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

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]

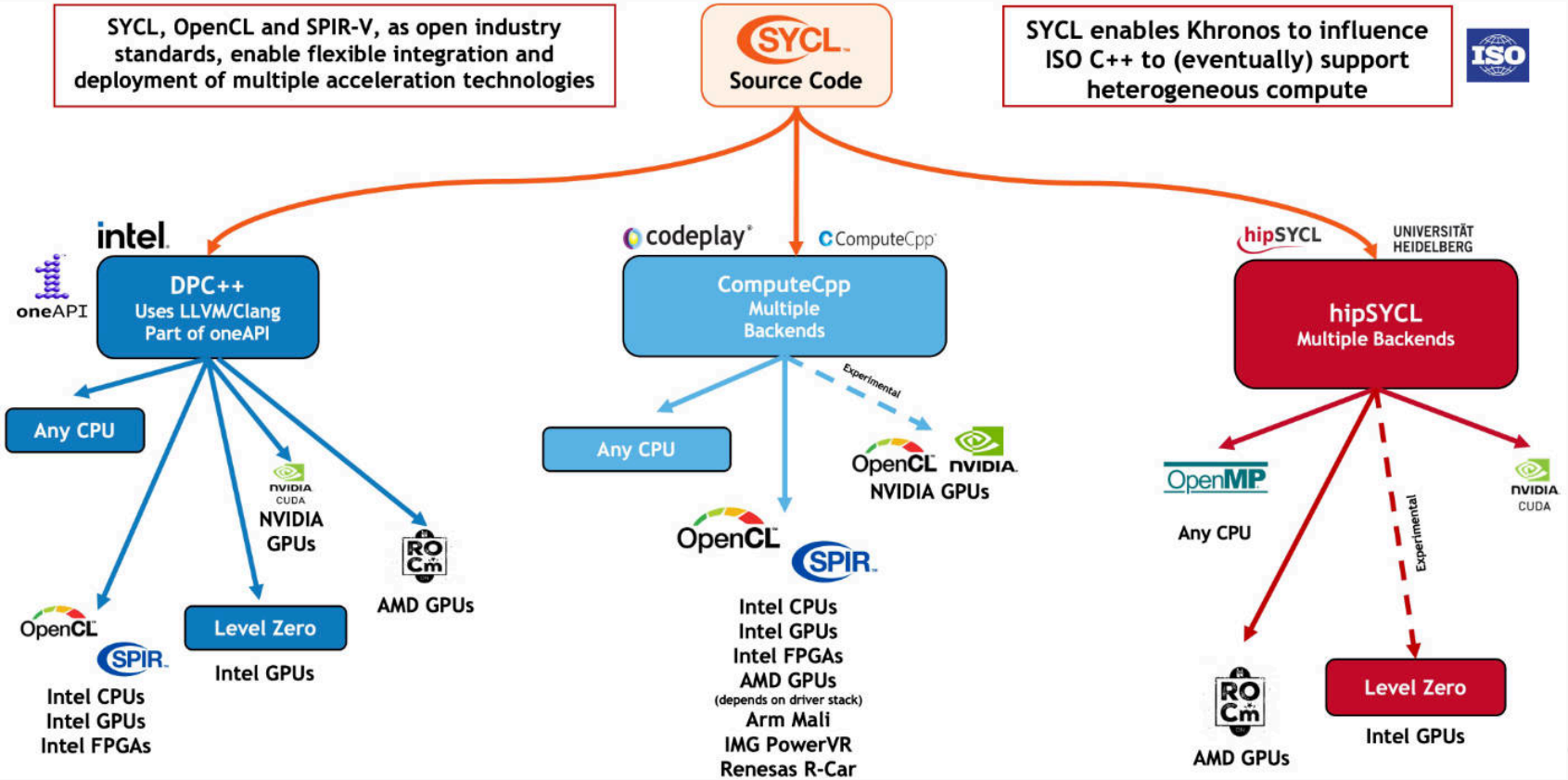
[1] Compute Unified Device Architecture
[2] Heterogeneous-Compute Interface
[3] Open Computing Language
[4] Data Parallel C++
[5] SYCL implementation exist *https://github.com/oneapi-src/oneDPL*

