Dive into OpenCL C++

Bartosz Sochacki
Adam Stański

Agenda

- I know OpenCL C and C++, how much do I have to learn?
- Porting OpenCL C code to OpenCL C++
- Writing generic code with OpenCL C++
- Value added

I know OpenCL C and C++, how much do I have to learn?

- C++14 based language with restrictions
- Implements all OpenCL C functionality + new extensions
- OpenCL C++ vs. C++
- OpenCL C vs. OpenCL C++

OpenCL C++ vs. C++

Similarities:

- OpenCL C++ is static subset of C++14 language
- Subset of C++14 standard library: type traits, iterators, array, tuples, …

Differences:

- Restrictions
- New attributes
- Built-in half and vector types
- OpenCL specific types and libraries
- namespace cl

OpenCL C++ vs. OpenCL C

Similarities:

- Special types: `images`, `pipes`, `events`, `device_queue`, …
- Built-in half and vector types
- `kernel` attribute
- Similar attributes
- Similar restrictions
- Performance

Differences:

- C++14 based
- Facelifted special OpenCL types
- All headers must be included
- Address space library
- Device enqueue uses lambdas (no blocks support)
- New functionality

Q: I know OpenCL C and C++, how much do I have to learn?

A: If you know both languages, OpenCL C++ will be easy to learn

Porting OpenCL C code to OpenCL C++

- **Keep in mind:**
  - All types and functions are in namespace cl
  - Address space pointers and storage classes
  - New C++ interface of OpenCL special types: `image`*, `sampler`, `pipe` …
  - OpenCL C `convert_*` built-in functions were replaced by one `convert_cast<>` template function
  - OpenCL C `as_type` built-in functions were replaced by on `as_type<>` template function

How to port existing OpenCL C code to OpenCL C++: Example 1: address space pointers

OpenCL C

```
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  out[tid] = in[tid];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>

kernel void myKernel(cl::global_ptr<int[]> in,
                     cl::global_ptr<int[]> out)
{
  size_t tid = cl::get_global_id(0);
  out[tid] = in[tid];
}
```

How to port existing OpenCL C code to OpenCL C++: Example 1: address space pointers

OpenCL C

```
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  out[tid] = in[tid];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>
```

```
kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  out[tid] = in[tid];
}
```

How to port existing OpenCL C code to OpenCL C++: Example 1: address space pointers

OpenCL C

```
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  out[tid] = in[tid];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>

kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  out[tid] = in[tid];
}
```

How to port existing OpenCL C code to OpenCL C++: Example 1: address space pointers

OpenCL C

```
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  out[tid] = in[tid];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>

kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  out[tid] = in[tid];
}
```

Summary

- Headers
- Address space classes
- Namespace cl

How to port existing OpenCL C code to OpenCL C++: Example 2: address space storage classes

OpenCL C

```
__attribute__((reqd_work_group_size(256, 1, 1)))
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  size_t lid = get_local_id(0);
  local size_t arr[256];

  arr[lid] = in[tid];
  work_group_barrier(CLK_LOCAL_MEM_FENCE);
  out[tid] = arr[get_local_size(0) - lid - 1];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_synchronization>
#include <opencl_memory>

[[cl::required_work_group_size(256, 1, 1)]]
kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  size_t lid = cl::get_local_id(0);
  cl::local<size_t[256]> arr;

  arr[lid] = in[tid];
  cl::work_group_barrier(cl::mem_fence::local);
  out[tid] = arr[cl::get_local_size(0) - lid - 1];
}
```

How to port existing OpenCL C code to OpenCL C++: Example 2: address space storage classes

OpenCL C

```
__attribute__((reqd_work_group_size(256, 1, 1)))
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  size_t lid = get_local_id(0);
  local size_t arr[256];

  arr[lid] = in[tid];
  work_group_barrier(CLK_LOCAL_MEM_FENCE);
  out[tid] = arr[get_local_size(0) - lid - 1];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_synchronization>
#include <opencl_memory>

[[cl::required_work_group_size(256, 1, 1)]]
kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  size_t lid = cl::get_local_id(0);
  cl::local<size_t[256]> arr;

  arr[lid] = in[tid];
  cl::work_group_barrier(cl::mem_fence::local);
  out[tid] = arr[cl::get_local_size(0) - lid - 1];
}
```

How to port existing OpenCL C code to OpenCL C++: Example 2: address space storage classes

