Inference with ArrayFire and oneAPI

Introduction

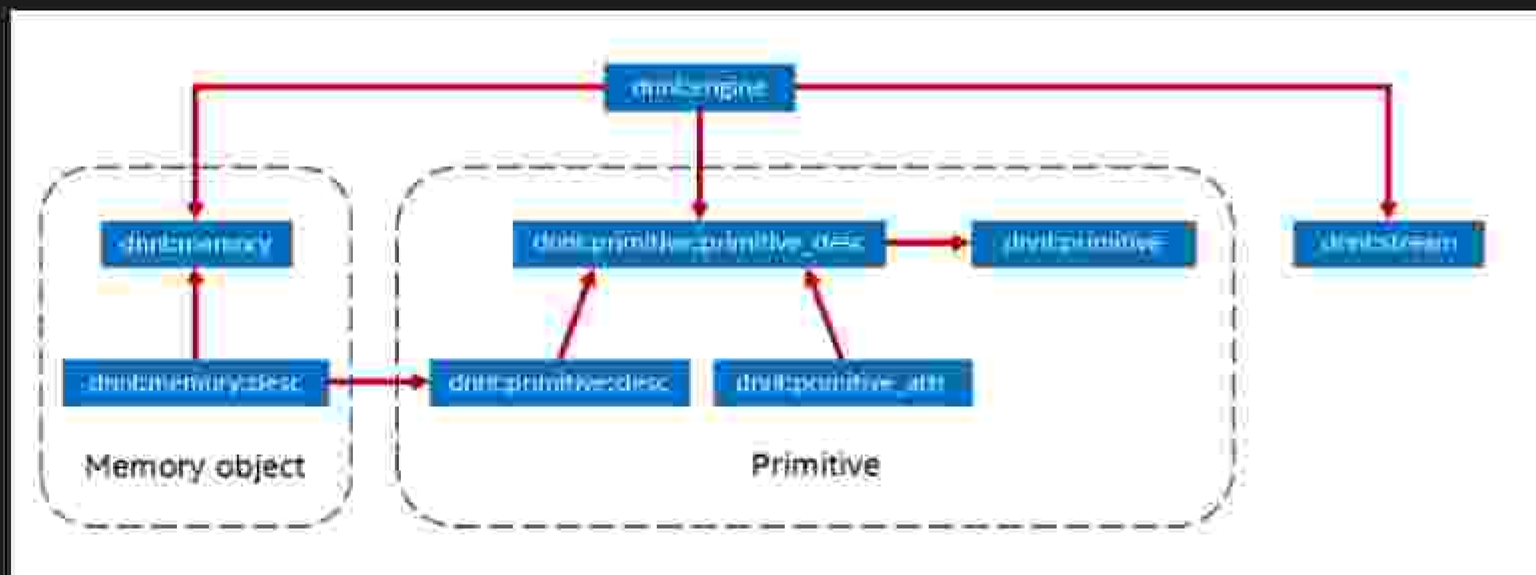
- ArrayFire the open-source C++ library
- Perform heterogenous computation on a variety of backends with an easy to use API
- Good at GPGPU computation, computer vision, image and signal processing

Flashlight

- Many derivative projects rely on ArrayFire under the hood
- Facebook's Flashlight is an open source C++ framework for training and inference
- Leverages ArrayFire, so support is there for OpenCL

oneDNN

- Library of basic building blocks for deep learning
- Optimized on Intel Architecture Processors, Intel Processor Graphics and Xe Architecture graphics.



oneDNN main components

- **Primitive (dnnl::primitive)** is an object that encapsulates a particular computation such as forward convolution. Differs from function by having an immutable state.
- **Engine: (dnnl::engine)** is an abstraction of a computational device: a CPU, a specific GPU card in the system
- **Streams (dnnl::stream)** encapsulate execution context tied to a particular engine. In our example will correspond to OpenCL command queues
- **Memory (dnnl::memory)** encapsulates handle to memory allocated on a specific engine, tensor dimensions, data type, and memory format.

