SYCL and OpenCL
State of the Nation

Michael Wong | ISOCP VP
Codeplay Vice President of R & D
SYCL Working Group Chair
Chair C++ Standard SG5, SG14
michael@codeplay.com | wongmichael.com

Ronan Keryell | Xilinx
SYCL Spec Editor
rkeryell@xilinx.com
A tale of two cities
OH, East is East, and West is West, and never the twain shall meet...

-Rudyard Kipling
Khronos is an International Industry Consortium of over 100 companies creating royalty-free, open standard APIs to enable software to access hardware acceleration for 3D graphics, Virtual and Augmented Reality, Parallel Computing, Neural Networks and Vision Processing.
OpenCL - Low-level Parallel Programming

- Low-level, explicit programming of heterogeneous parallel compute resources
  - One code tree can be executed on CPUs, GPUs, DSPs and FPGA ...

- OpenCL C or C++ language to write kernel programs to execute on any compute device
  - Platform Layer API - to query, select and initialize compute devices
  - Runtime API - to build and execute kernels programs on multiple devices

- The programmer gets to control:
  - What programs execute on what device
  - Where data is stored in various speed and size memories in the system
  - When programs are run, and what operations are dependent on earlier operations
OpenCL 2.2 Released in May 2017

- **2011**
  - OpenCL 1.2
  - Becomes industry baseline for heterogeneous parallel computing

- **2013**
  - OpenCL 2.0
  - Enables new class of hardware
    - SVM
    - Generic Addresses
    - On-device dispatch

- **2015**
  - OpenCL 2.1
  - SPIR-V 1.0
  - SPIR-V in Core
    - Kernel Language Flexibility

- **2017**
  - OpenCL 2.2
  - SPIR-V 1.2
  - SPIR-V 1.2
    - OpenCL C++ Kernel Language
      - Static subset of C++14
      - Templates and Lambdas
    - Pipes
      - Efficient device-scope communication between kernels
  - Code Generation Optimizations
    - Specialization constants at SPIR-V compilation time
    - Constructors and destructors of program scope global objects
    - User callbacks can be set at program release time

[https://www.khronos.org/opencl/](https://www.khronos.org/opencl/)
OpenCL Ecosystem

Hardware Implementers
Desktop/Mobile/Embedded/FPGA

OpenCL 2.2 - Top to Bottom C++

SYCL™
Single Source C++ Programming

OpenCL
Core API and Language Specs

SPIR™
Portable Kernel Intermediate Language

100s of applications using OpenCL acceleration
Rendering, visualization, video editing, simulation, image processing, vision and neural network inferencing
OpenCL Conformant Implementations

Vendor timelines are first conformant submission for each spec generation

- **Dec08**
  - OpenCL 1.0 Specification

- **Jun10**
  - OpenCL 1.1 Specification

- **Nov11**
  - OpenCL 1.2 Specification

- **Nov13**
  - OpenCL 2.0 Specification

- **Nov15**
  - OpenCL 2.1 Specification

Desktop:
- 2.0 | Dec14
- 1.2 | May15
- 2.0 | Jul14
- 1.2 | May15
- 2.0 | Apr17

Mobile:
- 2.0 | Nov15
- 1.2 | Feb11
- 1.2 | Mar16
- 1.2 | Sep14

Embedded:
- 1.2 | May15
- 1.2 | Aug15

FPGA:
- 1.0 | Jul13
- 1.2 | Dec14
- 1.0 | May09
- 1.1 | Aug10
- 1.2 | May12
- 1.0 | May10
- 1.1 | Feb11
- 1.0 | May09
- 1.1 | Jun10
- 1.1 | Mar11
- 1.2 | Dec12
- 1.1 | Aug12
- 1.2 | Sep13
- 1.1 | Nov12
- 1.2 | Apr14
- 1.1 | Apr12
- 1.2 | Dec14
- 1.1 | May13
- 1.2 | Sep14

