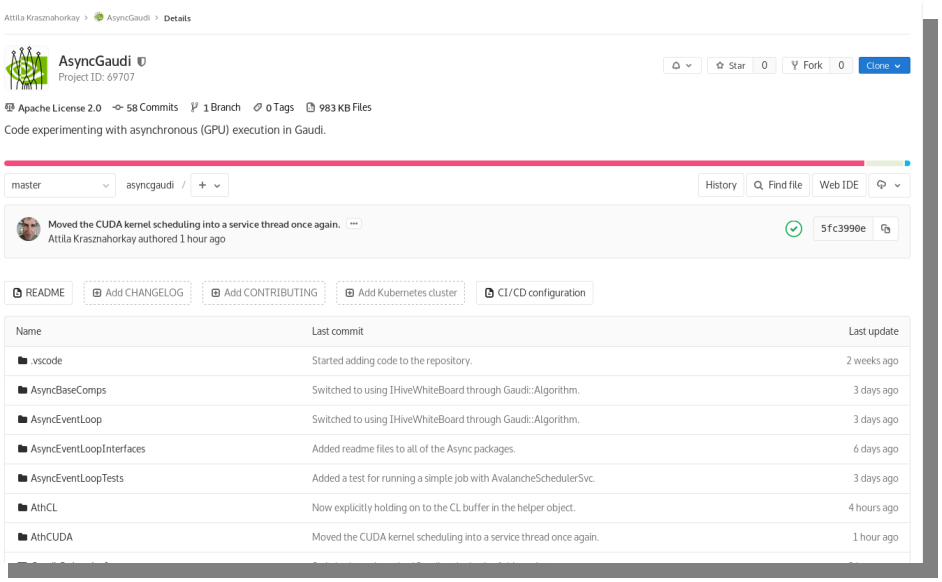
AsyncGaudi

Attila Krasznahorkay

The Repository

- https://gitlab.cern.ch/akraszna/asyncgaudi
- It now holds all of my latest CUDA code, and a bit of OpenCL as well



AthCUDA::KernelRunnerSvc

- It’s this service that does the heavy lifting
  - It can be configured through a few parameters how many tasks it should allow onto the GPU, and how many it should run on the CPU
  - If a task is to be run on the GPU, the function call returns very quickly
    - The actual CUDA kernel launch is scheduled through a TBB task. In such a way that only a single scheduling task would be run at the same time. This way we avoid mutexes used by CUDA, and we use the same thread pool that is used for the Gaudi algorithms as well

```
class KernelRunnerSvc : public extends< Service, IKernelRunnerSvc > {

public:
   /// Inherit the base class's constructor(s)
   using extends::extends;

   /// @name Interface inherited from @c IService
   /// @{

   /// Initialise the service
   virtual StatusCode initialize() override;

   /// Finalise the service
   virtual StatusCode finalize() override;

   /// @}

   /// @name Interface inherited from @c AthCUDA::IKernelRunnerSvc
   /// @{

   /// Execute a user specified kernel task
   ///
   /// If a GPU is available at runtime, and it is not doing other things
   /// at the moment, this function offloads the calculation to the GPU,
   /// and returns right away. The user is expected to return control in the
   /// calling thread to the framework, as the kernel task will notify the
   /// framework when the task gets finished.
   ///
   /// If a GPU is not available for any reason, the function just executes
   /// the task on the CPU in the caller thread, and returns only once the
   /// task is finished.
   ///
   /// @param task The task to be executed on the CPU or GPU
   /// @return A code describing what happened to the task
   ///
   virtual StatusCode
   execute( std::unique_ptr< IKernelTask > task ) override;

   /// @}
```

Defining The Test

- Now we need to define what sort of test we want to run
- Currently my test executable (https://gitlab.cern.ch/akraszna/asyncgaudi/blob/master/AthCUDA/AthCUDATests/util/cpuGpuBurn.cxx) only runs 2 algorithms. One that produces some “test particles” very quickly, and one that performs a dummy calculation on them.
- Since AvalancheSchedulerSvc can not balance a single algorithm that would run on the CPU in one event and on the GPU in another, as async::SchedulerSvc can, we need to find some fixed configuration
- Like: X CPU algorithms that run for X, Y and Z amount of time, and Y GPU algorithms that perform X, Y and Z number of floating point operations