Writing elegant host-side CUDA code in Modern C++

Eyal Rozenberg

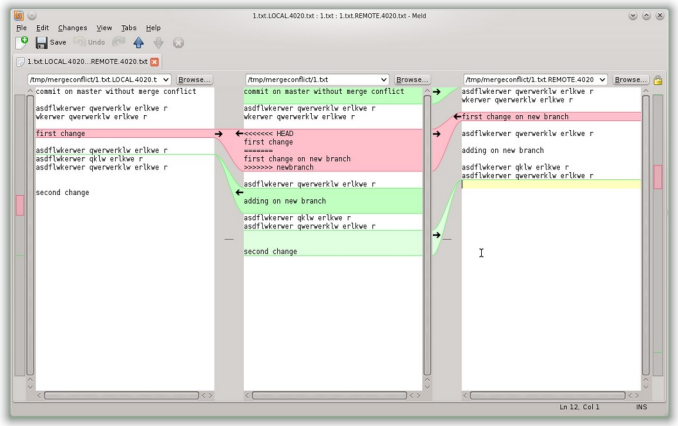
Target audience and difficulty level

- Assumed prior knowledge:
  - You're generally familiar with CUDA
  - You've written a kernel or two, and the code which launches it
  - You have some experience writing C++ ("modern" or otherwise)
- That's it!
- Advanced / experienced developers will realize the potential through the simpler examples.
- We can dive in deeper during the Q&A session

Session overview

We are going to:

- Recap some issues with the CUDA host-side API
- Focus on a sample CUDA program
- Perform a sequence of transformations
  - Some using the C++ CUDA API wrappers, some are more generic
- Consider the difference between successive versions
- Appreciate the final resulting program
- Focus on whatever you like in a Q&A segment



Issues with CUDA’s existing API

- (Almost) all API calls return a status/error indicator:
  - You need an extra block for every call, to check the return value
  - … or you an ugly macro to wrap your calls:

```
#define gpuErrchk(ans) { gpuAssert((ans), __FILE__, __LINE__); }
```

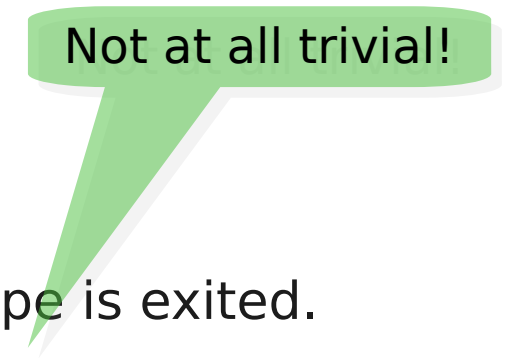
  - You can't chain calls within expressions, i.e. nothing like

```
current_device().properties().compute_capability().major
```

Issues with CUDA's existing API (cont...)

- No automatic scoped resources (RAII/CADRe), so whenever you:
  - Allocate memory
  - Create a CUDA stream
  - Create a CUDA event
  - (other driver-related stateful activity)

  ... you must ensure they are released when the scope is exited.



Issues with CUDA's existing API (cont...)

- Long similar-sounding names with no namespaces.
  - Example: There are 19 different functions API functions of the form `cudaMemcpyXXXXX()` :
    - `cudaMemcpy2DArrayToArray()`
    - `cudaMemcpyToArrayAsync()`
    - etc.
- Numeric IDs of the same type for different entities
  - Good luck not mixing up IDs of devices, streams, events etc.

Comparison with device-side CUDA

- CUDA device-side code has been C++'ish for nearly a decade
  - Templates, namespaces, classes, references, dependent types etc.
- Partial support for modern C++ is introduced gradually:
  - C++11: CUDA 7, 8.
  - C++14: CUDA 8, 9.
  - C++17: CUDA 11.
- It is now common to use: ranged-for loops, lambdas, compile-time computation with constexpr etc.
- There is room for improvement in supporting templated kernels.

Subject for
a different talk
some other time.

A solution

- I was rather annoyed with this situation when I started working with GPUs.
- Eventually found myself writing a few convenience wrappers
  - … while at CWI, Amste
- Finally decided to make into a proper FOSS library and release it:

```
https://github.com/eyalroz/cuda-api-wrappers/
```

- It's a C++11 library, to maximize usability.
- Not perfect (yet) – for many reasons, including:
  - No explicit support for task graphs
  - No support for OpenGL, EGL & Direct3D inter-operability
  - Not enough example programs and missing unit tests

Time to do something about it!

Let's look at some code...

Questions?

Q&A Time...

`https://github.com/eyalroz/cuda-api-wrappers/`