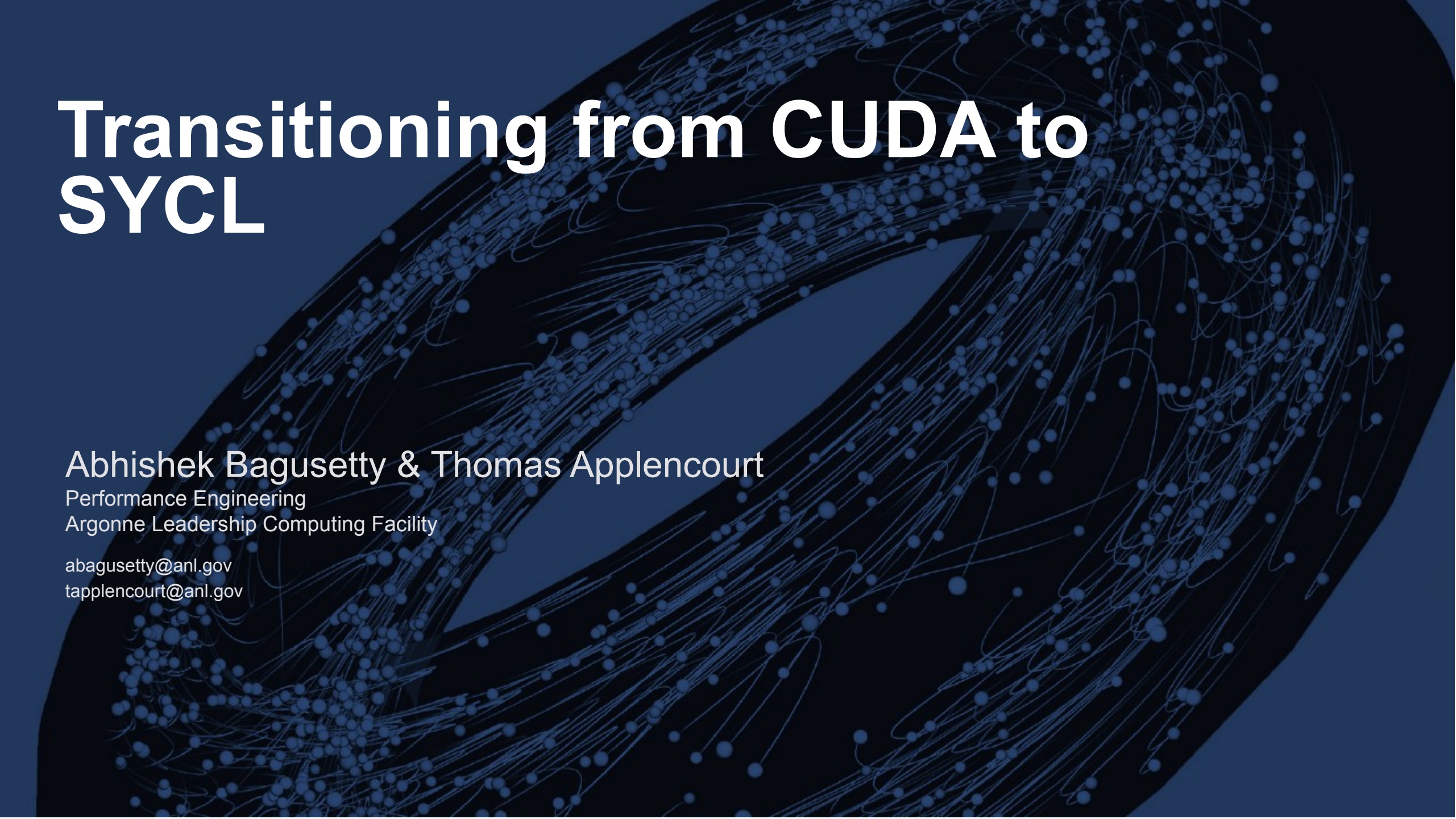
Transitioning from CUDA to SYCL



Abhishek Bagusetty & Thomas Applencourt
Performance Engineering
Argonne Leadership Computing Facility

abagusetty@anl.gov
tapplencourt@anl.gov

Agenda

Introduction: Why SYCL ?

(Examples) SYCL – Thomas Applencourt

(Examples) Porting CUDA to SYCL – Abhishek Bagusetty

A few practical case studies

Pre-Exascale & Exascale Compute Architectures

Machine	GPU Models	DOE Facility	GPU	No of GPUs per node
	CUDA, SYCL	ALCF Polaris	Nvidia A100	4
	CUDA, SYCL	NERSC Perlmutter	Nvidia A100	4
	HIP, SYCL	OLCF Frontier	AMD MI250x	4/8
	SYCL	ALCF Aurora	Intel Max Series	6/12
	HIP, (SYCL)	LLNL El Capitan	AMD MI300A	4

Aurora

Intel® Data Center GPU Max Series

4th Gen Intel XEON Max Series CPU *with High Bandwidth Memory*

Platform
HPE Cray-Ex

Racks - 166
Nodes - 10,624
CPUs - 21,248
GPUs – 63,744

Interconnect
HPE Slingshot 11
Dragonfly topology with adaptive routing
Cassini NIC, 200 Gb/s (25 GB/s), 8 per node
Network Switch:
25.6 Tb/s per switch (64 200 Gb/s ports)
Links with 25 GB/s per direction

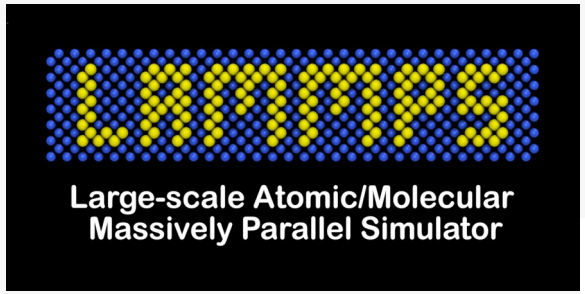
Peak FP64 Performance
≧ 2 exaFLOPS

Memory
10.9PiB of DDR @ 5.95 PB/s
1.36PiB of CPU HBM @ 30.5 PB/s
8.16PiB of GPU HBM @ 208.9 PB/s

Network
2.12 PB/s Peak Injection BW
0.69 PB/s Peak Bisection BW

Storage
230PB DAOS Capacity
31 TB/s DAOS Bandwidth

Applications/Libraries using SYCL

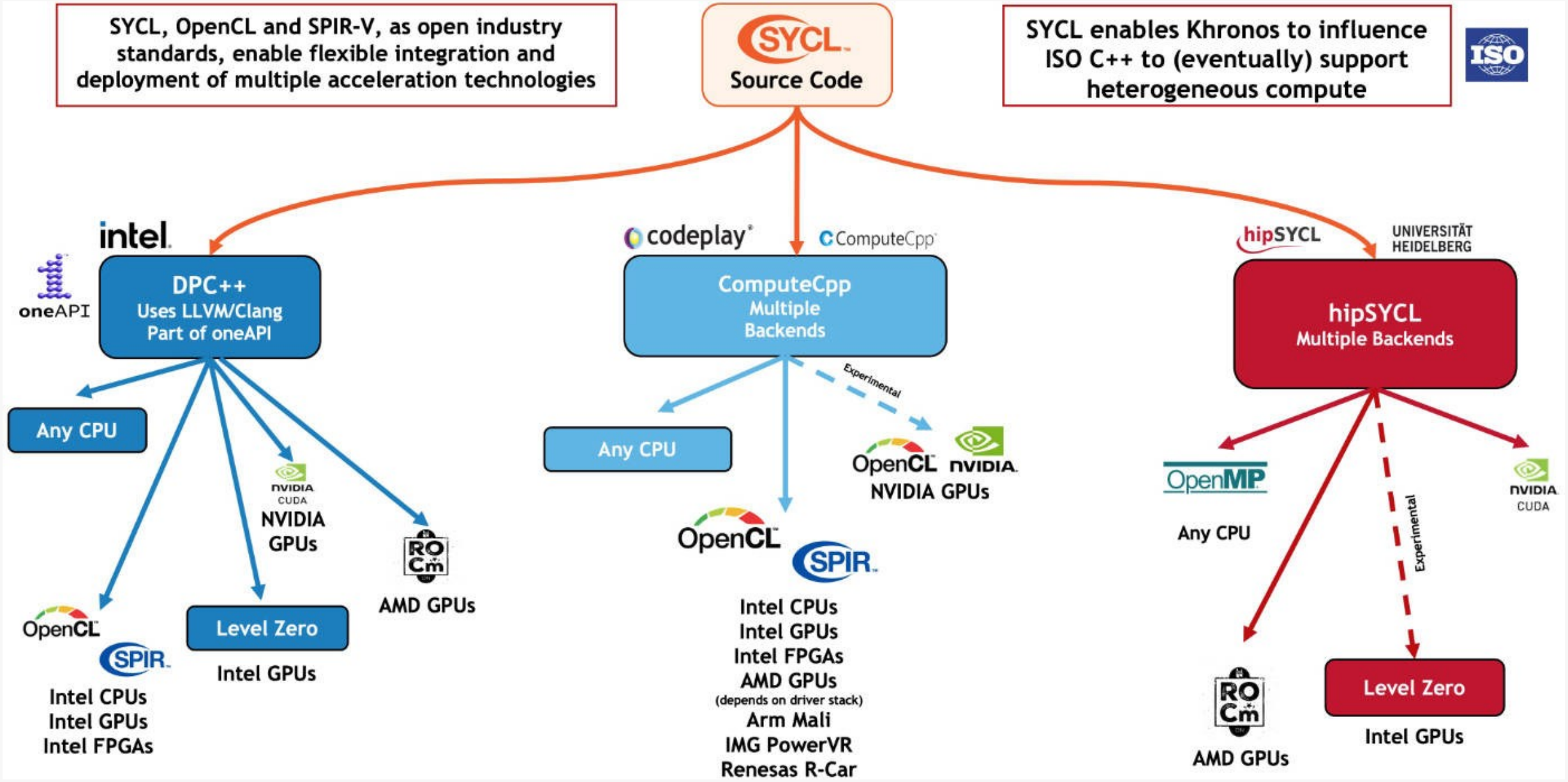


What is SYCL ?

