

SYCL on ALCF Machines

Abhishek (Abhi) Bagusetty

Performance Engineering Group
Argonne Leadership Computing Facility

abagusetty@anl.gov

Agenda

Introduction: Heterogeneous Computing

Why SYCL ?

SYCL as Portable Programming Model

Execution Model

Memory Model

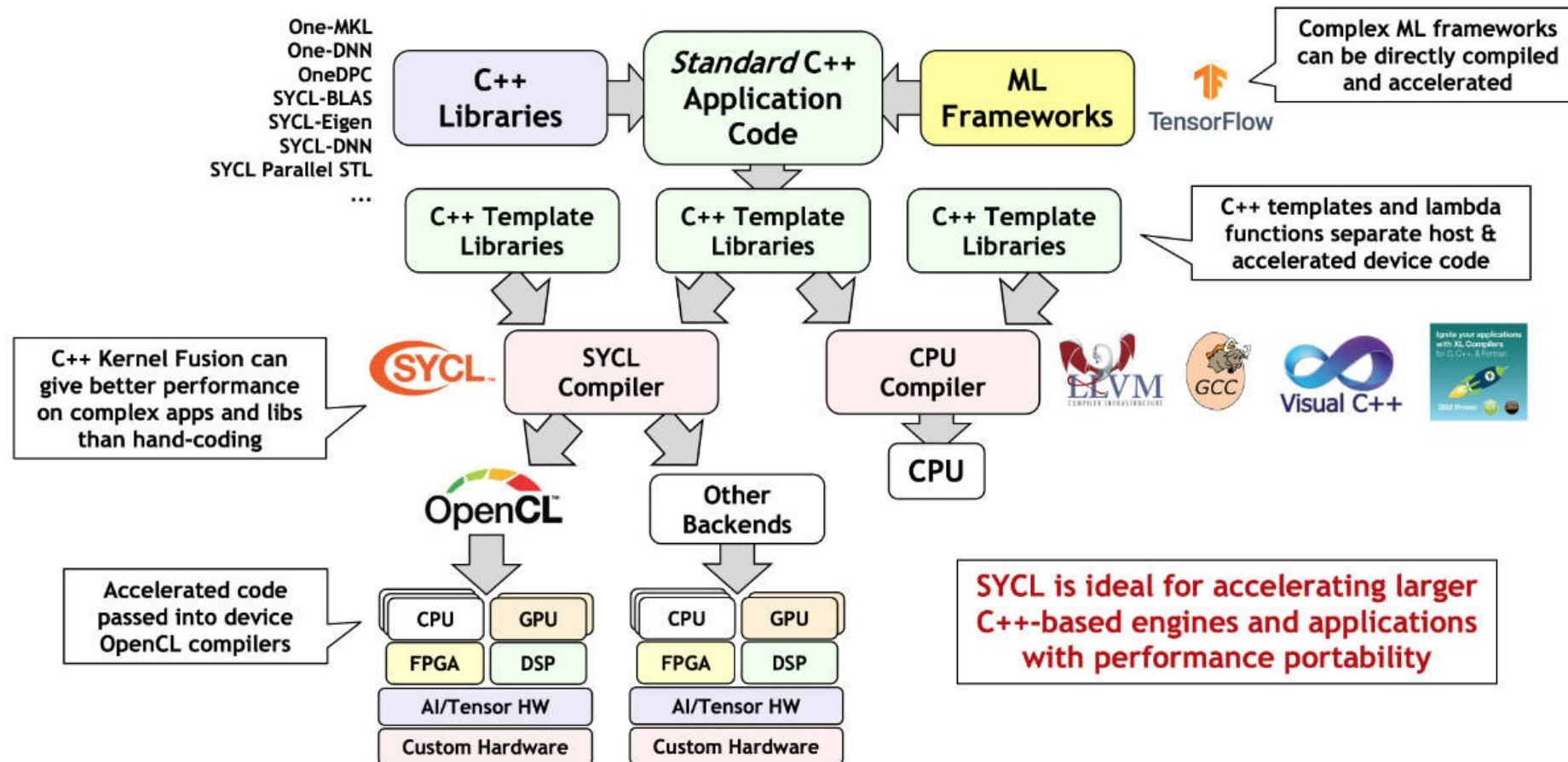
Best Practices

A few practical case studies

SYCL – A Portable Programming Model

A C++-based programming model for Intra-node parallelism

- SYCL is a specification and “not” an implementation, currently compliant to C++17 ISO standards
- Cross-platform abstraction layer, heavily backed by industry
- Open-source, vendor agonistic
- Single-source model (Unified development model across NVIDIA, AMD & Intel GPUs)



Khronos Group

The Khronos Group consortium develops and maintains some of the key open standards driving today's compute, graphics, and media innovation.

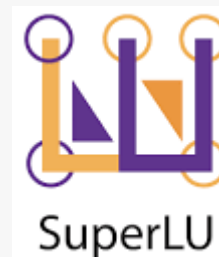
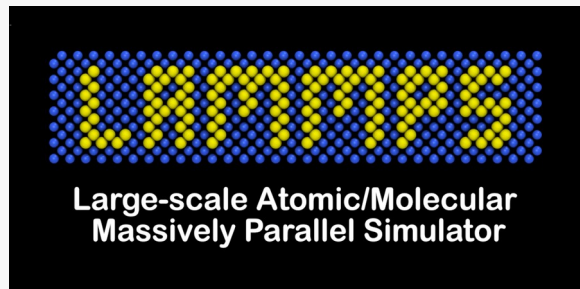
Active Standards



Exploratory Groups



Applications/Libraries using SYCL



How to Join SYCL Resources

SYCL Slack: <https://khr.io/slack>

SYCL Resources: <https://www.khronos.org/sycl/resources>

Issues or Questions or Contribute: <https://github.com/intel/llvm/issues>

SYCL API Reference Documentation: <https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html>

