SYCL – Introduction and Hands-on

Thomas Applencourt + people on next slide - apl@anl.gov

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

Book Keeping

Get the example and the presentation:

```
git clone https://github.com/alcf-perfengr/sycltrain
cd sycltrain/presentation/2020_07_30_ATPESC
```

People who are here to help:

- Collen Bertoni - bertoni@anl.gov
- Brian Homerding - bhomerding@anl.gov
- Nevin ”:)” Liber -nliber@anl.gov
- Ben Odom - benjamin.j.odom@intel.com
- Dan Petre - dan.petre@intel.com)

Table of contents

1. Introduction

2. Theory

3. Hands-on

4. Conclusion

Introduction

What programming model to use to target GPU?

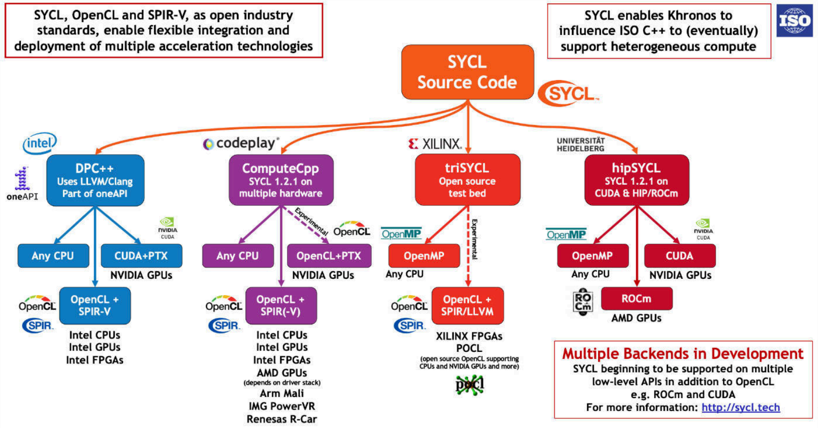
- OpenMP (pragma based)
- Cuda (proprietary)
- Hip (low level)
- OpenCL (low level)
- Kokkos, RAJA, OCCA (high level, abstraction layer, academic projects)

What is SYCL™?[1]

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

[1] SYCL Doesn't mean "Someone You Couldn't Love". Sadly.

SYCL Implementations



SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

SYCL Source Code

intel — DPC++
Uses LLVM/Clang
Part of oneAPI

- Any CPU
- CUDA+PTX — NVIDIA GPUs
- OpenCL + SPIR-V — Intel CPUs, Intel GPUs, Intel FPGAs

codeplay — ComputeCpp
SYCL 1.2.1 on multiple hardware

- Any CPU
- Experimental: OpenCL+PTX — NVIDIA GPUs
- OpenCL + SPIR(-V) — Intel CPUs, Intel GPUs, Intel FPGAs, AMD GPUs (depends on driver stack), Arm Mali, IMG PowerVR, Renesas R-Car

XILINX — triSYCL
Open source test bed

- OpenMP — Any CPU
- Experimental: OpenCL + SPIR/LLVM — XILINX FPGAs, POCL (open source OpenCL supporting CPUs and NVIDIA GPUs and more)

UNIVERSITÄT HEIDELBERG — hipSYCL
SYCL 1.2.1 on CUDA & HIP/ROCm

- OpenMP — Any CPU
- CUDA — NVIDIA GPUs
- ROCm — AMD GPUs

Multiple Backends in Development
SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL
e.g. ROCm and CUDA
For more information: http://sycl.tech [2]

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

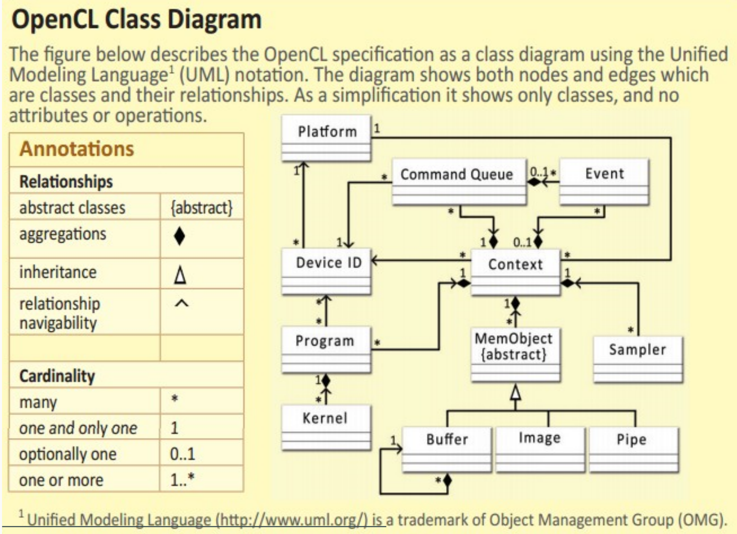
Goal of this presentation

1. Give you a feel of SYCL
2. Go through code examples (and make you do some homework)
3. Teach you enough so that can search for the rest if you interested
4. Question are welcomed! [3]

[3] Please just talk, or use slack

Theory

A picture is worth a thousand words[4]



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	Δ
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).

