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
1. Introduction

2. DPCPP ecosystem

3. Theory
   - Context And Queue
   - Unified Shared Memory

4. Kernel Submission

5. Buffer innovation

6. Conclusion
Introduction
What programming model to target Accelerator?

- CUDA\(^1\) / HIP\(^2\) / OpenCL\(^3\)
- OpenMP (pragma based)
- Kokkos, raja, OCCA (high level, abstraction layer, academic project)
- SYCL (high level) / DPCPP\(^4\)
- Parallel STL\(^5\)

---

\(^1\)Compute Unified Device Architecture

\(^2\)Heterogeneous-Compute Interface

\(^3\)Open Computing Language

\(^4\)Data Parallel C++

\(^5\)SYCL 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)
   • The current stable SYCL specification is SYCL2020
SYCL Implementation

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.

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

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/](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/issues
  2. Use apt-get
  3. Download OneAPI pre-installed binary
  4. Ask your sys-admin to install it for you :)
DPCPP ecosystem
1. This is not a CUDA to DPCPP source to source compiler.
2. "Tool Assisted Porting"

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

---

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\textsuperscript{12}

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language\textsuperscript{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

<table>
<thead>
<tr>
<th>Relationships</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract classes</td>
<td>{abstract}</td>
</tr>
<tr>
<td>aggregations</td>
<td>(\diamondsuit)</td>
</tr>
<tr>
<td>inheritance</td>
<td>(\uparrow)</td>
</tr>
<tr>
<td>relationship navigability</td>
<td>(\wedge)</td>
</tr>
</tbody>
</table>

### Cardinality

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>many</td>
<td>(\ast)</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>

\textsuperscript{1} Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).

\textsuperscript{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);
}
```
1. Context are explicit! You need to take care of them.
2. Contexts are used for memory isolation.
3. In particular, a kernel submitted to a queue (Q1) can only access memory that has been allocated in the same context as the one used to create Q1.
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
- Queue submissions is asynchronous\textsuperscript{13}

\textsuperscript{13} Use event to synchronize
Theory

Unified Shared Memory
Unified Shared Memory

All the info here: https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#table.USM.allocation.characteristics

Allocate memory on a device:

- `sycl::malloc_device` Only accessible on this device
- `sycl::malloc_shared` Accessible on device and on the host\textsuperscript{14}

API:

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

\textsuperscript{14}And possibly on other device too
Allocation example

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);
    const int N{1729};
    float *A = sycl::malloc_device<float>(N,D,C);
}
```

Implicit

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

int main() {
    const int N{1729};

    sycl::queue Q1{sycl::gpu_selector{}};
    float *A = sycl::malloc_device<float>(N, Q1);

    sycl::queue Q2{sycl::gpu_selector{}};
    float *B = sycl::malloc_device<float>(N, Q2);

    Q1.memcpy(A, B, N*sizeof(float)).wait();
}
```
```cpp
#include <CL/sycl.hpp>
namespace sycl = cl::sycl;

void f_implicit(const int N){
    sycl::queue Q{sycl::gpu_selector{}}; // One queue == One context
    float *A = sycl::malloc_device<float>(N,Q);
    float *B = sycl::malloc_device<float>(N,Q);
    Q.memcpy(A,B,N*sizeof(float)).wait();
}

void f_explicit(const int N){
    sycl::platform P{sycl::gpu_selector{}};
    sycl::device D = P.get_devices(sycl::info::device_type::gpu)[0];
    sycl::context C(D);

    sycl::queue Q1(C,D); // 2 Queues but same context!
    float *A = sycl::malloc_device<float>(N,Q1);
    sycl::queue Q2(C,D);
    float *B = sycl::malloc_device<float>(N,Q2);
    Q1.memcpy(A,B,N*sizeof(float)).wait();
}

int main(){
    const int N{10}; f_explicit(N); f_implicit(N);
}
```
1. Platform->Devices->Context->Queue
2. Unified Shared Memory Allocation
1. Define your kernel (as a functor)

2. Use a parallel for + range to submit you 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);
}
```
Kernel Submission
A kernel is submitted to a Queue
The kernel is invoked once for each work item
local work size Work items are grouped into a work group
The total number of all work items is specified by the global work size

15 similar to MPI_rank
16 similar to pragma omp simdlen/safelen
17 Credit The OpenCL Programming Book by Fixstars
**Implicit Loop: Example!**

```
1. global_work_size = 24; local_work_size = 8

**SYCL / OpenCL / CUDA / Hip:**

```
1. Q.parallel_for(sycl::nd_range<1>(sycl::range<1>(global_work_size),
   sycl::range<1>(local_work_size)),
   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_id < group_work_size){
   #pragma omp parallel for simd
   for (local_id=0; local_id++; local_id < local_work_size) {
      const int global_id = local_id + group_id*local_work_size
      mykernel(global_id, local_id)
   }
}
```
Buffer innovation
1. Buffers *encapsulate* your data
2. Accessors *describe* how you access those data
3. Buffer destruction will cause *synchronization*
```cpp
#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++)
    }
```

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)
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?
Hands-on

# 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
```