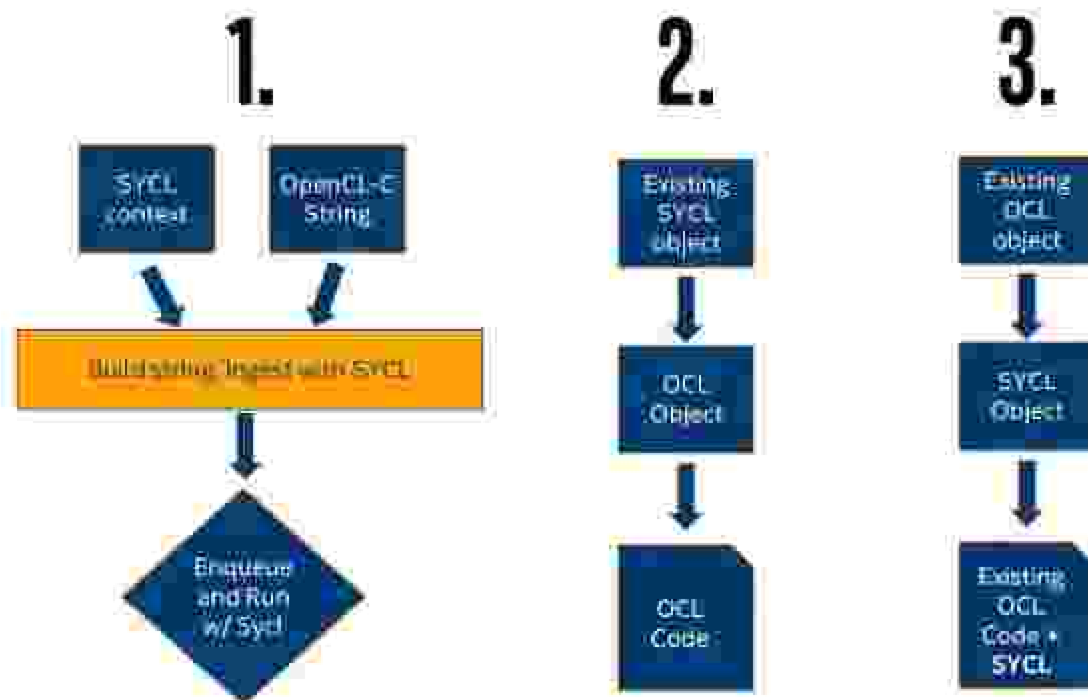
Rewrite everything from scratch, right?

- With shift towards data-centric AI, preprocessing steps more important than ever
- ArrayFire allows high-level expression of complicated pre-processing tasks
- oneAPI provides many useful low-level DNN components
- Can we join the two?

General SYCL OpenCL Interop

Interoperability Types

We contend there are three types of OpenCL* and SYCL* programming interoperability



General SYCL OpenCL Interop

```
queue q{gpu_selector()};          // Create Command Queue Targeting GPU
program p(q.get_context());       // Create program from the same context as q

// Compile OpenCL vecAdd kernel. which is expressed as a C++ Raw String as indicated by R"
p.build_with_source(R"( __kernel void vecAdd(__global int *a,
                                             __global int *b,
                                             __global int *c)
                       {
                              int i=get_global_id(0);
                              c[i] = a[i] + b[i];
                       } )");
  // buffers here ...
  q.submit([&](handler& h) {
          // accessors here ...
          // Set buffers as arguments to the kernel
          h.set_args(A, B, C);
          // Launch vecAdd kernel from the p program object across N elements.
          h.parallel_for(range<1> (N), p.get_kernel("vecAdd"));
  });
```

General SYCL OpenCL Interop

DPC++/SYCL Objects with **.get()**

- cl::sycl::platform::get() -> cl_platform_id
- cl::sycl::context::get() -> cl_context
- cl::sycl::device::get() -> cl_device_id
- cl::sycl::queue::get() -> cl_command_queue
- cl::sycl::event::get() -> cl_event
- cl::sycl::program::get() -> cl_program
- cl::sycl::kernel::get() -> cl_kernel

DPC++/SYCL Constructors Using OpenCL objects

- cl::sycl::platform::platform(cl_platform_id)
- cl::sycl::context::context(cl_context, ...)
- cl::sycl::device::device(cl_device_id)
- cl::sycl::queue::queue(cl_command_queue,...)
- cl::sycl::event::event(cl_event, ...)
- cl::sycl::program::program(context, cl_program)
- cl::sycl::kernel::kernel(cl_kernel, ...)
- cl::sycl::buffer::buffer(cl_mem, ...)
- cl::sycl::image::image(cl_mem, ...)

oneDNN / OpenCL Interop

- Almost identical, consistent with DPC++:
- Provides interop with both SYCL and OpenCL

