oneDNN Generic GPU Vendor

Denis Samoilov
09/12/2024

UXL FOUNDATION
Unified Acceleration

oneDNN GPU Support

- oneDNN GPU support is implemented via OpenCL and SYCL runtimes
  - oneDNN with OpenCL GPU runtime supports only Intel vendor (though it's technically possible to enable others)
  - oneDNN with SYCL GPU runtime supports Intel, NVIDIA, AMD and Generic vendors
- oneDNN with SYCL GPU runtime is the main configuration to support multiple vendors
- oneDNN has optimized and reference implementations for Intel, NVIDIA and AMD vendors
- oneDNN with SYCL GPU runtime requires oneAPI DPC++ compiler that supports the target vendor
  - There are Intel oneAPI DPC++ compiler that is distributed as part of oneAPI toolkit and Open Source oneAPI DPC++ compiler that can be built from source
  - The Intel oneAPI DPC++ compiler can be used with plugins to enable additional vendors: NVIDIA, AMD
- oneDNN is allowed to have different additional dependencies for different vendors such as cuDNN for NVIDIA or MIOpen for AMD
- How to build oneDNN with GPU support for different vendors:

  -
    ```
    cmake .. -DDNNL_CPU_RUNTIME=SYCL -DDNNL_GPU_RUNTIME=SYCL -DNNL_GPU_VENDOR=NVIDIA
    ```
  - Supported values for the vendor are Intel (default), NVIDIA, AMD and Generic
  - Documentation for the supported vendors can be found here

Generic GPU Vendor

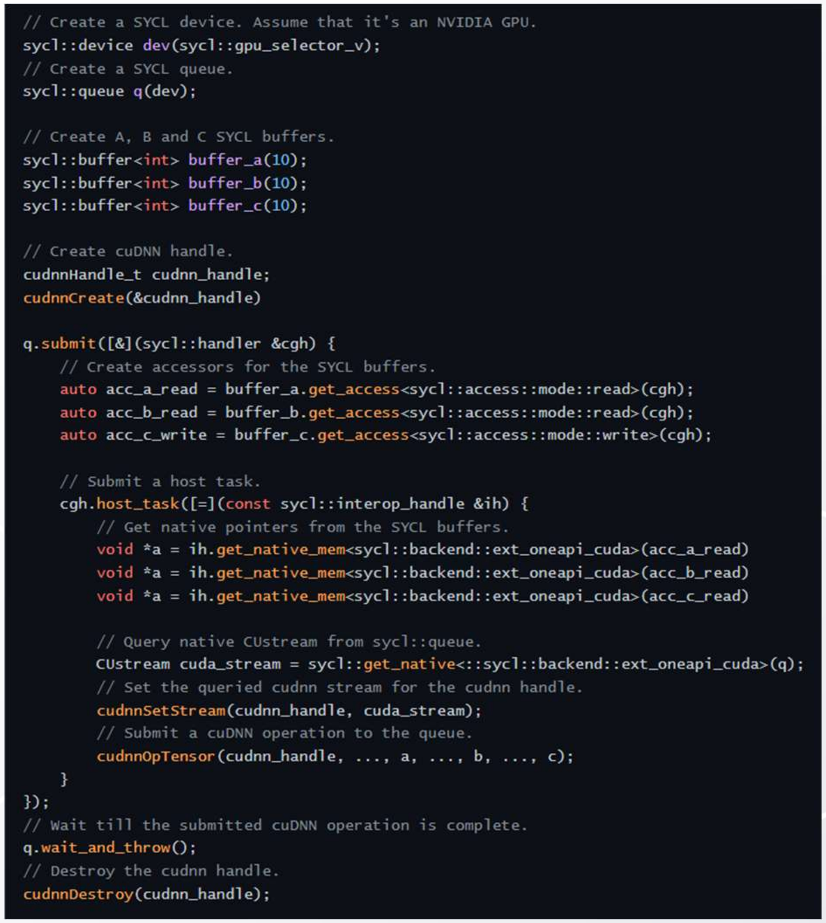
- The generic GPU vendor was introduced to provide initial support for different vendors
- The initial support means providing reference implementations that are generic by nature and not optimized for any vendor
- The generic kernels are implemented in generic SYCL (no vendor specific extensions are used)
- The generic GPU vendor is ready to be extended to support accelerators as well. There is no support for accelerators yet
- The generic vendor requires the oneAPI DPC++ compiler to support the target device (GPU or accelerator)
- If there are multiple GPUs and/or accelerators in the system, it is the user's responsibility to ensure that the correct SYCL device representing the target device is selected at runtime
- The list of supported oneDNN primitives for the generic vendor can be found here

Vendor Specific Optimizations

- The generic vendor is meant to provide the initial support for new vendors and therefore the performance of the generic kernels can be suboptimal
- There are three options to optimize oneDNN for the target vendor:
  - Implement SYCL kernels using vendor specific extensions
  - Use vendor performance libraries
  - Use backend kernel language when applicable. For example, if backend is OpenCL then the OpenCL kernels can be compiled and used with SYCL with the interoperability API
  - In all cases a new GPU vendor must be added (“DNNL_GPU_VENDOR=NEW_VENDOR”)
- The GPU specific code is organized around vendors. Each vendor has a dedicated and isolated space for development:

```
gpu
├── intel/                # Intel vendor-specific code
├── nvidia/               # NVIDIA vendor-specific code
├── amd/                  # AMD vendor-specific code
├── new_vendor/           # A new vendor-specific code
└── generic/              # Generic (vendor agnostic)code
    ├── sycl/             # Generic SYCL kernels (no vendor specific extensions)
    ├── ref_concat.hpp    # Kernel language agnostic implementations (concat via reorders)
```

Use Vendor Performance Libraries



```
// Create a SYCL device. Assume that it's an NVIDIA GPU.
sycl::device dev(sycl::gpu_selector_v);
// Create a SYCL queue.
sycl::queue q(dev);

// Create A, B and C SYCL buffers.
sycl::buffer<int> buffer_a(10);
sycl::buffer<int> buffer_b(10);
sycl::buffer<int> buffer_c(10);

// Create cuDNN handle.
cudnnHandle_t cudnn_handle;
cudnnCreate(&cudnn_handle)

q.submit([&](sycl::handler &cgh) {
    // Create accessors for the SYCL buffers.
    auto acc_a_read = buffer_a.get_access<sycl::access::mode::read>(cgh);
    auto acc_b_read = buffer_b.get_access<sycl::access::mode::read>(cgh);
    auto acc_c_write = buffer_c.get_access<sycl::access::mode::write>(cgh);

    // Submit a host task.
    cgh.host_task([=](const sycl::interop_handle &ih) {
        // Get native pointers from the SYCL buffers.
        void *a = ih.get_native_mem<sycl::backend::ext_oneapi_cuda>(acc_a_read)
        void *a = ih.get_native_mem<sycl::backend::ext_oneapi_cuda>(acc_b_read)
        void *a = ih.get_native_mem<sycl::backend::ext_oneapi_cuda>(acc_c_read)

        // Query native CUstream from sycl::queue.
        CUstream cuda_stream = sycl::get_native<::sycl::backend::ext_oneapi_cuda>(q);
        // Set the queried cudnn stream for the cudnn handle.
        cudnnSetStream(cudnn_handle, cuda_stream);
        // Submit a cuDNN operation to the queue.
        cudnnOpTensor(cudnn_handle, ..., a, ..., b, ..., c);
    }
});
// Wait till the submitted cuDNN operation is complete.
q.wait_and_throw();
// Destroy the cudnn handle.
cudnnDestroy(cudnn_handle);
```

- oneAPI DPC++ compiler backend has to be compatible with the performance library. For example, oneAPI DPC++ compiler with CUDA backend can be used with cuDNN
- SYCL specification defines a way to interoperate with the backend
  - SYCL objects can be queried for backend native objects. For example, “sycl::queue” for “Custream”, “sycl::contex” for “Cucontext”, “sycl::device” for “Cudevice” and “sycl::buffer” for a memory pointer
  - The pointer returned by “malloc_shared” or “malloc_device” can be used as is
- Using the interoperability API the vendor performance libraries can be embedded into the SYCL programming model

How To Contribute

- oneDNN is an open-source project that has contribution guidelines and defines an RFC process. Please make yourself familiar with them
  - When it comes to external contributions that impact some aspects of the library's architecture a formal proposal (RFC) is required. Adding a new vendor will requires a formal proposal
- The first step to enable a new vendor is to make sure that the oneAPI DPC++ compiler supports it
- The second step is to build oneDNN for a generic vendor and try to run it on the target device
  - **Warning**: while the vendor is generic the oneAPI DPC++ compiler may require some additional flags to compile generic SYCL kernels for the target device. See here for more details
- The third step is to add support for the new vendor to add optimized kernels. The generic kernels are allowed to be enabled in a combination with the optimized ones so the coverage will not be impacted
- Do no hesitate to ask questions on the public github. We try to answer them within a few days

Thank you!

UXL FOUNDATION
Unified Acceleration