Argonne NATIONAL LABORATORY

SYCL/DPC++ – An Introduction

Thomas Applencourt - apl@anl.gov

Argonne Leadership Computing Facility
Argonne National Laboratory
9700 S. Cass Ave
Argonne, IL 60349

Table of contents

Goal of this talk

1. Give you a feel of SYCL/DPCPP (15 min)
2. Tease you enough so you want to play with SYCL during the Hands-on (30 min)
3. Answer any Question (easy or hard[1]) that you can have.

[1] But not too hard, like ”What does SYCL mean? Is this an acronym?”

Introduction

What programming model to use to target GPU?

- Parallel STL[2]
- OpenMP (pragma based)
- CUDA[3] / HIP[4] / OpenCL[5] (low level)
- Kokkos, raja, OCCA (high level, abstraction layer, academic project)
- SYCL (high level) / DPCPP[6]

---

[2] We have many SYCL implementation backend for that. See *https://spec.oneapi.com/versions/0.5.0/oneAPI/Elements/onedpl/onedpl_root.html*

[3] Compute Unified Device Architecture

[4] Heterogeneous-Compute Interface

[5] Open Computing Language

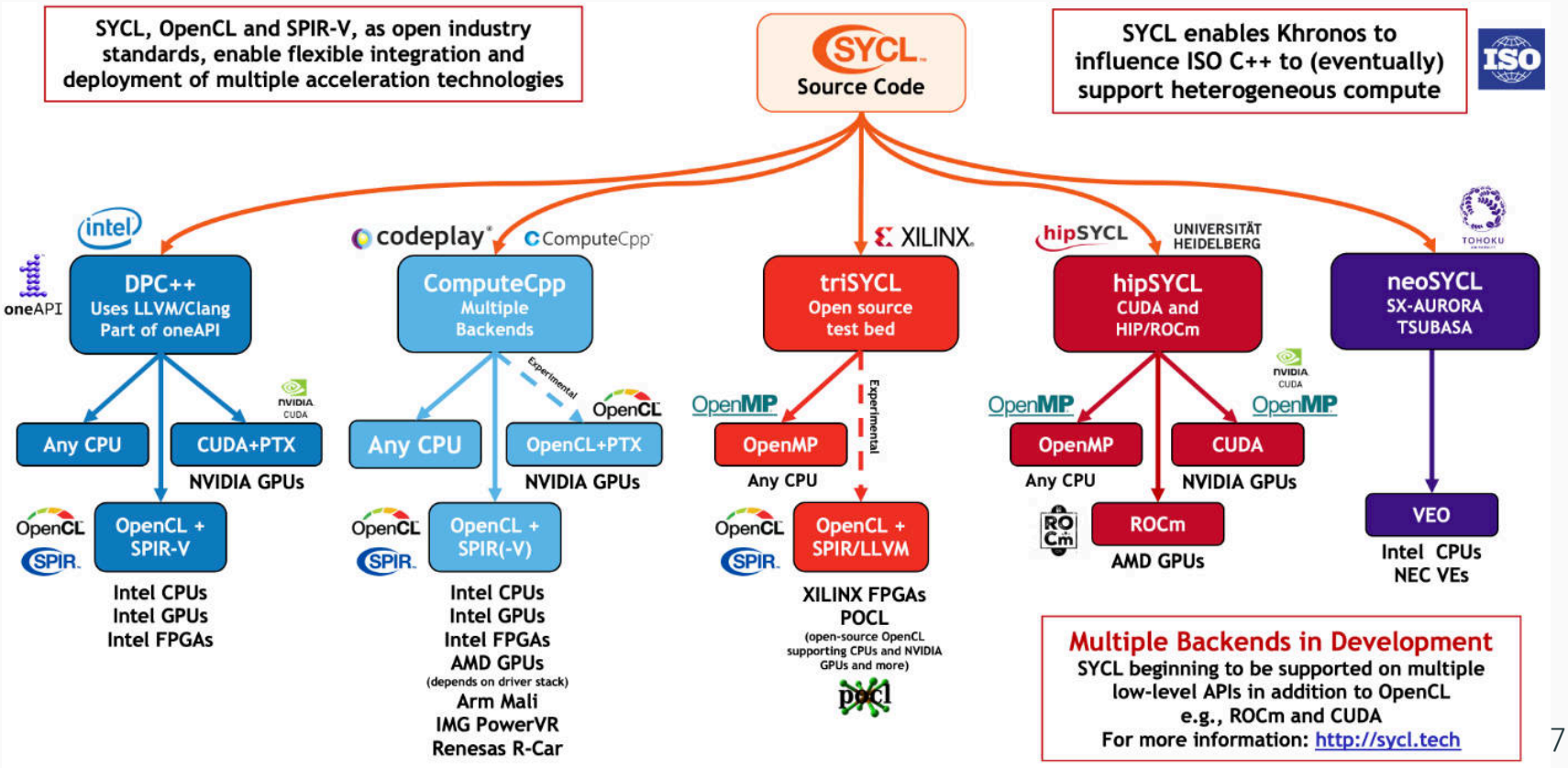
[6] Data Parallel C++

What is SYCL™?

1. Target C++ programmers (template, lambda)
   - No language extension
   - No pragmas
   - No attribute
2. Borrow lot of concept from battle tested OpenCL (platform, device, work-group, range)
3. Single Source (two compilation pass)
4. Implicit or Explicit data-transfer
5. SYCL is a Specification developed by the Khronos Group (OpenCL, SPIR, Vulkan, OpenGL)

- The current stable SYCL specification is SYCL2020

SYCL Implementation



7

[7] Credit: Khronos groups (*https://www.khronos.org/sycl/*)

What is DPCPP?

