General Pourpose GPU Computing

And a horde of APIs

Máté Ferenc Nagy-Egri
Wigner GPU Lab

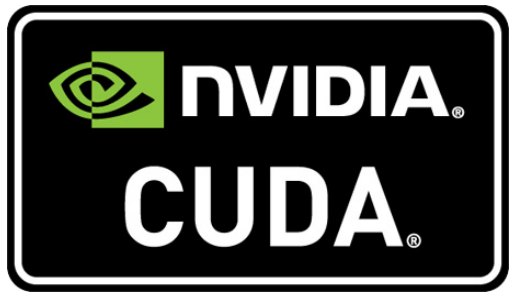
What is an API?

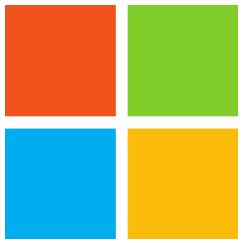
- Application Programming Interface
  - It is the surface of a library with which the programmer interacts
  - The library exposes types and functions with certain gurantees and features that the programmer wishes to use

FOR THE HOOOOOOORDEEE!

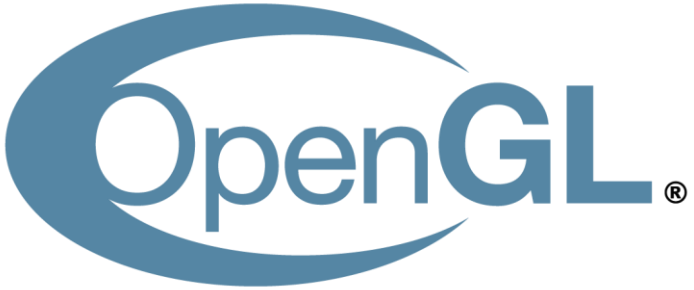
HORDE OF APIS

Non-exhaustive list of APIs





C++AMP



OpenACC

Directives for Accelerators

OpenMP

Compute Unified Device Architecture

- Nvidia's answer to the increasing developer pressure, that they want to do GPGPU without having to jump through flaming hoops
- First version released in 2007
- Two levels of access
  - C driver API
  - C language extension, in which special operators appear to launch kernels and separate host and device side code (starting from CUDA 7.0 partial C++11 support)
- Closed source technology, it is always tuned for the actual generation of Nvidia hardware
- Google got fed up with the slow pace at which the canonical compiler, ‚nvcc' is gaining features, so they mainlined their fork of Clang with CUDA language support

Compute Unified Device Architecture

```
__device__ __global__ saxpy(double a, const double* x, double* y)
{
    y[i] = a * x[i] + y[i];
}

int main()
{
    size_t length = 65536;
    size_t size = length * sizeof(double);

    double* x = (double*)malloc(size), *y = (double*)malloc(size);
    double** x_dev = (double**)cudaMalloc(size), **y_dev = (double**)cudaMalloc(size);

    cudaMemcpy(x_dev, x, size, cudaMemcpyHostToDevice);
    cudaMemcpy(y_dev, y, size, cudaMemcpyHostToDevice);

    saxpy<<<length, 1>>>(2.0, x_dev, y_dev)
}
```

Open Computing Language

- Apple Inc.’s answer to the success of CUDA, but because of high interest, they handed over the technology to the Khronos Consortium
- First version released in 2009
- Separate host and device code
  - C API for the host side code
  - C99 extension for device code (starting from OpenCL 2.1 static C++14)
- Open standard, anyone can implement it and participate in development of the standard (as our we)

Open Computing Language

saxpy.cpp

saxpy.cl

```
#pragma OPENCL EXTENSION cl_khr_fp64 : enable

__kernel void saxpy(double a,
                    __global double* x,
                    __global double* y)
{
   size_t i = get_global_id(0);

   y[i] = a * x[i] + y[i];
}
```

Open Computing Language

saxpy.cpp

```
int main()
{
   cl_double a = 2.0;
   std::vector<cl_double> x(length, 1.0);
   std::vector<cl_double> y(length, 1.0);

   cl::Buffer x_buf(context, x.begin(), x.end(), true);
   cl::Buffer y_buf(context, x.begin(), x.end(), false);
   cl::KernelFunctor<double, cl::Buffer, cl::Buffer> kernel(program, „saxpy");

   kernel(cl::EnqueueArgs(command_queue,
                          cl::NDRange(static_cast<cl::size_type>(x.size()))),
                          a,
                          x_buf,
                          y_buf).wait();
}
```

DirectX Compute Shader

- Microsoft's leading 3D graphics API's GPGPU spin-off
- First released in 2009, debut in DirectX 11
- Integrates in a trivial manner into the pipeline
- Host side C++ API reflects graphics usage scenarios
- Device side language is HLSL (High-level Shading Language)
- DirectX 12 introduces Shader Model 6.0
  - Clang fork for C++ shaders inbound (developed by MS)