Desktop:
- 2.0 | Dec14
- 1.2 | May15
- 2.0 | Jul14
- 1.2 | May15
- 2.0 | Apr17

Mobile:
- 2.0 | Nov15
- 1.2 | Feb11
- 1.2 | Mar16
- 1.2 | Sep14

Embedded:
- 1.2 | May15
- 1.2 | Aug15

FPGA:
- 1.0 | Jul13
- 1.2 | Dec14
- 1.0 | May09
- 1.1 | Aug10
- 1.2 | May12
- 1.0 | May10
- 1.1 | Feb11
- 1.0 | May09
- 1.1 | Jun10
- 1.1 | Mar11
- 1.2 | Dec12
- 1.1 | Aug12
- 1.2 | Sep13
- 1.1 | Nov12
- 1.2 | Apr14
- 1.1 | Apr12
- 1.2 | Dec14
- 1.1 | May13
- 1.2 | Sep14

Vendor timelines are first conformant submission for each spec generation.
OpenCL as Language/Library Backend

- Caffe: C++ based Neural network framework
- Halide: Language for image processing and computational photography
- C++ AMP: Multi-coreWare open source project on Bitbucket
- SYCL: Single Source C++ Programming for OpenCL
- aparapi: Java language extensions for parallelism
- OpenCV: Vision processing open source project
- OpenACC: Compiler directives for Fortran, C and C++
- TensorFlow: Open source software library for machine learning

Hundreds of languages, frameworks and projects using OpenCL to access vendor-optimized, heterogeneous compute runtimes

Over 4,000 GitHub repositories using OpenCL: tools, applications, libraries, languages - up from 2,000 two years ago
What is SYCL?

High-level C++ abstraction layer for OpenCL
Full coverage for all OpenCL features
Interop to enable existing OpenCL code with SYCL
Single-source compilation
Automatic scheduling of data movement
SYCL Example

// Create a device queue.
class: sycl::queue device_queue;

// Create buffers.
class: sycl::range<1> n_items{array_size};
class: sycl::buffer<class: sycl::cl_int, 1> in_buffer(in.data(), n_items);
class: sycl::buffer<class: sycl::cl_int, 1> out_buffer(out.data(), n_items);

// Submit a kernel and associated data movement operations.
device_queue.submit([&](class: sycl::handler &cgh) {
   // Defines the kernels access requirements.
   auto in_accessor = in_buffer.get_access<class: sycl::access::mode::read>(cgh);
   auto out_accessor = out_buffer.get_access<class: sycl::access::mode::write>(cgh);
   // Defines the kernel itself.
   cgh.parallel_for<class VecScalMul>(n_items, [=](class: sycl::id<1> wiID) {
      out_accessor[wiID] = in_accessor[wiID]*2;
   });
});
Separating Storage & Access

Buffers manage data across host CPU and one or more devices.

Buffers and accessors type safe access across host and device.

Accessors are used to describe access.

Buffers manage data across host CPU and one or more devices.

Buffers and accessors type safe access across host and device.

Accessors are used to describe access.

Buffers manage data across host CPU and one or more devices.

Buffers and accessors type safe access across host and device.

Accessors are used to describe access.

Buffers manage data across host CPU and one or more devices.

Buffers and accessors type safe access across host and device.

Accessors are used to describe access.
Data Dependency Task Graphs

Buffer A
- Read Accessor
- Write Accessor
CG A

Buffer B
- Read Accessor
- Write Accessor
CG B

Buffer C
- Read Accessor
- Read Accessor
- Write Accessor
CG C

Buffer D
- Read Accessor
- Write Accessor

CG A

CG B

CG C
Example: Vector Add
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {

}
```
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {
    cl::sycl::buffer<T, 1> inputABuf(inA.data(), out.size());
    cl::sycl::buffer<T, 1> inputBBuf(inB.data(), out.size());
    cl::sycl::buffer<T, 1> outputBuf(out.data(), out.size());
}
```

