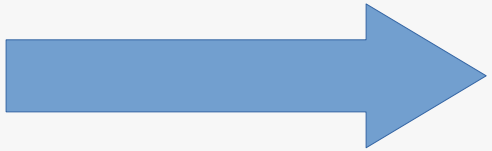
Porting from CUDA to SYCL



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.

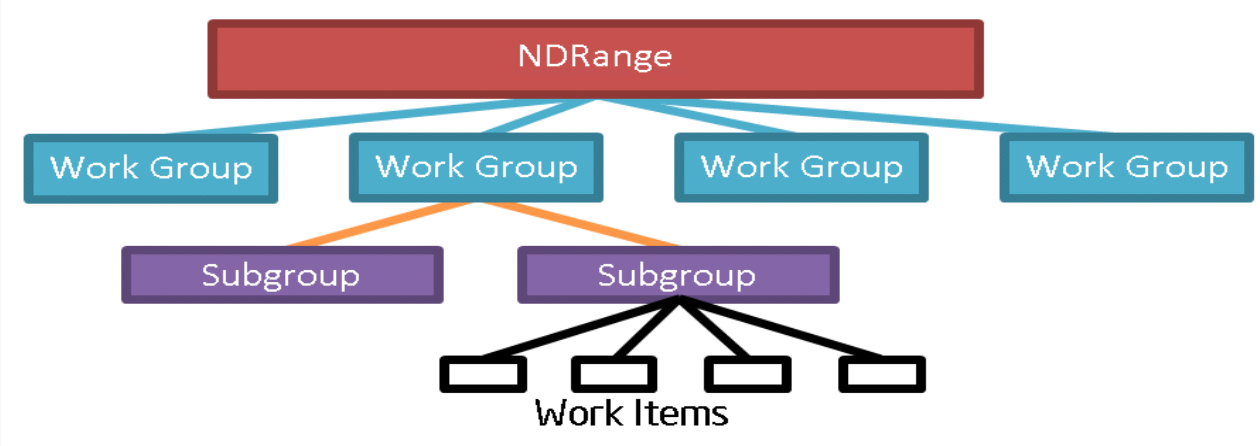
An **nd range** has three components

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

Argonne NATIONAL LABORATORY

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.

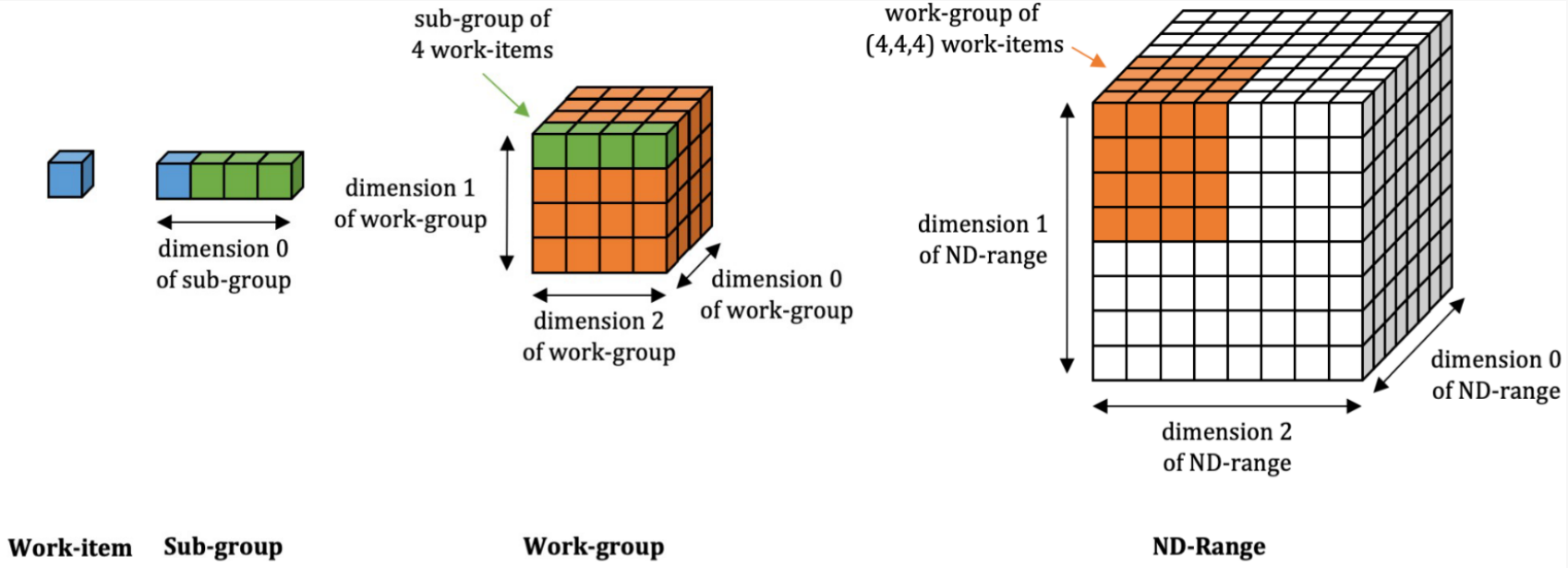
Argonne NATIONAL LABORATORY

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

Argonne NATIONAL LABORATORY

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

Argonne NATIONAL LABORATORY

Vector Addition: SYCL Buffer memory model

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

void main() {
 using namespace sycl;
 float A[1024], B[1024], C[1024];
 {
  buffer<float, 1> bufA { A, range<1> {1024} };
  buffer<float, 1> bufB { B, range<1> {1024} };
  buffer<float, 1> bufC { C, range<1> {1024} };

  queue myQueue;
  myQueue.submit([&](handler& cgh) {
   auto accA = bufA.get_access<access::read>(cgh);
   auto accB = bufB.get_access<access::read>(cgh);
   auto accC = bufC.get_access<access::write>(cgh);

   cgh.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
    accC[i] = accA[i] + accB[i];
   });
  }).wait();
 }

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

Host Code

Device Code

Host Code

Create SYCL buffers using host pointers.