SYCL – Motivation

oneAPI Implementation of SYCL = C++ and SYCL* standard and extensions

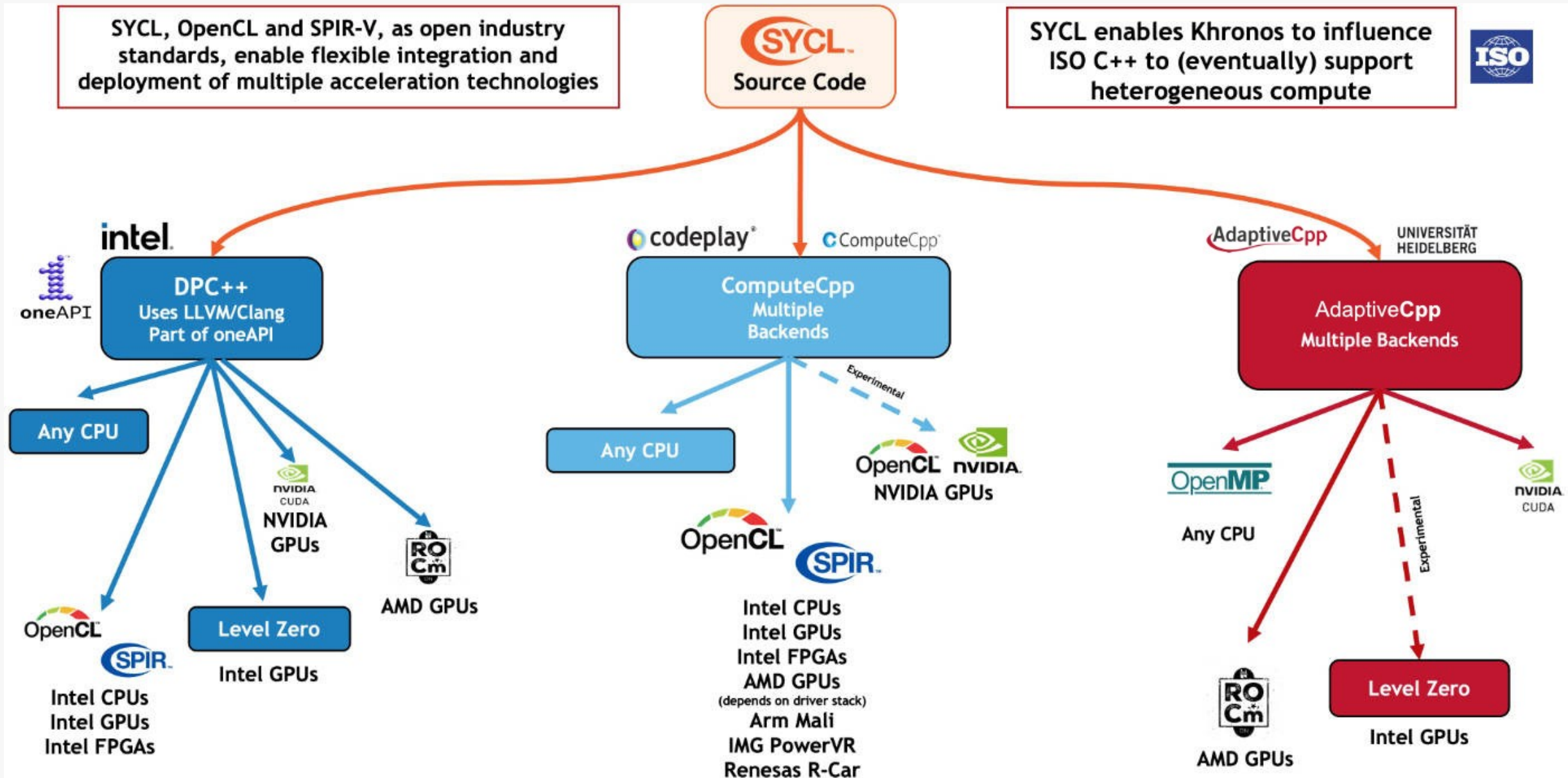
Based on modern C++

- ✓ C++ productivity features and familiar constructs

Standards-based, cross-architecture

- ✓ Incorporates the SYCL standard for data parallelism and heterogeneous programming

SYCL – Compiler Players



SYCL Programming Model

Concepts	CUDA	SYCL
Device Selection	<code>cudaSetDevice()</code>	<code>sycl::queue q{ sycl::gpu_selector{} };</code>
Memory Allocation	<code>cudaMalloc, cudaFree</code>	USM: <code>malloc_device, malloc_shared</code> (or) Buffers
Data Transfer	<code>cudaMemcpy</code>	Buffers auto-manage or memcpy with USM
Kernel Launch	<code>myKernel<<<grid,block>>>(args...)</code>	<code>q.parallel_for(nd_range, [=] (sycl::nd_item<>) { myKernel(args...) });</code>
Thread Indexing	<code>threadIdx.x + blockIdx.x * blockDim.x</code>	<code>item.get_id(0)</code> or <code>nd_item.get_global_id(0)</code>
Thread Groups	<code>blockIdx, threadIdx, warps (32)</code>	work-groups, work-items, subgroups (size may differ)
Streams/Queues	<code>cudaStream_t</code>	<code>sycl::queue</code> (in-order or out-of-order)
Libraries	cuBLAS, NCCL, cuRAND, cupy, etc	oneMKL, oneCCL, oneDPL, dpnp, etc.

