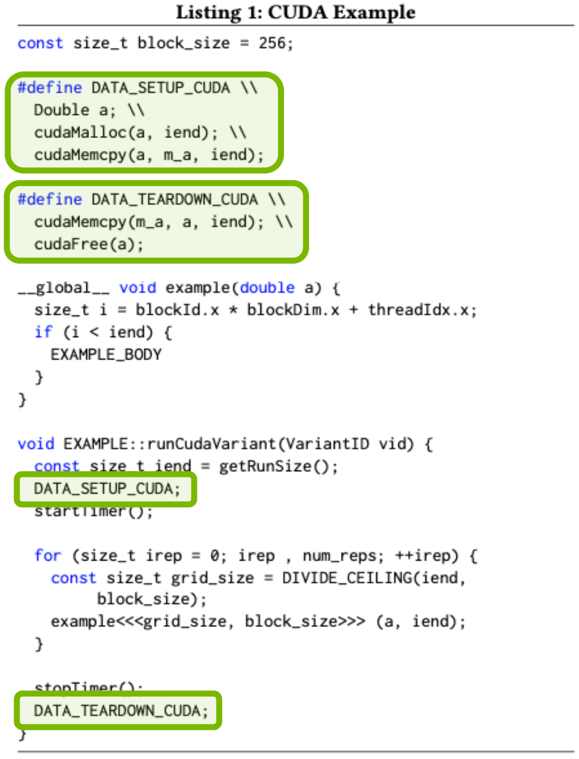
Listing 2: SYCL Example



```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \\
  sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
 { // Buffer Scope
  const size_t iend = getRunSize();
  DATA_SETUP_SYCL;
  startTimer();

  for (size_t irep = 0; irep , num_reps; ++irep) {
    const size_t grid_size = block_size *
                      DIVIDE_CEILING(iend, block_size);
    q.submit([&] (sycl::handler& h) {
      auto a =
           d_a.get_access<sycl::access::mode::read_write>(h);

      h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                                  {grid_size, block_size},
                          [=] (sycl::nd_item<1> item) {

        size_t i = item.get_group(0) *
             item.get_local_range().get(0) +
             item.get_local_id(0);
        if (i < iend) {
          EXAMPLE_BODY
        }
      });
    });
  }
 } // Buffer Destruction
  stopTimer();
}
```

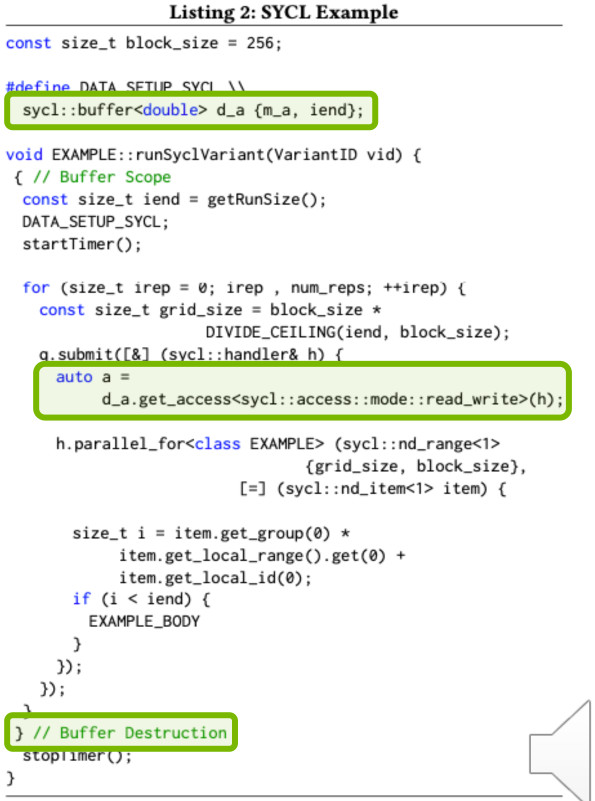
PORTING FOR COMPARABILITY

- Block size and grid size
- Indexing
- Memory management



Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \\
  Double a; \\
  cudaMalloc(a, iend); \\
  cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \\
  cudaMemcpy(m_a, a, iend); \\
  cudaFree(a);

