Argonne NATIONAL LABORATORY

# SYCL/DPC++ – And 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. Theory

3. Conclusion

# Introduction

# What programming model to use to target GPU?

- OpenMP (pragma based)
- CUDA[1] / Hip / OpenCL (low level)
- Kokkos, raja, OCCA (high level, abstraction layer, academic project)
- SYCL (high level)

[1] Compute Unified Device Architecture

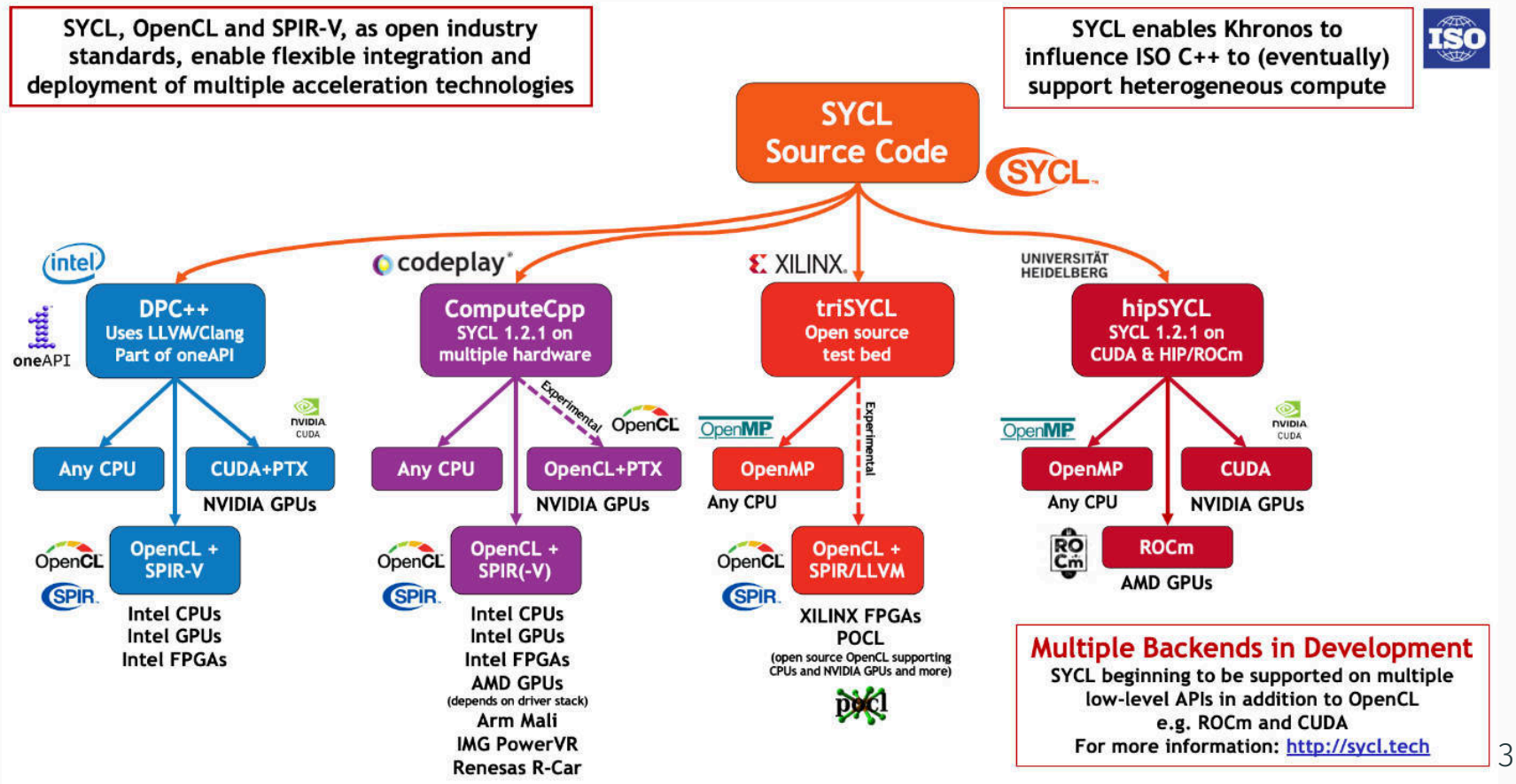
# What is SYCL™?