The buffers synchronise upon destruction.
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {
    cl::sycl::buffer<T, 1> inputABuf(inA.data(), out.size());
    cl::sycl::buffer<T, 1> inputBBuf(inB.data(), out.size());
    cl::sycl::buffer<T, 1> outputBuf(out.data(), out.size());
    cl::sycl::queue defaultQueue;
}
```
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {
    cl::sycl::buffer<T, 1> inputABuf(inA.data(), out.size());
    cl::sycl::buffer<T, 1> inputBBuf(inB.data(), out.size());
    cl::sycl::buffer<T, 1> outputBuf(out.data(), out.size());
    cl::sycl::queue defaultQueue;
    defaultQueue.submit([&](cl::sycl::handler &cgh) { 
    
    });
}
```

Create a command group to define an asynchronous task
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {
    cl::sycl::buffer<T, 1> inputABuf(inA.data(), out.size());
    cl::sycl::buffer<T, 1> inputBBuf(inB.data(), out.size());
    cl::sycl::buffer<T, 1> outputBuf(out.data(), out.size());
    cl::sycl::queue defaultQueue;
    defaultQueue.submit([&](cl::sycl::handler &cgh) {
        auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
        auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
        auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);
    });
}
```
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T> kernel;

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {
    cl::sycl::buffer<T, 1> inputABuf(inA.data(), out.size());
    cl::sycl::buffer<T, 1> inputBBuf(inB.data(), out.size());
    cl::sycl::buffer<T, 1> outputBuf(out.data(), out.size());
    cl::sycl::queue defaultQueue;
    defaultQueue.submit([&] (cl::sycl::handler &cgh) {
        auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
        auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
        auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);
        cgh.parallel_for<kernel<T>>(cl::sycl::range<1>(out.size()),
        [=](cl::sycl::id<1> idx) {
```
You must provide a name for the lambda

Create a parallel_for to define the device code
Example: Vector Add

```cpp
#include <CL/sycl.hpp>

template <typename T> kernel;

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> &out) {
    cl::sycl::buffer<T, 1> inputABuf(inA.data(), out.size());
    cl::sycl::buffer<T, 1> inputBBuf(inB.data(), out.size());
    cl::sycl::buffer<T, 1> outputBuf(out.data(), out.size());
    cl::sycl::queue defaultQueue;
    defaultQueue.submit([&](cl::sycl::handler &cgh) {
        auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
        auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
        auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);
        cgh.parallel_for<kernel<T>>(cl::sycl::range<1>(out.size())),
        [=](cl::sycl::id<1> idx) {
            outputPtr[idx] = inputAPtr[idx] + inputBPtr[idx];
        });
    });
}
Example: Vector Add

template <typename T>
void parallel_add(std::vector<T> inA, std::vector<T> inB, std::vector<T> out);

int main() {
    std::vector<float> inputA = { /* input a */ };  
    std::vector<float> inputB = { /* input b */ };  
    std::vector<float> output = { /* output */ };  

    parallel_add(inputA, inputB, output);
    ...
}

The SYCL Ecosystem

C++ Application

C++ Template Library → SYCL for OpenCL → OpenCL

- CPU
- GPU
- APU
- Accelerator
- FPGA
- DSP
Benefits of Data Dependency Graphs

Allows you to describe your problems in terms of relationships
  Don’t need to enqueue explicit copies

Removes the need for complex event handling
  Dependencies between device functions are automatically constructed
  Allows the runtime to make data movement optimizations
  Preemptively copy data to a device before executing device functions

Avoid unnecessarily copying data back to the host after executing a device function
  Avoid copying data to a device that you don’t need
Convergence with ISO C++

Industry working to bring heterogeneous compute to standard ISO C++
C++17 Parallel STL hosted by Khronos
SYCL has lead to several ISO C++ proposals for enabling heterogeneous compute
Executors - for scheduling work
Data Movement in C++ - position paper on data movement for heterogeneous compute
“Managed pointers” or “channels” - for sharing data
Affinity support
Hoping to target C++ 20 but timescales are tight
What can I do with a Parallel For Each?