__global__ void example(double a) {
  size_t i = blockId.x * blockDim.x + threadIdx.x;
  if (i < iend) {
    EXAMPLE_BODY
  }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
  const size_t iend = getRunSize();
  DATA_SETUP_CUDA;
  startTimer();

  for (size_t irep = 0; irep , num_reps; ++irep) {
    const size_t grid_size = DIVIDE_CEILING(iend,
         block_size);
    example<<<grid_size, block_size>>> (a, iend);
  }

  stopTimer();
  DATA_TEARDOWN_CUDA;
}
```

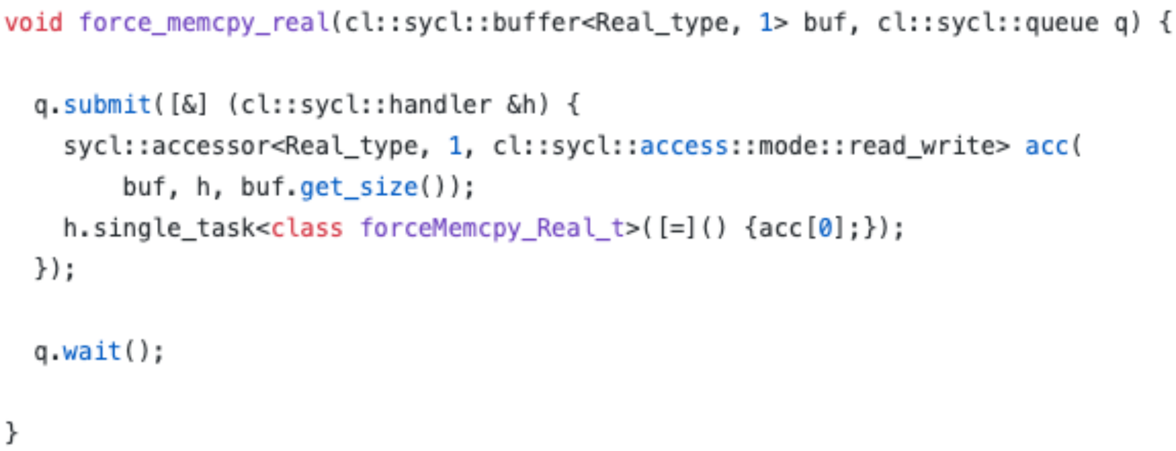


Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \\
  sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
 { // Buffer Scope
  const size_t iend = getRunSize();
  DATA_SETUP_SYCL;
  startTimer();

  for (size_t irep = 0; irep , num_reps; ++irep) {
    const size_t grid_size = block_size *
                      DIVIDE_CEILING(iend, block_size);
    q.submit([&] (sycl::handler& h) {
      auto a =
          d_a.get_access<sycl::access::mode::read_write>(h);

      h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                          {grid_size, block_size},
                  [=] (sycl::nd_item<1> item) {

        size_t i = item.get_group(0) *
             item.get_local_range().get(0) +
             item.get_local_id(0);
        if (i < iend) {
          EXAMPLE_BODY
        }
      });
    });
  }
 } // Buffer Destruction
  stopTimer();
}
```

DATA MOVEMENT

- No explicit data movement in SYCL.



```
void force_memcpy_real(cl::sycl::buffer<Real_type, 1> buf, cl::sycl::queue q) {

  q.submit([&] (cl::sycl::handler &h) {
    sycl::accessor<Real_type, 1, cl::sycl::access::mode::read_write> acc(
        buf, h, buf.get_size());
    h.single_task<class forceMemcpy_Real_t>([=]() {acc[0];});
  });

  q.wait();

}
```

- DPC++ USM proposal would allow for a direct performance comparison including data movement.