OpenCL C

```
__attribute__((reqd_work_group_size(256, 1, 1)))
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  size_t lid = get_local_id(0);
  local size_t arr[256];

  arr[lid] = in[tid];
  work_group_barrier(CLK_LOCAL_MEM_FENCE);
  out[tid] = arr[get_local_size(0) - lid - 1];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_synchronization>
#include <opencl_memory>

[[cl::required_work_group_size(256, 1, 1)]]
kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  size_t lid = cl::get_local_id(0);
  cl::local<size_t[256]> arr;

  arr[lid] = in[tid];
  cl::work_group_barrier(cl::mem_fence::local);
  out[tid] = arr[cl::get_local_size(0) - lid - 1];
}
```

How to port existing OpenCL C code to OpenCL C++: Example 2: address space storage classes

OpenCL C

```
__attribute__((reqd_work_group_size(256, 1, 1)))
kernel void myKernel(global int *in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  size_t lid = get_local_id(0);
  local size_t arr[256];

  arr[lid] = in[tid];
  work_group_barrier(CLK_LOCAL_MEM_FENCE);
  out[tid] = arr[get_local_size(0) - lid - 1];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_synchronization>
#include <opencl_memory>

[[cl::required_work_group_size(256, 1, 1)]]
kernel void myKernel(cl::global_ptr<int[ ]> in,
                     cl::global_ptr<int[ ]> out)
{
  size_t tid = cl::get_global_id(0);
  size_t lid = cl::get_local_id(0);
  cl::local<size_t[256]> arr;

  arr[lid] = in[tid];
  cl::work_group_barrier(cl::mem_fence::local);
  out[tid] = arr[cl::get_local_size(0) - lid - 1];
}
```

Summary

- Attributes
- Address space storage
- Enumerables

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                    read_only image2d_t img,
                    sampler_t s,
                    global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
              image2d<float4, image_access::sample> img,
              sampler s,
              global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                                      read_only image2d_t img,
                                      sampler_t s,
                                      global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                          image2d<float4, image_access::sample> img,
                          sampler s,
                          global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                          read_only image2d_t img,
                          sampler_t s,
                          global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                    image2d<float4, image_access::sample> img,
                    sampler s,
                    global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                        read_only image2d_t img,
                        sampler_t s,
                        global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                 image2d<float4, image_access::sample> img,
                 sampler s,
                 global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                        read_only image2d_t img,
                        sampler_t s,
                        global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                  image2d<float4, image_access::sample> img,
                  sampler s,
                  global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                              read_only image2d_t img,
                              sampler_t s,
                              global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                    image2d<float4, image_access::sample> img,
                    sampler s,
                    global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                      read_only image2d_t img,
                      sampler_t s,
                      global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                image2d<float4, image_access::sample> img,
                sampler s,
                global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

How to port existing OpenCL C code to OpenCL C++: Example 3: images

OpenCL C

```
kernel void myKernel(
                                read_only image2d_t img,
                                sampler_t s,
                                global float4 *out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords = (int2)(tidX, tidY);
  int w = get_image_width(img);
  out[w * tidY + tidX] = read_imagef(img, s, coords);
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_image>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(
                      image2d<float4, image_access::sample> img,
                      sampler s,
                      global_ptr<float4[ ]> out)
{
  size_t tidX = get_global_id(0), tidY = get_global_id(1);
  int2 coords(tidX, tidY);
  int w = img.width();
  out[w * tidY + tidX] = img.sample(s, coords);
}
```

Summary

- Type safe special types
- Vector initialization
- Methods

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
     read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                     global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
     r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                                global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
     read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                              global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
     r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                                        global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
     read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                                        global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
     r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
    read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                     global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
    r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
     read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                     global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
    r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
    read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                     global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
    r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

How to port existing OpenCL C code to OpenCL C++: Example 4: pipes

OpenCL C

```
kernel void myKernel(read_only pipe int pipe_in,
                     global int *out)
{
  size_t tid = get_global_id(0);
  reserve_id rid = reserve_read_pipe(pipe_in, 1);
  if(is_valid_reserve_id(rid) &&
     read_pipe(pipe_in, rid, 0, &out[tid]))
  {
    commit_read_pipe(pipe_in, rid);
  }
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_pipe>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(pipe<int, pipe_access::read> pipe_in,
                     global_ptr<int[]> out)
{
  size_t tid = get_global_id(0);
  auto r = pipe_in.reserve(1);
  if(r &&
     r.read(0, out[tid]))
  {
    r.commit();
  }
}
```