```cpp
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_policy,
            std::begin(v1), nElems, 1);

std::for_each(sycl_named_policy
               <class KernelName>,
               std::begin(v), std::end(v),
               [=](float f) { f * f + f });

Workload is distributed on the GPU cores

(mileage may vary, implementation-specific behaviour)
```
What can I do with a Parallel For Each?

```cpp
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_heter_policy(cpu, gpu, 0.5),
            std::begin(v1), nElems, 1);

std::for_each(sycl_heter_policy<class kName>(cpu, gpu, 0.5),
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

Workload is distributed on all cores!

(mileage may vary, implementation-specific behaviour)
Parallel STL

Open-source implementation of the ISO C++ parallelism TS (N4505)
https://github.com/KhronosGroup/SyclParallelSTL

```cpp
std::vector<int> v = {...};

sycl::sycl_execution_policy<class sort_kernel> policy1;
sort(policy1, v.begin(), v.end());

sycl::sycl_execution_policy<class negate_kernel> policy2;
transform(policy2, v.begin(), v.end(), v.begin(), std::negate<int>());
```
### Parallel overloads available in SYCL Parallel STL

<table>
<thead>
<tr>
<th>Function</th>
<th>Parallel Overload Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacent_difference</td>
<td>☑️</td>
</tr>
<tr>
<td>copy</td>
<td>☑️</td>
</tr>
<tr>
<td>count_if</td>
<td>☑️</td>
</tr>
<tr>
<td>fill_n</td>
<td></td>
</tr>
<tr>
<td>find_if</td>
<td>☑️</td>
</tr>
<tr>
<td>generate</td>
<td></td>
</tr>
<tr>
<td>inner_product</td>
<td>☑️</td>
</tr>
<tr>
<td>is_partitioned</td>
<td></td>
</tr>
<tr>
<td>max_element</td>
<td></td>
</tr>
<tr>
<td>mismatch</td>
<td></td>
</tr>
<tr>
<td>partial_sort</td>
<td></td>
</tr>
<tr>
<td>reduce</td>
<td>☑️</td>
</tr>
<tr>
<td>remove_if</td>
<td></td>
</tr>
<tr>
<td>replace_if</td>
<td></td>
</tr>
<tr>
<td>rotate_copy</td>
<td></td>
</tr>
<tr>
<td>set_intersection</td>
<td></td>
</tr>
<tr>
<td>stable_partition</td>
<td></td>
</tr>
<tr>
<td>transform</td>
<td></td>
</tr>
<tr>
<td>exclusive_scan</td>
<td>☑️</td>
</tr>
<tr>
<td>uninitialized_copy</td>
<td></td>
</tr>
<tr>
<td>unique_copy</td>
<td></td>
</tr>
<tr>
<td>adjacent_find</td>
<td>☑️</td>
</tr>
<tr>
<td>copy_if</td>
<td>☑️</td>
</tr>
<tr>
<td>equal</td>
<td></td>
</tr>
<tr>
<td>find</td>
<td>☑️</td>
</tr>
<tr>
<td>find_if_not</td>
<td>☑️</td>
</tr>
<tr>
<td>generate_n</td>
<td></td>
</tr>
<tr>
<td>inplace_merge</td>
<td></td>
</tr>
<tr>
<td>is_sorted</td>
<td></td>
</tr>
<tr>
<td>merge</td>
<td></td>
</tr>
<tr>
<td>move</td>
<td></td>
</tr>
<tr>
<td>partial_sort_copy</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
</tr>
<tr>
<td>replace</td>
<td></td>
</tr>
<tr>
<td>reverse</td>
<td></td>
</tr>
<tr>
<td>search</td>
<td></td>
</tr>
<tr>
<td>set_symmetric_difference</td>
<td></td>
</tr>
<tr>
<td>stable_sort</td>
<td></td>
</tr>
<tr>
<td>transform</td>
<td></td>
</tr>
<tr>
<td>inclusive_scan</td>
<td>☑️</td>
</tr>
<tr>
<td>uninitialized_fill</td>
<td></td>
</tr>
<tr>
<td>uninitialized_copy</td>
<td></td>
</tr>
<tr>
<td>all_of</td>
<td></td>
</tr>
<tr>
<td>copy_n</td>
<td></td>
</tr>
<tr>
<td>exclusive_scan</td>
<td>☑️</td>
</tr>
<tr>
<td>find_end</td>
<td></td>
</tr>
<tr>
<td>for_each</td>
<td>☑️</td>
</tr>
<tr>
<td>includes</td>
<td></td>
</tr>
<tr>
<td>is_heap</td>
<td></td>
</tr>
<tr>
<td>is_sorted_until</td>
<td></td>
</tr>
<tr>
<td>min_element</td>
<td></td>
</tr>
<tr>
<td>none_of</td>
<td></td>
</tr>
<tr>
<td>partition</td>
<td></td>
</tr>
<tr>
<td>remove_copy</td>
<td></td>
</tr>
<tr>
<td>replace_copy</td>
<td></td>
</tr>
<tr>
<td>reverse_copy</td>
<td></td>
</tr>
<tr>
<td>search_n</td>
<td></td>
</tr>
<tr>
<td>set_difference</td>
<td></td>
</tr>
<tr>
<td>sort</td>
<td>☑️</td>
</tr>
<tr>
<td>transform</td>
<td>☑️</td>
</tr>
<tr>
<td>uninitialized_copy</td>
<td></td>
</tr>
<tr>
<td>uninitialized_fill</td>
<td></td>
</tr>
</tbody>
</table>
### Demo Results - Running std::sort
(Running on Intel i7 6600 CPU & Intel HD Graphics 520)