PERFORMANCE ANALYSIS METHODOLOGY

- Hardware – NVIDIA V100 GPU
- hipSYCL – git revision 1779e9a
- CUDA – version 10.0.130

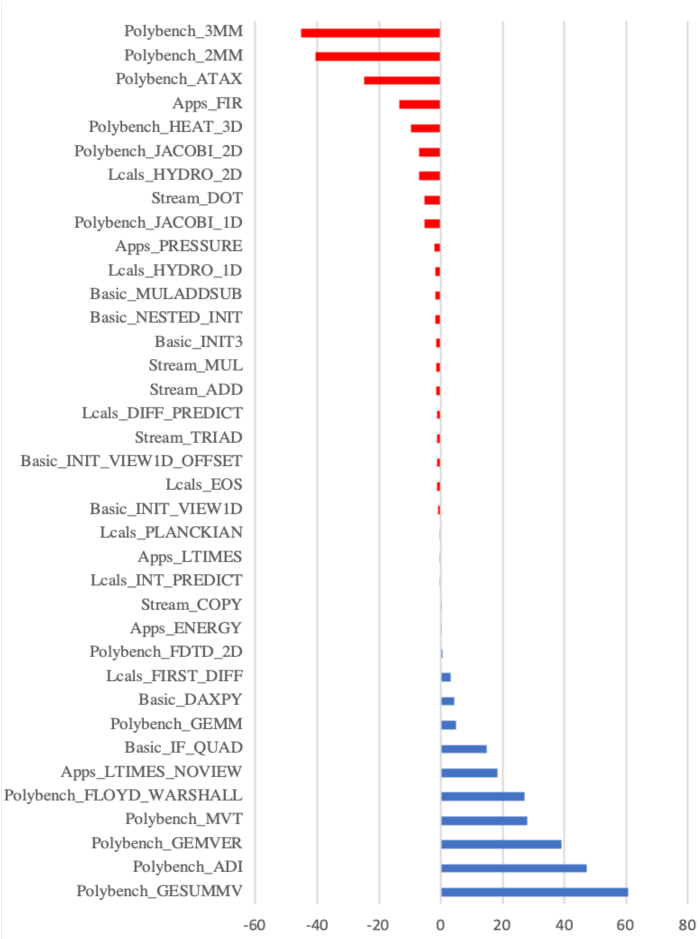
- Utilized nvprof to collect kernel timing without the time spent on memory transfer.

