IWOCL 2024

The 12th International Workshop on OpenCL and SYCL

An Online Compiler for SYCL Kernels and Some Related Ideas

James Brodman, Intel Corporation

Ben Ashbaugh, Michael Kinsner, Steffen Larsen, Greg Lueck, John Pennycook, Roland Schulz – Intel Corporation

Gordon Brown – Codeplay

APRIL 8-11, 2024 | CHICAGO, USA | IWOCL.ORG

What is “online compilation”? Why do we need it?

- Applications want to customize kernels based on input data, device features, or other parameters
- Specialization constants not powerful enough
  - E.g. aren’t “constexpr”, so can’t use as template parameters
- Not practical to predefine all possible variants of kernel
  - Too many variations, leads to code explosion
- Want to dynamically generate source code for a kernel and then compile it when the application runs
- OpenCL has this feature, as does CUDA (with NVRTC)

Example usage – part 1

```
namespace syclex = sycl::ext::oneapi::experimental;

int main() {
  sycl::queue q;
  sycl::context ctxt = q.get_context();

  std::string src = /*...*/;
  auto src_bundle = syclex::create_kernel_bundle_from_source(
    ctxt, syclex::source_language::sycl, src);

  auto exe_bundle = syclex::build(src_bndl);
  sycl::kernel myiota = exe_bundle.ext_oneapi_get_kernel("myiota");

  float start = 3.14f;
  float *ptr = sycl::malloc_shared<float>(NUM, q);
  q.submit([&](sycl::handler &cgh) {
    cgh.set_arg(0, start);
    cgh.set_arg(1, ptr);
    cgh.parallel_for(sycl::nd_range{{NUM},{WGSIZE}}, myiota);
  });
}
```

String defining kernel, dynamically generated

Leverages existing SYCL "kernel bundle" framework

Gets a "kernel" object from compiled string

Can use existing SYCL APIs to launch a "kernel" object

The problem with kernel argument indices

```
cgh.set_arg(0, start);
cgh.set_arg(1, ptr);
```

Sets kernel arg #0

Sets kernel arg #1

Lambda expression captures kernel arguments, but no defined order of captures in C++.

```
float start = 3.14f;
float *ptr = sycl::malloc_shared<float>(NUM, q);

auto lambda = [=](sycl::nd_item<> it) {
  int id = it.get_global_linear_id();
  ptr[id] = start + static_cast<float>(id);
};
```

SYCL doesn't define whether this kernel has two arguments or one "struct" argument.

```
struct mykernel {
  float start;
  float *ptr;

  void operator()(sycl::nd_item<> it) {
    int id = it.get_global_linear_id();
    ptr[id] = start + static_cast<float>(id);
  }
};
```

Solve with new kernel syntax – free function kernels

Kernel is just a plain function, arguments are obvious

This "property" identifies the function as an nd-range kernel

```
SYCL_EXT_ONEAPI_FUNCTION_PROPERTY((syclex::nd_range_kernel<1>))

void myiota(float start, float *ptr) {
  sycl::nd_item<1> it = syclex::this_work_item::get_nd_item<1>();

  int id = it.get_global_linear_id();
  ptr[id] = start + static_cast<float>(id);
}
```

Need some new way to get iteration index. Functions in the "this_work_item" namespace return the iteration index. These are available as an extension even for traditional kernels.

We support this syntax for nd-range and single-task kernels (not for simple range kernels)

Putting it all together

```
namespace syclex = sycl::ext::oneapi::experimental;

int main() {
  sycl::queue q;
  sycl::context ctxt = q.get_context();

  std::string src = R"""(
    SYCL_EXT_ONEAPI_FUNCTION_PROPERTY((syclex::nd_range_kernel<1>))
    void myiota(float start, float *ptr) {
      sycl::nd_item<1> it = syclex::this_work_item::get_nd_item<1>();
      int id = it.get_global_linear_id();
      ptr[id] = start + static_cast<float>(id);
    }
  )""";

  auto src_bundle = syclex::create_kernel_bundle_from_source(ctxt, syclex::source_language::sycl, src);

  auto exe_bundle = syclex::build(src_bndl);
  sycl::kernel myiota = exe_bundle.ext_oneapi_get_kernel("myiota");

  float start = 3.14f;
  float *ptr = sycl::malloc_shared<float>(NUM, q);
  q.submit([&](sycl::handler &cgh) {
    cgh.set_arg(0, start);
    cgh.set_arg(1, ptr);
    cgh.parallel_for(sycl::nd_range{{NUM},{WGSIZE}}, myiota);
  });
}
```

Remember, this string would be dynamically generated

Digression on free function kernels

Free function kernels – not just for online compilation

```
SYCL_EXT_ONEAPI_FUNCTION_PROPERTY((syclex::nd_range_kernel<1>))
void iota(float start, float *ptr) {
  sycl::nd_item<1> it = syclex::this_work_item::get_nd_item<1>();
  int id = it.get_global_linear_id();
  ptr[id] = start + static_cast<float>(id);
}

int main() {
  sycl::queue q;
  sycl::context ctxt = q.get_context();

  auto exe_bundle =
    syclex::get_kernel_bundle<iota, sycl::bundle_state::executable>(ctxt);
  sycl::kernel myiota = exe_bundle.ext_oneapi_get_kernel<iota>();

  float start = 3.14f;
  float *ptr = sycl::malloc_shared<float>(NUM, q);
  q.submit([&](sycl::handler &cgh) {
    cgh.set_arg(0, start);
    cgh.set_arg(1, ptr);
    cgh.parallel_for(sycl::nd_range{{NUM},{WGSIZE}}, myiota);
  });
}
```

Kernel defined exactly as before, but definition is in main program (not in a string)

Get "kernel" object from function pointer template parameter

Launch kernel exactly as in online compilation case

Why?

