

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

1. Introduction
2. DPCPP (and the associated ecosystem)
3. Theory
4. Conclusion

Goal of this talk

1. Give you a feel of SYCL/DPCPP
2. Tease you enough so you want to play with SYCL during the Hands-on¹
3. Question are welcomed!

¹Or at least watch me going through some examples... on Exclusive Hardware!

Introduction

What programming model to use to target GPU?

- Parallel STL
- OpenMP (pragma based)
- CUDA² / HIP³ / OpenCL⁴ (low level)
- Kokkos, raja, OCCA (high level, abstraction layer, academic project)
- SYCL (high level) / DPCPP⁵

²Compute Unified Device Architecture

³Heterogeneous-Compute Interface

⁴Open Computing Language

⁵Data Parallel C++

What is SYCL™?

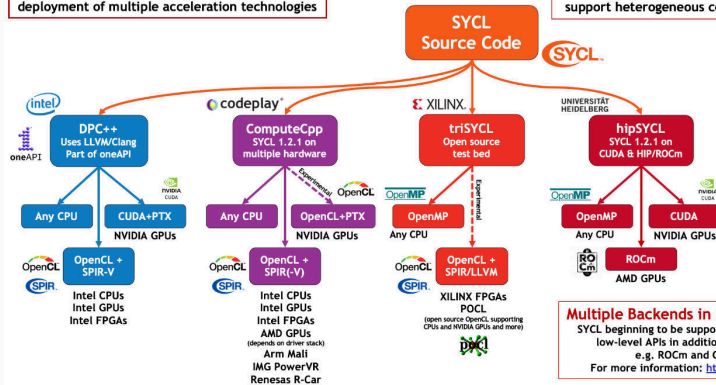
1. Target C++ programmers (template, lambda)
 - No language extension
 - No pragmas
 - No attribute
2. Borrow lot of concept from battle tested OpenCL (platform, device, work-group, range)
3. Single Source (two compilation pass)
4. **Implicit data-transfer**
5. SYCL is a Specification developed by the Khronos Group (OpenCL, SPIR, Vulkan, OpenGL)
 - The current stable SYCL specification is 1.2.
 - SYCL 2020 is expected to be approved at the end of the years⁶.

⁶<https://www.khronos.org/registry/SYCL/specs/sycl-2020-provisional.pdf>

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>

⁷Credit: Khronos groups (<https://www.khronos.org/sycl/>)

What is DPCPP¹⁰?

- Intel implementation of SYCL⁸
- The name of the SYCL-aware Intel compiler⁹

⁸Obvious from the name isn't it?

⁹So you don't need to pass `-fsycl`

¹⁰Data Parallel C++: <https://github.com/intel/llvm/releases>

DPCPP (and the associated ecosystem)

DPCPP a high potential SYCL implementation


DPCPP implement the SYCL Standard + extension¹¹


- Unnamed Lambda
- Unified Shared Memory
- Reduction
- Magic introspection function
- Explicit SIMD


Many of DPCPP extension are now merged in the new SYCL2020 standard!

¹¹<https://github.com/intel/llvm/tree/sycl/sycl/doc/extensions>

A note on USM

**Cppcon**
The C++ Conference
September 13-16





Michael Wong & Gordon Brown

	Explicit USM (minimum)	Restricted USM (optional)	Concurrent USM (optional)	System USM (optional)
Consistent pointers	✓	✓	✓	✓
Pointer-based structures	✓	✓	✓	✓
Explicit data movement	✓	✓	✓	✓
Shared access	✗	✓	✓	✓
Concurrent access	✗	✗	✓	✓
System allocations (malloc/new)	✗	✗	✗	✓

codeplay[®] | Microsoft Windows | © 2020 Codeplay Software Ltd.

Heterogeneous Programming in C++ with SYCL 2020

ansatz

- SYCL2020: with Native programming model (OpenCL, Cuda, ...)
 - Creating sycl Buffer with *cl_mem*
 - Creating sycl Kernel from Opencl Kernel
- DPCPP: With OpenMP. Not yet memory compatibly. But on the pipeline¹²

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

1. This is **not** a CUDA to DPCPP source to source compiler¹³.
2. "Tool Assisted Porting", not part of th
3. *module load dpct* on JLSE

¹³It's a little more complex than a perl script

¹⁴*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 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>

¹⁵*<https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html>*

1. Lot of profiler available ¹⁶.
2. Will trace the underlying back-end calls

```
$ iprof ./4_buffer  
Running on Intel(R) Gen12LP HD Graphics NEO  
A[ 0 ] = 0
```

```
== OpenCL ==
```

```
API calls | 1 Hostnames | 1 Processes | 1 Threads
```

Name	Time	Time(%)	Calls	Average
clBuildProgram	112.49ms	97.69%	1	112.49ms
clCreateBuffer	1.50ms	1.30%	1	1.50ms
clEnqueueNDRangeKernel	584.20us	0.51%	1	584.20us
Total	115.15ms	100.00%	4	

```
[...]
```

¹⁶For iprof module load iprof

Theory

First thing first: λ !