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   10.60%  692.74ms      4460  155.32us  1.2470us  101.74ms  [CUDA memcpy HtoD]
                    2.64%  172.26ms     16000  10.766us  9.7910us  13.120us  rajaperf::lcals::first_diff(double*, double*, long)
```

PERFORMANCE SUITE

Results

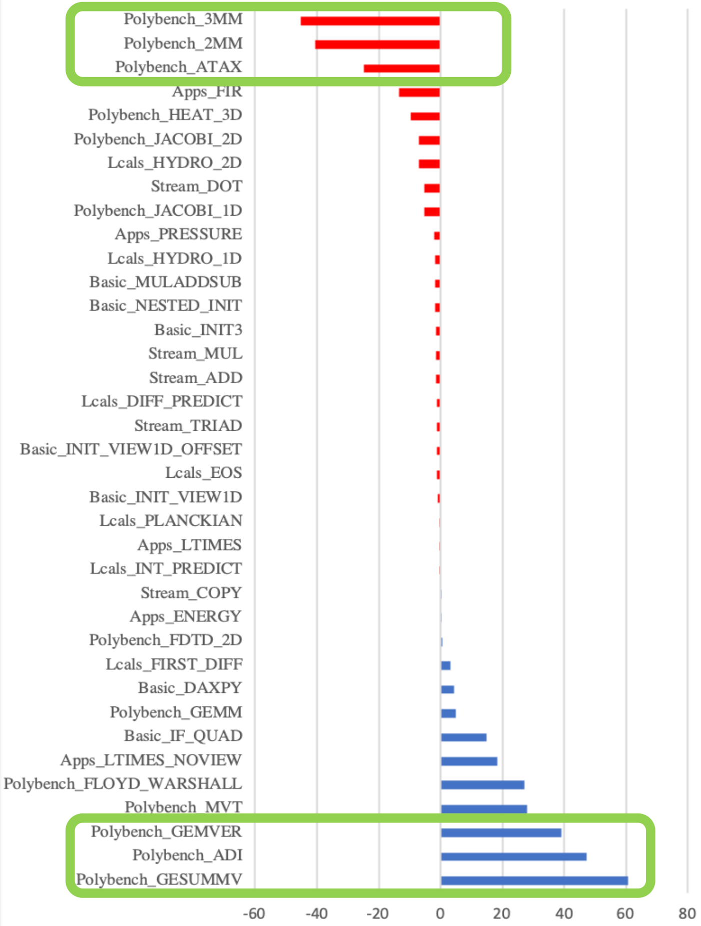
- Problem size is scaled by a factor of five to fill the GPU.
- Five kernels were not measured due to missing features.
- Most kernels are show similar performance.

