Bringing performant support for Nvidia hardware to SYCL

Ruyman Reyes Castro
Principal Software Engineer, Programming Models

IWOCL 2020
Codeplay - Enabling AI to be Open, Safe and Accessible to all

Products

**Acoran**
Integrates all the industry standard technologies needed to support a very wide range of AI and HPC

**ComputeCpp**
C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

**ComputeAorta**
The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Addressable Markets

- High Performance Compute (HPC)
- Automotive ADAS, IoT, Cloud Compute
- Smartphones & Tablets
- Medical & Industrial

Technologies: Artificial Intelligence, Vision Processing, Machine Learning, Big Data Compute

Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland with ~80 employees

Customers

Many Major Companies

© 2020 Codeplay Software Ltd.
What have we done

CUDA backend for Intel SYCL implementation

- Does not require OpenCL
- All contributions in the open

SYCL Standard contributions

- Experience of porting SYCL to non-OpenCL backend
- Multiple extensions that enable CUDA-specific features
- Overall porting experience
DPC++ and SYCL (and Codeplay)

Data Parallel C++ : C++ and SYCL* standard and extensions

• “Incorporates” the SYCL standard for data parallelism and heterogeneous programming Data Parallel C++ ⇔ DPC++

DPC++ Extends SYCL 1.2.1

• Fast-moving open collaboration feeding into the SYCL standard
• Open source implementation with goal of upstream LLVM
• DPC++ extensions aim to become core SYCL, or Khronos extensions

Codeplay involvement

• Contribute back to the community from an independent codebase
• Explore extensions and actively participate on oneAPI initiative
Disclaimer and Trademarks

• NVIDIA, the NVIDIA logo and CUDA are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and/or other countries

• Codeplay is not associated with NVIDIA for this work and it is purely using public documentation and widely available code
Summary

• Using SYCL for CUDA
• Overall design of SYCL for CUDA
• Compiler implementation
• Runtime implementation
• Interoperability with existing libraries
• Conclusions and future work
Using SYCL for CUDA
Using SYCL for CUDA

• Build or get a binary package of DPC++
  • Daily builds of master in https://github.com/intel/llvm/releases
  • Detailed instructions in https://github.com/intel/llvm/blob/sycl/sycl/doc/GetStartedGuide.md

• Compile your code using the CUDA target triple
  ```
  clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice \
  simple-sycl-app.cpp -o simple-sycl-app-cuda.exe
  ```
  No changes required to your SYCL code

• Run your application with the CUDA backend enabled
  ```
  SYCL_BE=PI_CUDA ./simple-sycl-app-cuda.exe
  ```
  Env var used by default device selection
Design of SYCL for CUDA
SYCL for CUDA

SYCL 1.2.1 was intended for OpenCL 1.2

• If a SYCL 2.2 ever existed, it was based on OpenCL 2.2
• What could be a good alternative target to demonstrate SYCL as a High Level Model?
• Let’s have an open discussion about SYCL for non-OpenCL!

Sure let’s do Vulkan!

• Not that simple, SYCL was never designed for Graphics
• Already a potential path via clspv + clvk

Have you heard about CUDA?

• Existing OpenCL + PTX path (available on ComputeCpp) not great
  • Difficult to maintain but no customer base
• Native CUDA support will be better to expand the ecosystem
SYCL 1.2.1 on CUDA

• What can work?
  • Platform model (Platform/Device/Context)
  • Buffers, copy
  • NDRange kernels

• What cannot work
  • Interoperability (no OpenCL types!)
  • Images and samplers
    • CUDA images are sampled on construction
    • SYCL/OpenCL Images are sampled in the kernel
  • SYCL program class
    • OpenCL compilation model does not match CUDA (e.g. options are different)

We have created a number of proposals and provide feedback to the SYCL WG to make those implementable on a future SYCL version
Main outcome: SYCL “generalization”

**SYCL Next**

- **Host backend**
  - Mandatory
  - No external dependencies from SYCL library
  - Must execute on the main CPU of the system

- **OpenCL backend**
  - Optional
  - Requires libOpenCL.so in the system

- **CUDA backend**
  - Optional
  - Requires libcuda.so in the system and an NVIDIA linker

https://github.com/KhronosGroup/SYCL-Shared/blob/master/proposals/sycl_generalization.md
A SYCL module represents a collection of functions and symbols that can be used for all devices in the associated context. A SYCL module can store different versions of the same functions and symbols in different representations. Each of these versions is called a device image.