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
```

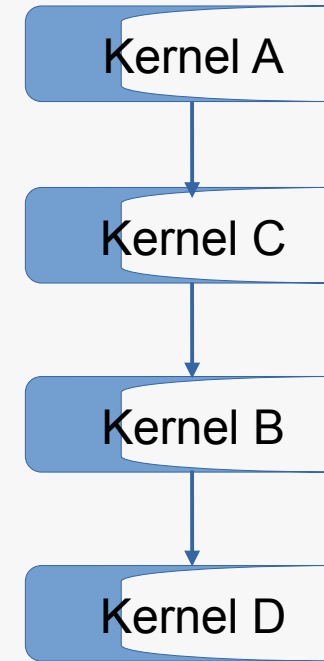
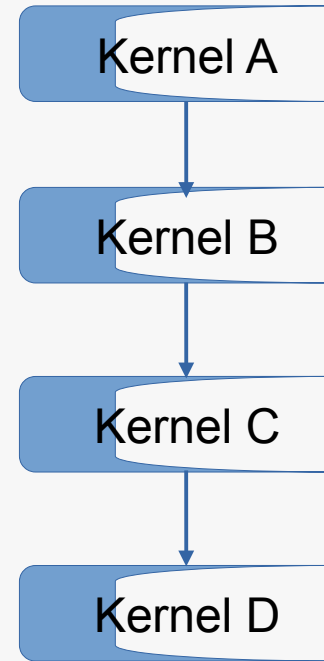
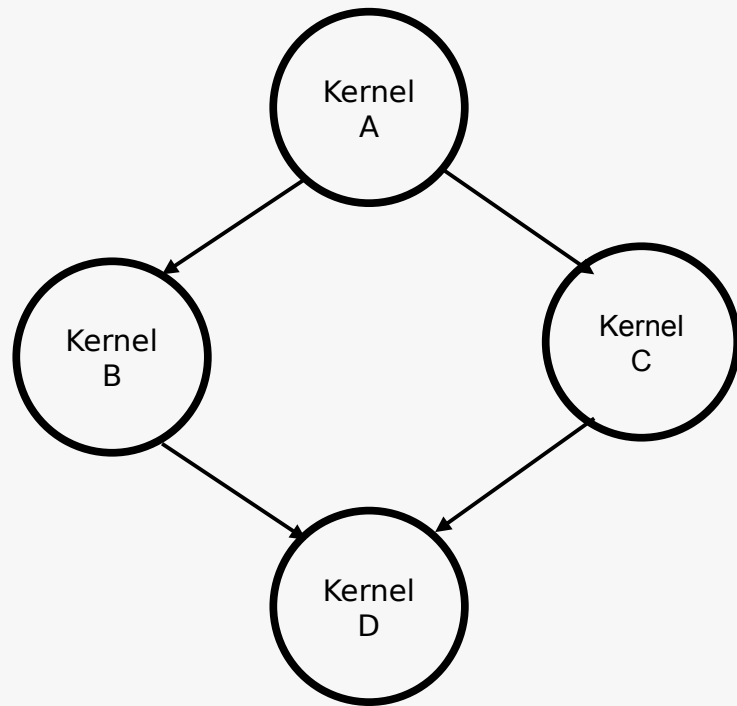
- **Context**

- 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

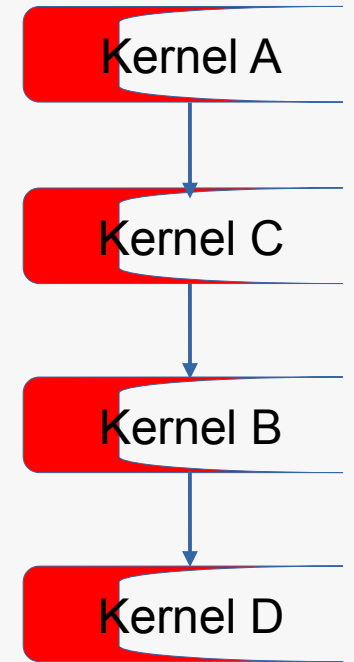
- **Queue (aka CUDA Stream)**

- 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 means 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

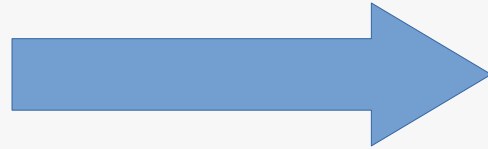
```
event memcpy(void* dest, const void* src, size_t numBytes);  
event memcpy(void* dest, const void* src, size_t numBytes, event depEvent);  
event memcpy(void* dest, const void* src, size_t numBytes,  
             const std::vector<event>& depEvents);
```

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)

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

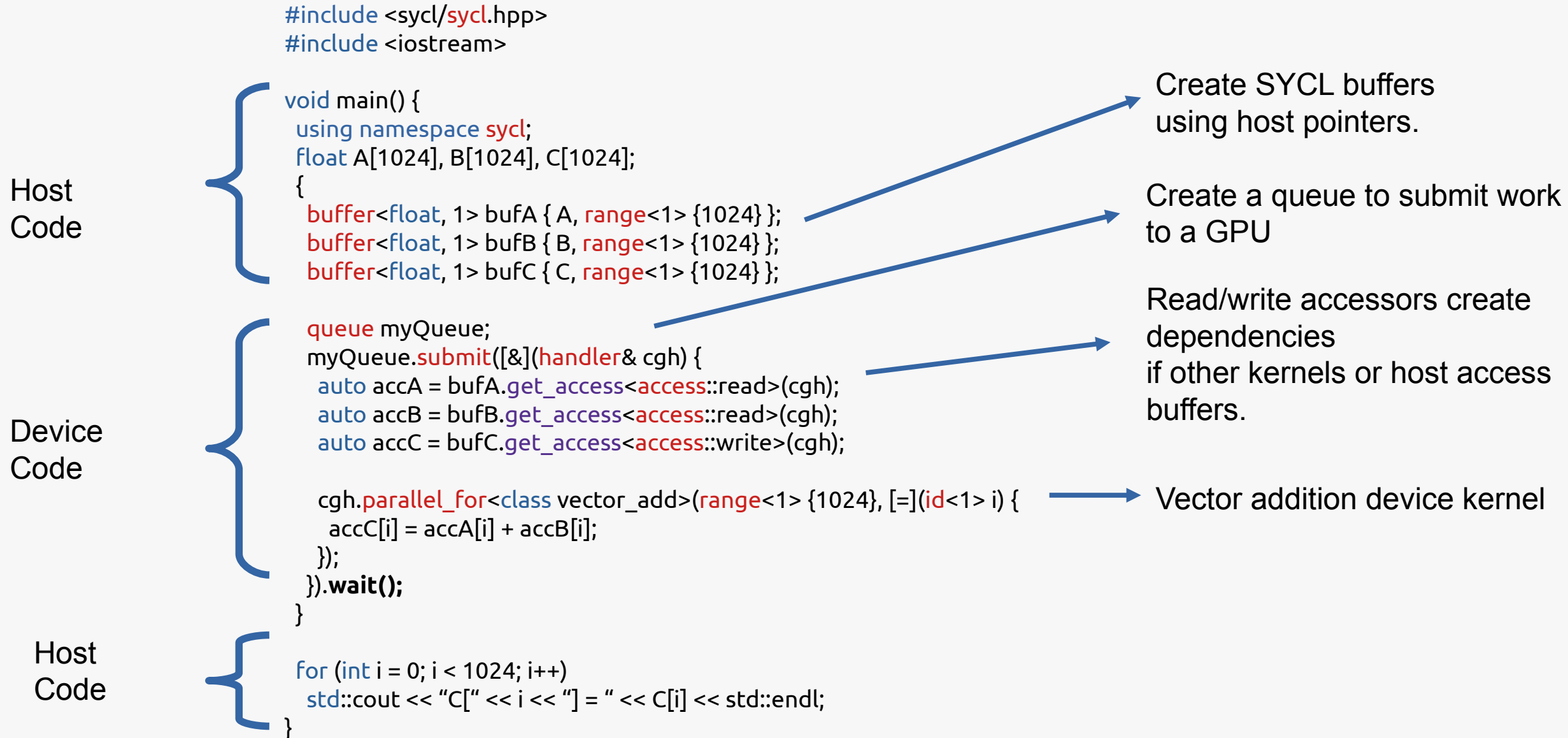
- SYCL's Global/Device allocated memory is only **valid** on the **device**
- More importantly not accessible from host

Thread Indexing

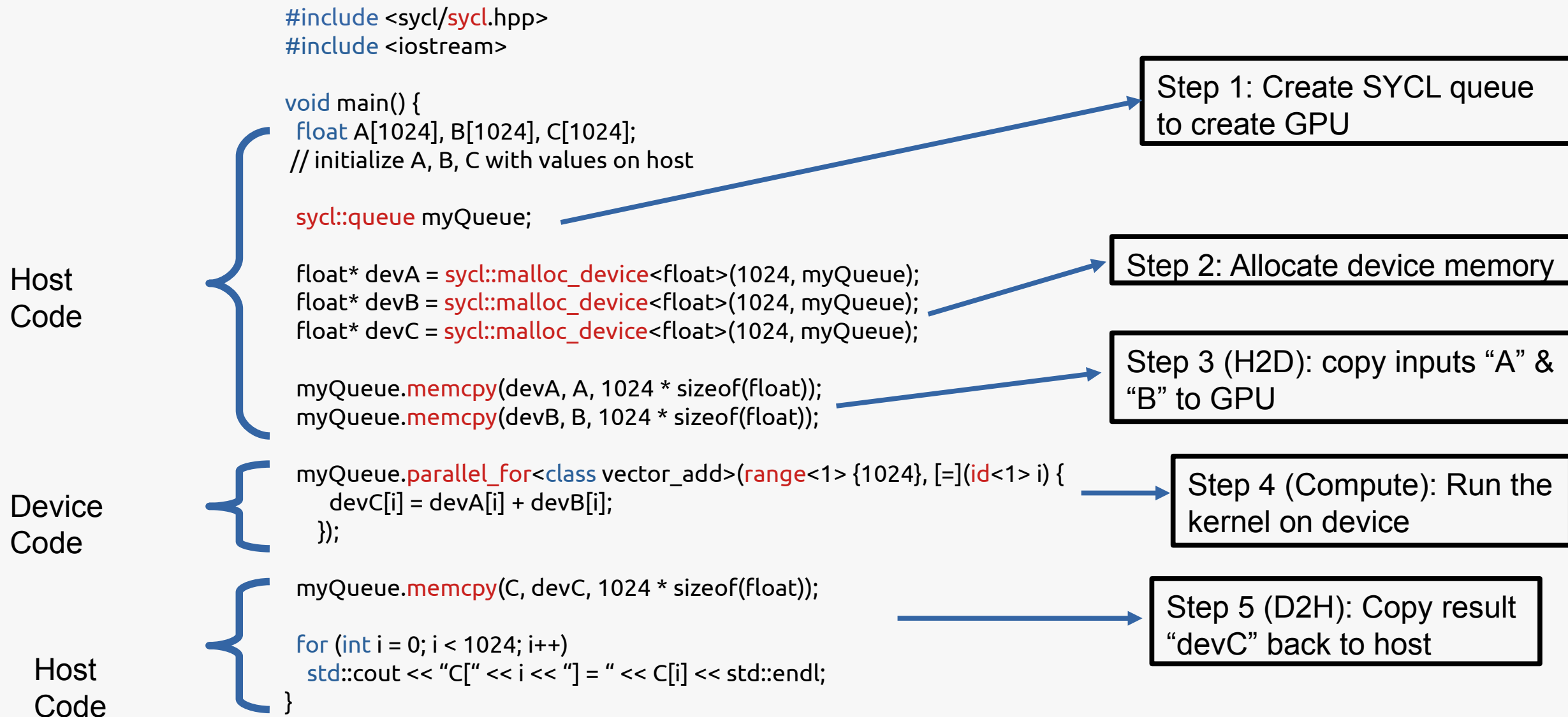
CUDA	SYCL
gridDim.x/y/z	sycl::nd_item.get_group_range(2/1/0)
blockIdx.x/y/z	sycl::nd_item.get_group(2/1/0)
blockDim.x/y/z	sycl::nd_item.get_local_range().get(2/1/0)
threadIdx.x/y/z	sycl::nd_item.get_local_id(2/1/0)
warpSize	sycl::nd_item.get_sub_group().get_local_range().get(0)

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

Vector Addition: SYCL Buffer memory model



Vector Addition: SYCL USM memory model



Vector Addition: SYCL USM memory model

Host
Code