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



[6]

[6] 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 NVDIA 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 :)

DPCPP ecosystem

SYCLomatic: CUDA to SYCL translator[9]

1. A CUDA to SYCL source to source compiler
2. Used by a few Apps with some of success

[9] *https://github.com/oneapi-src/SYCLomatic*

oneMKL[10]

> *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

[10] *https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html*

OneDPL[11]

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.*

[11] https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/dpc-library.html

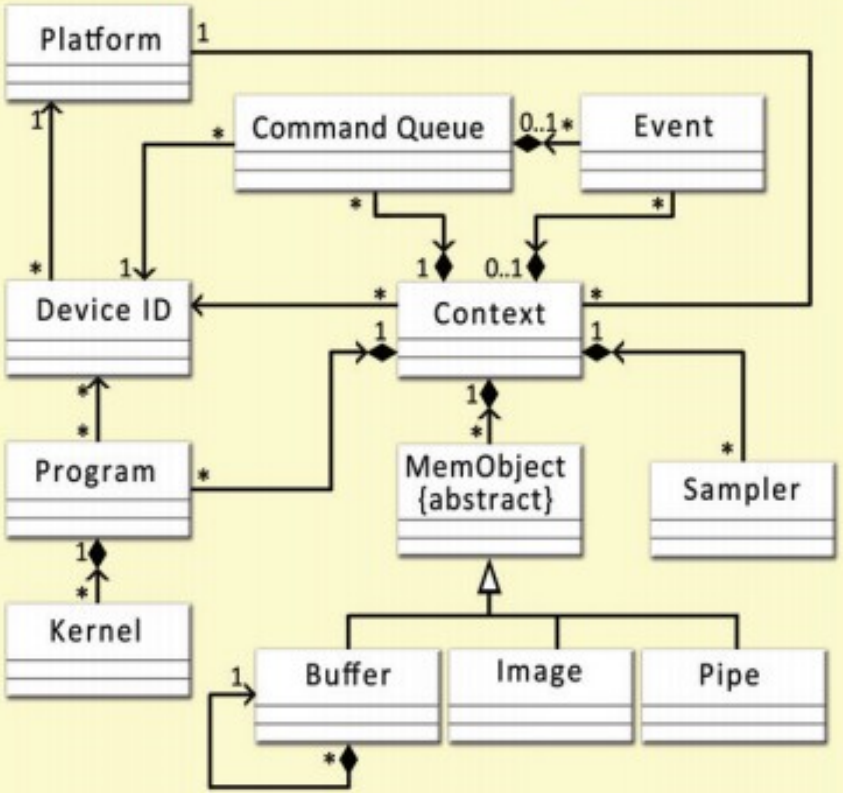
Theory

A picture is worth a thousand words[12]

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).

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

Theory

Context And Queue

(Platform ->) Devices -> Context -> 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

```
1  #include <CL/sycl.hpp>
2  namespace sycl = cl::sycl;
3
4  int main() {
5    sycl::platform P(sycl::gpu_selector{});
6    sycl::device D = P.get_devices(sycl::info::device_type::gpu)[0];
7    sycl::context C(D);
8    sycl::queue Q(C,D);
9  }
```

How to create a Queue

Explicit

```
1  #include <CL/sycl.hpp>
2  namespace sycl = cl::sycl;
3
4  int main() {
5    sycl::platform P{sycl::gpu_selector{}};
6    sycl::device D = P.get_devices(sycl::info::device_type::gpu)[0];
7    sycl::context C(D);
8    sycl::queue Q(C,D);
9  }
```