- SYCL is “not” a programming model but a “language specification”

– Heuristics looks similar to OpenCL-C bindings
– C++ single source (coexists host and device source code)
– Two distinct memory models (USM and/or Buffer)
– Asynchronous programming (overlaps device-compute, copy, host operations)
– Portability (functional and performance)
– Productivity

SYCL – Compiler Players



Queues & Contexts

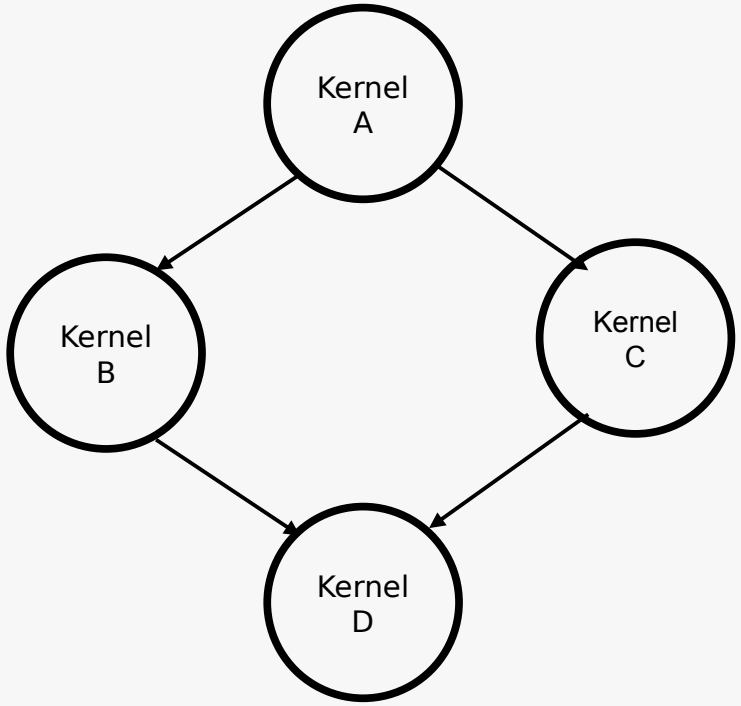
- "SYCL Queues" provide mechanism to **submit** work to a **device** or **sub-device**
- "SYCL Contexts" is well known to be over-looked

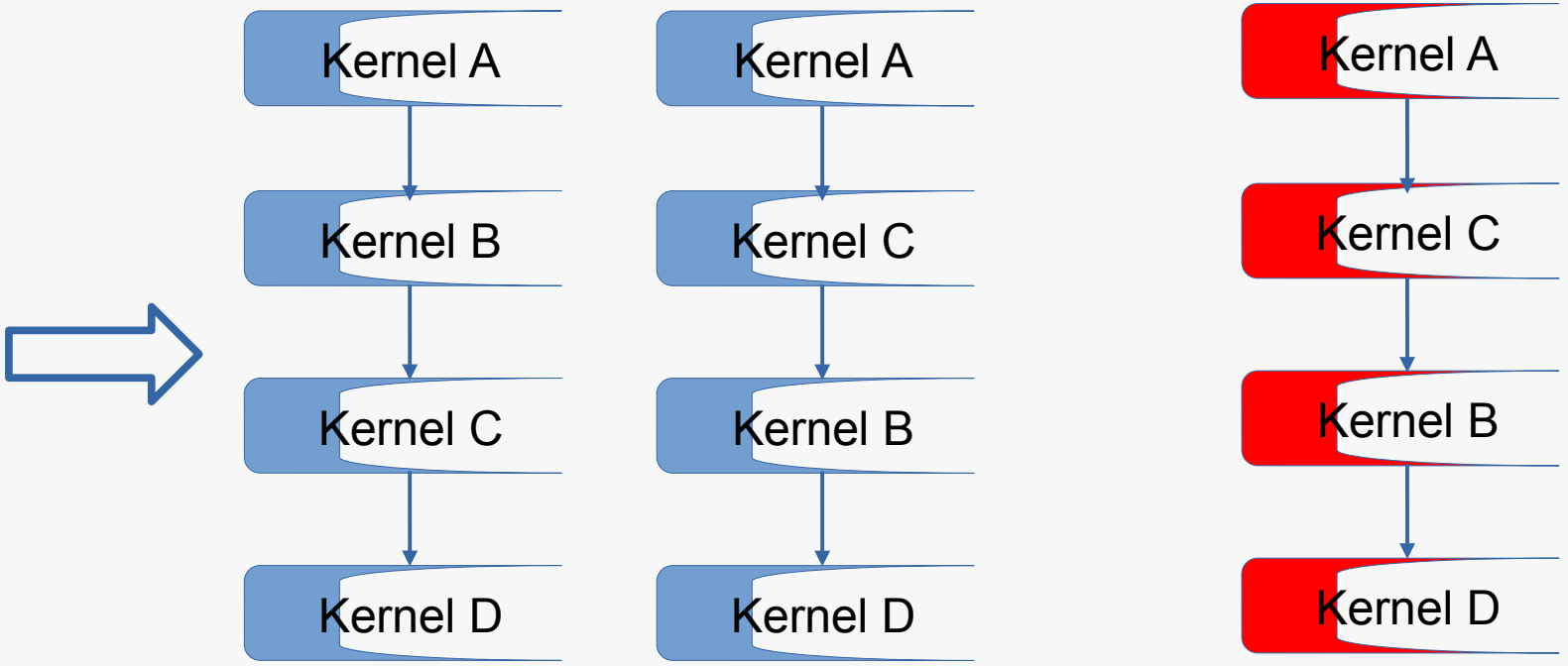
```
sycl::queue myQue;  // implicitly creates a SYCL context
```

- **Context**
  - Contexts are used for resources isolation and sharing
  - A SYCL context may consist of one or multiple devices
  - Both root-devices and sub-devices can be within single context (all from same SYCL platform)
  - Memory created can be shared only if their associated queue(s) are created using the same context

- **Queue (aka CUDA Stream)**
  - SYCL queue is always attached to a single device in a possibly multi-device context
    - ✓ Executes "**asynchronously**" from host code
    - ✓ SYCL queue can execute tasks enqueued in either "**in-order**" or "**out-of-order (default)**"
    - ✓ SYCL queue (in-order) is similar to CUDA stream (FIFO)

Queues (out-of-order vs in-order)





(OOO queue) This means commands are allowed to be overlapped and re-ordered or executed concurrently providing dependencies are honoured to ensure consistency.

(In-order) This mean commands must execute strictly in the order they were enqueued.

```
auto outOfOrderQueue = sycl::queue{gpu_selector_v};
auto inOrderQueue = sycl::queue{gpu_selector_v, sycl::property::queue::in_order{}};
```

Devices

- Devices are the target for acceleration offload

SYCL sub-devices ↔ CUDA Multi-Instance GPU (MIG) mode ↔ OpenCL sub-devices

- Explicit Scaling: Partitioning of a SYCL root device into multiple sub-devices based on NUMA boundary
  - ✓ SYCL queues are further created based on “sub-devices” (better performance)
- Implicit Scaling: SYCL unpartitioned/root device is directly used to create a SYCL queue

```
// EXPLICIT SCALING (better performance)


sycl::platform platform(sycl::gpu_selector{});
auto const& gpu_devices = platform.get_devices(sycl::info::device_type::gpu);
for (auto const& gpuDev : gpu_devices) {
  if(gpu_dev.get_info<sycl::info::device::partition_max_sub_devices>() > 0) {
    auto SubDev =
gpuDev.create_sub_devices<sycl::info::partition_property::partition_by_affinity_domain>(sycl::info::partition_affinity_domain::numa);

    for (auto const& tile : SubDev) {
      Que = sycl::queue(tile);
    }
  }
}

// IMPLICIT SCALING

sycl::platform platform(sycl::gpu_selector{});
auto const& gpu_devices = platform.get_devices(sycl::info::device_type::gpu);
for (auto const& gpuDev : gpu_devices) {
   Que = sycl::queue(gpuDev);
}
```

SYCL Events for Task-dependencies

- Performs book-keeping of tasks for data-transfer between host-device
- Similar to CUDA/HIP events

- SYCL runtime inherently has a DAG dependency paradigms
- Using SYCL dependency infrastructure provided via “SYCL Event”

- Task dependency between a communication-computation tasks

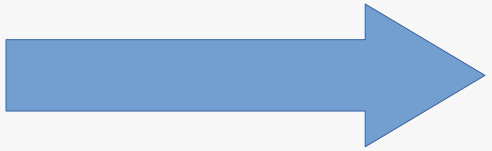
```
event memcpy(void* dest, const void* src, size_t numBytes);
event memcpy(void* dest, const void* src, size_t numBytes, event depEvent);
event memcpy(void* dest, const void* src, size_t numBytes,
             const std::vector<event>& depEvents);
```