```
#include <sycl/sycl.hpp>
#include <iostream>
```

```
void main() {
    float A[1024], B[1024], C[1024];
    // initialize A, B, C with values on host
```

```
    sycl::queue myQueue;
```

```
    float* devA = sycl::malloc_device<float>(1024, myQueue);
    float* devB = sycl::malloc_device<float>(1024, myQueue);
    float* devC = sycl::malloc_device<float>(1024, myQueue);
```

```
    myQueue.memcpy(devA, A, 1024 * sizeof(float));
    myQueue.memcpy(devB, B, 1024 * sizeof(float));
```

Device
Code

```
    myQueue.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
        devC[i] = devA[i] + devB[i];
    });
```

```
    myQueue.memcpy(C, devC, 1024 * sizeof(float));
```

Host
Code

```
    for (int i = 0; i < 1024; i++)
        std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

SYCL queue (by-default) is out-of-order. (i.e., the execution starts when possible. Duty of programmer to assure correct dependencies

myQueue.wait(), wait for H2D to complete before starting the kernel

myQueue.wait(), wait for the kernel to finish

myQueue.wait(), wait for D2H to complete before printing "C"

Vector Addition: SYCL USM memory model

```
#include <sycl/sycl.hpp>
#include <iostream>
```

SYCL queue (in-order) i.e., FIFO
like `cudaStream_t`

Host
Code

```
void main() {
    float A[1024], B[1024], C[1024];
    // initialize A, B, C with values on host

    sycl::queue myQueue(sycl::property_list{sycl::property::queue::in_order{}});

    float* devA = sycl::malloc_device<float>(1024, myQueue);
    float* devB = sycl::malloc_device<float>(1024, myQueue);
    float* devC = sycl::malloc_device<float>(1024, myQueue);

    myQueue.memcpy(devA, A, 1024 * sizeof(float));
    myQueue.memcpy(devB, B, 1024 * sizeof(float));
```

Device
Code

```
myQueue.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
    devC[i] = devA[i] + devB[i];
});
```

Host
Code

```
myQueue.memcpy(C, devC, 1024 * sizeof(float));

for (int i = 0; i < 1024; i++)
    std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

`myQueue.wait()`, wait for D2H to
complete before printing "C"

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 PrgEnv-gnu  
cuda-toolkit-standalone/12.9.0 spack-pe-base  
cmake mpiwrappers/cray-mpich-oneapi-upstream
```

To get a compute node:

```
qsub -I -l select=1,walltime=0:60:00,filesystems=home:grand -A alcf_training -q HandsOnHPC
```

Access to ALCF Aurora

<https://docs.alcf.anl.gov/aurora/getting-started-on-aurora/>

To login:

```
ssh username@aurora.alcf.anl.gov
```

Password:

Modules to Load:

```
module restore
```

To get a compute node:

```
qsub -I -l select=1,walltime=0:60:00,filesystems=home:eagle -A alcf_training -q HandsOnHPC
```