<table>
<thead>
<tr>
<th>size</th>
<th>$2^{16}$</th>
<th>$2^{17}$</th>
<th>$2^{18}$</th>
<th>$2^{19}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>std::seq</td>
<td>0.27031s</td>
<td>0.620068s</td>
<td>0.669628s</td>
<td>1.48918s</td>
</tr>
<tr>
<td>std::par</td>
<td>0.259486s</td>
<td>0.478032s</td>
<td>0.444422s</td>
<td>1.83599s</td>
</tr>
<tr>
<td>std::unseq</td>
<td>0.24258s</td>
<td>0.413909s</td>
<td>0.456224s</td>
<td>1.01958s</td>
</tr>
<tr>
<td>sycl_execution_policy</td>
<td>0.273724s</td>
<td>0.269804s</td>
<td>0.277747s</td>
<td>0.399634s</td>
</tr>
</tbody>
</table>
Eigen Linear Algebra Library

SYCL backend in mainline
Focused on Tensor support, providing support for machine learning/CNNs
Equivalent coverage to CUDA
Working on optimization for various hardware architectures (CPU, desktop and mobile GPUs)

https://bitbucket.org/eigen/eigen/
TensorFlow

SYCL backend support for all major CNN operations
Complete coverage for major image recognition networks
  GoogLeNet, Inception-v2, Inception-v3, ResNet, ....
Ongoing work to reach 100% operator coverage and optimization for various hardware architectures (CPU, desktop and mobile GPUs)

https://github.com/tensorflow/tensorflow
SYCL Ecosystem

• Single-source heterogeneous programming using STANDARD C++
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL

• Fast and powerful path for bring C++ apps and libraries to OpenCL
  - C++ Kernel Fusion - better performance on complex software than hand-coding
  - Halide, Eigen, Boost.Compute, SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL GTX
  - triSYCL, ComputeCpp, VisionCpp, ComputeCpp SDK ...