Data-transfer showing “event” returns & dependencies

```
template <typename KernelName, int Dims, typename... Rest>
event parallel_for(nd_range<Dims> executionRange,
                   const std::vector<event>& depEvents, Rest&&... rest);
```

kernel-launch showing “event” returns & dependencies

Porting from CUDA to SYCL



Execution Model: CUDA vs SYCL

CUDA	SYCL
thread	work-item
block	work-group
grid	nd-range

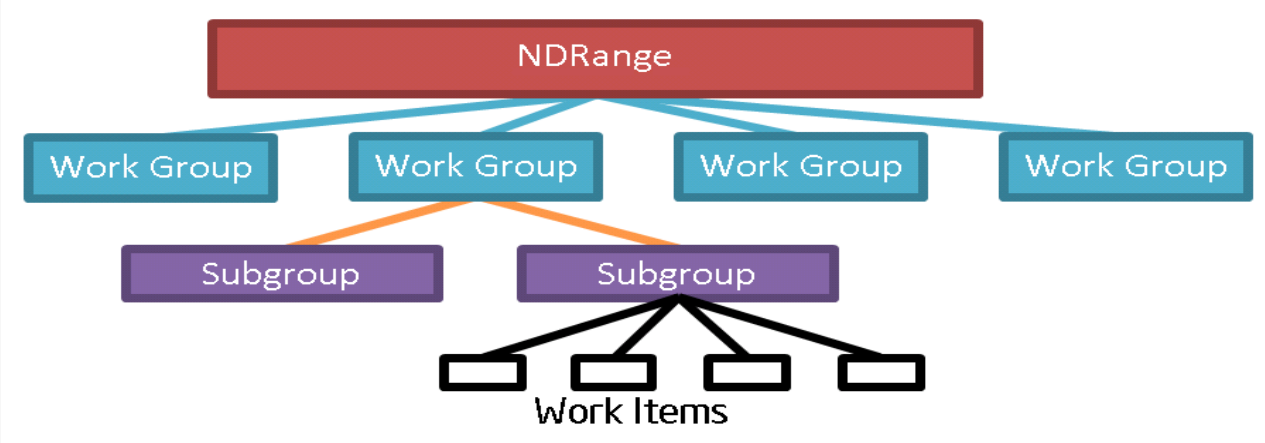
A **grid** is an array of thread blocks launched by a kernel.

An **nd range** has three components

- global range (total work items)
- local range (work-items per work-group)
- number of work groups (total work groups)

CUDA - warp (vs) SYCL - sub groups

CUDA	SYCL
thread	work-item
warp	**sub-group**
block	work-group
grid	nd-range



Sub-groups are subset of the work-items that are executed simultaneously or with additional scheduling guarantees.

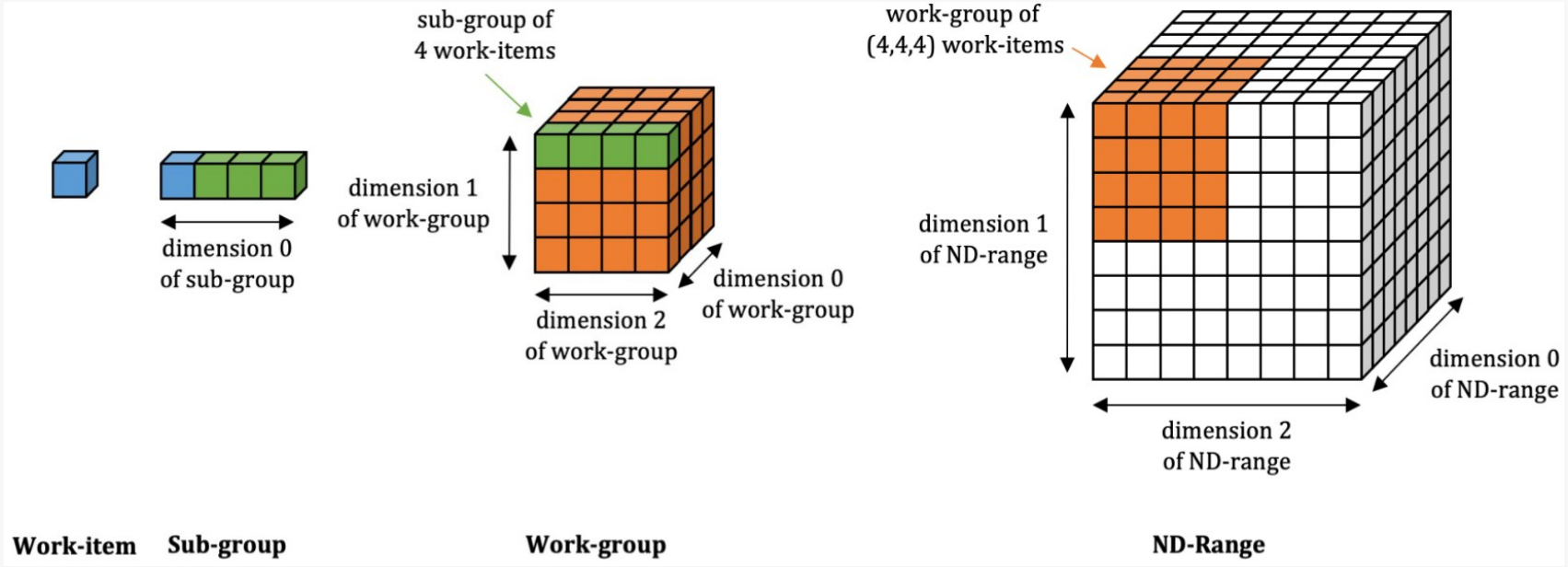
Leveraging sub-groups will help to map execution to low-level hardware and may help in achieving higher performance.

Why use SYCL - sub groups ?

Sub-Group = subset of work-items within a work-group.

A subset of work-items within a work-group that execute with additional guarantees and often map to SIMD hardware.

•Work-items in a sub-group can communicate directly using shuffle operations, without repeated access to local or global memory, and may provide better performance.
•Work-items in a sub-group have access to sub-group collectives, providing fast implementations of common parallel patterns.



Memory Model: CUDA vs SYCL

CUDA		SYCL	
Memory Type	**Scope**	**Memory Type**	**Scope**
Register memory	Thread	Private memory	Work-item
Shared memory	Block	Local memory	Work-group
Global memory	Grid (all threads)	Global memory	All work Items

Allocation Type	Initial Location	Accessible By		Migratable To	
`device`	`device`	`host`	No	`host`	No
		`device`	Yes	`device`	N/A
		Another `device`	Optional (P2P)	Another `device`	No
`host`	`host`	`host`	Yes	`host`	N/A
		Any `device`	Yes	`device`	No
`shared`	Unspecified	`host`	Yes	`host`	Yes
		`device`	Yes	`device`	Yes
		Another `device`	Optional	Another `device`	Optional

https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html#table.USM.allocation.characteristics

Memory Model: Global Memory

CUDA		SYCL	
Memory Type	**Scope**	**Memory Type**	**Scope**
Register memory	Thread	Private memory	Work-item
Shared memory	Block	Local memory	Work-group
Global memory	**Grid (all threads)**	**Global memory**	**All work Items**

```
// allocating device memory

    float *A_dev;
    cudaMalloc((void **)&A_dev, array_size * sizeof(float));
```

```
// allocating device memory

    sycl::queue q(syckl::gpu_selector{});
    float *A_dev = sycl::malloc_device<float>(array_size, q);
```

- SYCL's Global/Device allocated memory is only **valid** on the **device**
- More importantly not accessible from host

Thread Indexing

CUDA	SYCL
gridDim.x/y/z	sycl::nd_item.get_group_range(2/1/0)
blockIdx.x/y/z	sycl::nd_item.get_group(2/1/0)
blockDim.x/y/z	sycl::nd_item.get_local_range().get(2/1/0)
threadIdx.x/y/z	sycl::nd_item.get_local_id(2/1/0)
warpSize	sycl::nd_item.get_sub_group().get_local_range().get(0)

- Always ensure the mapping of CUDA's (x/y/z) dimensions map to (2/1/0) in SYCL

Portable Math Libraries ?