Implicit

```
1  #include <CL/sycl.hpp>
2  namespace sycl = cl::sycl;
3
4  int main() {
5   sycl::queue Q{sycl::gpu_selector{}};
6  // sycl::device D = Q.get_device();
7  // sycl::context C = Q.get_context();
8  }
```

A note on Queue

- Queue are out-of-order by default
  - But can created in order

    ```
    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[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

Allocation example

```
1  #include <CL/sycl.hpp>
2  namespace sycl = cl::sycl;
3
4  int main() {
5   sycl::queue Q{sycl::gpu_selector{}};
6   const int N{1729};
7   float *A = sycl::malloc_device<float>(N,Q);
8  }
```

Kernel Submission

Parallel for

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

```
1  #include <CL/sycl.hpp>
2  #include <numeric>
3  namespace sycl = cl::sycl;
4
5  int main() {
6   const int N{1729};
7   sycl::queue Q{sycl::gpu_selector{}};
8   int *A = sycl::malloc_shared<int>(N,Q);
9   Q.parallel_for(N, [=](sycl::item<1> id) { A[id] = id; }).wait();
10  assert(std::accumulate(A, A+N, 0.) == N*(N-1)/2);
11 }
```

ND Range

```
global_work_size = 1024 ; local_work_size = 8
```

SYCL / OpenCL / CUDA / Hip:

```
Q.parallel_for(sycl::nd_range<1>(sycl::range<1>(global_work_size),
                                 sycl::range<1>(local_work_size)),
               kernel);
```

OpenMP:

```
const int group_work_size = global_work_size / local_work_size;
#pragma omp team distribute
for (int group_id=0; group_id++; group_id < group_work_size){
    #pragma omp parallel for
    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)
    }
}
```

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!

Memory management: SYCL innovation

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

Buffer Example

```
1  #include <CL/sycl.hpp>
2  namespace sycl = cl::sycl;
3
4  int main(int argc, char **argv) {
5    const int N= 100;
6    std::vector<int> A(N);
7    sycl::queue Q;
8    {
9      sycl::buffer bufferA{A};
10     Q.submit([&](sycl::handler &cgh) {
11       sycl::accessor accessorA{bufferA, cgh,
12                                sycl::write_only, sycl::no_init};
13       cgh.parallel_for(N, [=](sycl::id<1> idx) { accessorA[idx] = idx;});
14     });
15   }
16   for (size_t i = 0; i < N; i++)
17     std::cout << "A[ " << i << " ] = " << A[i] << std::endl;
18 }
```

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

Interoperability

Please see ”Using Interoperability Mode in SYCL 2020” for more info
https://www.iwocl.org/iwocl-2022/program