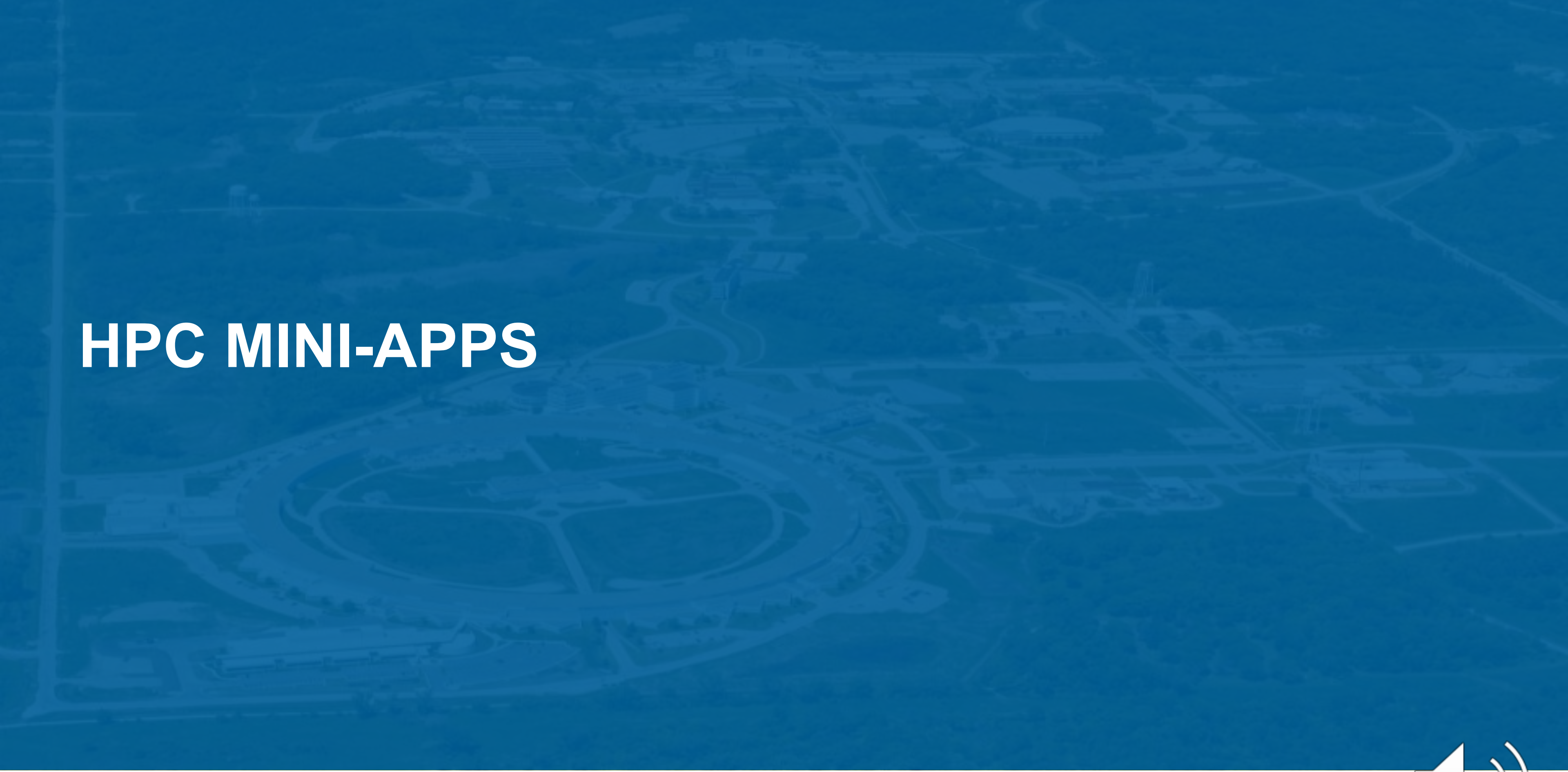


PERFORMANCE SUITE

Results

- Problem size is scaled by a factor of five to fill the GPU
- Five kernels were not measured due to missing features
- Most kernels are show similar performance
- Memory bandwidth utilization.
- CUDA is using non-coherent memory loads.





HPC MINI-APPS

N-BODY SIMULATION MINI-APP

- Simulation of point masses.

- Position of the particles are computed using finite difference methods.

- Each particle stores the position, velocity and acceleration.

