SYCL – A gentle Introduction

Thomas Applencourt - apl@anl.gov, Abhishek Bagusetty

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

1. Introduction
2. DPCPP ecosystem
3. Theory
   - Context And Queue
   - Unified Shared Memory
4. Kernel Submission
5. Buffers
6. Interopt
7. Show me number!
8. Conclusion
Introduction
What programming model to target Accelerator?

- CUDA¹ / HIP² / OpenCL³
- OpenMP (pragma based)
- Kokkos, raja, OCCA (high level, abstraction layer, academic project)
- SYCL (high level) / DPCPP⁴
- Parallel STL⁵

¹Compute Unified Device Architecture
²Heterogeneous-Compute Interface
³Open Computing Language
⁴Data Parallel C++
⁵SYCL implementation exist [https://github.com/oneapi-src/oneDPL](https://github.com/oneapi-src/oneDPL)
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)

6. Nice interoperability with other programming model (OpenMP, CUDA, Hip, OpenCL)
   - The current stable SYCL specification is SYCL2020
SYCL Implementation

Credit: Khronos groups (https://www.khronos.org/sycl/)
What is DPCPP?

• Intel implementation of SYCL
• The name of the SYCL-aware Intel compiler\(^7\) 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++\(^8\).

\(^7\)So you don’t need to pass -fsycl
\(^8\)I know marketing is confusing...
How to install SYCL: Example with Intel implementation

- Intel implementation work with Intel and NVidia Hardware
  1. Install from source https://github.com/intel/llvm/
  2. Use apt-get
  3. Download OneAPI pre-installed binary
  4. Ask your sys-admin to install it for you :)

Argonne National Laboratory
DPCPP ecosystem
1. A CUDA to SYCL source to source compiler
2. Used by a few Apps with some of success

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

- Some OpenMP Apps are using oneMKL as a "portability layer" for BLAS/LAPCK
- Sadly not yet installed on ThetaGPU

The Intel® oneAPI DPC++ Library is a companion to the Intel® oneAPI DPC++/C++ Compiler and provides an alternative for C++ developers who create heterogeneous applications and solutions. Its APIs are based on familiar standards—C++ STL, Parallel STL (PSTL), Boost.Compute, and SYCL*—to maximize productivity and performance across CPUs, GPUs, and FPGAs.

---

Theory
A picture is worth a thousand words

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language (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

<table>
<thead>
<tr>
<th>Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract classes</td>
</tr>
<tr>
<td>aggregations</td>
</tr>
<tr>
<td>inheritance</td>
</tr>
<tr>
<td>relationship navigability</td>
</tr>
</tbody>
</table>

Cardinality

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>many</td>
<td>*</td>
</tr>
<tr>
<td>one and only one</td>
<td>1</td>
</tr>
<tr>
<td>optionally one</td>
<td>0..1</td>
</tr>
<tr>
<td>one or more</td>
<td>1..*</td>
</tr>
</tbody>
</table>


12 and this is a UML diagram so maybe more!
Theory

Context And Queue
1. (A platform a collection of devices sharing the same backend)
2. A context is a bundle of devices used for memory isolation
3. A queue use a context and a device to dispatch work or to allocate memory

```cpp
#include <CL/sycl.hpp>
namespace sycl = cl::sycl;

int main() {
    sycl::platform P(sycl::gpu_selector{});
    sycl::device D = P.get_devices(sycl::info::device_type::gpu)[0];
    sycl::context C(D);
    sycl::queue Q(C,D);
}
```
How to create a Queue

Explicit

```cpp
#include <CL/sycl.hpp>
namespace sycl = cl::sycl;

int main() {
  sycl::platform P{sycl::gpu_selector{}};
  sycl::device D = P.get_devices(sycl::info::device_type::gpu)[0];
  sycl::context C(D);
  sycl::queue Q(C,D);
}
```

Implicit

```cpp
#include <CL/sycl.hpp>
namespace sycl = cl::sycl;

int main() {
  sycl::queue Q{sycl::gpu_selector{}};
  // sycl::device D = Q.get_device();
  // sycl::context C = Q.get_context();
}
```
A note on Queue

- Queue are out-of-order by default
  - But can be created in order
    
    ```cpp
    sycl::queue
    Q{sycl::property_list{sycl::property::queue::in_order{}}}
    ```

- Queue submissions are asynchronous\(^{13}\)