https://github.com/KhronosGroup/SYCL-Shared/blob/master/proposals/sycl_modules.md

This is a high-level abstraction, NOT a mapping of a SPIR-V or LLVM module
Host task

Command group that runs a task on the host inside the SYCL DAG

```cpp
auto cgh = [=] (handler& cgh) {
    auto accB = bufB.get_access<access::mode::write,
        access::target::host_buffer>(cgh);

    h.codeplay_host_task([=](){
        std::ifstream ifs(some_file_name, std::ifstream::in);
        std::for_each(std::begin(accB), std::end(accA), [&] (auto& elem) {
            if (!ifs.good()) {
                elem = 0;
            } else {
                elem = ifs.get();
            }
        });
    });
    qA.submit(cgh);
};
```

https://github.com/codeplaysoftware/standards-proposals/blob/master/host_task/host_task.md
Compiler implementation
Leveraging existing CUDA support

- Current LLVM tip has CUDA support
- This was contributed by Google back in 2016 [https://research.google/pubs/pub45226/](https://research.google/pubs/pub45226/)
- Includes a CUDA runtime implementation and a PTX backend
- The PTX backend is the interesting part!
Driver (linking)
Converting local memory to Shared memory

Converting local memory to Shared memory

```
buffer<cl::sycl::cl_int, 1> buf(data, range<1>(size));
myQueue.submit([&](handler &cgh) {
    auto ptr = buf.get_access<access::mode::read_write>(cgh);
    accessor<cl::sycl::cl_int, 1, access::mode::read_write,
             access::target::local>
        tile(range<1>(2), cgh);
    cgh.parallel_for<example_kernel>(
        nd_range<1>(range<1>(size), range<1>(2)), [=](nd_item<1> item) {

        tile[pos] = ptr[item.get_global_linear_id()];
    }
});
```

Local memory allocation

Usage as an accessor

Multiple allocations of local memory are allowed
Converting local memory to Shared memory

```c
extern __shared__ int s[];
int *integerData = s;    // nI ints
float *floatData = (float*)&integerData[nI]; // nF floats
char *charData = (char*)&floatData[nF]; // nC chars
```

CUDA Dynamic Shared memory

Declarations, each pointer refers to an element

Using CUDA Dynamic Shared memory in the CUDA runtime:
Passing the total size of the allocation as last argument

```c
myKernel<<<gridSize, blockSize, nI*sizeof(int)+nF*sizeof(float)+nC*sizeof(char)>>>(...);
```

Local to Shared transformation

Create a global symbol to the CUDA shared memory address space

Transform all pointers to CUDA shared memory into a 32 bit integer

Replace all uses of the pointers by offsets into the shared memory

```c
define void @kernel(i8 addrspace(3)* %arg1, i32 addrspace(3)* %arg2) {
    @kernel.shared = external addrspace(3) global [0 x i8], align 4
}

define void @kernel(i32 %0, i32 %1) {
    @kernel.shared_mem = external addrspace(3) global [0 x i8], align 4
    @kernel_shared_mem = external addrspace(3) global [0 x i8], align 4
}

entry:
    %arg1 = getelementptr [0 x i8], [0 x i8] addrspace(3)* @kernel_shared_mem, i32 0, i32 %0
    %2 = getelementptr [0 x i8], [0 x i8] addrspace(3)* @kernel_shared_mem, i32 0, i32 %1
    %arg2 = bitcast i8 addrspace(3)* %2 to i32 addrspace(3)*
```
Runtime implementation
The PI API