Constructs a closure: an unnamed function object capable of capturing variables in scope.

Lambda Hello World¹⁷

```
1  int main() {
2      //[] -> Capture; [=] capture by value, [&] capture by reference
3      //() -> Parameter
4      //{ } -> Body
5      [](){};
6      return 0;
7  }

1  void printVector(vector<int> v) {
2      const offset = 0;
3      // lambda expression to print vector
4      for_each(v.begin(), v.end(), [=](int i) {
5          cout << offset + i << endl;
6      });
7  }
```

¹⁷Yes, it look like APL

A picture is worth a thousand words¹⁸

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language¹ (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 ^

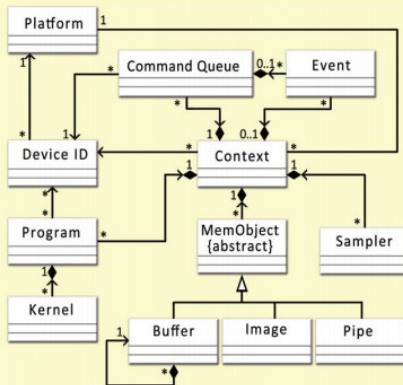
Cardinality

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).

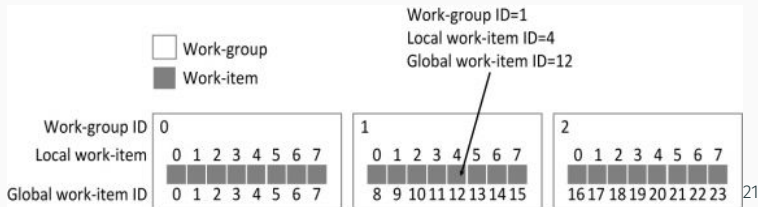
¹⁸and this is a UML diagram so maybe more!

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

Implicit Loop

- A Kernel is invoked once for each **work item**¹⁹
- **local work size** Work items are grouped into a **work group**²⁰
- The total number of all work items is specified by the **global work size**



¹⁹similar to *MPI_rank*

²⁰similar to *pragma omp simdlen/safelen*

²¹Credit The OpenCL Programming Book by Fixstars

Implicit Loop: Example!

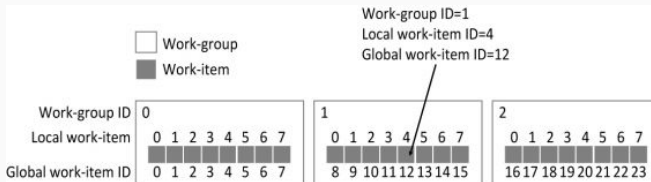
```
1  global_work_size = 24 ; local_work_size = 8
```

SYCL / Opencl / CUDA

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

OpenMP²²

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



²²Using chunking / tilling / vectorization technique

SYCL Example

```
1  std::vector<int> A(global_range);
2  {
3      sycl::buffer<sycl::cl_int, 1> bufferA(A.data(), A.size());
4      sycl::queue myQueue;
5      myQueue.submit([&](sycl::handler &cgh) {
6          auto accessorA = bufferA.get_access<sycl::access::mode::write>(cgh);
7          cgh.parallel_for<class hello_world>(
8              sycl::range<1>(global_range),
9              [=](sycl::id<1> idx) { accessorA[idx] = idx[0];}
10         );
11     });
12 }
```

Teasing: SYCL Explicit SIMD Example

```
1  q.submit([&](handler &cgh) {
2      auto PA = bufa.get_access<access::mode::read>(cgh);
3      auto PB = bufb.get_access<access::mode::read>(cgh);
4      auto PC = bufc.get_access<access::mode::write>(cgh);
5      cgh.parallel_for<class Test>(
6          GlobalRange * LocalRange, [=](id<1> i) SYCL_ESIMD_KERNEL {
7          using namespace sycl::INTEL::gpu;
8          unsigned int offset = i * VL * sizeof(float);
9          simd<float, VL> va = block_load<float, VL>(PA, offset);
10         simd<float, VL> vb = block_load<float, VL>(PB, offset);
11         simd<float, VL> vc = va + vb;
12         block_store(PC, offset, vc);
13     });
14 }
```

"Pro" tips

1. Queue creation are expensive. Try to reuse the queue²³
2. The easiest way to achieved parallel execution of kernel, is to use mutiple queue.
3. You need also to pass ID around, sometime it can be tedious.
dpcpp provide an extension to emulate OpenCL
get_global_id()
4. Use compatibility and USM to ease the porting

²³You already pass *mpi_com_world* around, just do the same things.

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

1.2.1 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/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)

Thank you! Do you have any questions?