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=amdgc-n-d-amdhsa -Xsycl-target-backend --offload-arch=gfx9xx  
simple-sycl-app.cpp -o simple-sycl-app-rocm
```

*Tested for ROCm 6.4.2, gfx90a for AMD250x on Frontier

oneAPI core elements



oneDPL

oneAPI Data Parallel C++ Library

A companion to the DPC++ Compiler for programming oneAPI devices with APIs from C++ standard library, Parallel STL, and extensions.



oneDNN

oneAPI Deep Neural Network Library

High performance implementations of primitives for deep learning frameworks.



oneCCL

oneAPI Collective Communications Library

Communication primitives for scaling deep learning frameworks across multiple devices.



Level Zero

oneAPI Level Zero

System interface for oneAPI languages and libraries.



oneDAL

oneAPI Data Analytics Library

Algorithms for accelerated data science.



oneTBB

oneAPI Threading Building Blocks

Library for adding thread-based parallelism to complex applications on multiprocessors.



oneMKL

oneAPI Math Kernel Library

High performance math routines for science, engineering, and financial applications.

Porting to SYCL using Math Libraries (oneMKL/oneMath, oneDPL)

oneMKL consists of three parts:

- The oneMKL specification - part of the “oneAPI” specification
 - ✓ oneMKL component provides mathematical routines for HPC applications, etc
- An open-source library implementing the MKL API - “[oneMKL Interfaces](#)”
- The UXL (Unified Acceleration) Foundation develops these specifications
- The specification is open-source, available on GitHub
- The original Intel optimized math routines - for clarity, Intel® MKL



Porting to SYCL using Math Libraries (oneMKL Interface)

Implements the oneMKL specification, dispatching to other libraries underneath

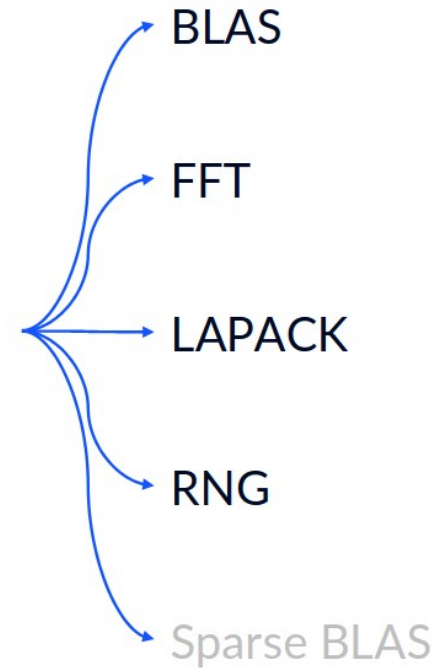
- ✓ Intel (Intel's MKL)
 - ✓ Nvidia (cuBLAS, cuRAND, cuFFT etc.)
 - ✓ AMD (rocBLAS, rocFFT etc.)
 - ✓ And SYCL-supported devices ("generic" SYCL code)
-
- SYCL compiler implementations (supported): DPC++ and AdaptiveCpp
 - (AdaptiveCpp support varies by backend but is being worked on)

Porting to SYCL using Math Libraries (oneMKL domains)

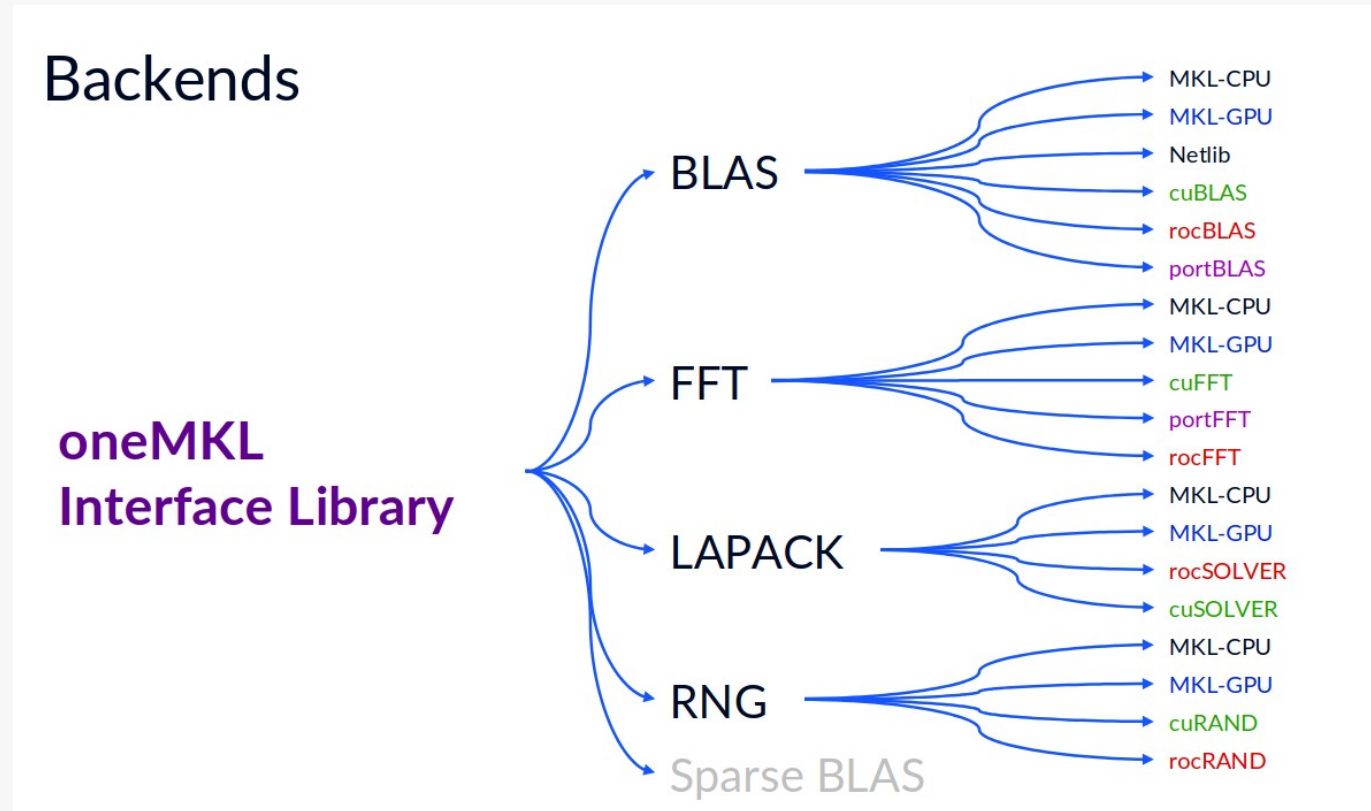
Domains

- BLAS
- LAPACK
- DFT
- RNG
- Sparse BLAS

**oneMKL
Interface Library**



Porting to SYCL using Math Libraries (oneMath backends)



*SparseBLAS is currently a Work-In-Progress

Support for cuSPARSE - <https://github.com/oneapi-src/oneMKL/pull/527>

Support for rocSPARSE - <https://github.com/oneapi-src/oneMKL/pull/544>

Porting to SYCL using Math Libraries (oneMKL Usage)

Runtime Dispatch	Static Dispatch
<pre>\$> clang++ -fsycl -I\$ONEMKLROOT/include app.cpp \$> clang++ -fsycl app.o -L\$ONEMKLROOT/lib -lonemkl</pre>	<pre>\$> clang++ -fsycl -I\$ONEMKLROOT/include app.cpp \$> clang++ -fsycl app.o -L\$ONEMKLROOT/lib -lonemkl_blas_mklcpu -lonemkl_blas_cublas</pre>
<pre>#include "oneapi/mkl.hpp" cpu_dev = sycl::device(sycl::cpu_selector()); gpu_dev = sycl::device(sycl::gpu_selector()); sycl::queue cpu_queue(cpu_dev); sycl::queue gpu_queue(gpu_dev); oneapi::mkl::blas::column_major::gemm(cpu_queue, transA, transB, m, ...); oneapi::mkl::blas::column_major::gemm(gpu_queue, transA, transB, m, ...);</pre>	<pre>#include "oneapi/mkl.hpp" cpu_dev = sycl::device(sycl::cpu_selector()); gpu_dev = sycl::device(sycl::gpu_selector()); sycl::queue cpu_queue(cpu_dev); sycl::queue gpu_queue(gpu_dev); oneapi::mkl::blas::column_major::gemm(oneapi::mkl::backend_selector<oneapi::mkl::backend::mklcpu> {cpu_queue}, transA, transB, m, ...); oneapi::mkl::blas::column_major::gemm(oneapi::mkl::backend_selector<oneapi::mkl::backend::cublas> {gpu_queue}, transA, transB, m, ...);</pre>
• oneMKL can build with support for multiple vendors at once • oneMKL can automatically dispatch to the correct backend library • Backends are lazily dlopened (a bit of overhead)	• (a) Uses templated backend selector API, (b) Links with required backend wrapper libraries • Avoids overhead of dispatch tables by linking directly against backend libraries

Porting to SYCL using Math Libraries (CMake Magic)

oneMKL is already installed, then find oneMKL Dependency via CMake for a project

```
find_package(oneMKL REQUIRED)

// Link via runtime dispatch
target_link_library(mytarget PRIVATE
MKL::onemkl)

// (or) Link against specific domain & backend
target_link_library(mytarget PRIVATE
    MKL::onemkl_<domain>_<backend>)
```

oneMKL is “not” available, Build and automatically link via FetchContent in CMake

```
include(FetchContent)
set(BUILD_FUNCTIONAL_TESTS OFF)
set(BUILD_EXAMPLES OFF)
set(ENABLE_<BACKEND_NAME>_BACKEND ON)
FetchContent_Declare(
    onemkl_interface_library
    GIT_REPOSITORY https://github.com/oneapi-src/oneMKL.git
    GIT_TAG develop
)
FetchContent_MakeAvailable(onemkl_interface_library)
target_link_libraries(myTarget PRIVATE onemkl)
// (or) for a specific backend
target_link_libraries(myTarget PRIVATE
onemkl_<domain>_<backend>)
```

Porting to SYCL using Math Libraries (Install oneMKL Interfaces)

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

```
cd oneMKL
```

```
mkdir -p build-cuda && cd build-cuda
```

```
cmake .. -DCMAKE_CXX_COMPILER=icpx/clang++ \  
  -DENABLE_CUBLAS_BACKEND=True \  
  -DENABLE_CUSOLVER_BACKEND=True \  
  -DENABLE_CURAND_BACKEND=True \  
  -DENABLE_MKLCPU_BACKEND=False \  
  -DENABLE_MKLGPU_BACKEND=False \  
  -DBUILD_FUNCTIONAL_TESTS=False -DBUILD_EXAMPLES=False \  
  -DCMAKE_INSTALL_PREFIX=<where-to-install>
```

```
make -j; make install
```

It's pretty easy (for instance oneMKL for Nvidia GPUs)

- ✓ Enable the backends you want (CUBLAS, CUSOLVER or CURAND) or (ROCBLAS, ROSOLVER or ROCRAND)
- ✓ Disable building functional-tests and examples

Don't forget to add the install-directory <where-to-install> to the system path

```
export PATH=<where-to-install>/bin:$PATH
```

```
export CPATH=<where-to-install>/include:$CPATH
```

```
export LD_LIBRARY_PATH=<where-to-install>/lib:$LD_LIBRARY_PATH
```

Porting to SYCL using Math Libraries (Hands-on)

Objective: Learn oneMKL GEMM API, how to compile and run

https://github.com/codeplaysoftware/syclacademy/tree/main/Code_Exercises/oneMath_gemm

Login to compute node on ALCF Polaris

```
module use /soft/modulefiles/  
module load oneapi/release
```

Source

```
git clone -b main --depth=1 https://github.com/codeplaysoftware/syclacademy.git  
cd syclacademy/Code_Exercises/oneMath_gemm
```

Compile