Summary

- Objects instead of keywords
- Smart reserve id

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
  enqueue_kernel(queue,
                 CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                 ndrange_1d(1),
                 ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
  queue.enqueue_kernel(
                       enqueue_policy::wait_kernel,
                       ndrange(1),
                       [=]( ) { *a = *b + *c; });
}
```

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
  enqueue_kernel(queue,
                 CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                 ndrange_1d(1),
                 ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
  queue.enqueue_kernel(
                 enqueue_policy::wait_kernel,
                 ndrange(1),
                 [=]() { *a = *b + *c; });
}
```

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
  enqueue_kernel(queue,
                 CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                 ndrange_1d(1),
                 ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
  queue.enqueue_kernel(
                 enqueue_policy::wait_kernel,
                 ndrange(1),
                 [=]() { *a = *b + *c; });
}
```

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
  enqueue_kernel(queue,
                 CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                 ndrange_1d(1),
                 ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
  queue.enqueue_kernel(
                       enqueue_policy::wait_kernel,
                       ndrange(1),
                       [=]() { *a = *b + *c; });
}
```

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
   enqueue_kernel(queue,
                  CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                  ndrange_1d(1),
                  ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
   queue.enqueue_kernel(
                  enqueue_policy::wait_kernel,
                  ndrange(1),
                  [=]() { *a = *b + *c; });
}
```

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
    enqueue_kernel(queue,
                   CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                   ndrange_1d(1),
                   ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
    queue.enqueue_kernel(
                   enqueue_policy::wait_kernel,
                   ndrange(1),
                   [=]() { *a = *b + *c; });
}
```

How to port existing OpenCL C code to OpenCL C++: Example 5: device enqueue

OpenCL C

```
kernel void myKernel(queue_t queue,
                     global int *a,
                     global int *b,
                     global int *c)
{
  enqueue_kernel(queue,
                 CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                 ndrange_1d(1),
                 ^{ *c = *a + *b; });
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_device_queue>
#include <opencl_memory>
using namespace cl;

kernel void myKernel(device_queue queue,
                     global_ptr<int> a,
                     global_ptr<int> b,
                     global_ptr<int> c)
{
  queue.enqueue_kernel(
                 enqueue_policy::wait_kernel,
                 ndrange(1),
                 [=]() { *a = *b + *c; });
}
```

Summary

- Compile time validation
- Lambdas instead of blocks

Conversion from OpenCL C to OpenCL C++ is straightforward

Writing generic code with OpenCL C++ Example: generic image operations

Writing generic code with OpenCL C++
Example: generic image operations

```
kernel void myKernel1(
    image2d<float4, image_access::read> in1,
    image2d<float4, image_access::read> in2,
    image2d<float4, image_access::write> out)
{
  filter_image(in1, in2, out, blend);
}
```

```
kernel void myKernel2(
     image3d<int4, image_access::read> in1,
     image3d<int4, image_access::read> in2,
     image3d<int4, image_access::write> out)
{
  filter_image(in1, in2, out, blend);
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
kernel void myKernel1(
    image2d<float4, image_access::read> in1,
    image2d<float4, image_access::read> in2,
    image2d<float4, image_access::write> out)
{
  filter_image(in1, in2, out, blend);
}
```

```
kernel void myKernel2(
    image3d<int4, image_access::read> in1,
    image3d<int4, image_access::read> in2,
    image3d<int4, image_access::write> out)
{
  filter_image(in1, in2, out, blend);
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
auto blend = [](auto val1, auto val2) { return (val1 + val2) / 2; };

template<class T, class U, class F>
void filter_image(T in1, T in2, U out, F filter)
{
  auto pos = coords<typename T::integer_coord>();
  auto val1 = in1.read(pos);
  auto val2 = in2.read(pos);
  out.write(pos, filter(val1, val2));
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
auto blend = [](auto val1, auto val2) { return (val1 + val2) / 2; };

template<class T, class U, class F>
void filter_image(T in1, T in2, U out, F filter)
{
  auto pos = coords<typename T::integer_coord>();
  auto val1 = in1.read(pos);
  auto val2 = in2.read(pos);
  out.write(pos, filter(val1, val2));
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
auto blend = [](auto val1, auto val2) { return (val1 + val2) / 2; };

template<class T, class U, class F>
void filter_image(T in1, T in2, U out, F filter)
{
  auto pos = coords<typename T::integer_coord>();
  auto val1 = in1.read(pos);
  auto val2 = in2.read(pos);
  out.write(pos, filter(val1, val2));
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
auto blend = [](auto val1, auto val2) { return (val1 + val2) / 2; };

template<class T, class U, class F>
void filter_image(T in1, T in2, U out, F filter)
{
  auto pos = coords<typename T::integer_coord>();
  auto val1 = in1.read(pos);
  auto val2 = in2.read(pos);
  out.write(pos, filter(val1, val2));
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
template<class T>
T coords() { return get_global_id(0); }

template<>
int2 coords<int2>() { return int2(get_global_id(0), get_global_id(1)); }

template<>
int3 coords<int3>() { return int3(get_global_id(0), get_global_id(1), get_global_id(2) ); }
```