• More information at http://sycl.tech

Developer Choice
The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language
Low Level Control
‘GPGPU’-style separation of device-side kernel source code and host code

Single-source C++
Programmer Familiarity
Approach also taken by C++ AMP and OpenMP
Vision and Neural Net Acceleration Challenge

Neural Net Training Frameworks

Vision/Al Applications

Vision and Neural Net Inferencing Runtime

Trained Networks

Desktop and Cloud Hardware

Embedded/Mobile/Desktop Vision/Inferencing Hardware
OpenCL and Vulkan

OpenCL 1.2
OpenCL C Kernel Language

2011

OpenCL for DSPs
- Embedded imaging, vision and inferencing
- Flexible reduced precision
- Conformance without IEEE 32 Floating Point
- Explicit DMA

OpenCL 2.1
SPIR-V in Core

2015

SYCL 1.2
C++11 Single source programming

OpenCL 2.2
C++ Kernel Language

2017

SYCL 2.2
C++14 Single source programming

Industry working to bring Heterogeneous compute to standard ISO C++
C++17 Parallel STL hosted by Khronos
Executors - for scheduling work
“Managed pointers” or “channels” - for sharing data

Help bring OpenCL-class compute to Vulkan
Safety Critical APIs

New Generation APIs for safety certifiable vision, graphics and compute e.g. ISO 26262 and DO-178B/C

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL ES 2.0 - 2007
Shader programmable pipeline

OpenGL SC 1.0 - 2005
Fixed function graphics subset

OpenGL SC 2.0 - April 2016
Shader programmable pipeline subset

OpenVX SC 1.1 Released 1st May 2017
Restricted “deployment” implementation executes on the target hardware by reading the binary format and executing the pre-compiled graphs

Khronos SCAP
'Safety Critical Advisory Panel'
Guidelines for designing APIs that ease system certification.
Open to Khronos member AND industry experts

OpenCL SC TSG Formed
Working on OpenCL SC 1.2
Eliminate Undefined Behavior
Eliminate Callback Functions
Static Pool of Event Objects

Experience and Guidelines
C++ Std Timeline/status
https://isocpp.org/std/status
Conclusion:
Future Heterogeneous/Distributed directions in C++

- C++ will be the convergence for many Heterogeneous Programming Standards
- ISO C++ driving towards a future TS in Heterogeneous and Safety/Security
  - Executors enables multiple resources
  - Asynchronous Algorithm enables latency hiding
  - Context
  - Affinity before inaccessible memory
  - Data movement to access inaccessible memory
  - Exception handling in a concurrent environment
Will the two galaxies ever join?
Bof on Wednesday 12:15-1:15 Room 405-406-407 led by Hal Finkel, ANL

Distributed & Heterogeneous Programming in C++ (DHPC++) for HPC

Ben Sanders (AMD, HCC/HiP, HSA)
Carter Edwards (SNL, Kokkos, C++)
CJ Newburn (Nvidia, HiHat, Agency, CUDA)
David Beckinsale (LLNL, Raja)
Hartmut Kaiser (LSU, HPX)
Michael Wong (Codeplay, ISOcpp, Khronos, C/C++, SYCL)
Ronan Keryell (Xilinx, Khronos, SYCL)
Xinmin Tian (Intel/Altera, OpenMP C++)

» goo.gl/y5HQQH (slide)
» goo.gl/yh1ytV (Google sheet of questions)
SYCL Implementations and ecosystem

Implementations

ComputeCpp by Codeplay
https://www.codeplay.com/products/computecpp

triSYCL https://www.github.com/triSYCL/triSYCL

Sycl-gtx https://www.github.com/ProGTX/sycl-gtx

Ecosystem

SYCL ParallelSTL
SYCLBLAS
TensorFlow both SYCL and triSYCL
ParallelSTL with Ranges
For Codeplay, these are our layer choices

We have chosen a layer of standards, based on current market adoption:
- TensorFlow and OpenCV
- SYCL
- OpenCL (with SPIR)
- LLVM as the standard compiler back-end

The actual choice of standards may change based on market dynamics, but by choosing widely adopted standards and a layering approach, it is easy to adapt.
For Codeplay, these are our products

Device-specific programming
- LLVM

Higher-level language enabler
- OpenCL SPIR

C/C++-level programming
- SYCL

Graph programming
- TensorFlow
- OpenCV
Community Edition
Available now for free!