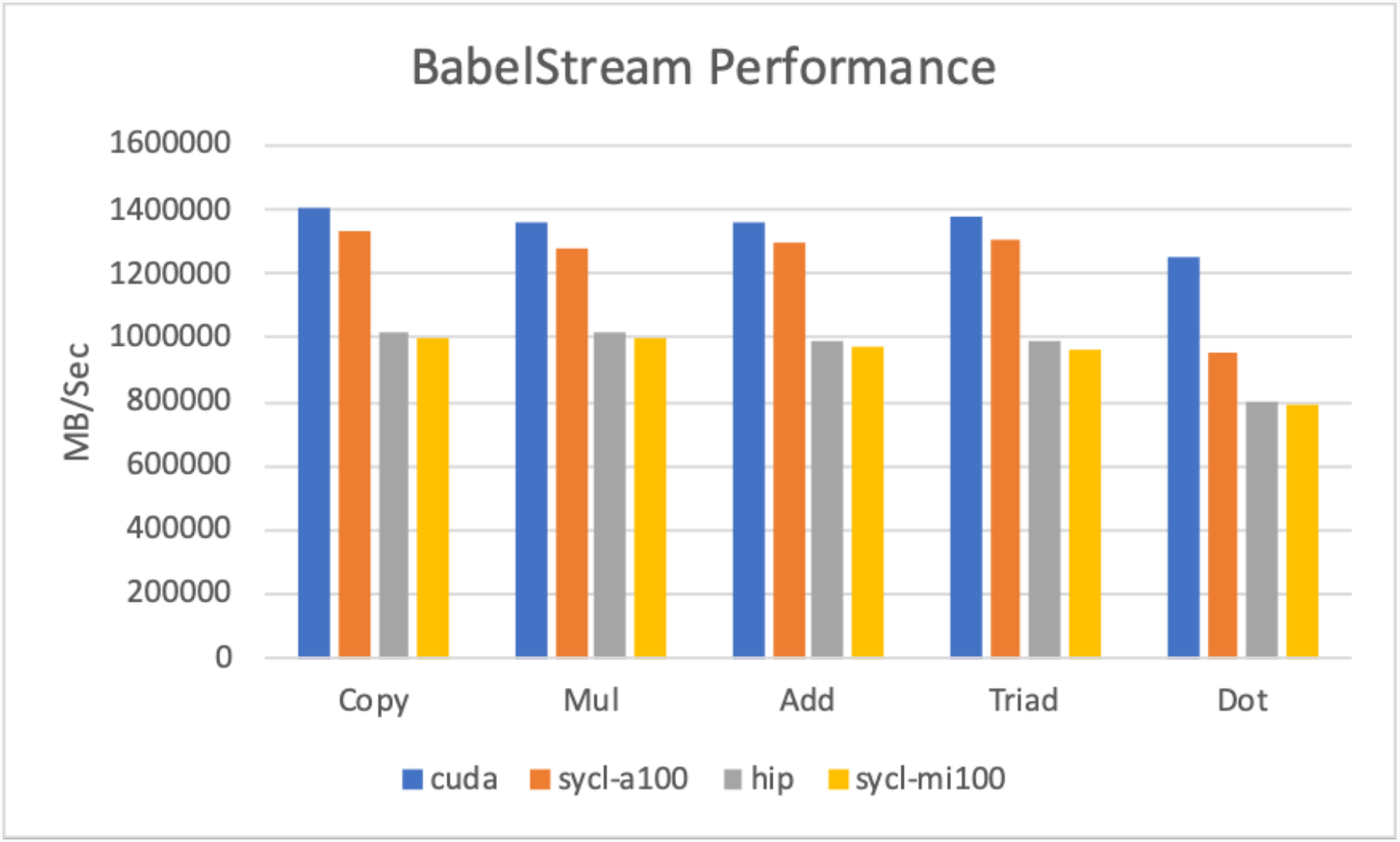
1. SYCL Object from Backend Object
2. Backend Object from SYCL Object
3. Schedule a Backend Specific Command (host task)