- Open-source implementation of the oneMKL Data Parallel C++ (DPC++) interface
- Works with multiple devices (backends)
- Uses vendor, device-specific libraries underneath

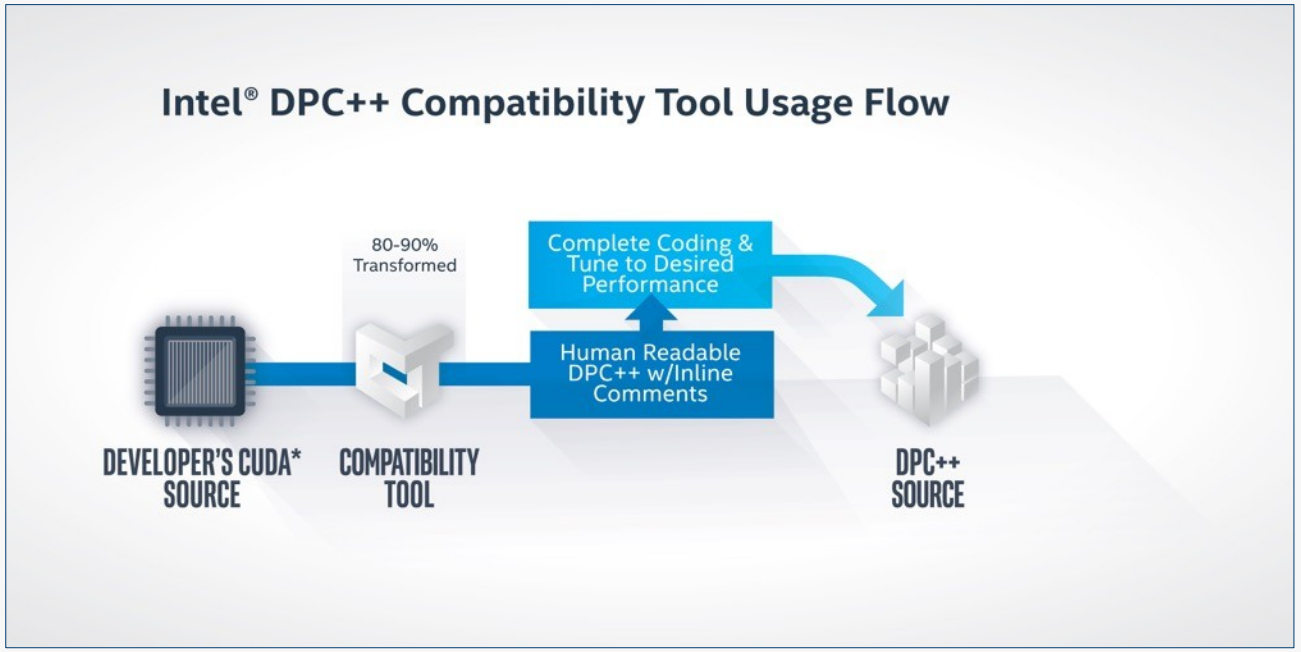
Note: Apart of device-backend, supports host-CPU interface: Intel MKL, NETLIB

	NVIDIA	AMD	Intel
BLAS	cuBLAS	rocBLAS	oneMKL
Linear Solvers	cuSOLVER	In-works (rocSOLVER)	oneMKL
Random Numbers	cuRAND	rocRAND	oneMKL
FFT	cuFFT	rocFFT	oneMKL
Sparse	cuSparse	rocSparse	oneMKL

How to port existing CUDA to SYCL ?

Intel® DPC++ Compatibility Tool

Assist in migrating CUDA* applications to SYCL/DPC++, extending user choices

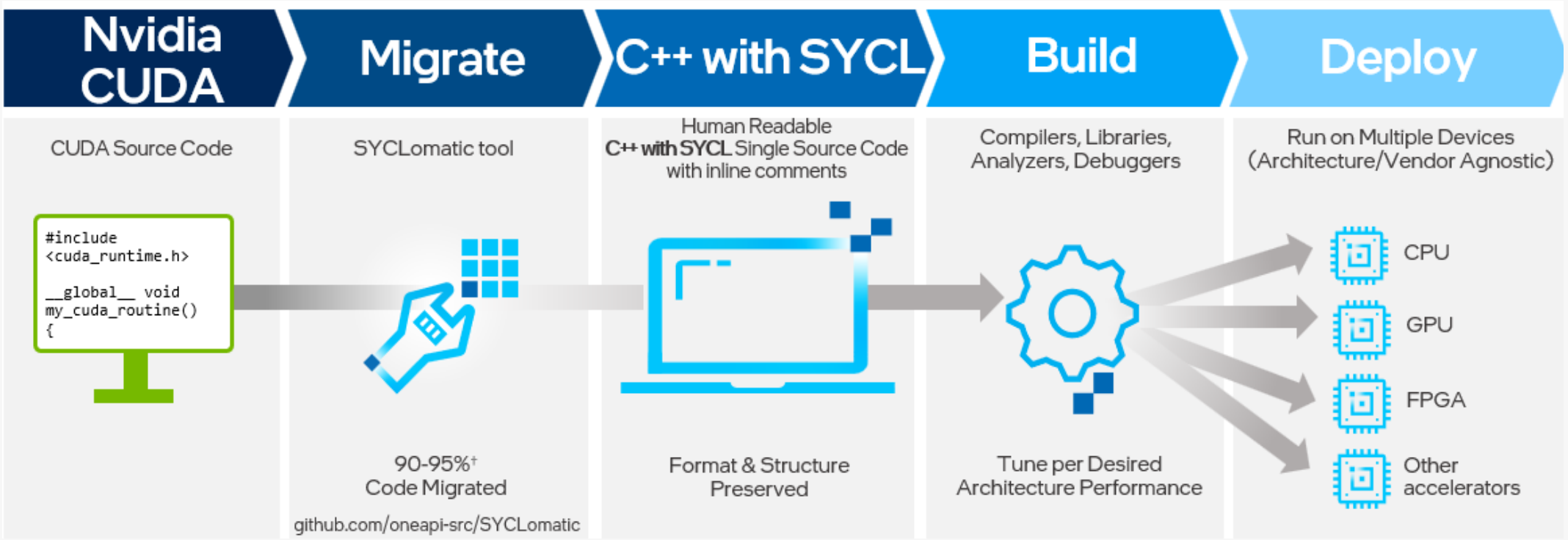


- Assists developers migrating code written in CUDA* to DPC++
- Target is to migrate up to 90-95% of code automatically
- Inline comments are provided to help developer complete code

https://developer.codeplay.com/products/computecpp/ce/guides/sycl-for-cuda-developers/cuda-to-sycl-examples
https://www.intel.com/content/www/us/en/developer/articles/training/intel-dpcpp-compatibility-tool-training.html

Porting CUDA projects to SYCL

SYCLomatic: A New CUDA*-to-SYCL* Code Migration Tool
(previously referred to as DPCT, DPC++ Compatibility Tool)



https://developer.codeplay.com/products/computecpp/ce/guides/sycl-for-cuda-developers/cuda-to-sycl-examples

Instructions to build SYCLomatic (Optional)

Repository: https://github.com/oneapi-src/SYCLomatic

Setup Environment:

```
export SYCLOMATIC_HOME=~/workspace
export
PATH_TO_C2S_INSTALL_FOLDER=~/workspace/c2s_install
mkdir $SYCLOMATIC_HOME
cd $SYCLOMATIC_HOME

git clone https://github.com/oneapi-src/SYCLomatic.git
```

Build Instructions:

```
cd $SYCLOMATIC_HOME
mkdir build
cd build
cmake -G Ninja -DCMAKE_INSTALL_PREFIX=$PATH_TO_C2S_INSTALL_FOLDER  -DCMAKE_BUILD_TYPE=Release
-DLLVM_ENABLE_PROJECTS="clang"  -DLLVM_TARGETS_TO_BUILD="X86;NVPTX" ../SYCLomatic/llvm
ninja install-c2s
```

Post Installation:

```
export PATH=$PATH_TO_C2S_INSTALL_FOLDER/bin:$PATH
export CPATH=$PATH_TO_C2S_INSTALL_FOLDER/include:$CPATH
```

ALCF Polaris: `module load oneapi/upstream` already has SYCLomatic

SYCL tutorial

Porting CUDA + X(OpenMP, MPI)

- Sources with CUDA + X (OpenMP, MPI) can also be ported to SYCL +X using DPCT/SYCLomatic tool
- Sample: https://github.com/argonne-lcf/CUDA_to_SYCL_samples
- Running the CUDA version on ALCF Polaris with 4 Nvidia A100 GPUs

```
$ mpiexec -np 4 -ppn 4 --depth=4 --cpu-bind depth ./set_polaris_affinity.sh ./hello_cuda_affinity.out
MPI 000 - OMP 000 - HWT 0-3 (Running on: 001) - Node x3004c0s25b1n0 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID c7
MPI 001 - OMP 000 - HWT 4-7 (Running on: 004) - Node x3004c0s25b1n0 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID 85
MPI 002 - OMP 000 - HWT 8-11 (Running on: 010) - Node x3004c0s25b1n0 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID 46
MPI 003 - OMP 000 - HWT 12-15 (Running on: 012) - Node x3004c0s25b1n0 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID 07
```

Porting CUDA + X(OpenMP, MPI)

- Initialize MPI
- Check for CUDA_VISIBLE_DEVICES vars
- `cudaGetDeviceCount` - Get total number of GPUs seen by each MPI rank
- `cudaGetDeviceProperties` - Get total number of GPUs
  seen by each MPI rank

```
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    char name[MPI_MAX_PROCESSOR_NAME];
    int resultlength;
    MPI_Get_processor_name(name, &resultlength);

    // If CUDA_VISIBLE_DEVICES is set, capture visible GPUs
    const char* gpu_id_list;
    const char* cuda_visible_devices = getenv("CUDA_VISIBLE_DEVICES");
    if (cuda_visible_devices == NULL) gpu_id_list = "N/A";
    else gpu_id_list = cuda_visible_devices;

    // Find how many GPUs CUDA runtime sees
    int num_devices = 0;
    cudaGetDeviceCount(&num_devices);

    // Loop over available GPUs for this MPI rank
    for (int i = 0; i < num_devices; i++) {
        cudaDeviceProp prop;
        cudaGetDeviceProperties(&prop, i);

        // Build runtime GPU ID list
        if (i > 0) rt_gpu_id_list.append(",");
        rt_gpu_id_list.append(std::to_string(i));

        // Extract Bus ID
        char bus_id_str[8];
        snprintf(bus_id_str, sizeof(bus_id_str), "%02x", prop.pciBusID);
```