Writing generic code with OpenCL C++ Example: generic image operations

```
template<class T>
T coords() { return get_global_id(0); }
template<>
int2 coords<int2>() { return int2(get_global_id(0), get_global_id(1)); }
template<>
int3 coords<int3>() {
   return int3(get_global_id(0), get_global_id(1), get_global_id(2) ); }

auto blend = [](auto val1, auto val2) { return (val1 + val2) / 2; };

template<class T, class U, class F>
void filter_image(T in1, T in2, U out, F filter)
{
  auto pos = coords<typename T::integer_coord>();
  auto val1 = in1.read(pos);
  auto val2 = in2.read(pos);
  out.write(pos, filter(val1, val2));
}
```

```
kernel void myKernel1(
    image2d<float4, image_access::read> in1,
    image2d<float4, image_access::read> in2,
    image2d<float4, image_access::write> out)
{
  filter_image(in1, in2, out, blend);
}

kernel void myKernel2(
     image3d<int4, image_access::read> in1,
     image3d<int4, image_access::read> in2,
     image3d<int4, image_access::write> out)
{
  filter_image(in1, in2, out, blend);
}
```

Writing generic code with OpenCL C++ Example: generic image operations

```
auto max_channel = [](auto val1, auto val2) { return max(val1, val2); };

kernel void myKernel1(
    image2d<float4, image_access::read> in1,
    image2d<float4, image_access::read> in2,
    image2d<float4, image_access::write> out)
{
  filter_image(in1, in2, out, max_channel );
}

kernel void myKernel2(
      image3d<int4, image_access::read> in1,
      image3d<int4, image_access::read> in2,
      image3d<int4, image_access::write> out)
{
  filter_image(in1, in2, out, max_channel );
}
```

Writing generic code with OpenCL C++
Example: matrix implementation

Writing generic code with OpenCL C++ Example: matrix implementation

```
template <class T, size_t N>
struct matrix
{
    make_vector_t<T, N> m[N];
};
```

Writing generic code with OpenCL C++ Example: matrix implementation

```
matrix& operator+=(const matrix& rhs)
{
        for (size_t i = 0; i < N; ++i) m[i] += rhs.m[i]; return *this;
}
```

Writing generic code with OpenCL C++ Example: matrix implementation

```
auto operator[ ](const ulong2& idx) const
{
        return index(m[idx.x], idx.y);
}
```

Writing generic code with OpenCL C++
Example: matrix implementation

```
matrix transpose( ) const
{
    matrix temp;
        for (size_t i = 0; i < N; ++i)
            for (size_t j = 0; j < N; ++j)
                temp[{i, j}] = *this[{j, i}];
    return temp;
}
```

Writing generic code with OpenCL C++ Example: matrix implementation

```
matrix operator*(const matrix & rhs) const
{
    auto r = rhs.transpose( );
    matrix res;
    for (size_t i = 0; i < N; ++i)
        for (size_t j = 0; j < N; ++j)
            res[{i, j}] = dot(m[i], r[j]);
    return res;
}
```

Writing generic code with OpenCL C++ Example: matrix usage

```
void fun( )
{
    matrix<float, 4> m{ {{1,0,0,0},
                          {0,1,0,0},
                          {0,0,1,0},
                          {0,0,0,1}} };
    m[{3,3}] = 2;
    m = m * m.transpose( );
}
```

Value added

- Modern language with rich standard library
- Increases productivity with generic programming
- Metaprogramming allows to solve many problems at compile time
- Flexibility

KHRONOS™
GROUP