\(^{13}\)More about that later
Theory

Unified Shared Memory
Unified Shared Memory

• `sycl::malloc_host` Pinned Memory
• `sycl::malloc_device` Only accessible on this device
• `sycl::malloc_shared` Accessible on device and on the host\(^\text{14}\)

API:

• `sycl::malloc_device` and `sycl::malloc_shared` are bound to a Context and a Device
• Hence to a Queue

\(^{14}\)And possibly on other device too
#include <CL/sycl.hpp>
namespace sycl = cl::sycl;

int main() {
  sycl::queue Q{sycl::gpu_selector{}};
  const int N{1729};
  float *A = sycl::malloc_device<float>(N, Q);
}
Kernel Submission
Parallel for

1. Define your kernel (as a functor)
2. Use a parallel for + range to submit your kernel to a Queue.

```cpp
#include <CL/sycl.hpp>
#include <numeric>
namespace sycl = cl::sycl;

int main() {
    const int N{1729};
    sycl::queue Q{sycl::gpu_selector{}};
    int *A = sycl::malloc_shared<int>(N, Q);
    Q.parallel_for(N, [=](sycl::item<1> id) { A[id] = id; }).wait();
    assert(std::accumulate(A, A+N, 0.) == N*(N-1)/2);
}
```
ND Range

```c
1  global_work_size = 1024 ; local_work_size = 8

SYCL / OpenCL / CUDA / Hip:
```
2  Q.parallel_for(sycl::nd_range<1>(sycl::range<1>(global_work_size),
3                        sycl::range<1>(local_work_size)),
4                        kernel);
```

OpenMP:
```
1  const int group_work_size = global_work_size / local_work_size;
2  #pragma omp team distribute
3  for (int group_id=0; group_id++ < group_work_size){
4      #pragma omp parallel for
5        for (local_id=0; local_id++ < local_work_size) {
6            const int global_id = local_id + group_id*local_work_size
7            mykernel(global_id, local_id)
8        }
9  }
```
Buffers
How to Handle Dependency?

1. Use "in-order" queue, may give up on some parallelism
2. Use "out-of-order" queue, need to put event/dependency everywhere
3. Use buffers!
1. Buffers **encapsulate** your data
2. Accessors **describe** how you access those data
3. Accessors will be use to perform an optimal scheduling
4. Buffer destruction will cause **synchronization**
```
#include <CL/sycl.hpp>
namespace sycl = cl::sycl;

int main(int argc, char **argv) {
    const int N = 100;
    std::vector<int> A(N);
    sycl::queue Q;

    sycl::buffer bufferA{A};
    Q.submit([&](sycl::handler &cgh) {
        sycl::accessor accessorA{bufferA, cgh, sycl::write_only, sycl::no_init};
        cgh.parallel_for(N, [=](sycl::id<1> idx) { accessorA[idx] = idx; });
    });

    for (size_t i = 0; i < N; i++)
}
```

This code will not perform any ‘H2D‘ transfer!
Buffer conclusion

- alloc/free in C is hard, we invested RAII in C++
- Handling dependency is hard with USM, we invested SYCL buffer
Interopt
Please see "Using Interoperability Mode in SYCL 2020" for more info
https://www.iwocl.org/iwocl-2022/program

1. SYCL Object from Backend Object
2. Backend Object from SYCL Object
3. Schedule a Backend Specific Command (host task)
Show me number!
Where does the number come from?

- Data provided by our friends from Codeplay
- As usual with data, take then with a grain of salt
- I encourage you to bench your code by yourself
- And open a Bug if the performance are not good enough!\(^\text{15}\)

\(^{15}\text{And if the same performance bugs exists on some Intel GPU, please contact me directly :)}\)
Conclusion
Conclusion

1. For better or worse, SYCL is C++
2. Many vendors (Intel, Nvidia, AMD) and hardware (CPU, GPU, FPGA) supported
3. Implicit data-movement by default (Buffer / Accessors concepts), but can you USM if preferred
4. Good interopt with other programming model
5. Competitive Performance
Lot of goods resources online

SYCL 2020 Spec

Examples
1. https://github.com/alcf-perfengr/sycltrain

Documentations (online and books)
1. https://sycl.tech/
Thank you! Do you have any questions?
# Assuming you are already theta

git clone https://github.com/alcf-perfengr/sycltrain

# Then read the readme in

cat ./sycltrain/presentation/2021_08_05_ATPESC/README.md