- Intel implementation of SYCL[8]
- The name of the SYCL-aware Intel compiler[9] who is packaged with Intel OneAPI SDK.
- Intel SYCL compiler is open source and based on LLVM *https://github.com/intel/llvm/*. This is what is installed on ThetaGPU, hence the compiler will be named *clang++*[10].

[8] Obvious from the name isn't it?
[9] So you don't need to pass *-fsycl*
[10] I know marketing is confusing...

DPCPP (and the associated ecosystem)

DPCPP a high potential SYCL implementation

DPCPP implement the SYCL Standard + extension[11]

- Magic introspection function
- Explicit SIMD

Many of DPCPP extension (Unnamed Lambda, Unified Shared Memory) are now merged in the new SYCL2020 standard!

[11] *https://github.com/intel/llvm/tree/sycl/sycl/doc/extensions*

Interoperability

- SYCL2020: with Native programming model (OpenCL, Cuda, ...)
  - CUstream <-> sycl::queue
  - etc
- DPCPP: With OpenMP[12]

[12] *https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/software-development-process/composability/c-c-openmp-and-dpc-composability.html*

DPCT: CUDA to DPCPP translator[13]

1. This is not a CUDA to DPCPP source to source compiler.
2. ”Tool Assisted Porting”

[13] *https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/software-development-process/migrating-code-to-dpc/migrating-from-cuda-to-dpc.html*

oneMKL[14]

> *oneMKL interfaces are an open-source implementation of the oneMKL Data Parallel C++ (DPC++) interface according to the oneMKL specification. It works with multiple devices (back-ends) using device-specific libraries underneath.*