Porting CUDA + X(OpenMP, MPI)

- Porting the affinity code to Intel GPUs was giving incorrect GPU_IDs and Bus_ID
- `CUDA_VISIBLE_DEVICES` would not work with Intel GPU. Switch to `ZE_AFFINITY_MASK`
- Couldn't port `cudaDeviceProp` attribute `` `pciBusID` `` in CUDA to its SYCL equivalent

```
$ export OMP_NUM_THREADS=1
$ export CPU_BIND_SCHEME="--cpu-bind=list:1-8:9-16:17-24:25-32:33-40:41-48:53-60:61-68:69-76:77-84:85-92:93-100"
$ mpiexec -n 12 -ppn 12 ${CPU_BIND_SCHEME} gpu_tile_compact.sh ./hello_cuda_affinity_v1.out | sort
MPI 000 - OMP 000 - HWT 1-8 (Running on: 001) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 001 - OMP 000 - HWT 9-16 (Running on: 009) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 002 - OMP 000 - HWT 17-24 (Running on: 017) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 003 - OMP 000 - HWT 25-32 (Running on: 025) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 004 - OMP 000 - HWT 33-40 (Running on: 033) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 005 - OMP 000 - HWT 41-48 (Running on: 041) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 006 - OMP 000 - HWT 53-60 (Running on: 053) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 007 - OMP 000 - HWT 61-68 (Running on: 061) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 008 - OMP 000 - HWT 69-76 (Running on: 069) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 009 - OMP 000 - HWT 77-84 (Running on: 077) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 010 - OMP 000 - HWT 85-92 (Running on: 085) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
MPI 011 - OMP 000 - HWT 93-100 (Running on: 093) - Node x4305c0s5b0n0 - RT_GPU_ID 0 - GPU_ID N/A - Bus_ID ffffff
```

For appropriate binding of CPUs and GPUs in Aurora, please refer to https://docs.alcf.anl.gov/aurora/running-jobs-aurora/#running-mpiopenmpsycl-applications

Porting CUDA + X(OpenMP, MPI)

- `CUDA_VISIBLE_DEVICES` would not work with Intel GPU. Switch to `ZE_AFFINITY_MASK`
- CUDA's `cudaDeviceProp` attribute `` `pciBusID` `` was manually ported to SYCL API via: `sycl::device` attributes `get_info<sycl::ext::intel::info::device::pci_address>()`

```
$ export OMP_NUM_THREADS=1
$ export CPU_BIND_SCHEME="--cpu-bind=list:1-8:9-16:17-24:25-32:33-40:41-48:53-60:61-68:69-76:77-84:85-92:93-100"
$ mpiexec -n 12 -ppn 12 ${CPU_BIND_SCHEME} gpu_tile_compact.sh ./hello_cuda_affinity_v2.out | sort
MPI 000 - OMP 000 - HWT 1-8 (Running on: 001) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 0.0 - Bus_ID 0000:18
MPI 001 - OMP 000 - HWT 9-16 (Running on: 009) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 0.1 - Bus_ID 0000:18
MPI 002 - OMP 000 - HWT 17-24 (Running on: 017) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 1.0 - Bus_ID 0000:42
MPI 003 - OMP 000 - HWT 25-32 (Running on: 025) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 1.1 - Bus_ID 0000:42
MPI 004 - OMP 000 - HWT 33-40 (Running on: 033) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 2.0 - Bus_ID 0000:6c
MPI 005 - OMP 000 - HWT 41-48 (Running on: 041) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 2.1 - Bus_ID 0000:6c
MPI 006 - OMP 000 - HWT 53-60 (Running on: 057) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 3.0 - Bus_ID 0001:18
MPI 007 - OMP 000 - HWT 61-68 (Running on: 061) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 3.1 - Bus_ID 0001:18
MPI 008 - OMP 000 - HWT 69-76 (Running on: 069) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 4.0 - Bus_ID 0001:42
MPI 009 - OMP 000 - HWT 77-84 (Running on: 077) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 4.1 - Bus_ID 0001:42
MPI 010 - OMP 000 - HWT 85-92 (Running on: 085) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 5.0 - Bus_ID 0001:6c
MPI 011 - OMP 000 - HWT 93-100 (Running on: 093) - Node x4206c0s0b0n0 - RT_GPU_ID 0 - GPU_ID 5.1 - Bus_ID 0001:6c
```

For appropriate binding of CPUs and GPUs in Aurora, please refer to https://docs.alcf.anl.gov/aurora/running-jobs-aurora/#running-mpiopenmpsycl-applications

Common Mistakes with Porting to SYCL

Mistakes: Thread Indexing Assumptions

- CUDA is heavily modeled on C conventions, where as SYCL is based on C++
- In CUDA: Dimension ordering is: X (fastest), then Y, then Z.
- In SYCL: Dimension ordering is reversed: Dimension 2 (Z), Dimension 1 (Y), then Dimension 0 (X).
- SYCLomatic helps with porting this common pitfall

```
// CUDA
__global__ void my_kernel(float* data) {
    int x = threadIdx.x;
    int y = threadIdx.y;
    int z = threadIdx.z;

    int idx = z * blockDim.y * blockDim.x + y * blockDim.x +
x;
    data[idx] += 1.0f;
}

// SYCL
h.parallel_for(nd_range<3>(global, local), [=](nd_item<3>
item) {
    int x = item.get_local_id(2); // Dimension 2 maps to
CUDA x
    int y = item.get_local_id(1); // Dimension 1 maps to
CUDA y
    int z = item.get_local_id(0); // Dimension 0 maps to
CUDA z

    int idx = z * blockDimY * blockDimX + y * blockDimX + x;
    data[idx] += 1.0f;
});
```

Mistakes: Synchronization & Execution Semantics

- CUDA: Conditional Synchronization
- Only threads that enter the `__syncthreads()` call synchronize. Other threads proceed without synchronization, leading to potential race conditions.
- CUDA allows threads to proceed even if they don't synchronize

CUDA:

```
__global__ void kernel() {
    int tid = threadIdx.x;

    // Some computation
    if (tid < 10) {
        // Only threads with tid < 10 will synchronize here
        __syncthreads();
    }

    // Code after synchronization
}
```

Mistakes: Synchronization & Execution Semantics

- SYCL: Conditional Synchronization
- If not all threads in the work-group synchronize, it can lead to deadlocks because waiting threads never reach the barrier
- SYCL requires all threads in a work-group to synchronize; missing this leads to deadlocks.

SYCL:

```
q.submit([&](sycl::handler& h) {
    h.parallel_for(sycl::range<1>(256), [=](sycl::id<1> i) {
        int tid = i[0];
        if (tid < 10) {
            sycl::group_barrier(); // Only threads with tid < 10
synchronize
        }
    });
});
```

Guidance: Ensure all threads perform work-group synchronizations/barrier. SYCL provides an experimental feature such as `sycl::ext::oneapi::experimental::get_ballot_group`

Mistakes: Usage of CUDA's constant memory

- Constant Memory in CUDA is a read-only memory space that is cached and optimized for efficient access by all threads in a block.
- It's intended for values that are uniform across all threads in a kernel.
- It's cached and has faster access than global memory when accessed by multiple threads.
- Size Limit: 64 KB.

CUDA:

```cuda
// CUDA example with constant memory
__constant__ float const_data[1024];

__global__ void kernel() {
    int idx = threadIdx.x;
    printf("Value: %f\n", const_data[idx]);
}

int main() {
    float data[1024] = { ... }; // Initialize data
    cudaMemcpyToSymbol(const_data, data, sizeof(float) * 1024);
    kernel<<<1, 1024>>>();
}
```

Mistakes: Usage of CUDA's constant memory

- SYCL does not have a direct equivalent to CUDA constant memory.
- In SYCL, memory spaces are abstracted differently, and there's no specific constant memory model.
- There are two possible solutions:
  (a) Constant variables whose values are known at compile-time
  (b) Constant variables whose values are dynamically assigned at runtime

SYCL:

```
int main() {
    queue q;
    device_global<float[1024]> const_data;

    q.submit([&](handler& h) {
        h.parallel_for(range<1>(1024), [=](id<1> idx) {
            printf("Value: %f\n", const_data[idx]);
        });
    }).wait();
}
```

Mistakes: Usage of CUDA's constant memory

- There are two possible solutions:

- (a) Constant variables whose values are known at compile-time

  `static constexpr float const_data[1024] = {...}`

  (b) Constant variables whose values are dynamically assigned at runtime

  `device_global` allows defining data that is consistent across device kernels, offering a cross-platform alternative to constant memory.

SYCL:

```
int main() {
    queue q;
    device_global<float[1024]> const_data;

    q.submit([&](handler& h) {
        h.parallel_for(range<1>(1024), [=](id<1> idx) {
            printf("Value: %f\n", const_data[idx]);
        });
    }).wait();
}
```

Mistakes: Implicit Synchronization – cudaFree

- In CUDA, cudaFree(ptr) implicitly synchronizes the device.
- All previously issued kernels must complete before the memory is freed.
- Guarantees that, no running kernel will access freed memory.
- No explicit `cudaDeviceSynchronize()` is needed before freeing.
- This protects against common "use-after-free" bugs.

CUDA:

```
float* d_data;
cudaMalloc(&d_data, size);
kernel<<<blocks, threads>>>(d_data);
// No cudaDeviceSynchronize() needed
cudaFree(d_data); // Implicit synchronization
```

Mistakes: Implicit Synchronization – cudaFree

- In SYCL, freeing Unified Shared Memory (USM) with sycl::free() does not synchronize automatically.
- Operations on the queue are asynchronous by default.
- Freeing device memory before pending kernels finish can cause undefined behavior or crashes.