- At each timestep the force of all particles acting on one another is calculated.
  - $O(n^2)$

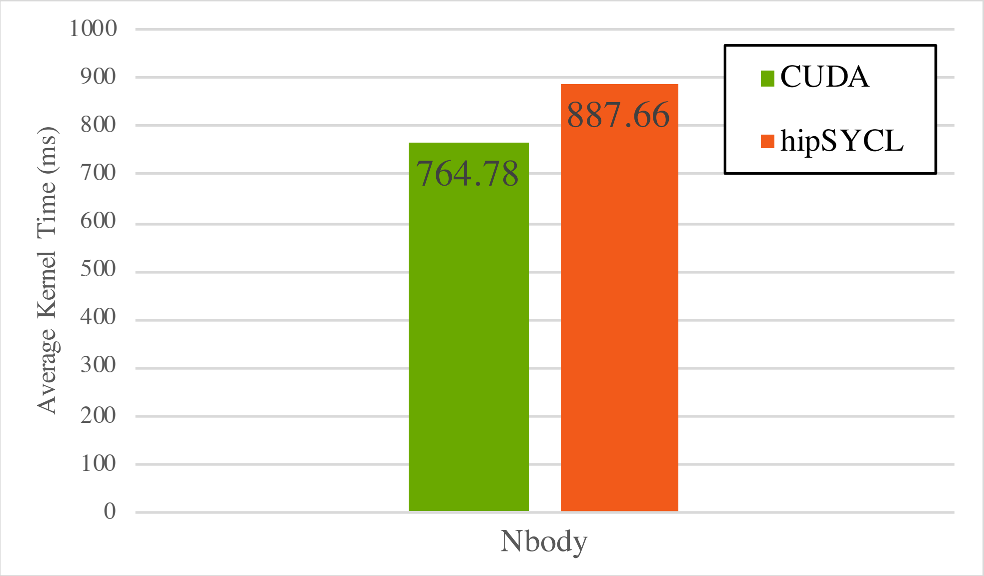
N-BODY

Results

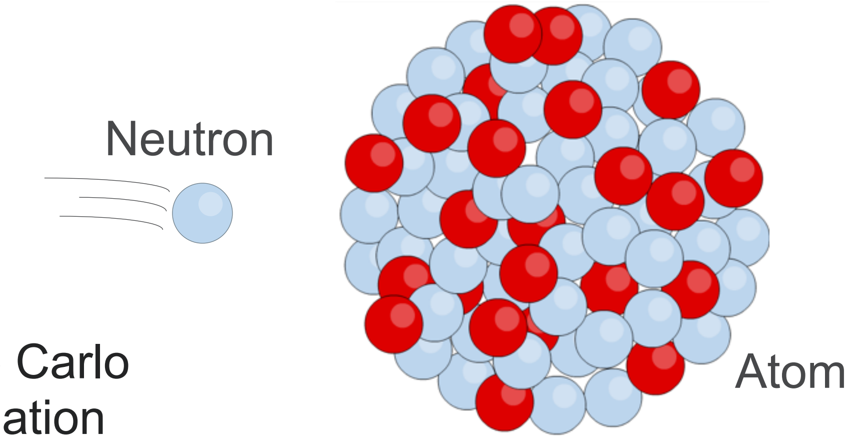
Similar performance metrics

- Memory throughput
- Occupancy

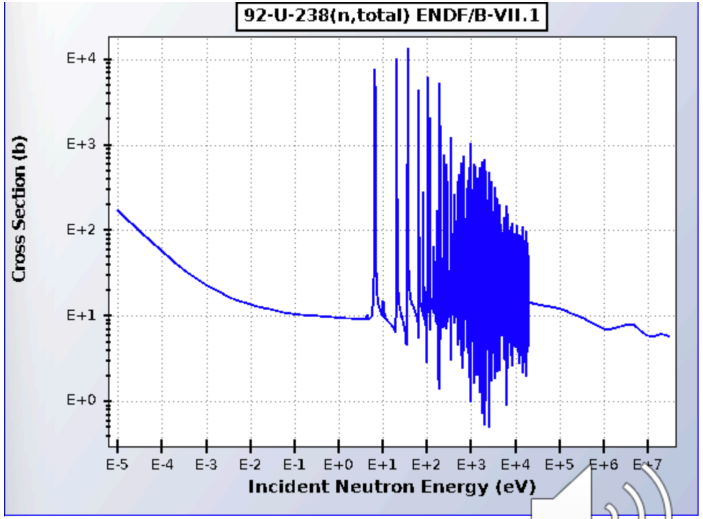
Metric	SYCL	CUDA
FP Instructions (single)	128000000	128000000
Control-Flow Instructions	28000048	25004048
Load/Store Instructions	16018000	16018000
Misc Instructions	**4010096**	26192



XSBench