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

[14] *https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html*

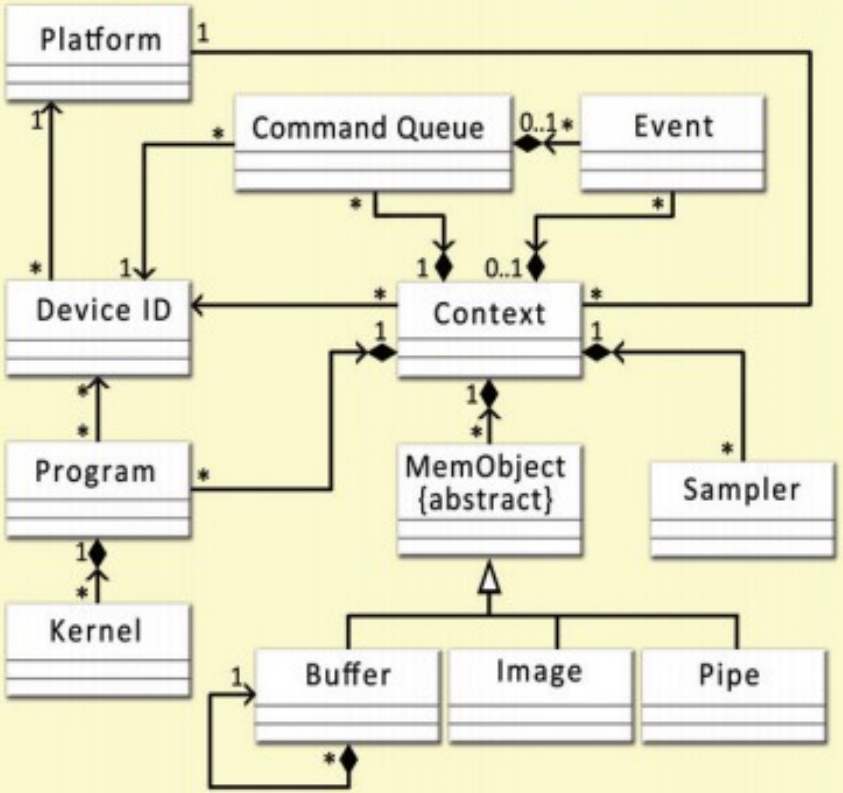
Theory

A picture is worth a thousand words[15]

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language[1] (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.

Annotations	
Relationships	
abstract classes	{abstract}
aggregations	♦
inheritance	△
relationship navigability	^
Cardinality	
many	*
one and only one	1
optionally one	0..1
one or more	1..*



[1] Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).

[15] and this is a UML diagram so maybe more!

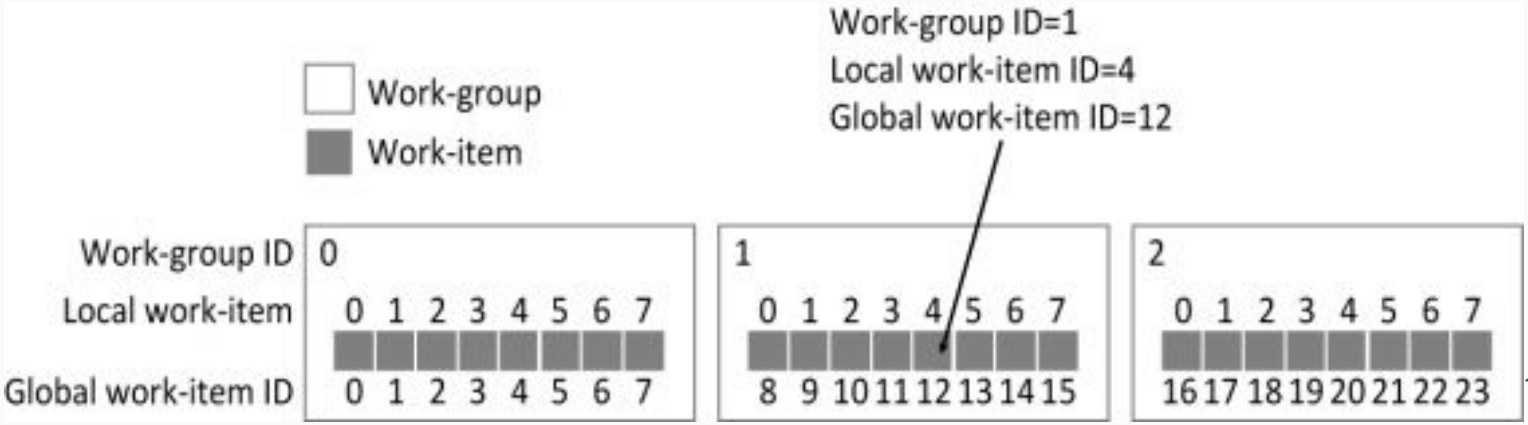
Memory management: SYCL innovation

1. Buffers encapsulate your data
2. Accessors describe how you access those data
3. Buffer destruction will cause synchronization