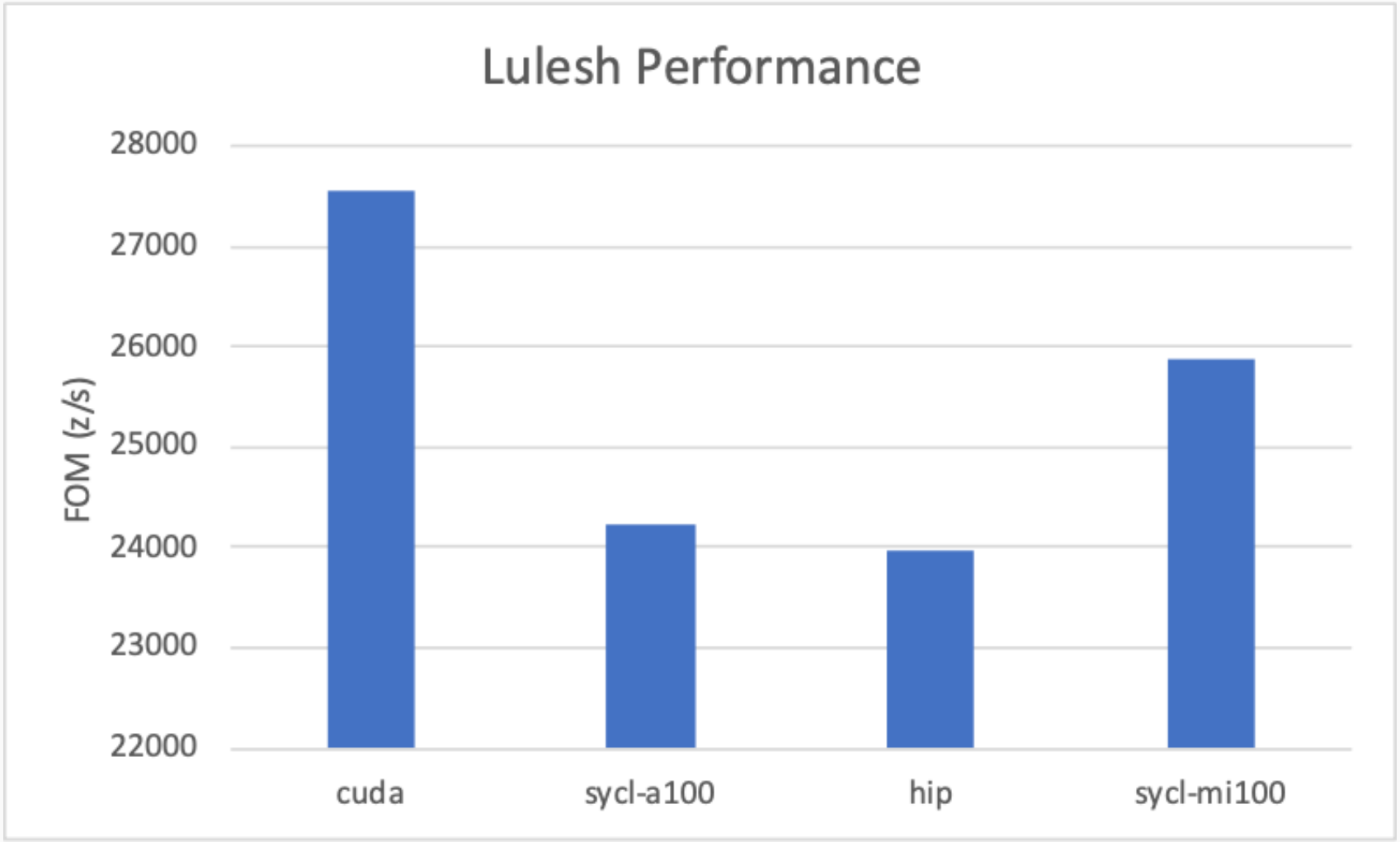
Show me number!