- Mini-app representing key kernel in Monte Carlo neutron transport for nuclear reactor simulation
- Driven by large tables of **cross section** data that specifies probabilities of interactions between neutron and different types of atoms
- Features a highly randomized memory access pattern that is typically challenging to get running efficiently on most HPC architectures
- Open source, available on github
  - ➢ github.com/ANL-CESAR/XSBench



Example of cross section data for 1 atom type

XSBENCH

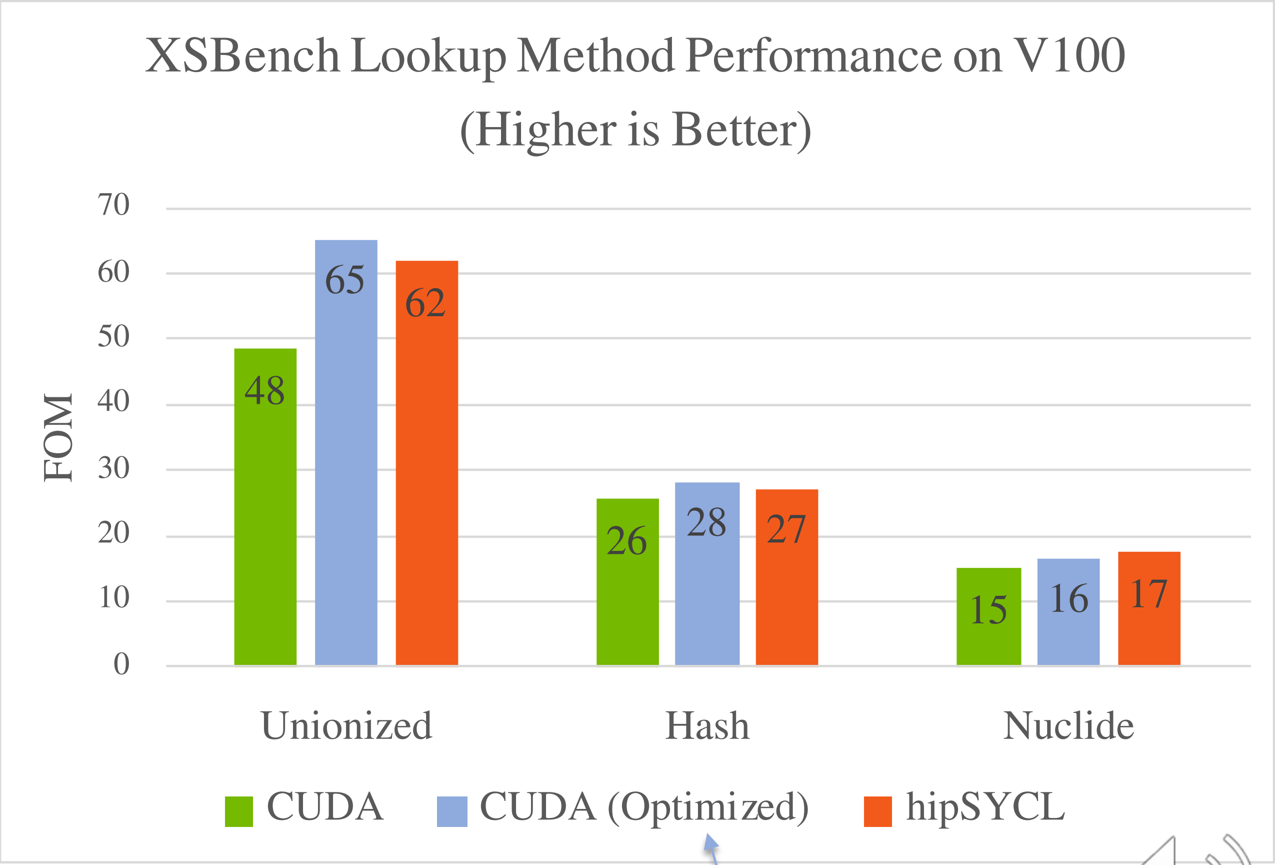
Results

hipSYCL