Open Grahpics Library

- The Khronos variant of Microsoft's Direct Compute
- First release in 2012, debut with OpenGL 4.3
- Integrates in a trivial manner into the pipeline
- Host side C API is ARCHAIC!
- Device side language is GLSL (OpenGL Shading Language)
  - Does not seem that this will change, at least in a portable manner

C++ Accelerated Massive Parallelism

- Concept API for standardizing GPGPU parallelism
- Developed by Microsoft, first release in 2012
- C++ language extension that built on the current state of C++ concepts
- Does not restrict the format of the intermediate injected into the binary
- (It’s future remains a mystery)

Open Computing Language

```
double a = 2.0;
std::vector<double> x(length, 1.0);
std::vector<double> y(length, 1.0);

concurrency::array<double> arr_x(concurrency::extent<1>(length), x.cbegin(), x.cend(), acc_view);
concurrency::array<double> arr_y(concurrency::extent<1>(length), x.cbegin(), x.cend(), acc_view);

concurrency::parallel_for_each(acc_view,
                               arr_x.get_extent(),
                               [a,
                                x_view = concurrency::array_view<double>(arr_x),
                                y_view = concurrency::array_view<double>(arr_y)]
(const concurrency::index<1> idx) restrict(amp,cpu)
{
    y_view[idx] = a * x_view[idx] + y_view[idx];
});
```

Open Multi-Processing

- Handled by an independant consortium (OpenMP Architecture Review Board)
- Originally, a NUMA-aware CPU parallel technology
- Since 2013 and its version 4.0 it supports GPU offload
- Open standard, anyone can implement it
- C, C++, Fortran languages supported through #pragma directives

Open Accelerators

- Nvidia initiative to create an OpenMP-like pragma driven C, C++, Fortran compatible GPGPU API
- Open standard, anyone can implement it
- In 2013 a study layed the foundations of how to merge OpenACC contstructs into OpenMP
  - Perhaps some later version will feature the fruits
- The newest 2.0 version is supported by GCC 5 and up
  - Both MP and ACC may be compiled to PTX or HSAIL

Open MP/ACC

```
double a = 2.0;
std::vector<double> x(length, 1.0);
std::vector<double> y(length, 1.0);

#pragma omp target
{
    #pragma omp parallel for
    for (int i = 0; i < length; ++i) y[i] = a * x[i] + y[i];
}
```

SYCL

- Following up on the criticism OpenCL has recieved, SYCL is a template library
  - v1.2 public beta may be downloaded here
  - v2.2 reference implementation may be obtained from Github
- Host and device code separate through the use of the template library
  - No language extension!
  - Pure CPU implementation may exist (triSYCL is such an implementation)
- The means of compiling is not specified, though the format of the injected intermediate is
  - Under the hood relies on a functioning OpenCL implementation

SYCL

```
double a = 2.0;
std::vector<double> x(length, 1.0);
std::vector<double> y(length, 1.0);

cl::sycl::buffer<double, 1> x_buf(x.data(), cl::sycl::range<1> { length });
cl::sycl::buffer<double, 1> y_buf(y.data(), cl::sycl::range<1> { length });

cl::sycl::queue().submit([&](cl::sycl::handler &cgh)
{
    auto x_view = x_buf.get_access<access::mode::read>(cgh);
    auto y_view = y_buf.get_access<access::mode::readwrite>(cgh);

    cgh.parallel_for<class saxpy>(cl::sycl::range<1>{ length },
                                  [=](cl::sycl::id<1> idx)
    {
        y_view[index] = a *  x_view[index] + y_view[index];
    });
});
```

Heterogeneous Compute Compiler

- AMD at the end of 2015 announced their Boltzmann Initiative which consists of multiple tools
  - ROCm (Radeon Open Compute), a set of middleware tools (languages/compilers) which serve as the basis of interaction between software and hardware
  - HCC is a Clang fork, which is capable of consuming multiple input APIs and emit code for the ROCm middleware
    - C++AMP 1.2, OpenMP 3.1, Parallel STL
- Open platform, can be implemented by anyone

Heterogeneous Compute Compiler

```
double a = 2.0;
std::vector<double> x(length, 1.0);
std::vector<double> y(length, 1.0);

hc::array_view<double, 1> x_view(length, x.data());
hc::array_view<double, 1> y_view(length, y.data());

hc::parallel_for_each(hc::extent<1>(length),
                      [=](hc::index<1> i) [[hc]]
{
    y_view[i] = a * x_view[i] + y_view[i];
});
```

Questions?

THANK YOU FOR YOUR ATTENTION