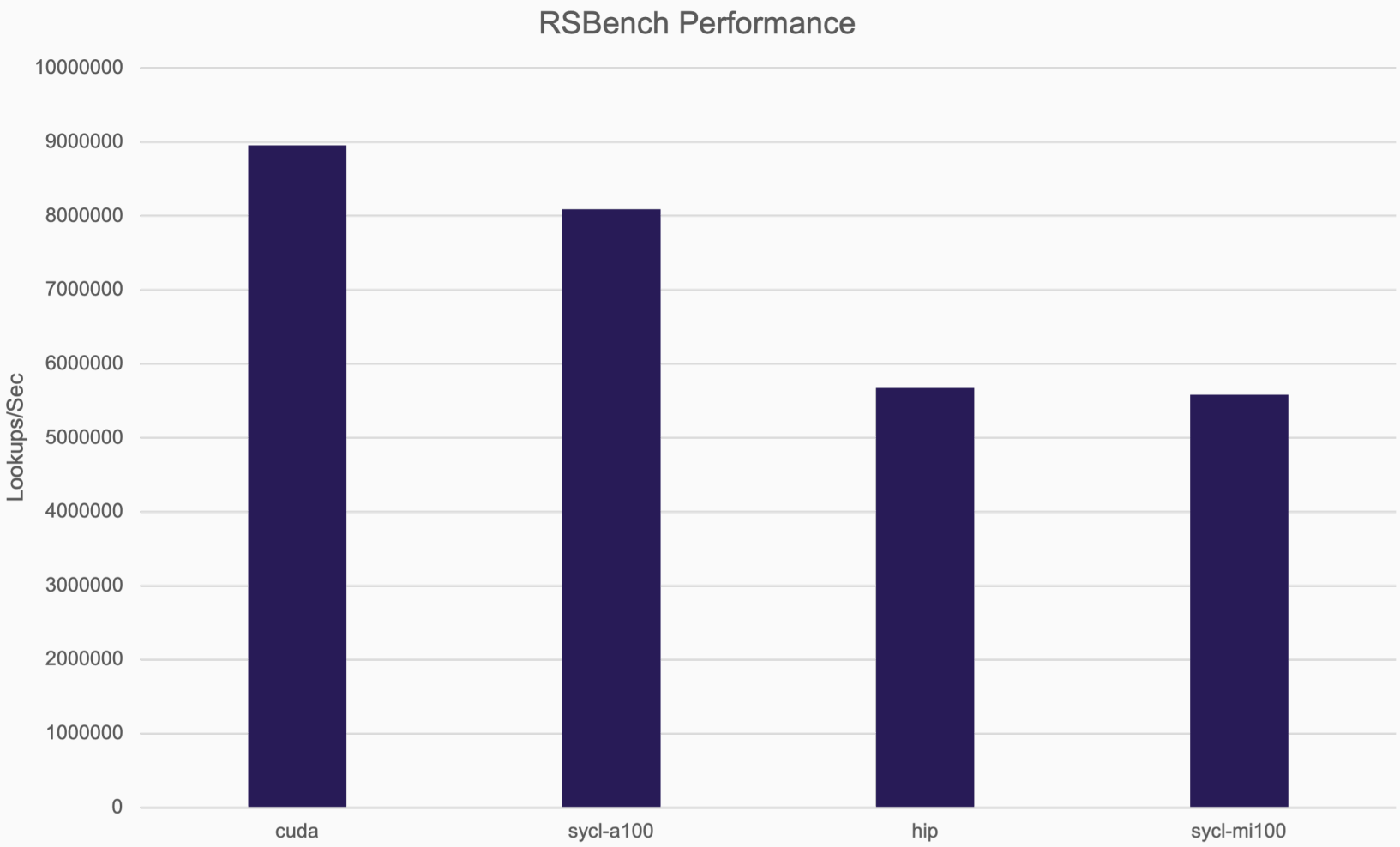
Where does the number come from?