```
export MKLROOT=/soft/compilers/oneapi/release/onemath  
  
icpx -std=c++17 -O3 -fsycl -fsycl-targets=nvptx64-nvidia-cuda -Xsycl-target-backend --cuda-gpu-arch=sm_80 -I${MKLROOT}/include -L${MKLROOT}/lib -lonemkl ./solution_onemath_usm_gemm.cpp  
-o solution_onemath_usm_gemm.out
```

oneDPL: Accelerated C++ on Heterogeneous System

- High productivity and portable performance for heterogeneous computing – CPUs, GPUs, and FPGAs
- APIs based on standards and familiar extensions – C++ STL, SYCL, Boost.Compute
- Optimized C++ standard algorithms implemented on top of SYCL, OpenMP, oneTBB
- Interoperable with DPC++ and other oneAPI libraries
- Integrated with Intel® DPC++ Compatibility Tool to simplify migration of CUDA* applications using Thrust* API to DPC++ code

- ✓ C++17 is the minimal supported version of the C++ standard
- ✓ Header names start with oneapi/dpl:
`#include <oneapi/dpl/algorithm>`
- ✓ All functionality is provided in namespace `oneapi::dpl`
short alias: `namespace dpl = oneapi::dpl;`

oneDPL: 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)
```

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 );
```

Portability of oneMKL for Intel, Nvidia, AMD GPUs

Open-source

<https://github.com/oneapi-src/oneMKL>

```
#include <sycl/sycl.hpp>
#include <oneapi/mkl.hpp> // oneMKL header

int main() {
    sycl::queue q(sycl::property_list{sycl::property::queue::in_order{}});
    std::cout << "Device: " << q.get_device().get_info<sycl::info::device::name>() << std::endl
    << std::endl;

    // Allocate device-arrays
    double* A_dev          = sycl::malloc_device<double>(M*N, q);
    double* B_dev          = sycl::malloc_device<double>(N*P, q);
    double* C_dev_oneMKL = sycl::malloc_device<double>(M*P, q);

    // Copy inputs from host to device
    q.memcpy(A_dev, A, (M*N) * sizeof(double));
    q.memcpy(B_dev, B, (N*P) * sizeof(double));

    auto gemm_event = oneapi::mkl::blas::column_major::gemm(q, transB, transA,
        n, m, k,
        alpha,
        B_dev, ldB,
        A_dev, ldA,
        beta,
        C_dev_oneMKL, ldC);

    gemm_event.wait();

    .....
}
```



Standard SYCL and oneMKL headers



Set up SYCL device & print name



Allocate “device” memory for matrices



Copy inputs from host to device



Call to oneMKL GEMM API

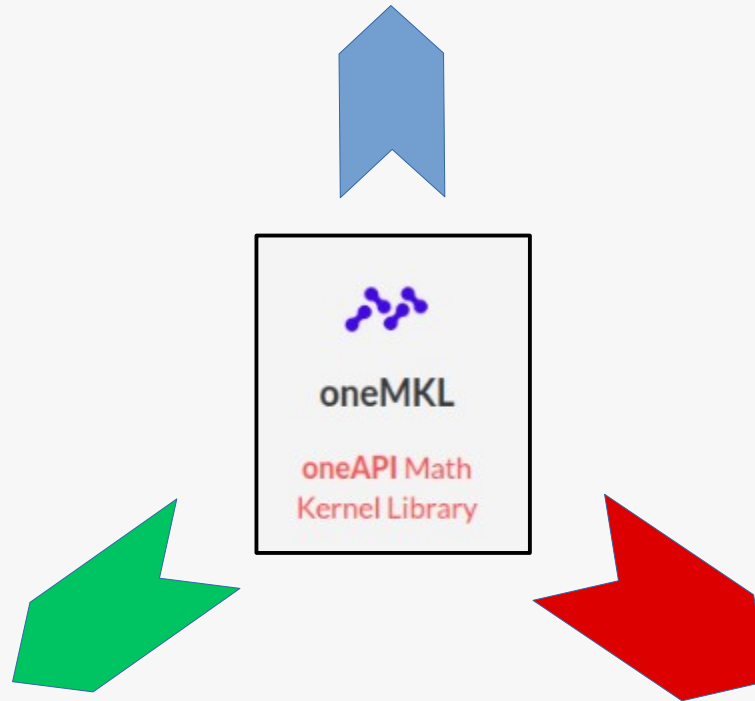


Synchronize for the results to finish

Portability of oneMKL for Intel, Nvidia, AMD GPUs

```
clang++ -std=c++17 -O3 -qmk1 -L$MKLR00T/lib -lonemkl sycl_onemkl_gemm.cpp -o  
sycl_onemkl_gemm.out
```

Compile for Intel PVC on ALCF Aurora



Compile for Nvidia A100 on ALCF Polaris

```
clang++ -std=c++17 -O3 -fsycl -fsycl-targets=nvptx64-  
nvidia-cuda -Xsycl-target-backend --cuda-gpu-arch=sm_80  
-L$MKLR00T/lib -lonemkl sycl_onemkl_gemm.cpp -o  
sycl_onemkl_gemm.out
```

Compile for AMD MI250X on OLCF Frontier

```
clang++ -std=c++17 -O3 -fsycl -Xsycl-target-  
backend=amdgc-n-amd-amdhsa --offload-arch=gfx90a -  
L$MKLR00T/lib -lonemkl sycl_onemkl_gemm.cpp -o  
sycl_onemkl_gemm.out
```

oneMKL for Nvidia A100 & AMD MI250X

```
abagusetty@x3004c0s31b1n0 ~ $ export CUBLAS_LOGINFO_DBG=1; export CUBLAS_LOGDEST_DBG=stdout
abagusetty@x3004c0s31b1n0 ~ $ ./sycl_onemkl_gemm_v1.out
Problem size: c(600,2400) = a(600,1200) * b(1200,2400)
Device: NVIDIA A100-SXM4-40GB
```

```
I! cuBLAS (v11.4) function cublasStatus_t cublasCreate_v2(cublasContext**) called:
i! handle: type=cublasHandle_t; val=POINTER (IN HEX:0x0x14a709ed7a38)
I! ....
```

```
I! cuBLAS (v11.4) function cublasStatus_t cublasDgemm_v2(cublasHandle_t, cublasOperation_t, cublasOperation_t, int, int, int, const double*, const double*, int, const double*, int, const double*, double*, int) called:
```

```
I! ....
```

```
oneMKL: Test PASSED
```

```
abagusetty@login04 /lustre/orion/gen243/scratch/abagusetty $ ROCBLAS_LAYER=4 ./sycl_onemkl_gemm.out
Problem size: c(600,2400) = a(600,1200) * b(1200,2400)
Device: gfx90a:sramecc+:xnack-
```

```
oneMKL: Test PASSED
```

```
- { rocblas_function: "rocblas_dgemm", atomics_mode: atomics_allowed, transA: 'N', transB: 'N', M: 2400, N: 600, K: 1200,
alpha: 1.0, lda: 2400, ldb: 1200, beta: 0.0, ldc: 2400, call_count: 1 }
```

- Performance: oneMKL uses cuBLAS & rocBLAS APIs underneath ensuring performance
- Standard rocBLAS/cuBLAS logging tools also work with oneMKL APIs

Portability of oneDPL for Intel, Nvidia, AMD GPUs

Open-source <https://github.com/oneapi-src/oneDPL>

```
#include <oneapi/dpl/execution>
#include <oneapi/dpl/algorithm>
#include <sycl/sycl.hpp>
#include <cassert>

int main()
{
    sycl::queue q(sycl::gpu_selector_v);
    std::cout << "Device: " <<
q.get_device().get_info<sycl::info::device::name>() << std::endl <<
std::endl;
    int data_host[4] = {3, 7, 2, 5};
    int* data = sycl::malloc_device<int>(4, q); // inputs
    int* data_y = sycl::malloc_device<int>(4, q); // outputs
    q.memcpy(data, data_host, 4*sizeof(int)).wait();

    // subtracts 10 to all values in data
    std::transform( oneapi::dpl::execution::make_device_policy(q),
                    data, data + 4, /* first1, last1 */
                    data_y, /* output */
                    [](const auto& x){return x-10;}); /* unary op */

