Overview of SYCL/ComputeCPP
Heterogeneous C++ Language

Peter Žužek, Staff Software Engineer, SYCL
peter@codeplay.com

Khronos Booth, Supercomputing 2017
Agenda

• SYCL
• ComputeCpp
• Example SYCL code
• SYCL Ecosystem
Agenda

• SYCL
• ComputeCpp
• Example SYCL code
• SYCL Ecosystem
• Khronos Open Standard for Compute
• Single source programming
• Completely standard C++11
• SYCL 1.2 builds on top of OpenCL 1.2
• Abstracts low level details
• http://sycl.tech/
SYCL Implementations

• triSYCL
  - github.com/triSYCL/triSYCL
• sycl-gtx
  - github.com/ProGTX/sycl-gtx
• Xilinx SYCL for FPGA
• ComputeCpp
  - codeplay.com/products/computesuite/computecpp
Host and Device

• Can run on OpenCL devices in the system
• Also provides a host device
  – Option to still run kernel code if no OpenCL device available
  – Allows for kernel debugging

![Diagram of SYCL Runtime, OpenCL, Host, Device X, Device Y, Device Z connections]
Kernel restrictions

• OpenCL 1.2 device restrictions still apply to kernels:
  - No exception handling
  - No recursion
  - No RTTI
  - No dynamic allocation
  - No function pointers
Runtime Dependency Handling

- Data on SYCL device represented by a buffer
  - For use in kernels need to request access
- Kernel submission is asynchronous
  - Execution scheduled according to data requirements
  - Blocks when data synchronized with host

```cpp
image<2> imageA(inputDataA, ...)
image<2> imageB(inputDataB, ...)
queue gpuQ; queue dspQ;
dspQ.submit(filterA(imageA));
gpuQ.submit(filterB(imageB));
dspQ.submit(combine(imageA, imageB));
imageA.
get_access<access::mode::host>();
```
Resilience to failure

- Exception handling
  - OpenCL error thrown as exceptions
- Asynchronous kernel errors handled by user functions
- Optional fallback queue
  - Used in case of error
  - All dependencies automatically adjusted
Agenda

• SYCL
• ComputeCpp
• Example SYCL code
• SYCL Ecosystem
• Codeplay’s implementation of SYCL
• Two main parts:
  − Device compiler based on LLVM/Clang
  − ComputeCpp runtime is wholly-owned Codeplay IP
• Compiles kernels into SPIR
  − Experimental support for SPIR-V and PTX backends
ComputeCpp variants

• Community Edition
  - Free to download from our website
  - Helps with expanding the SYCL ecosystem

• Custom builds
  - Hardware vendors
  - Renesas R-Car
Agenda

• SYCL
• ComputeCpp
• Example SYCL code
• SYCL Ecosystem
Vector addition in SYCL

```cpp
template <typename T>
vector<T> add_vectors(const vector<T>& a, const vector<T>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});
    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;
    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });

    return c;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a,
                        const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});

    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });

        return c;
    });
}

int main() {
    vector<int> a{1, 2, 3, 4, 5};
    vector<int> b{6, 7, 8, 9, 10};
    auto c = add_vectors(a, b);
    return 0;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a,
                         const vector<int>& b) {

    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});

    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });

    return c;
}

int main() {
    vector<int> a{1, 2, 3, 4, 5};
    vector<int> b{6, 7, 8, 9, 10};
    auto c = add_vectors(a, b);
    return 0;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a, const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});

    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });

    return c;
}
```

int main() {
    vector<int> a{1, 2, 3, 4, 5};
    vector<int> b{6, 7, 8, 9, 10};
    auto c = add_vectors(a, b);
    return 0;
}
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a, 
const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});

    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });

    return c;
}

int main() {
    vector<int> a{1, 2, 3, 4, 5};
    vector<int> b{6, 7, 8, 9, 10};
    auto c = add_vectors(a, b);
    return 0;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a, const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});

    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });

    return c;
}

int main() {
    vector<int> a{1, 2, 3, 4, 5};
    vector<int> b{6, 7, 8, 9, 10};
    auto c = add_vectors(a, b);
    return 0;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a, const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});

    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});

    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });
    return c;
}

int main() {
    vector<int> a{1, 2, 3, 4, 5};
    vector<int> b{6, 7, 8, 9, 10};
    auto c = add_vectors(a, b);
    return 0;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a, const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});
    buffer<int, 1> bufC(c.data(), range<1>{N});

    int main() {
        vector<int> a{1, 2, 3, 4, 5};
        vector<int> b{6, 7, 8, 9, 10};
        auto c = add_vectors(a, b);
        return 0;
    }

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
    });

    return c;
}
```
Vector addition in SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

#include <vector>
using std::vector;

vector<int> add_vectors(const vector<int>& a, const vector<int>& b) {
    const auto N = a.size();
    buffer<int, 1> bufA(a.data(), range<1>{N});
    buffer<int, 1> bufB(b.data(), range<1>{N});
    vector<int> c(N);
    buffer<int, 1> bufC(c.data(), range<1>{N});
    queue myQueue;

    myQueue.submit([&](handler& cgh) {
        auto A = bufA.get_access<access::mode::read>(cgh);
        auto B = bufB.get_access<access::mode::read>(cgh);
        auto C = bufC.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class add>(
            range<1>{N},
            [=](id<1> i) {
                C[i] = A[i] + B[i];
            }
        );
    });

    return c;
}
```
Agenda

• SYCL
• ComputeCpp
• Example SYCL code
• SYCL Ecosystem
ComputeCpp SDK

• Samples on how to use SYCL
• https://github.com/codeplaysoftware/computeccpp-sdk
Parallel STL

• Implementation of the C++17 parallel algorithms
• SYCL execution policy
• https://github.com/KhronosGroup/SyclParallelSTL

```cpp
std::for_each(sycl_policy, std::begin(v), std::end(v),
    [=](int& elem) {
        elem *= 2;
    });
```
Linear Algebra libraries

- Eigen
  - High level C++ library of template headers
  - https://github.com/ville-k/sycl_starter

- SYCL BLAS
  - Implementation of BLAS routines using SYCL
  - https://github.com/codeplaysoftware/sycl-blas
Tensorflow

- Open Source Machine Learning library from Google
- Simpler to port to SYCL than OpenCL due to C++ approach
- SYCL port part of the main repository
- https://github.com/tensorflow/tensorflow