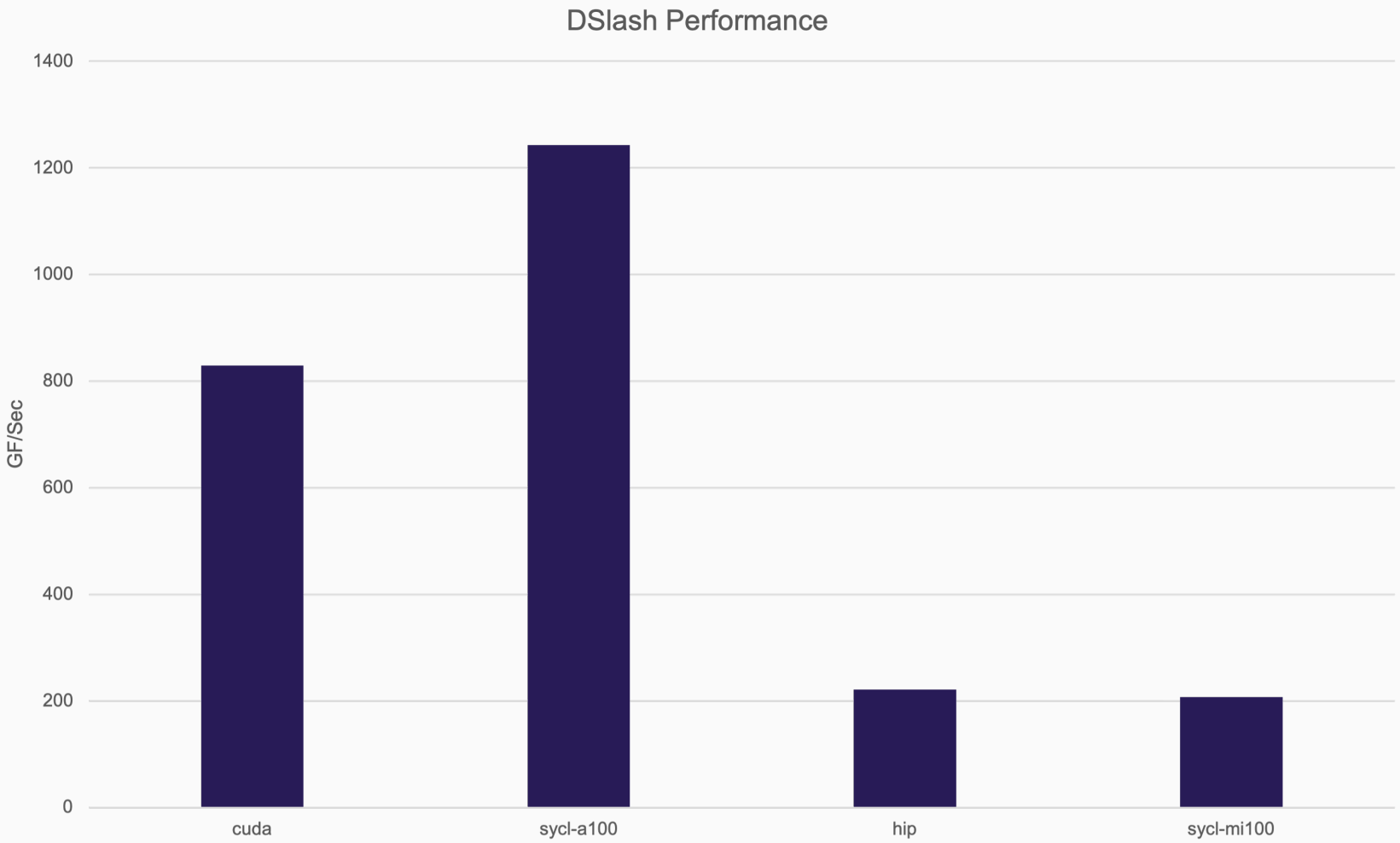
- Data provided by our friends from Codeplay
- As usual with data, take then with a grain of salt
- I encourage you to bench your code by yourself
- And open a Bug if the performance are not good enough![15]

[15] And if the same performance bugs exists on some Intel GPU, please contact me directly :)


BabelStream Performance
MB/Sec
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Copy
Mul
Add
Triad
Dot
cuda
sycl-a100
hip
sycl-mi100


Lulesh Performance
28000
27000
26000
25000
24000
23000
22000
FOM (z/s)
cuda
sycl-a100
hip
sycl-mi100


RSBench Performance
Lookups/Sec
10000000
9000000
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
cuda
sycl-a100
hip
sycl-mi100


DSlash Performance
GF/Sec
1400
1200
1000
800
600
400
200
0
cuda
sycl-a100
hip
sycl-mi100

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

1. *https://www.khronos.org/files/sycl/sycl-2020-reference-guide.pdf*
2. *https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.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)

Q&A

Thank you! Do you have any questions?

Hands-on

```
# Assuming you are alrady theta
git clone https://github.com/alcf-perfengr/sycltrain
# Then read the readme in
cat ./sycltrain/presentation/2021_08_05_ATPESC/README.md
```