Progress with single source OpenCL

Mark Grimes

12/Oct/2015

University of BRISTOL

Refresher on SYCL

- Pronounced “sickle”. No idea what it stands for.
- Allows C++11 (with some limitations) to be run on any OpenCL device, from the same source file as the host code. Code example in backup, or just google.
- AMD have an experimental open source implementation [triSYCL], but only runs on CPUs (OpenMP underlying).
- Codeplay have a full implementation to be released “later in 2015” [ComputeCpp].
    - Possible to get free previews with a confidentiality agreement.
    - No idea what license options the released version will have.
- No way of knowing what other implementations are yet to be announced.

ComputeCpp in detail

- Requires OpenCL 1.2 with the `cl_khr_spir` extension.
  - Hence no nVidia GPU support. There are reports that Windows hotfix drivers support OpenCL 1.2, so maybe it’s coming?
  - SPIR - Standard Portable Intermediate Representation.
- Modified version of clang (“`compute++`”). Two step build:
  1. `compute++` with source → outputs header file with SPIR binary wrapped in C char array.
  2. Build with your favourite compiler (could be `compute++`), including the header from the previous step.
  - Maybe there’s some incantation to do it all in one step? Not much documentation other than the example build scripts.

Current status

- Finally found someone with enough authority to sign the NDA.
- Experimental compiler with little documentation.
- A lot of teething troubles, mostly down to a requirement of a more recent glibc than my test platform (but only memcpy?).
  - Have a version of glibc that seems to play nice* with the rest of the system, forcing link to that with rpath.
- Managed to get test EDAnalyzers running on host in CMSSW_6_2_0_SLHC26 (remember this is for HGCal development). So far just calculating closest PFHit by brute force.
- Problems with Xeon Phi (later), so took a step back to stand alone “hello world” programs.

* (but see later)

Current status - GPU

- Running on AMD “Hawaii” seems to go okay.
  - Although the GPU sometimes disappears and I have to raise a ticket for a sysadmin to restart the X server (?).
  - Not sure if that’s because of something I’ve done or not.
  - Get some warnings about incompatible systems. Not sure what this means.
    - Incompatible type formats? I thought SYCL was supposed to sort this out.
    - Results look okay, although currently very simple kernels.

Current status - accelerator

- Running on Xeon Phi just segfaults.
  - Running the generated SPIR through a “traditional” OpenCL program works. Must be host code.
  - gdb says crash is in a no argument function in my alien glibc (get_nprocs). Link going wrong?
  - Problem with the system version is just
    `undefined reference to ` + "`" + `memcpy@GLIBC_2.14’`?
  - Apparently this is some famous glibc regression?
  - Tried various hacks to just link `memcpy.o` from newer version; redefine symbol etcetera.
  - Could do with some help.

Current limitations

Not yet a full implementation of SYCL. When actually running, so far I've found (ComputeCpp v2015.06):

- Complains if kernels are chained together in the same enque command. Thought this was part of the standard, so might be in next version.
- Currently no OpenCL builtins. E.g. `sqrt` just segfaults.
- Hierarchical parallelism (i.e. workgroups) currently only supported on host. I've not tested yet.
- Needs plenty of validation, e.g.

```
output[index]=input[index]*input[index];
```

...gives incorrect results.

```
float temp=input[index];
output[index]=temp*temp;
```

...works. Maybe related to Hawaii warning? Can't remember which system this was on.

Backup

“Hello CMSSW” in SYCL

N.B. This slide from my last talk using triSYCL. Need slight code changes for ComputeCpp.

```
void TestAnalyser::analyze( const edm::Event& event, const edm::EventSetup& eventSetup )
{
      edm::Handle<reco::PFRecHitCollection> hRecHits;
      event.getByToken( inputToken_, hRecHits );

      size_t inputSize=hRecHits->size();
      cl::sycl::queue myQueue;

      cl::sycl::buffer<reco::PFRecHit> inputBuffer( hRecHits->data(), hRecHits->size() );
      cl::sycl::buffer<double> outputBuffer( inputSize*inputSize );

      myQueue.submit( [&]( cl::sycl::handler& myHandler )
      {
            auto inputAccess=inputBuffer.get_access<cl::sycl::access::read>(myHandler);
            auto outputAccess=outputBuffer.get_access<cl::sycl::access::write>(myHandler);

            myHandler.parallel_for<class MyKernel>( inputSize*inputSize, [=](int index)
            {
                  const auto& hitPosA=inputAccess[index%inputSize].position();
                  const auto& hitPosB=inputAccess[index/inputSize].position();
                  outputAccess[index]=std::sqrt( (hitPosA.x()-hitPosB.x())*(hitPosA.x()-hitPosB.x())
                                               + (hitPosA.y()-hitPosB.y())*(hitPosA.y()-hitPosB.y())
                                               + (hitPosA.z()-hitPosB.z())*(hitPosA.z()-hitPosB.z()) );
            });

      });

      auto outputAccess=outputBuffer.get_access<cl::sycl::access::read,cl::sycl::access::host_buffer>();
      size_t indexA=5, indexB=8;
      if( inputSize>8 ) edm::LogInfo("SYCLTest") << "The Cartesian distance between two arbitrary hits is "
                  << outputAccess[indexA*inputSize+indexB];
}
```

Access token à la edm::EDGetTokenT. Allows SYCL to automatically copy the required data to the GPU.

This part is the kernel that would run on the GPU

Access tokens also allow the queuing system to calculate dependencies between kernels. E.g. It knows a kernel with read access to `outputBuffer` would have to run after this one.

- [Trigkas] Angelos Trigkas, University of Edinburgh, August 2014, https://static.ph.ed.ac.uk/dissertations/hpc-msc/2013-2014/Investigation%20of%20the%20OpenCL%20SYCL%20Programming%20Model.pdf
- [triSYCL] https://github.com/amd/triSYCL
- [ComputeCpp] https://www.codeplay.com/products/computecpp
- [last talk] https://indico.cern.ch/event/440798/