- Or you can also use Unified shared memory

Submiting Loop

- A Kernel is invoked once for each work item [16]
- local work size Work items are grouped into a work group [17]
- The total number of all work items is specified by the global work size



[18]

[16] similar to *MPI_rank*
[17] similar to *pragma omp simdlen/safelen*
[18] Credit The OpenCL Programming Book by Fixstars

Example

Anatomy of a SYCL application [3.2]

Below is an example of a typical SYCL application which schedules a job to run in parallel on any OpenCL accelerator. USM versions of this example are shown on page page 15 of this reference guide.

```
#include <iostream>
#include <sycl/sycl.hpp>
using namespace sycl; // (optional) avoids need for "sycl::" before SYCL names
int main() {
    int data[1024]; // Allocates data to be worked on
    queue myQueue; // Create default queue to enqueue work

    // By wrapping all the SYCL work in a {} block, we ensure all
    // SYCL tasks must complete before exiting the block,
    // because the destructor of resultBuf will wait.
    {
        // Wrap our data variable in a buffer.
        buffer<int, 1> resultBuf { data, range<1> { 1024 } };

        // Create a command group to issue commands to the queue.
        myQueue.submit([&](handler & cgh) {
            // Request access to the buffer without initialization
            accessor writeResult { resultBuf, cgh, write_only, no_init };

            // Enqueue a parallel_for task with 1024 work-items.
            cgh.parallel_for(1024, [=](auto idx) {
                // Initialize each buffer element with its own rank number starting at 0
                writeResult[idx] = idx;
            }); // End of the kernel function
        });     // End of the queue commands
    }           // End of scope, so wait for the queued work to complete

    // Print result
    for (int i = 0; i < 1024; i++) {
        std::cout <<"data["<< i << "] = " << data[i] << std::endl;
    return 0;
}
```

Header file
SYCL programs must include the <sycl/sycl.hpp> header file to provide all of the SYCL features used in this example.

Namespace
SYCL names are defined in the sycl namespace.

Queue
This line implicitly selects the best underlying device to execute on. See queue class functions [4.6.5] on page 2 of this reference guide.

Buffer
All data required in a kernel must be inside a buffer or image or else USM is used. See buffer class functions [4.7.2] on page 3 of this reference guide.

Accessor
See accessor class functions in [4.7.6.x] on pages 4 and 5 of this reference guide.

Handler
See handler class functions [4.9.4] on page 9 of this reference guide.

Scopes
The ***kernel scope*** specifies a single kernel function compiled by a device compiler and executed on a device.

The ***command group scope*** specifies a unit of work which is comprised of a kernel function and accessors.

The ***application scope*** specifies all other code outside of a command group scope.

Conclusion

Conclusion

1. For better or worth, SYCL is C++
2. Many vendors (Intel, Nvidia, AMD) and hardware (CPU, GPU, FPGA) supported
3. Implicit data-movement by default (Buffer / Accessors concepts)

Lot of goods resources online

SYCL 2020 Spec

1. *https://www.khronos.org/files/sycl/sycl-2020-reference-guide.pdf*
2. *https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf*

Examples

1. *https://github.com/alcf-perfengr/sycltrain*
2. *https://github.com/codeplaysoftware/computecpp-sdk/tree/master/samples*
3. *https://github.com/jeffhammond/dpcpp-tutorial*

Documentations (online and books)

1. *https://sycl.tech/*
2. Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL (ISBN 978-1-4842-5574-2)

Q&A

Thank you! Do you have any questions?

Hands-on

```
# Assuming you are in theta
git clone https://github.com/alcf-perfengr/sycltrain
# Or
# git clone https://github.com/argonne-lcf/CompPerfWorkshop-2021

# Then on Theta GPU Compute node
module use /soft/thetagpu/compilers/dpcpp/modulefiles
module load dpcpp

cd sycltrain
# or
# cd CompPerfWorkshop-2021/13_sycl-oneAPI/

cd sycl_train/9_sycl_of_hell
make
```