```
Load #1
Load #2
Load #3
Load #4
Load #5
Load #6
Load #7
Load #8
Load #9
Load #10
Load #11
Load #12
FLOPS...
```

CUDA

```
Load #1
Load #2
FLOPS...
Load #3
Load #4
Load #5
Load #6
Load #7
Load #8
Load #9
FLOPS...
Load #10
FLOPS...
Load #11
FLOPS...
Load #12
FLOPS...
```



Uses `__ldg()` to force contiguous load instructions

CONCLUSIONS

- SYCL using hipSYCL is showing competitive performance on NVIDIA devices.
- Common performance analysis tool very useful. Many subtle details when using difference performance measurement tools on different devices with different programming models.
- Cross programming model studies can provide insight into optimization opportunities.

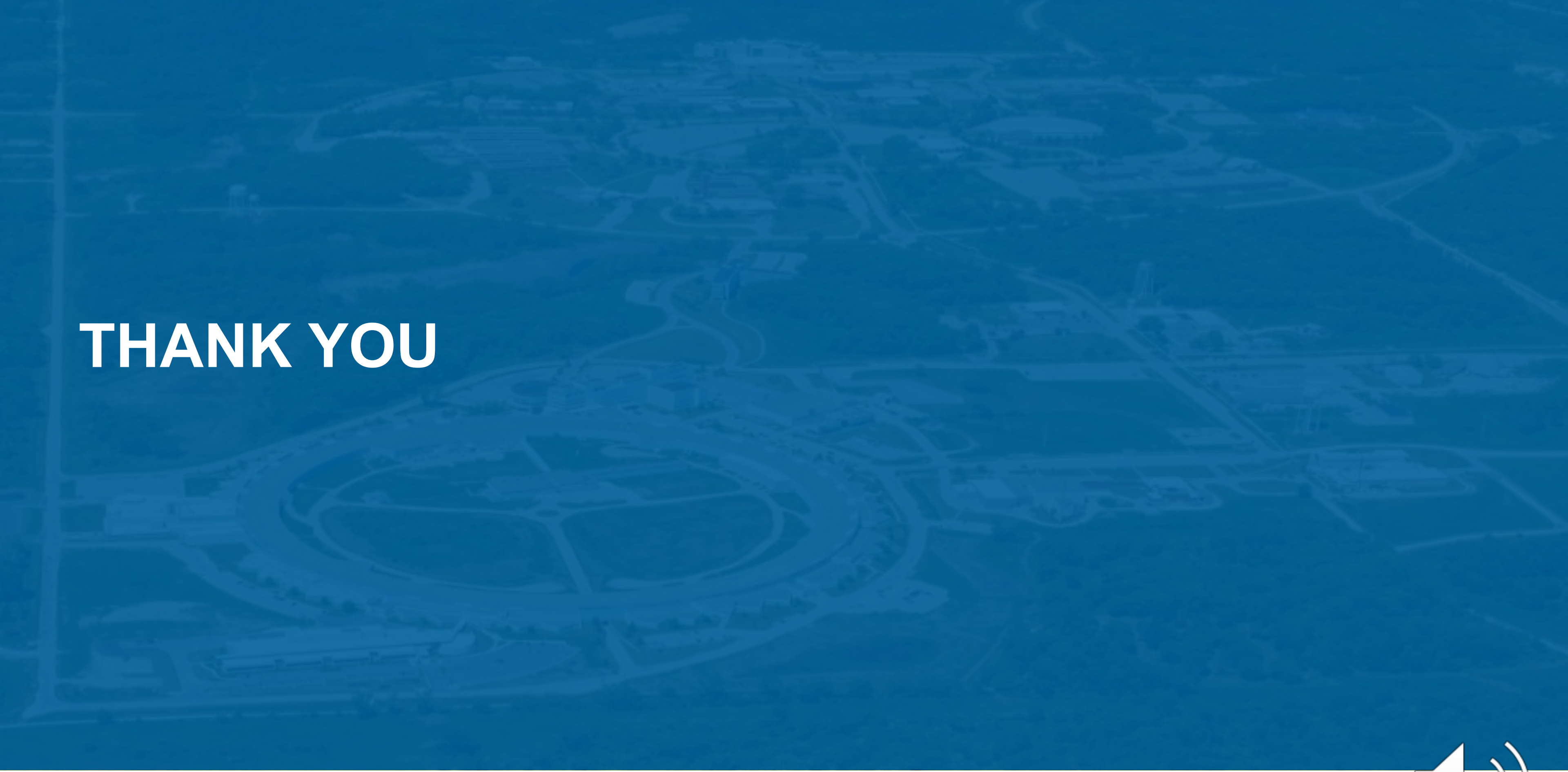
FUTURE WORK

- Utilize larger HPC codes running multi-node problem sizes.
- Investigate the performance of additional toolchains for SYCL and CUDA.
- Investigate performance of the same code across various GPUs.
- Explore the performance of Intel’s DPC++ extensions.

ACKNOWLEDGEMENTS

- ALCF, ANL and DOE
- ALCF is supported by DOE/SC under contract DE-AC02-06CH11357

- This research was supported 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) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation’s exascale computing imperative.



THANK YOU

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

REFERENCES

[1] 2020. Summit. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

[2] 2020. Aurora. https://press3.mcs.anl.gov/aurora

[3] 2020. Frontier. https://www.olcf.ornl.gov/frontier

[4] NVIDIA Corporation. 2020. CUDA C++ Programming Guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[5] Khronos OpenCL Working Group SYCL subgroup. 2018. SYCL Specification.

[6] Aksel Alpay. 2019. hipSYCL. https://github.com/illuhad/hipSYCL

[7] Richard D. Hornung and Holger E. Hones. 2020. RAJA Performance Suite. https://github.com/LLNL/RAJAPerf

[8] Fabio Barruffa. 2020. N-Body Demo. https://github.com/fbaru-dev/nbody-demo

[9] John R. Tramm. 2020. XSBench: The Monte Carlo macroscopic cross section lookup benchmark. https://github.com/ANL-CESAR/XSBench