    // Results:
    int result_ref[4] = {-7, -3, -8, -5};
    int* result_host = new int[4]; // copy back results to host for
comparing
    q.memcpy(result_host, data_y, 4*sizeof(int));
    for( int i=0; i<4; i++) {
        assert( result_ref[i] == result_host[i] );
    }

    std::cout << "oneDPL: Test PASSED\n";
    return 0;
}
```



Standard SYCL and oneDPL headers



Set up SYCL device & inputs



Calling std::transform from ISO C++ for GPU

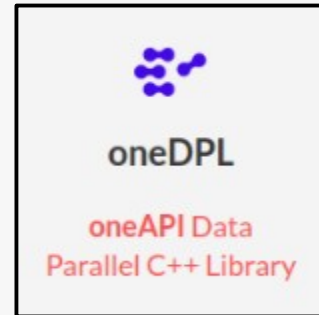


Synchronize for the results to verify

Portability of oneDPL for Intel, Nvidia, AMD GPUs

```
clang++ -std=c++17 -O3 sycl_onedpl.cpp -o sycl_onedpl.out
```

Compile for Intel PVC on ALCF Aurora



Compile for Nvidia A100 on ALCF Polaris

```
clang++ -std=c++17 -O3 -fsycl -fsycl-targets=nvptx64-  
nvidia-cuda -Xsycl-target-backend --cuda-gpu-arch=sm_80  
sycl_onedpl.cpp -o sycl_onedpl.out
```

Compile for AMD MI250X on OLCF Frontier

```
clang++ -std=c++17 -O3 -fsycl -Xsycl-target-  
backend=amdgc-n-amd-amdhsa --offload-arch=gfx90a  
sycl_onedpl.cpp -o sycl_onedpl.out
```

oneDPL for Nvidia A100 & AMD MI250X

```
abagusetty@x3004c0s31b1n0 ~ $ ./sycl_onedpl.out  
Device: NVIDIA A100-SXM4-40GB  
oneDPL: Test PASSED
```

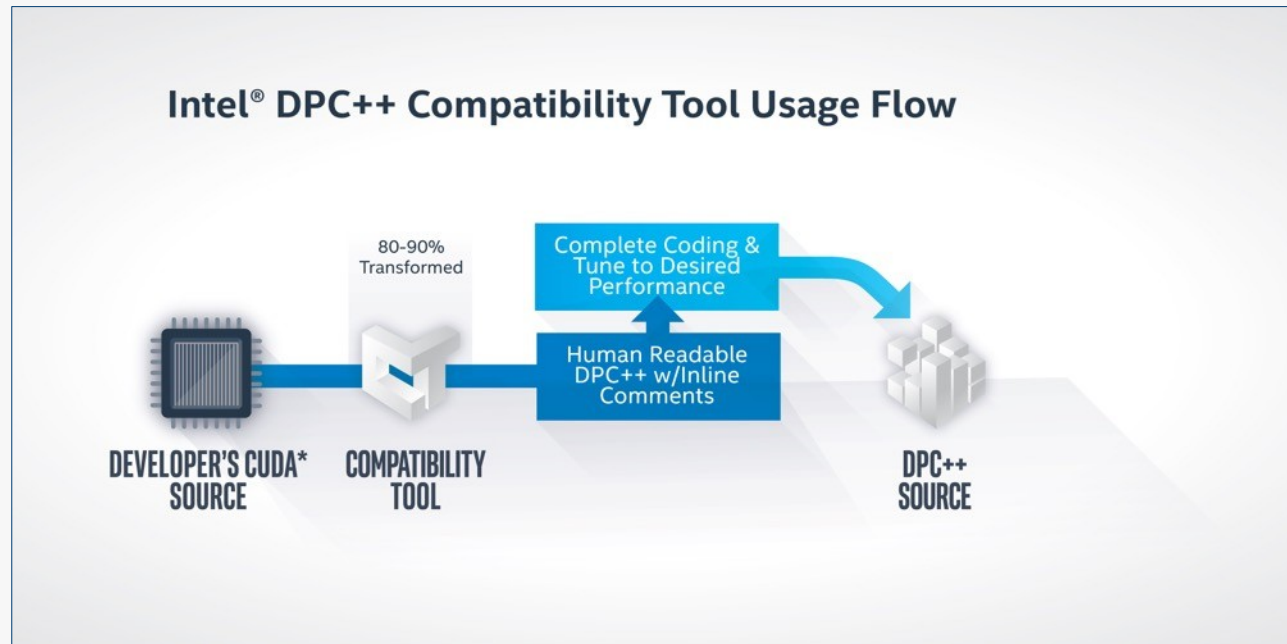
```
abagusetty@login04 /lustre/orion/gen243/scratch/abagusetty $ ./sycl_onedpl.out  
Device: gfx90a:sramecc+:xnack-  
oneDPL: Test PASSED
```

- Performance: oneDPL depends on the underlying portable SYCL kernels for Nvidia & AMD hardware

How to port existing CUDA to SYCL ?

Intel® DPC++ Compatibility Tool

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

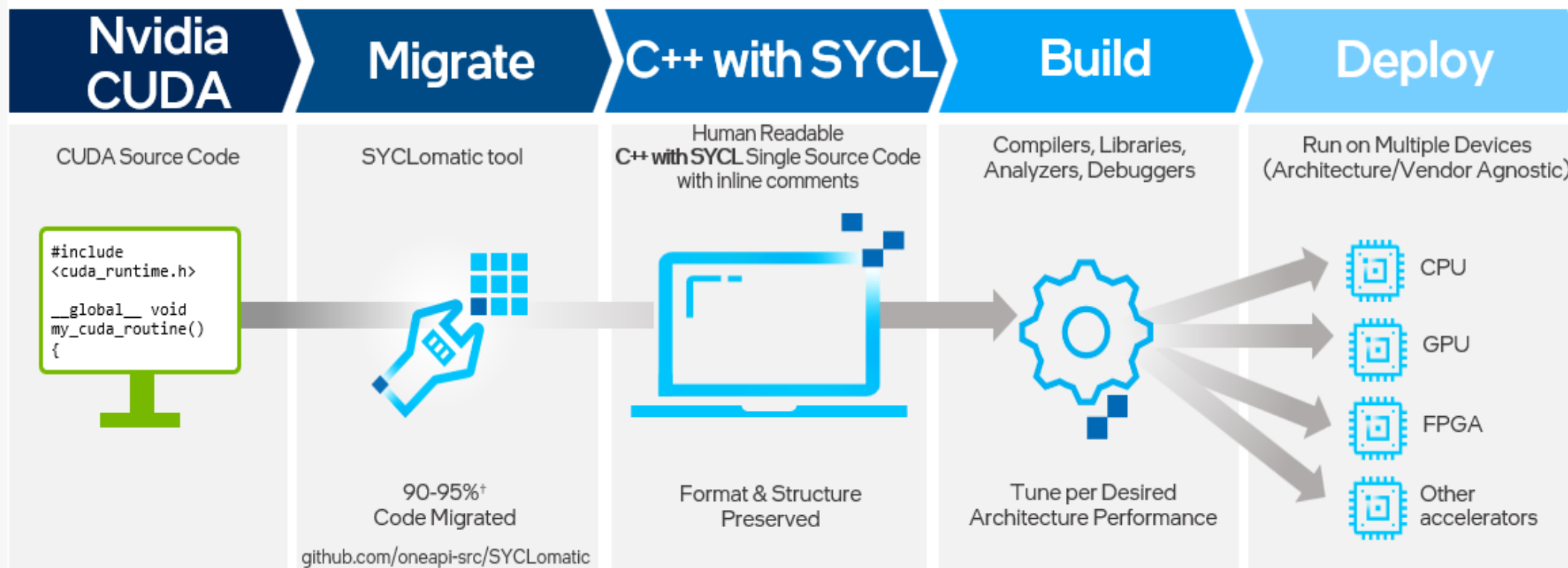


- Assists developers migrating code written in CUDA* to DPC++
- Target is to migrate up to 80-90% of code automatically
- Inline comments are provided to help developer complete code

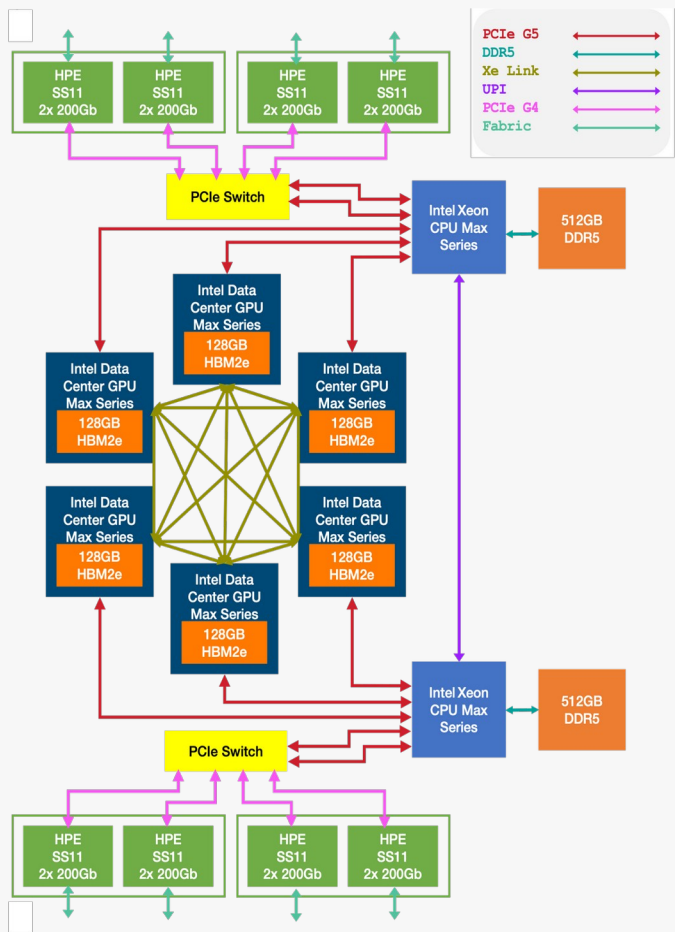
[Codeplay: CUDA to SYCL for Beginners](#)
[Intel Compatibility Tool](#)

Porting CUDA projects to SYCL

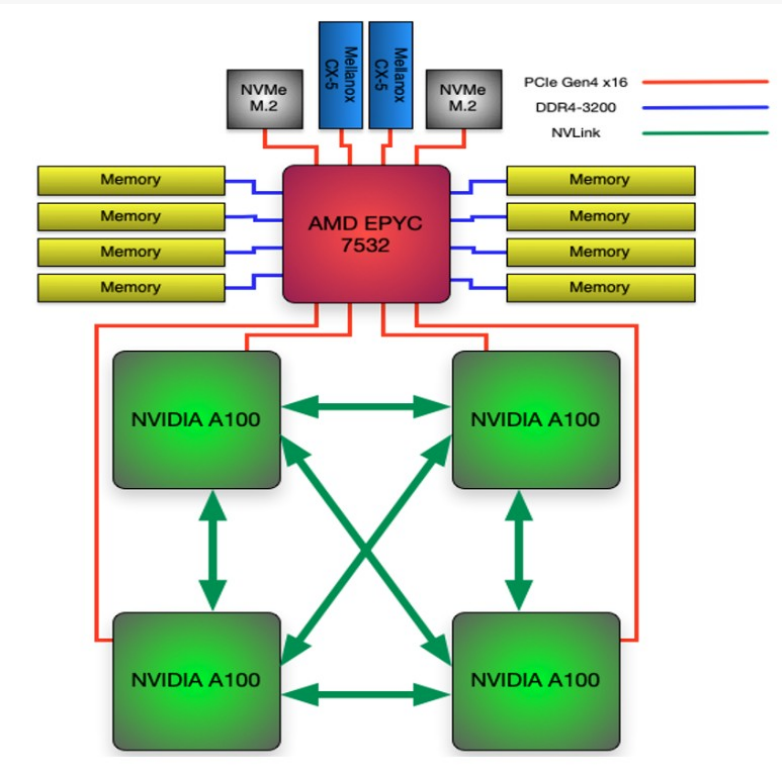
SYCLomatic: A New CUDA*-to-SYCL* Code Migration Tool
(previously referred to as DPCT, DPC++ Compatibility Tool)



Multi-GPU Programming



ALCF Aurora – 6 PVC (12 tiles/Xe-Stacks)



ALCF Polaris – 4 A100

Multi-GPU Programming

- SYCL queue provides the handle to submit work on a given device (CPU/GPU/FPGA, etc)