Visit:
computecpp.codeplay.com
Open source SYCL projects:

- ComputeCpp SDK - Collection of sample code and integration tools
- SYCL ParallelSTL - SYCL based implementation of the parallel algorithms
- VisionCpp - Compile-time embedded DSL for image processing
- Eigen C++ Template Library - Compile-time library for machine learning

All of this and more at: http://sycl.tech
triSYCL

- Open Source SYCL 1.2/2.2
- Uses C++17 templated classes
- Used by Khronos to define the SYCL and OpenCL C++ standard
  - Languages are now too complex to be defined without implementing...
- On-going implementation started at AMD and now led by Xilinx
- https://github.com/triSYCL/triSYCL
- OpenMP for host parallelism
- Boost.Compute for OpenCL interaction
- Prototype of device compiler for Xilinx FPGA
```cpp
#include <CL/sycl.hpp>

q.submit([&](auto &cgh) {
    // The kernel write a, so get a write accessor on it
    auto A = a.get_access<access::mode::write>(cgh);
    // Enqueue parallel kernel on a N*M 2D iteration space
    cgh.parallel_for<class init_a>({ N, M },
    [=] (auto index) {
        A[index] = index[0]*2 + index[1];
    });
});
```
Zynq UltraScale+ MPSoC overview: All Programmable...
#include <CL/sycl.hpp>
#include <iostream>
#include <boost/test/minimal.hpp>
using namespace cl::sycl;

constexpr size_t N = 300;
using Type = int;

int test_main(int argc, char *argv[]) {
  buffer<Type> a {N};
  buffer<Type> b {N};
  buffer<Type> c {N};

  { auto a_b = b.get_access<access::mode::discard_write>();
    // Initialize buffer with increasing numbers starting at 0
    std::iota(a_b.begin(), a_b.end(), 0);
  }

  { auto a_c = c.get_access<access::mode::discard_write>();
    // Initialize buffer with increasing numbers starting at 5
    std::iota(a_c.begin(), a_c.end(), 5);
  }

  queue q { default_selector {});

  // Launch a kernel to do the summation
  q.submit([&](handler &cgh) {
    auto a_a = a.get_access<access::mode::discard_write>(cgh);
    auto a_b = b.get_access<access::mode::read>(cgh);
    auto a_c = c.get_access<access::mode::read>(cgh);

    // A typical FPGA-style pipelined kernel
    cgh.single_task<class add>({=, 
      d_a = drt::accessor<decltype(a_a)> { a_a },
      d_b = drt::accessor<decltype(a_b)> { a_b },
      d_c = drt::accessor<decltype(a_c)> { a_c } {
        for (unsigned int i = 0 ; i < N; ++i)
          d_a[i] = d_b[i] + d_c[i];
    });
  });

  // Verify the result
  auto a_a = a.get_access<access::mode::read>();
  for (unsigned int i = 0 ; i < a.get_count(); ++i)
    BOOST_CHECK(a_a[i] == 5 + 2*i);

  return 0;
}
SPIR 2.0 “de facto” output in Clang 3.9.1

using: ModuleID = "device_compiler/single_task_vector_add_drt.kernel.bc"
source_filename = "device_compiler/single_task_vector_add_drt.cpp"
target triple = "spir64"

declare i32 @gxx_personality_v0(...) ; Function Attrs: nocinline norecurse nounwind uwtable
define spir_kernel void @TRISYCL_kernel_0(i32 %arrayidx.i13.i inbounds i32 i32 %arrayaddr.i13.i) spir64 attribute #0 = { nocinline norecurse nounwind uwtable "disable-tail-calls"="false" "less-precise-fp-math"="false" "no-frame-pointer-elt"="false" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "stack-protector-use-it"="true" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