SYCL:

```
auto* d_data = sycl::malloc_device<float>(size, queue);
queue.parallel_for(..., [=](...){ d_data[...] = ...; });

// WRONG: No sync before free
sycl::free(d_data, queue); // Potential use-after-free
```

Mistakes: Implicit Synchronization – cudaFree

- Proper Synchronization before Memory deallocation
- Always ensure that all commands depending on the memory are completed before calling `sycl::free()`
- Use `queue.wait()` to synchronize.
- But over using of synchronization leads to degraded performance

SYCL:

```
auto* d_data = sycl::malloc_device<float>(size, queue);
queue.parallel_for(..., [=](...){ d_data[...] = ...; });

queue.wait();          // Explicitly wait for completion
sycl::free(d_data, queue);
```

Mistakes: Warp-specific Programming CUDA vs SYCL

- CUDA offers warp intrinsics (e.g., `__shfl_sync`, `__ballot_sync`, `__any_sync`) assuming 32 threads per warp.
- SYCL replaces warps with sub-groups, but sub-group size is device-dependent (not always 32!)
- Operations vary across hardware (Intel, AMD, NVIDIA backends)

CUDA:

```
__global__ void shuffle_example(int *data) {
    int lane = threadIdx.x % 32;
    int value = data[threadIdx.x];

    // Shuffle value from thread (lane + 1) % 32
    int shuffled = __shfl_sync(0xFFFFFFFF, value, (lane + 1)
% 32);

    data[threadIdx.x] = shuffled;
}
```

Mistakes: Warp-specific Programming CUDA vs SYCL

- Assuming warp size = 32 can cause functional or performance issues.
- Direct translation of warp intrinsics (like shuffle) needs careful subgroup handling.
- Device portability issues — different subgroup sizes across vendors.

SYCL:

```
q.submit([&](sycl::handler &h) {
    h.parallel_for(sycl::nd_range<1>(256, 32), [=]
(sycl::nd_item<1> item) {
        auto sg = item.get_sub_group();
        int lane = sg.get_local_id();
        int value = data[item.get_global_id()];

        int shuffled = sycl::shift_group_left(sg, value, 1);

        data[item.get_global_id()] = shuffled;
    });
});
```

Mistakes: Warp-specific Programming CUDA vs SYCL

- Always use sub_group APIs
- Query subgroup size: `sg.get_local_range().size()` instead of assuming 32
- Design algorithms to handle variable subgroup sizes dynamically
- Synchronization (`__syncwarp()` in CUDA) must be adapted to `group_barrier(sg)` for correctness.

SYCL:

```
q.submit([&](sycl::handler &h) {
    h.parallel_for(sycl::nd_range<1>(256, 32), [=]
(sycl::nd_item<1> item) {
        auto sg = item.get_sub_group();
        int lane = sg.get_local_id();
        int value = data[item.get_global_id()];

        int shuffled = sycl::shift_group_left(sg, value, 1);

        data[item.get_global_id()] = shuffled;
    });
});
```

How to manage unsupported APIs

- DPCT provides warnings when a API can’t be ported. Please do create an issue here: https://github.com/oneapi-src/SYCLomatic/issues
- SYCL experimental extensions (These are extensions developed for DPC++ implementation and not yet part of SYCL specification)
- SYCL interoperability (Obtains native CUDA/HIP/L0 objects)
- When porting PTX or assembly for Nvidia hardware, feel free to reach out to us

Instructions to build SYCLomatic (Optional)

Repository: https://github.com/oneapi-src/SYCLomatic

Setup Environment:

```
export SYCLOMATIC_HOME=~/workspace
export
PATH_TO_C2S_INSTALL_FOLDER=~/workspace/c2s_install
mkdir $SYCLOMATIC_HOME
cd $SYCLOMATIC_HOME

git clone https://github.com/oneapi-src/SYCLomatic.git
```

Build Instructions:

```
cd $SYCLOMATIC_HOME
mkdir build
cd build
cmake -G Ninja -DCMAKE_INSTALL_PREFIX=$PATH_TO_C2S_INSTALL_FOLDER  -DCMAKE_BUILD_TYPE=Release
-DLLVM_ENABLE_PROJECTS="clang"  -DLLVM_TARGETS_TO_BUILD="X86;NVPTX" ../SYCLomatic/llvm
ninja install-c2s
```

Post Installation:

```
export PATH=$PATH_TO_C2S_INSTALL_FOLDER/bin:$PATH
export CPATH=$PATH_TO_C2S_INSTALL_FOLDER/include:$CPATH
```

ALCF Polaris: `module load oneapi/upstream` already has SYCLomatic

Things to notice with SYCLomatic

`https://github.com/oneapi-src/SYCLomatic`

1. SYCLomatic is a tool that bridges the cap between CUDA to SYCL

2. “dpct” namespace and headers are used for porting to SYCL

3. “dpct” headers and namespace are not standard SYCL APIs. They are merely wrappers/helpers

4. (Optional) For a SYCL specification complaint code (or) for production purpose: Consider manually replacing “dpct” with SYCL equivalents

Access to ALCF Polaris

https://docs.alcf.anl.gov/polaris/getting-started/

To login:

```
ssh username@polaris.alcf.anl.gov
Password:
```

Modules to Load:

```
module use /soft/modulefiles/
module load oneapi/upstream cmake
```

To get a compute node:

```
qsub -I -A PROJECTNAME -q QUEUENAME -l select=1:system=polaris -l walltime=60:00 -l filesystems=home:eagle
```

Experimental Support for CUDA and ROCm devices

Compiling With DPC++ for CUDA GPUs

The following command can be used to compile your code using DPC++ for CUDA backend:

```
clang++ -std=c++17 -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice -Xsycl-target-backend
--cuda-gpu-arch=sm_80 simple-sycl-app.cpp -o simple-sycl-app-cuda
```

Compiling With DPC++ for ROCm GPUs*

The following command can be used to compile your code using DPC++ for HIP backend:

```
clang++ -fsycl -fsycl-targets=amdgcn-amd-amdhsa -Xsycl-target-backend --offload-arch=gfx9xx
simple-sycl-app.cpp -o simple-sycl-app-rocm
```

*Currently tested for ROCm 4.2.0, gfx906 and gfx908 for MI50 and MI100 GPU targets respectively

Case Study 1: Transform reduce

Obvious differences in the inclusion of headers

```
#include <thrust/transform_reduce.h>

template<typename InputIterator, typename UnaryFunction, typename OutputType, typename
BinaryFunction>
OutputType thrust::transform_reduce(InputIterator first,
                                    InputIterator last,
                                    UnaryFunction unary_op,
                                    OutputType init,
                                    BinaryFunction binary_op)
```

From standard C++ Algorithms library

```
#include <oneapi/dpl/execution>
#include <oneapi/dpl/algorithm>

template< class ExecutionPolicy, class ForwardIt, class T,class BinaryReductionOp, class
UnaryTransformOp >
T transform_reduce( ExecutionPolicy&& policy,
                    ForwardIt first,
                    ForwardIt last,
                    T init,
                    BinaryReductionOp reduce,
                    UnaryTransformOp transform );
```

Case Study 1: Transform reduce

```
#include <thrust/transform_reduce.h>

template<typename InputIterator, typename UnaryFunction, typename OutputType, typename
BinaryFunction>
OutputType thrust::transform_reduce(InputIterator first,
                                    InputIterator last,
                                    UnaryFunction unary_op,
                                    OutputType init,
                                    BinaryFunction binary_op)
```

`ExecutionPolicy` arg, is an additional argument that enables parallelism on CPU or GPU

From standard C++ Algorithms library

```
#include <oneapi/dpl/execution>
#include <oneapi/dpl/algorithm>

template< class ExecutionPolicy, class ForwardIt, class T,class BinaryReductionOp,
class UnaryTransformOp >
T transform_reduce( ExecutionPolicy&& policy,
                    ForwardIt first,
                    ForwardIt last,
                    T init,
                    BinaryReductionOp reduce,
                    UnaryTransformOp transform );
```

Case Study 1: Transform reduce

```
#include <oneapi/dpl/execution>
#include <oneapi/dpl/algorithm>

template< class ExecutionPolicy, class ForwardIt, class T,class BinaryReductionOp,
class UnaryTransformOp >
T transform_reduce( ExecutionPolicy&& policy,
                    ForwardIt first,
                    ForwardIt last,
                    T init,
                    BinaryReductionOp reduce,
                    UnaryTransformOp transform );
```

1. oneDPL adopts uses the standard C++ Algorithms library

2. Most of the thrust algorithms maps to C++ parallel Algorithms library

3. How is oneDPL portable ?
   SYCL kernels are used underneath the oneDPL algorithms, that makes oneDPL portable to other vendors

Case Study 2: Porting cuBLAS API to oneMKL API

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float                 *C, int ldc)
```

Things to consider when porting existing cu* math libraries to oneMKL

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm(sycl::queue &queue,
                     onemkl::transpose transa, onemkl::transpose transb,
                     std::int64_t m, std::int64_t n, std::int64_t k,
                     Ts alpha,
                     const Ta    *a, std::int64_t lda,
                     const Tb    *b, std::int64_t ldb,
                     Ts beta,
                     Tc          *c, std::int64_t ldc,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Case Study 2: Porting cuBLAS API to oneMKL API

cuBLAS APIs are column-major by default &
oneMKL APIs provide both row-major & column-major

Choose `oneapi::mkl::blas::column_major` APIs

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float                 *C, int ldc)
```