```
sycl::queue q(sycl::gpu_selector_v);
```

- Single GPU obtained via:
 - `sycl::gpu_selector_v` provides access to the first GPU available
 - Exposing visibility of devices can be altered via setting the environment variables
 - `CUDA_VISIBLE_DEVICES`, `ROCR_VISIBLE_DEVICES`, `ZE_AFFINITY_MASK`

```
abagusetty@x3011c0s31b0n0 ~ $ CUDA_VISIBLE_DEVICE=3 sycl-ls --verbose
INFO: Output filtered by ONEAPI_DEVICE_SELECTOR environment variable, which is set to cuda:gpu.
To see device ids, use the --ignore-device-selectors CLI option.

[cuda:gpu] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-40GB 8.0 [CUDA 12.2]
```

Hands-on:

[SYCL-single-gpu](#)

Multi-GPU Programming – ALCF Polaris

- (Traditional) MPI + X: **Single-GPU per MPI** - Multiple GPUs can be exposed via binding each GPU to a MPI-processes, each MPI-process acts as if there is 1 GPU
- **Multi-GPU per MPI** All GPUs are visible to any given MPI-process

```
auto gpus = sycl::platform(sycl::gpu_selector_v).get_devices();  
...  
sycl::queue q0(gpus[0]);  
sycl::queue q1(gpus[1]);  
...
```

```
abagusetty@x3011c0s31b0n0 ~ $ sycl-ls --verbose  
INFO: Output filtered by ONEAPI_DEVICE_SELECTOR environment variable, which is set to cuda:gpu.  
To see device ids, use the --ignore-device-selectors CLI option.
```

```
[cuda:gpu] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-40GB 8.0 [CUDA 12.2]  
[cuda:gpu] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-40GB 8.0 [CUDA 12.2]  
[cuda:gpu] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-40GB 8.0 [CUDA 12.2]  
[cuda:gpu] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-40GB 8.0 [CUDA 12.2]
```

```
abagusetty@x3011c0s31b0n0 ~ $ nvidia-smi  
Thu Aug 1 16:18:57 2024
```

NVIDIA-SMI 535.154.05 Driver Version: 535.154.05 CUDA Version: 12.2									
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC				
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.			
						MIG M.			
0	NVIDIA A100-SXM4-40GB	On	00000000:07:00.0	Off	0%	Default			
N/A	30C	P0	52W / 400W	0MiB / 40960MiB		Disabled			
1	NVIDIA A100-SXM4-40GB	On	00000000:46:00.0	Off	0%	Default			
N/A	30C	P0	54W / 400W	0MiB / 40960MiB		Disabled			
2	NVIDIA A100-SXM4-40GB	On	00000000:85:00.0	Off	0%	Default			
N/A	29C	P0	52W / 400W	0MiB / 40960MiB		Disabled			
3	NVIDIA A100-SXM4-40GB	On	00000000:C7:00.0	Off	0%	Default			
N/A	32C	P0	55W / 400W	0MiB / 40960MiB		Disabled			

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage	
	ID	ID					
No running processes found							

Have a Productive Workshop!