entry: br label %for.body.i

for.body.i: ; preds = %for.body.i, %entry
%indvars.iv.i = phi i64 [ 0, %entry ], [ %indvars.iv.next.i, %for.body.i ]
%arrayidx.i11.i = getelementptr inbounds i32, i32 %arrayaddr.i11.i, align 4, !tb aa !7
%arrayidx.i11.i = getelementptr inbounds i32, i32 %arrayaddr.i11.i, align 4, !tb aa !7
%add.i = add nsw i32 %i1, %0
%arrayidx.i13.i = getelementptr inbounds i32, i32 %arrayaddr.i13.i, align 4, !tb aa !7
store i32 %add.i, i32 %arrayidx.i13.i, align 4, !tb aa !7
%indvars.iv.next.i = add nsw i64 %indvars.iv.i, 1
%exitcond.i = icmp eq i64 %indvars.iv.next.i, 300
br i1 %exitcond.i, label %"_ZZZ9test_mainIPcENK35_lclERN2c14sycl7handlerEENKU1vE_clEv.exit"!

"_ZZZ9test_mainIPcENK35_lclERN2c14sycl7handlerEENKU1vE_clEv.exit": preds = %for.body.i
ret void

attributes #0 = { nocinline norecurse nounwind uwtable "disable-tail-calls"="false" "less-precise-fp-math"="false" "no-frame-pointer-elt"="false" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "stack-protector-use-it"="true" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!!llvm.ident = !%(0)
!opencl.spir.version = !%(1)
!opencl.ocl.version = !%(2)

!0 = !%("clang version 3.9.1 "
!1 = !%(132 2, 132 0)
!2 = !%(132 1, 132 2)
!3 = !%(132 1, 132 1, 132 0)
!4 = !%("int "", "int "", "int "")
!5 = !%("", ", ", ")
!6 = !%("read_write", "read_write", "read_write")
!7 = !%(18, 16, i64 0)
!8 = !%("int", 18, i64 0)
!9 = !%("omnipotent char", 18, i64 0)
!10 = !%("Simple C++ TBAA")
After Xilinx SDx 2017.2 xocc ingestion...
After Xilinx SDx 2017.2 xocc ingestion...
FPGA layout
Code execution on real FPGA

rkeryell@xirjoant40:~/Xilinx/Projects/OpenCL/SYCL/triSYCL/branch/device/tests (device)$
device_compiler/single_task_vector_add_drt.kernel_caller
binary_size = 5978794
task::add_prelude
task::add_prelude
task::add_prelude
accessor(Accessor &a) : &a = 0x7ffd39395f40
 &buffer =0x7ffd39395f50
accessor(Accessor &a) : &a = 0x7ffd39395f30
 &buffer =0x7ffd39395f60
accessor(Accessor &a) : &a = 0x7ffd39395f20
 &buffer =0x7ffd39395f70
single_task &f = 0x7ffd39395f50
task::prelude
schedule_kernel &k = 0x1516060
Setting up
_ZN2cl4sycl6detail18instantiate_kernelIZZ9test_mainiPPcENK3$_1clERNS0_7handlerEE3addZZ9test_mainiS4_ENKS5_clES7_EU1vE_EEvT0_
aka TRISYCL_kernel_0
Name device xilinx_adm-pcie-7v3_lddr_3_0
serialize_accessor_arg index =0, size = 4, arg = 0
serialize_accessor_arg index =1, size = 4, arg = 0x1
serialize_accessor_arg index =2, size = 4, arg = 0x2
**** no errors detected
Any Questions?

- Khronos working on a comprehensive set of solutions for vision and inferencing
  - Layered ecosystem that include multiple developer and deployment options
- These slides and further information on all these standards at Khronos
  - www.khronos.org
- Please let us know if we can help you participate in any Khronos activities!
  - You are very welcome - and we appreciate your input!
- Please contact us with any comments or questions!
  - Michael Wong | michael@codeplay.com | wongmichael.com