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 years[2].

[2] https://www.khronos.org/registry/SYCL/specs/sycl-2020-provisional.pdf

# SYCL Implementation




[3]

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

# What is DPCPP[5]?

- Intel implementation of SYCL[4]
- The name of the SYCL-aware Intel compiler

[4] Obvious from the name isn't it?
[5] Data Parallel C++

# DPCPP specialty

DPCPP implement the SYCL Standard + HPC extension

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

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

# DPCT: CUDA to DPCPP translator

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

# oneMKL

> *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`

# Vtunes / Advisor / iprof

1. Lot of profiler available.
2. Will trace monitor 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 |

[...]
```

# Goal of this talk

1. Give you a feel of SYCL/DPCPP so you know if it's maybe for you or not.
2. Tease you enough so you want to play with SYCL during the Hands-on[6]
3. Question are welcomed!

[6] Or at least watch me going through some examples

# Theory

# First thing first: $\lambda$ !

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

Lambda Hello World[7]

```
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  }
```

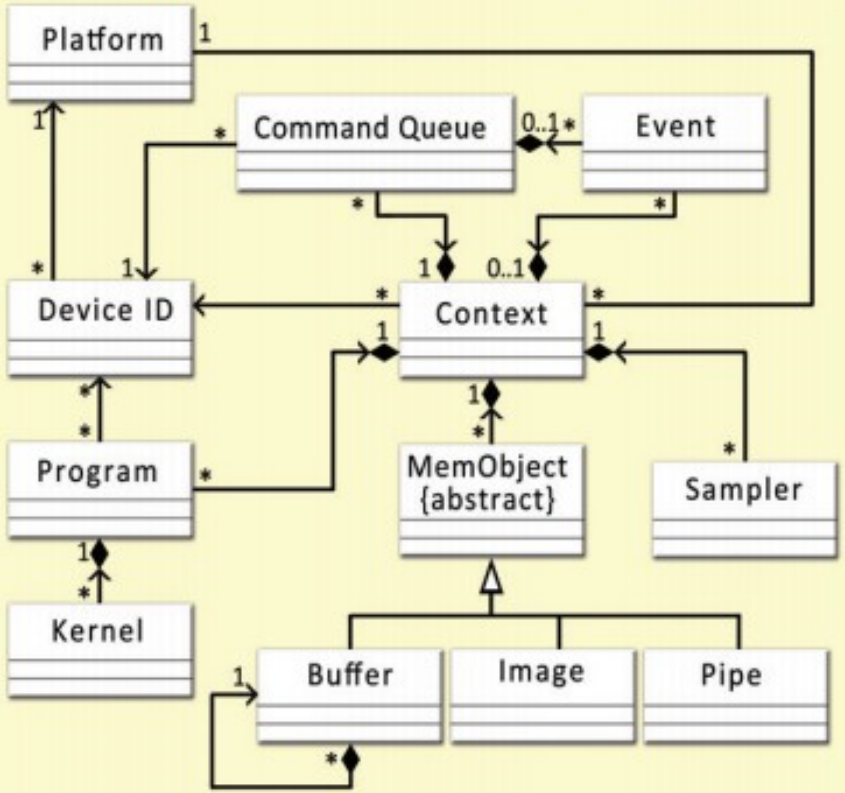