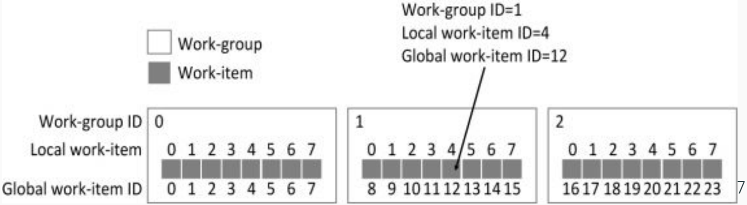
[4] and this is a UML diagram so maybe more!

Memory management: SYCL innovation

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

Implicit Loop

- A Kernel is invoked once for each work item [5]
- local work size Work items are grouped into a work group [6]
- The total number of all work items is specified by the global work size



[7]

[5] similar to *`MPI_rank`*

[6] similar to *`pragma omp simdlen/safelen`*

[7] Credit The OpenCL Programming Book by Fixstars

Implicit Loop: Example!

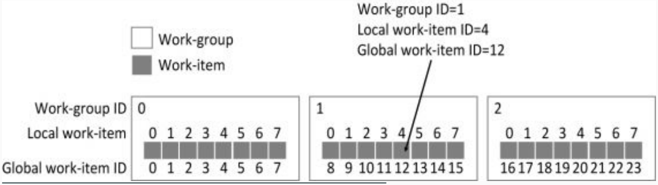
```
global_work_size = 24 ; local_work_size = 8
```

SYCL / Opencl / CUDA

```
parallel_for<global_work_size,local_work_size>(mykernel);
```

OpenMP[8]

```
# wG = work_group ; wC = work_item
for (wG_id=0; wG_id++; wG_id < (global_work_size / local_work_size)
    for (local_wI_id=0; local_wI_id++; local_wI_id < local_work_size)
        global_wI_id = local_wI_id + wG_id*local_wG_size
```



Work-group ID=1
Local work-item ID=4
Global work-item ID=12

[8] Using chunking / tilling / vectorization technique

Hands-on

Where do I run (cluster)?

Two Cluster are available for you [9]:

1. *`cooley.alcf.anl.gov`*. Nvidia GPU (Tesla K80)
2. *`devcloud.intel.com`*. Intel iGPU (Intel Iris ”Gen9”)
   - *`https://devcloud.intel.com/datacenter/learn/connect-with-ssh-linux-macos/`*[11]

[9] If you feel adventurous[10], install a SYCL compiler in your machine:

- *`https://software.intel.com/content/www/us/en/develop/articles/oneapi-repo-instructions.html`*
- or *`https://github.com/alcf-perfengr/sycltrain/blob/master/.travis.yml`*

[11] And if you are using windows you shouldn’t!

What do I run?

1. After logged to you the cluster, get the example and the presentation:

   ```
   git clone https://github.com/alcf-perfengr/sycltrain
   cd sycltrain/presentation/2020_07_30_ATPESC
   ```

2. Get a node

   ```
   ./fetch_a_node.devcloud.sh
   # ./fetch_a_node.cooley.sh
   ```

3. Source the correct environment

   ```
   cd sycltrain/presentation/2020_07_30_ATPESC
   source env.devcloud.rc
   #source env.cooley.rc
   ```

4. Compile and run examples

   ```
   cd 9_sycl_of_hell
   make run_0_tiny_sycl_info
   ```

Example and Exercise

1. Examples are available in the `9_sycl_of_hell` folder
2. Exercises are available in the `exercise` folder
   - 2.1 We will do the One Atom together [12]
   - 2.2 Harder problem will be covered at the end if we have time[13]

[12] `!="watching me"`

[13] But you are encouraged to do them by yourself... And send me the solution!

Lets go!

1. Examples are available in the `9_sycl_of_hell` folder
2. Exercises are available in the `exercise` folder
    2.1 We will do the One Atom together [14]
    2.2 Harder problem will be covered at the end if we have time[15]

[14] `!="watching me"`

[15] But you are encouraged to do them by yourself, and send me the solution!

Conclusion

Conclusion

1. 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

Spec

1. *https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf*
2. *https://www.khronos.org/files/sycl/sycl-121-reference-card.pdf*

Examples

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

Documentations

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

Q&A

Thanks you! Do you have any questions?