Create a queue to submit work to a GPU

Read/write accessors create dependencies if other kernels or host access buffers.

Vector addition device kernel

Argonne NATIONAL LABORATORY

Vector Addition: SYCL USM memory model

```
#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));

 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));

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

Host Code

Device Code

Host Code

Step 1: Create SYCL queue to create GPU

Step 2: Allocate device memory

Step 3 (H2D): copy inputs "A" & "B" to GPU

Step 4 (Compute): Run the kernel on device

Step 5 (D2H): Copy result "devC" back to host

Argonne NATIONAL LABORATORY

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];
   });
```

Host Code

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

 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"

Argonne NATIONAL LABORATORY

Vector Addition: SYCL USM memory model

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

Host Code

Device Code

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(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));

 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));

 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"

Argonne NATIONAL LABORATORY

Case Study 1: Equivalents for Nvidia Thrust Library ?

- Thrust is a C++ template library for CUDA based on the Standard Template Library (STL)
- A rich collection of data parallel primitives such as scan, sort, and reduce, etc.
- Thrust can be utilized in rapid prototyping of CUDA applications where robustness and absolute performance are crucial.

```
thrust::device_vector<int> x = h_vec;
// sort data on the device (This breaks the compile)
thrust::sort(x.begin(), x.end());
```

- oneDPL defines a subset of the C++ standard library which you can use with buffers and data parallel kernels.
- oneDPL extends Parallel STL with execution policies and companion APIs for running algorithms on oneAPI devices
- Extensions. An additional set of library classes and functions that are known to be useful in practice but are not (yet) included into C++ or SYCL specifications.

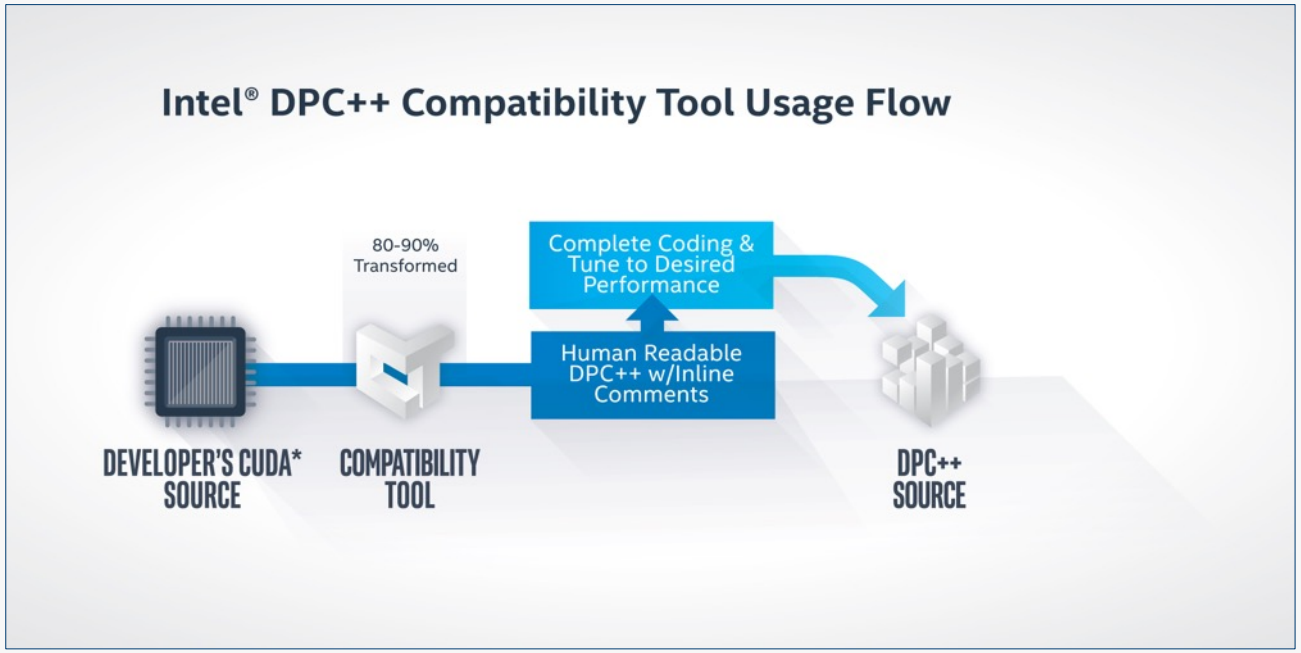
```
// sort x!
auto policy = dpstd::execution::make_device_policy<class oneapiSort>( q );
std::sort(policy, x, x+n_points);
q.wait();
```

- Note: oneDPL library is open-source and in-development. Not all features are supported.
- https://github.com/oneapi-src/oneDPL

Case Study 2: 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

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

Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software.

Case Study 2: Continued… AutoDock-GPU

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: What are my opions for cublas ?

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	In-works (cuFFT)	In-works (rocFFT)	In-works (onemkl::dft)

Argonne NATIONAL LABORATORY

Useful resources

https://github.com/argonne-lcf/sycltrain/tree/master/9_sycl_of_hell

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