API to construct oneDNN object

- Engine:
  - dnnl::ocl_interop::make_engine(cl_device_id, cl_context)
- Stream:
  - dnnl::ocl_interop::make_stream(const engine &, cl_command_queue)
- Memory (Buffer-based):
  - dnnl::memory(const memory::desc &, const engine &, cl_mem)
- Memory (USM-based):
  - dnnl::ocl_interop::make_memory(const memory::desc &, const engine &, ocl_interop::memory_kind, void *)

API to access OpenCL object(s)

- Engine:
  - dnnl::ocl_interop::get_device(const engine &)
    dnnl::ocl_interop::get_context(const engine &)
- Stream:
  - dnnl::ocl_interop::get_command_queue(const stream &)
- Memory (Buffer-based):
  - dnnl::ocl_interop::get_mem_object(const memory &)
- Memory (USM-based):
  - dnnl::memory::get_data_handle()

oneDNN / SYCL Interop

- Runtime differentiates between sycl and OpenCL runtimes, so interop functions are required for both

API to construct oneDNN object

- Engine:
  - dnnl::sycl_interop::make_engine(const cl::sycl::device &, const cl::sycl::context &)
- Stream:
  - dnnl::sycl_interop::make_stream(const engine &, cl::sycl::queue &)
- Memory (Buffer-based):
  - dnnl::sycl_interop::make_memory(const memory::desc &, const engine &, cl::sycl::buffer<T, ndims> &)
- Memory (USM-based):
  - dnnl::memory(const memory::desc &, const engine &, void *)

API to access SYCL object(s)

- Engine:
  - dnnl::sycl_interop::get_device(const engine &) dnnl::sycl_interop::get_context(const engine &)
- Stream:
  - dnnl::sycl_interop::get_queue(const stream &)
- Memory (Buffer-based):
  - dnnl::sycl_interop::get_buffer<T, ndims>(const memory &)
- Memory (USM-based):
  - dnnl::memory::get_data_handle()

Accessing ArrayFire af::array

- Share context and device from oneDNN with ArrayFire

```
#include <arrayfire.h> // ArrayFire headers
#include <af/opencl.h>
#include "oneapi/dnnl/dnnl_ocl.hpp" // oneDNN interop headers
#include "oneapi/dnnl/dnnl_sycl.hpp"
#include <CL/cl.h>

// get contexts from oneDNN
cl_device_id device_ = dnnl::ocl_interop::get_device(eng);
cl_context context_ = dnnl::ocl_interop::get_context(eng);
cl_command_queue queue_ = dnnl::ocl_interop::get_command_queue(s);
// share OpenCL details with ArrayFire
afcl::addDevice(device_, context_, queue_);
afcl::setDevice(device_, context_);
```

Accessing ArrayFire af::array

- Use ArrayFire for preprocessing images for oneDNN alexnet example in SDK

```
af::array images = af::constant(0.f, h, w, 3, batch); // create empty array within same context as oneDNN
images = read_images(directory);
images = af::resize(images, 227, 227) / 255.f;        // resize to alexnet input size and normalize [0-1]
images = af::reorder(images, 3, 2, 0, 1);             // hwcn -> nchw
...                                                   // additional pre-processing
af::sync();                                           // force all queued arrayfire operations to finish
```

Accessing ArrayFire af::array

- Return memory to oneDNN

```
cl_mem *src_mem = af_user_src.device<cl_mem>(); // get cl_mem from arrayfire

// create dnnl::memory with cl_mem for further inference with preprocessed inputs
dnnl::memory user_src_memory({{conv1_src_tz}, dt::f32, tag::nchw}, eng, *src_mem);

// additional alexnet network setup, loading of weights
...

// execute all primitive steps for full inference using our inputs
for (size_t i = 0; i < net.size(); ++i) {
    net.at(i).execute(s, net_args.at(i));
}

s.wait();             // wait until stream finishes writing to memory

af_user_src.unlock(); // return memory ownership to arrayfire so resources can be freed
```

Laziness prevails!

- OneAPI's reliance on the SYCL standard allows you to reuse your existing OpenCL code with minimal interop bookkeeping
- Don't be afraid to incorporate these new tools in your existing codebases
- Use ArrayFire!

Contact Us

ArrayFire.com

sales@arrayfire.com

800-570-1941

ArrayFire