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm(sycl::queue &queue,
                     onemkl::transpose transa, onemkl::transpose transb,
                     std::int64_t m, std::int64_t n, std::int64_t k,
                     Ts alpha,
                     const Ta    *a, std::int64_t lda,
                     const Tb    *b, std::int64_t ldb,
                     Ts beta,
                     Tc          *c, std::int64_t ldc,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Case Study 2: Porting cuBLAS API to oneMKL API

Return types from both the APIs are vastly different.

cuBLAS: Provides info of the status

oneMKL: provides an `sycl::event`

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float                 *C, int ldc)
```

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm(sycl::queue &queue,
                     onemkl::transpose transa, onemkl::transpose transb,
                     std::int64_t m, std::int64_t n, std::int64_t k,
                     Ts alpha,
                     const Ta    *a, std::int64_t lda,
                     const Tb    *b, std::int64_t ldb,
                     Ts beta,
                     Tc          *c, std::int64_t ldc,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Case Study 2: Porting cuBLAS API to oneMKL API

Scalar values (alpha & beta)

cuBLAS API: Requires a pointer

oneMKL API: Requires a scalar

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float                 *C, int ldc)
```

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm(sycl::queue &queue,
                     onemkl::transpose transa, onemkl::transpose transb,
                     std::int64_t m, std::int64_t n, std::int64_t k,
                     Ts alpha,
                     const Ta    *a, std::int64_t lda,
                     const Tb    *b, std::int64_t ldb,
                     Ts beta,
                     Tc          *c, std::int64_t ldc,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Case Study 2: Porting cuBLAS API to oneMKL API

Subtle details:

1. oneMKL (optional): Additional argument of a `sycl::event` to act as a dependency

2. `sycl::queue` is used for oneMKL APIs. There are no equivalents of a special BLAS handle similar to `cublasHandle_t` in oneMKL

Performance: oneMKL piggy-backs underneath by calling respective vendor APIs (cuBLAS/rocBLAS)

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float                 *C, int ldc)
```

```
namespace oneapi::mkl::blas::column_major {
    sycl::event gemm(sycl::queue &queue,
                     onemkl::transpose transa, onemkl::transpose transb,
                     std::int64_t m, std::int64_t n, std::int64_t k,
                     Ts alpha,
                     const Ta    *a, std::int64_t lda,
                     const Tb    *b, std::int64_t ldb,
                     Ts beta,
                     Tc          *c, std::int64_t ldc,
                     const std::vector<sycl::event> &dependencies = {})
}
```

Case Study 3: How to port a large project to SYCL

https://github.com/ccsb-scripps/AutoDock-GPU

```
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/cuda$ pwd
/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/cuda
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/cuda$ ls -ltr
total 2560
-rw-r--r--. 1 abagusetty jlse  5069 Jun 30 21:12 GpuData.h
-rw-r--r--. 1 abagusetty jlse  5996 Jun 30 21:12 auxiliary_genetic.cu
-rw-r--r--. 1 abagusetty jlse 41373 Jun 30 21:12 calcMergeEneGra.cu
-rw-r--r--. 1 abagusetty jlse 17544 Jun 30 21:12 calcenergy.cu
-rw-r--r--. 1 abagusetty jlse  4757 Jun 30 21:12 constants.h
-rw-r--r--. 1 abagusetty jlse  2671 Jun 30 21:12 kernel1.cu
-rw-r--r--. 1 abagusetty jlse  2464 Jun 30 21:12 kernel2.cu
-rw-r--r--. 1 abagusetty jlse 10983 Jun 30 21:12 kernel3.cu
-rw-r--r--. 1 abagusetty jlse 11317 Jun 30 21:12 kernel4.cu
-rw-r--r--. 1 abagusetty jlse 15668 Jun 30 21:12 kernel_ad.cu
-rw-r--r--. 1 abagusetty jlse 15405 Jun 30 21:12 kernel_adam.cu
-rw-r--r--. 1 abagusetty jlse  5368 Jun 30 21:12 kernels.cu
-rw-r--r--. 1 abagusetty jlse    30 Jun 30 22:00 kernels.o
```

Native CUDA kernels (files with extensions .cu)

SYCL (ported files with extensions dp.cpp)

```
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/dpcpp_out$ ls -ltr
total 2304
-rw-r--r--. 1 abagusetty jlse  19962 Jun 30 22:09 kernel_adam.dp.cpp
-rw-r--r--. 1 abagusetty jlse  21385 Jun 30 22:09 kernel_ad.dp.cpp
-rw-r--r--. 1 abagusetty jlse  21279 Jun 30 22:09 kernel4.dp.cpp
-rw-r--r--. 1 abagusetty jlse  20303 Jun 30 22:09 kernel3.dp.cpp
-rw-r--r--. 1 abagusetty jlse   6414 Jun 30 22:09 GpuData.h
-rw-r--r--. 1 abagusetty jlse   3818 Jun 30 23:31 kernel1.dp.cpp
-rw-r--r--. 1 abagusetty jlse   3809 Jun 30 23:32 kernel2.dp.cpp
-rw-r--r--. 1 abagusetty jlse   6341 Jun 30 23:35 auxiliary_genetic.dp.cpp
-rw-r--r--. 1 abagusetty jlse  22969 Jun 30 23:59 calcenergy.dp.cpp
-rw-r--r--. 1 abagusetty jlse  61223 Jul  1 00:05 calcMergeEneGra.dp.cpp
-rw-r--r--. 1 abagusetty jlse 105717 Jul  1 13:23 kernels.dp.cpp
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/dpcpp_out$
```

Case Study 3: How to port a large project to SYCL

https://github.com/ccsb-scripps/AutoDock-GPU

```
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/cuda$ pwd
/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/cuda
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/cuda$ ls -ltr
total 2560
-rw-r--r--. 1 abagusetty jlse  5069 Jun 30 21:12 GpuData.h
-rw-r--r--. 1 abagusetty jlse  5996 Jun 30 21:12 auxiliary_genetic.cu
-rw-r--r--. 1 abagusetty jlse 41373 Jun 30 21:12 calcMergeEneGra.cu
-rw-r--r--. 1 abagusetty jlse 17544 Jun 30 21:12 calcenergy.cu
-rw-r--r--. 1 abagusetty jlse  4757 Jun 30 21:12 constants.h
-rw-r--r--. 1 abagusetty jlse  2671 Jun 30 21:12 kernel1.cu
-rw-r--r--. 1 abagusetty jlse  2464 Jun 30 21:12 kernel2.cu
-rw-r--r--. 1 abagusetty jlse 10983 Jun 30 21:12 kernel3.cu
-rw-r--r--. 1 abagusetty jlse 11317 Jun 30 21:12 kernel4.cu
-rw-r--r--. 1 abagusetty jlse 15668 Jun 30 21:12 kernel_ad.cu
-rw-r--r--. 1 abagusetty jlse 15405 Jun 30 21:12 kernel_adam.cu
-rw-r--r--. 1 abagusetty jlse  5368 Jun 30 21:12 kernels.cu
-rw-r--r--. 1 abagusetty jlse    30 Jun 30 22:00 kernels.o
```

Native CUDA kernels (files with extensions .cu)

SYCL (ported files with extensions dp.cpp)