- Consistent syntax with online compiled kernels
  - No advantage to prohibiting this syntax in “normal” (non-online-compiled) kernels
- Familiar syntax to OpenCL and CUDA programmers
  - Eases migration to SYCL

Back to the online compiler

Backends can online compile other languages

```
namespace syclex = sycl::ext::oneapi::experimental;

int main() {
  sycl::queue q;
  sycl::context ctxt = q.get_context();

  std::string src = R"""(
    extern "C" __global__
    void myiota(float start, float *ptr) {
      size_t id = blockIdx.x * blockDim.x + threadIdx.x;
      ptr[id] = start + static_cast<float>(id);
    }
  )""";

  auto src_bundle = syclex::create_kernel_bundle_from_source(ctxt, syclex::source_language::cuda, src);

  auto exe_bundle = syclex::build(src_bndl);
  sycl::kernel myiota = exe_bundle.ext_oneapi_get_kernel("myiota");

  float start = 3.14f;
  float *ptr = sycl::malloc_shared<float>(NUM, q);
  q.submit([&](sycl::handler &cgh) {
    cgh.set_arg(0, start);
    cgh.set_arg(1, ptr);
    cgh.parallel_for(sycl::nd_range{{NUM},{WGSIZE}}, myiota);
  });
}
```

Kernel defined in CUDA source code

Launching is the same

Online compilation support in the DPC++ compiler

- SYCL – Supported on all backends
- CUDA – Supported only on CUDA backend
- OpenCL C – Supported on either OpenCL or Level Zero backend

SPIR-V can also be a “language”

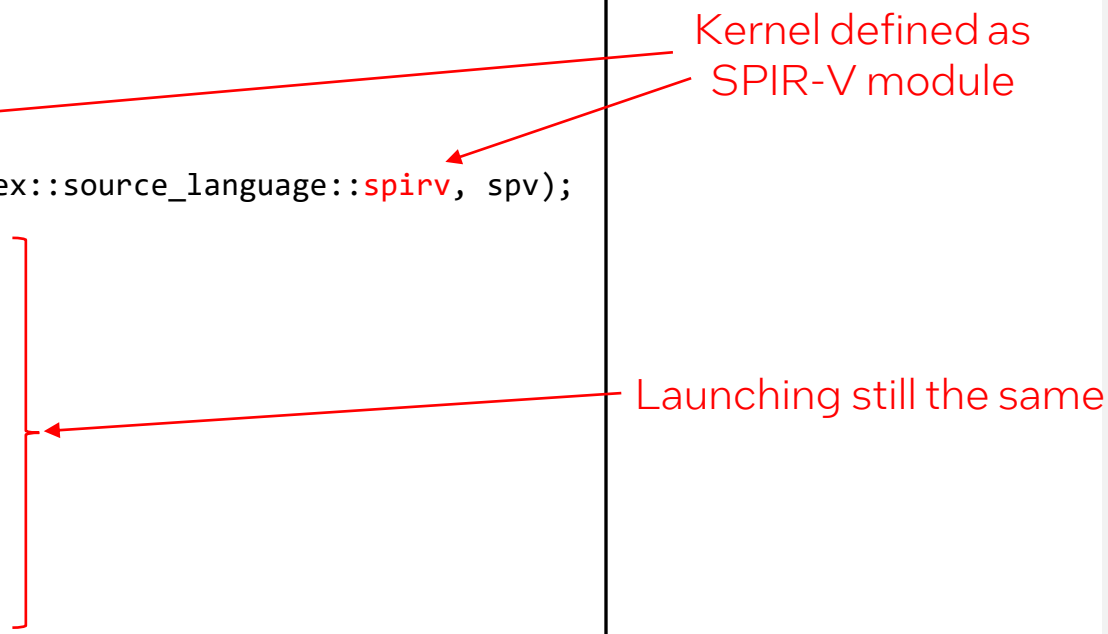
```
namespace syclex = sycl::ext::oneapi::experimental;

int main() {
  sycl::queue q;
  sycl::context ctxt = q.get_context();

  std::vector<std::byte> spv{/* binary SPIR-V module */};
  auto src_bundle = syclex::create_kernel_bundle_from_source(ctxt, syclex::source_language::spirv, spv);

  auto exe_bundle = syclex::build(src_bndl);
  sycl::kernel myiota = exe_bundle.ext_oneapi_get_kernel("myiota");

  float start = 3.14f;
  float *ptr = sycl::malloc_shared<float>(NUM, q);
  q.submit([&](sycl::handler &cgh) {
    cgh.set_arg(0, start);
    cgh.set_arg(1, ptr);
    cgh.parallel_for(sycl::nd_range{{NUM},{WGSIZE}}, myiota);
  });
}
```



- Enables ninjas to hand-code kernels in SPIR-V
- Enables any high-level compiler that generates SPIR-V (e.g. graph compiler)
- Supported on Level Zero or OpenCL backends

Experimental support in DPC++

- OpenCL C – Works now
- SPIR-V – Works now
- SYCL – In progress
- CUDA – Planned

Disclaimers


intel