https://github.com/intel/llvm/blob/sycl/sycl/include/CL/sycl/detail/pi.h

```
// Memory

pi_result piMemBufferCreate(pi_context context, pi_mem_flags flags, size_t size,
    void *host_ptr, pi_mem *ret_mem);

pi_result piMemImageCreate(pi_context context, pi_mem_flags flags,
    const pi_image_format *image_format,
    const pi_image_desc *image_desc, void *host_ptr,
    pi_mem *ret_mem);
```

```
pi_result OCL(piMemBufferCreate)(pi_context context, pi_mem_flags flags,
    size_t size, void *host_ptr, pi_mem *ret_mem) {
  pi_result ret_err = PI_INVALID_OPERATION;
  *ret_mem = cast<pi_mem>QlCreateBuffer(cast<cl_context>(context),
    cast<cl_mem_flags>(flags), host_ptr, cast<cl_image2d_desc>(
    cast<cl_mem>(image_desc),
    cast<cl_image_format>(image_format),
    PiMemBufferFlags(flags),
    PiMemImage2DDesc(host_ptr))),
  return ret_err;
}
```
PI CUDA plugin equivalent (example)

```c
pi_result cuda_bufferCreate(pi_context context, pi_mem_flags flags,
    size_t size, void *host_ptr,
    pi_mem *ret_mem) {
    // Need input memory object
    assert(ret_mem != nullptr);
    ScopedContext active(context);
    CUDeviceptr ptr;
    _pi_mem::alloc_mode allocMode = _pi_mem::alloc_mode::classic;
    if ((flags & PI_MEM_FLAGS_HOST_PTR_USE) && enableUseHostPtr) {
        retErr = PI_CHECK_ERROR(cuMemHostRegister(host_ptr, size,
            CU_MEMHOSTREGISTER_DEVICEMAP));
        retErr = PI_CHECK_ERROR(cuMemHostGetDevicePointer(&ptr, host_ptr, 0));
        allocMode = _pi_mem::alloc_mode::use_host_ptr;
    } else {
        retErr = PI_CHECK_ERROR(cuMemAlloc(&ptr, size));
    }
}
```

Errors are assertions for developers

Set the active CUDA context

Register memory if USE_HOSTPTR

Allocate CUDA memory if no flags

https://github.com/intel/llvm/blob/sycl/sycl/plugins/cuda/pi_cuda.cpp
Construct a PI mem object

```cpp
auto piMemObj = std::unique_ptr<pi_mem>(
    new _pi_mem{context, parentBuffer, allocMode, ptr, host_ptr, size});
```

PI Mem object, no longer a 1 2 1 map!
Interoperability
Using native libraries in SYCL

```cpp
auto cgH = [=] (codeplay::handler& cgh) {
    // Get device accessor to SYCL buffer (cannot be dereferenced directly in interop_task).
    auto accA = bufA.get_access<access::mode::read>(cgh);
    auto accB = bufB.get_access<access::mode::read_write>(cgh);

    h.interop_task([=](codeplay::interop_handle &handle) {
        third_party_api(handle.get_queue(), // Get the OpenCL command queue to use, can be the fallback
                        handle.get_buffer(accA), // Get the OpenCL mem object behind accA
                        handle.get_buffer(accB)); // Get the OpenCL mem object behind accB
        // Assumes call has finish when exiting the task
    });
};
qA.submit(cgH);
```

https://github.com/codeplaysoftware/standards-proposals/blob/master/interop_task/interop_task.md
Calling CUDA libraries

```cpp
queue.submit([&](cl::sycl::handler &cgh) {
    auto a_acc = a.get_access<cl::sycl::access::mode::read>(cgh);
    auto c_acc = c.get_access<cl::sycl::access::mode::read_write>(cgh);
    cgh.interop_task([&](cl::sycl::interop_handler ih) {
        auto sc = CublasScopedContextHandler(queue);
        auto handle = sc.get_handle(queue);
        auto a_ = sc.get_mem<cuDataType*>(ih, a_acc);
        auto c_ = sc.get_mem<cuDataType*>(ih, c_acc);
        cublasStatus_t err;
        ___err = cublasSgemm(handle, upper_lower, trans, n, k, (cuDataType*)&alpha,
                             a_, lda, (cuDataType*)&beta, c_, ldc);
    });
});
```

- Normal command group with dependencies
- Set the active CUDA context
- Obtain native CUDA pointers from buffers
- Call native cublasSgemm
Conclusions and future work
Preliminary performance results

BabelStream FP32 MB/s

http://uob-hpc.github.io/BabelStream

Platform: CUDA 10.1 on GeForce GTX 980
Internal experimental branch

BabelStream FP32 MB/s

- Copy
- Mul
- Add
- Triad
- Dot

- SYCL for CUDA
- CUDA
- OpenCL for CUDA
- Experimental branch
Conclusions

- DPC++ is a working SYCL 1.2.1 compiler with many extensions that enable oneAPI features
- CUDA backend is integrated into main trunk and is part of the DPC++ release
- Already lots of comments from community, issues and even contributed pull requests!
- Currently working towards conformance (as much as is possible) in SYCL 1.2.1
Participate!

- Join us in the intel/llvm repository
- Report issues and feature requests
- Review or contribute Pull requests