```
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/dpcpp_out$ ls -ltr
total 2304
-rw-r--r--. 1 abagusetty jlse  19962 Jun 30 22:09 kernel_adam.dp.cpp
-rw-r--r--. 1 abagusetty jlse  21385 Jun 30 22:09 kernel_ad.dp.cpp
-rw-r--r--. 1 abagusetty jlse  21279 Jun 30 22:09 kernel4.dp.cpp
-rw-r--r--. 1 abagusetty jlse  20303 Jun 30 22:09 kernel3.dp.cpp
-rw-r--r--. 1 abagusetty jlse   6414 Jun 30 22:09 GpuData.h
-rw-r--r--. 1 abagusetty jlse   3818 Jun 30 23:31 kernel1.dp.cpp
-rw-r--r--. 1 abagusetty jlse   3809 Jun 30 23:32 kernel2.dp.cpp
-rw-r--r--. 1 abagusetty jlse   6341 Jun 30 23:35 auxiliary_genetic.dp.cpp
-rw-r--r--. 1 abagusetty jlse  22969 Jun 30 23:59 calcenergy.dp.cpp
-rw-r--r--. 1 abagusetty jlse  61223 Jul  1 00:05 calcMergeEneGra.dp.cpp
-rw-r--r--. 1 abagusetty jlse 105717 Jul  1 13:23 kernels.dp.cpp
abagusetty@jlselogin2:/gpfs/jlse-fs0/users/abagusetty/AutoDock-GPU/dpcpp_out$
```

Performance: SYCL on NERSC Perlmutter A100 GPUs

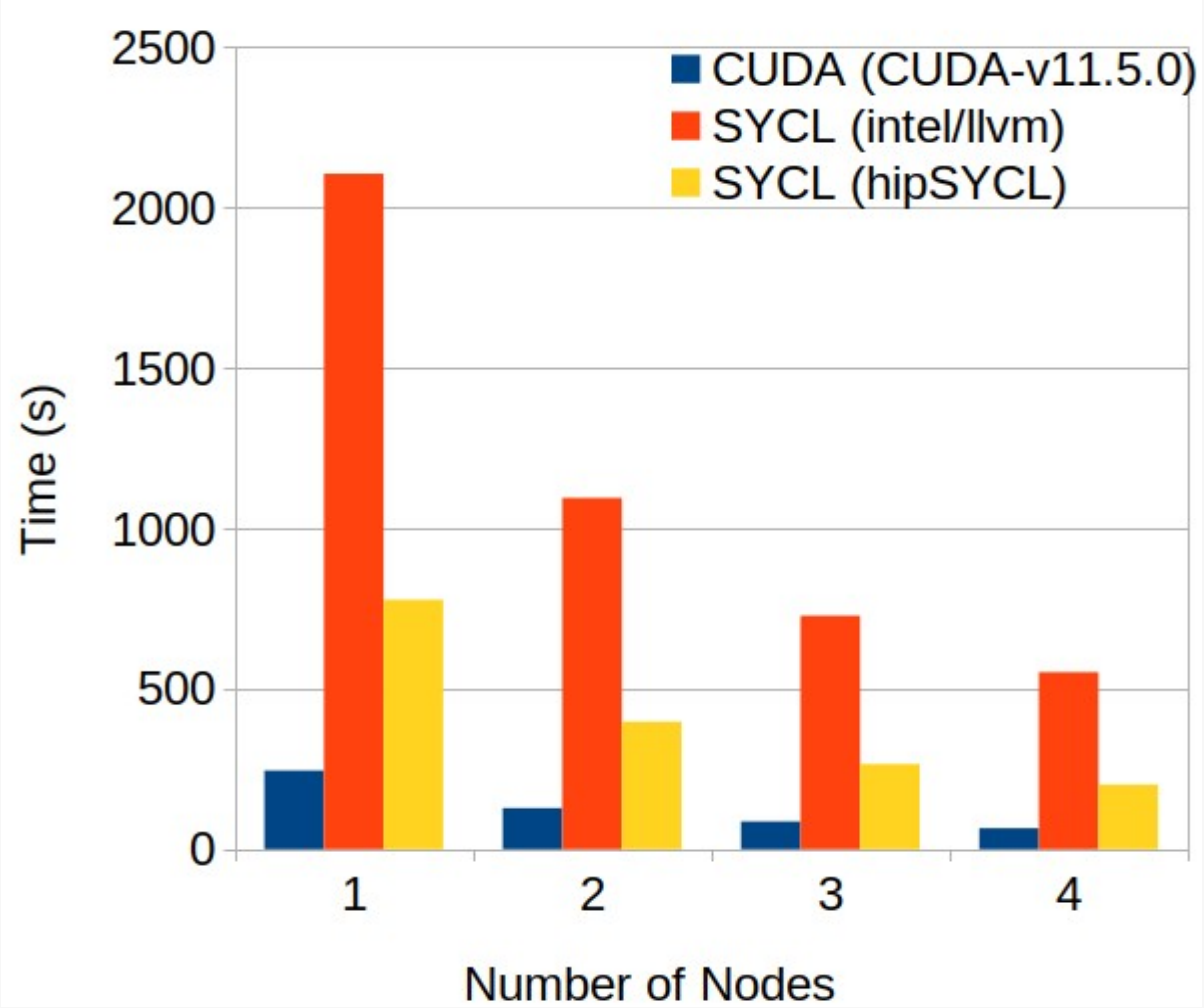
Performance Highlights

- Performance of NWChemEx-TAMM CCSD(T). Timings shown for the expensive (T) part.
- hipSYCL compiler offers best performance in comparison to intel/llvm DPC++ compiler on Nvidia GPUs
- hipSYCL (CUDA) backend is about 3x slower than optimized non-tensor core CUDA code
- intel/llvm (CUDA) backend is about 8.5x slower than optimized non-tensor core CUDA code

Functionality Highlights

- SYCL performance on NVidia devices is almost feature complete
- Areas to improve include:
  (a) better runtime support
  (b) improved integration with CMake
  (c) improved integration with portable Math libraries (like oneMKL)

(lower the better)



Note: Performance on ALCF Polaris is exactly same as Perlmutter, given similarities in system architecture, programming environment

Performance: SYCL on OLCF Crusher MI-250x GPUs

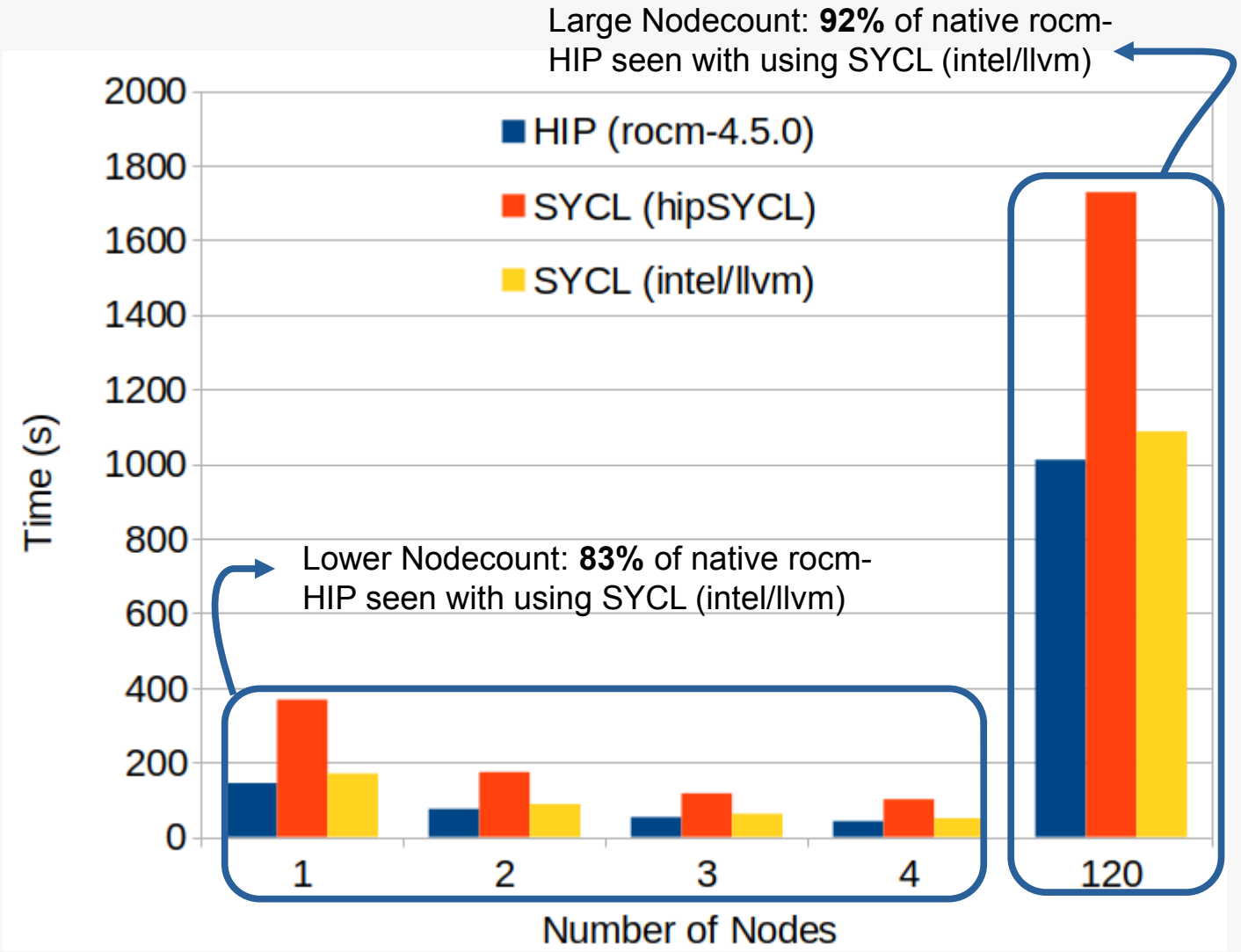
Performance Highlights

- Performance of NWChemEx-TAMM CCSD(T). Timings shown for the expensive (T) part.
- Portability using SYCL was **83-92%** performant in comparison to native rocm-HIP with NWChemEx-TAMM
- About ~**92%** of the native HIP performance was seen with intensive workload on large-node counts (120 nodes/960 GPUs)
- Consistent performance of **~83%** was seen with smaller node count of 1-4 nodes (i.e.,8-32 GPUs)
- intel/llvm compiler yielded the best portable SYCL performance on AMD MI-250X devices

Functionality Highlights

Basic features implementation is complete

intel/llvm SYCL portability can possibly close the performance gap with rocm-HIP, if resource are allocated for :
(a) better runtime support
(b) MI250x/*gfx-90a* arch specific performance tuning
(c) Matrix-cores ISA instructions through SYCL APIs



*Strong Scaling: Fixed problem size with varying resources

Useful Resources

oneAPI Beta downloads and documentation:

https://software.intel.com/content/www/us/en/develop/tools/oneapi.html

DPC++ Compatibility Tool Getting Started:

https://software.intel.com/content/www/us/en/develop/documentation/get-started-with-intel-dpcpp-compatibility-tool/top.html

DPC++ Compatibility Tool User Guide:

https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top.html

DevCloud access:

https://intelsoftwaresites.secure.force.com/devcloud/oneapi

Codeplay migration docs:

https://developer.codeplay.com/products/computecpp/ce/guides/sycl-for-cuda-developers

https://developer.codeplay.com/products/computecpp/ce/guides/sycl-for-cuda-developers/migration