[7] Yes, it look like APL

# A picture is worth a thousand words[8]

## 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).

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

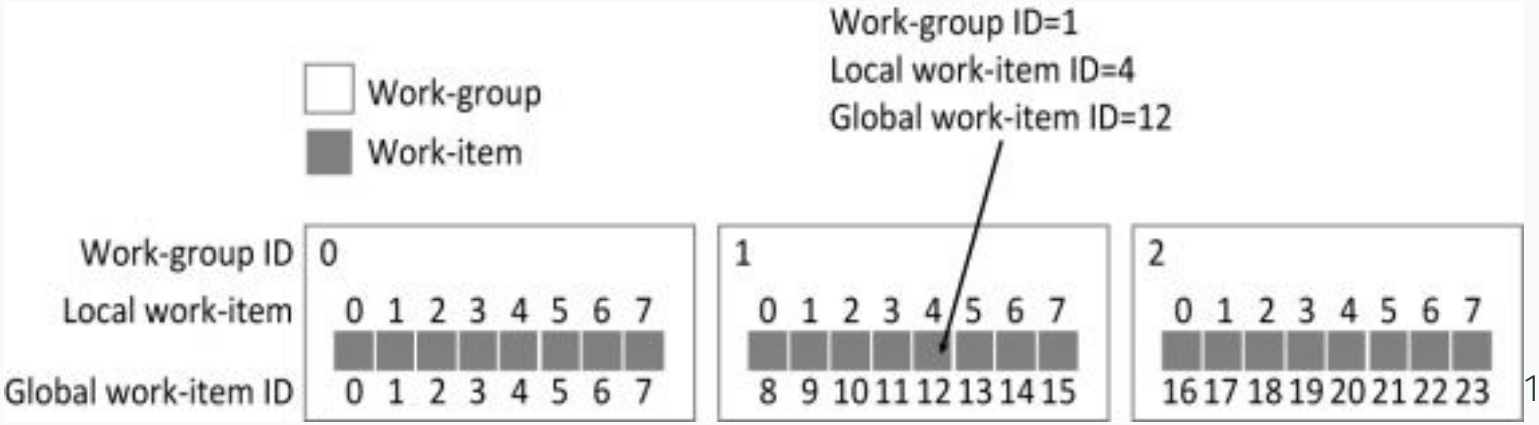
# Memory management: SYCL innovation

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

# Implicit Loop

- A Kernel is invoked once for each work item [9]
- local work size Work items are grouped into a work group [10]
- The total number of all work items is specified by the global work size



[11]

[9] similar to *MPI_rank*
[10] similar to *pragma omp simdlen/safelen*
[11] Credit The OpenCL Programming Book by Fixstars

# Implicit Loop: Example!

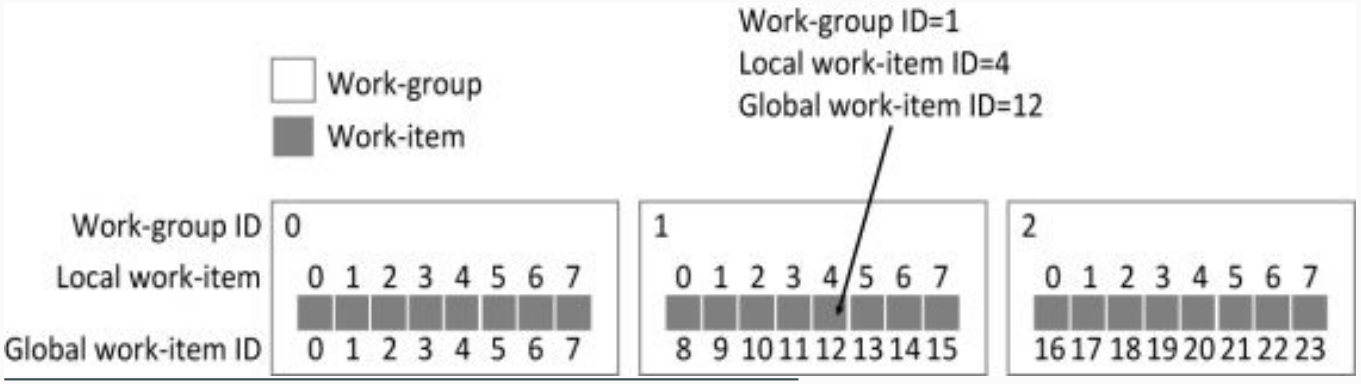
```
global_work_size = 24 ; local_work_size = 8
```

## SYCL / Opencl / CUDA

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

## OpenMP[12]

```
# 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
```




[12] Using 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  }
```

# ”Pro” tips

1. Queue creation are expensive. Try to reuse the queue[13]
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()`

[13] You 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

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

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?