



C2P2 Annual Seminar

Jason Thurow

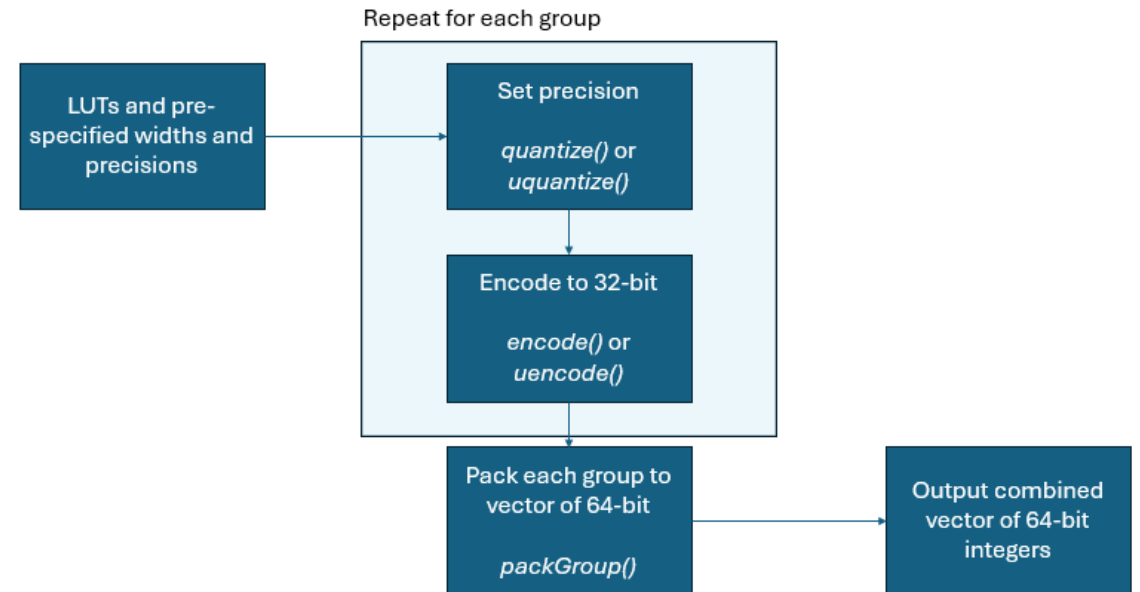
8/10/26

Advised by: Jahred Adelman (NIU)

Haider Abidi (BNL)

Summary of FPGA Work

- Development of Field Programmable Gate Array (FPGA) Pipelines for ATLAS EFTracking (Trigger)
- My project focused on converting Look-up Tables (LUTs) to fixed-point representation within Athena



After Last Year's Seminar

- Getting the fixed-point representation framework to work on real FPGAs on the EFTracking Testbed
- Modified existing EFTracking kernel launching software residing in athena to use my program
- The kernel itself does nothing, but serves as proof of concept that the correct vector inputs can be sent to FPGAs!
- Good stopping point, as EFTracking decided not to use FPGAs moving forward

New Project

- From Technology Choice Committee Report

Based on the evaluation of the pipelines, considering how they could be expected to develop in the future, and the balance of risks and opportunities of the technologies, **the Technology Choice Committee recommends a heterogeneous EF farm based on CPUs and GPUs** as the best technology solution for ATLAS. The committee emphasizes that the engagement and focus of the community will be needed to realise this.

- Working on GPUs now!

Project Overview

- There currently exist no GPU Compute Performance Monitoring tools for the EFTracking pipelines, and they will definitely be needed to improve performance, make debugging easier, and inform logistical decisions
- After consulting with other ATLAS collaborators, it seems best to pursue developing these tools inside the existing Athena PerfMon framework

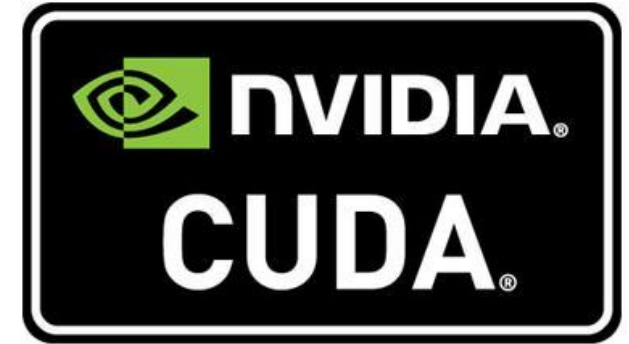


ATLAS EF Tracking / EFTRACK-961 Compute and Physics performance / EFTRACK-1011

Athena Monitoring of GPU tracking pipelines

Challenges

- PerfMon has no existing GPU performance monitoring, so this needs to be constructed from scratch
- Additionally, we need to be able to monitor different brands of GPUs, so we must develop for CUDA, SYCL, etc.
- However, PerfMon should remain, at least without a flag, independent of HPC tool. Loading CUDA dependencies when not needed would defeat the purpose of lightweight monitoring



Getting Started

- To begin, it was useful to try monitoring a simple example kernel that already exists in Athena, especially because the EFTracking pipelines were not in Athena at this time
- Developing for CUDA first
- CUDA has easy built-in tools for performance monitoring, I have experience running the CUDA pipelines, and a simple matrix multiplication example, AthExCUDA, already exists

Memory Monitoring

- Fortunately, CUDA has a way to instantaneously measure memory, which fits perfectly into how PerfMon measures changes in CPU memory by algorithm
- All that is needed is a new function and extensions to the PerfMon component data structure

```
inline double PMonMT::get_gpu_vmem() {  
    size_t freeMem, totalMem;  
    cudaMemGetInfo(&freeMem, &totalMem);  
    return (totalMem - freeMem)/1024.0; //kB  
}
```

GPU Time

- GPU Timing is much trickier
- One might be tempted to use a CPU clock, which is the approach used for PerfMon CPU timing
- However, this breaks when the GPU kernel runs asynchronously while the CPU performs other tasks, which is going to be the case on practically every Trigger job
- Solution: `cudaEventRecord`, which creates an event that can record the timing, only outputting the info when the CPU and GPU resynchronize

```
size_t CUDAEventTool::recordCUDAEventStart(const std::string& name, cudaStream_t stream){
    CUDAEventRecord record;
    record.name = name;

    cudaEventCreate(&record.start);
    cudaEventCreate(&record.stop);

    cudaEventRecord(record.start, stream);

    m_pendingCUDAEvents.push_back(record);
    size_t index = (m_pendingCUDAEvents.size() - 1);
    return index;
}

void CUDAEventTool::recordCUDAEventStop(size_t id, cudaStream_t stream){
    auto& record = m_pendingCUDAEvents[id];

    cudaEventRecord(record.stop, stream);

    record.stopped = true;
}
```

Multi-threading

- Unfortunately, the original setup I had is not thread safe, as any thread could modify the vector storing the events simultaneously
- Migrated into an Athena tool, as seen on previous slide
- Current goal is to make this thread safe, either by having a separate tool instance on each thread, or by indexing the event list by slot so that each slot can only modify its own vector

Questions?

Please feel free to send questions to
z1924270@students.niu.edu or anywhere else you can find
me! (Mattermost, hallway at NIU, etc.)