GPU Usage Status & Plans in ATLAS

Attila Krasznahorkay,
on behalf of a lot of people

Accelerators in ATLAS

- ATLAS does not use any accelerators in central production (yet)
  - Individual physics analysers may use GPU assisted deep learning tools and similar methods, but nothing is done in an “organised” way just yet
- The last round of evaluation for using GPUs was done during LHC’s Long Shutdown 1 (2013-2015)
  - Evaluating practically just CUDA at that time
  - Back then the conclusion was not to invest manpower into re-writing a significant amount of our software for GPUs
- What changed since?
  - At many computing centres we will start getting GPUs whether we explicitly asked for them or not 😛
  - Our build system and event data model improved a lot
  - Hopefully the programming models improved as well 😉

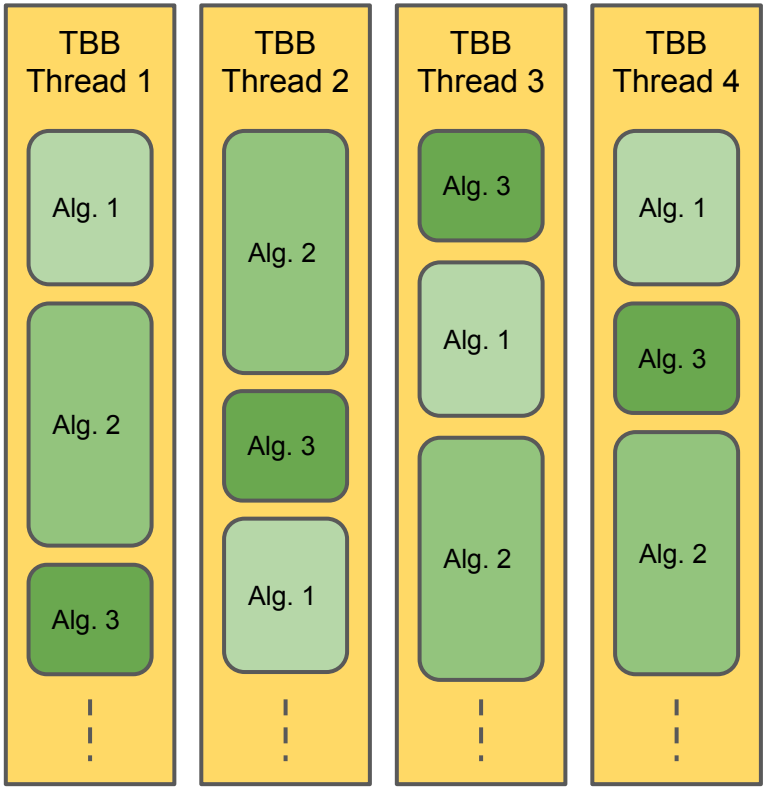
The (Evolving) Computing Landscape

- Is a complicated one…
  - We are clearly moving towards a very heterogeneous environment for the foreseeable future
- Many different accelerators are on the market
  - NVidia GPUs are the most readily available in general, and also used in Summit and Perlmutter
  - AMD GPUs are not used too widely in comparison, but will be in Frontier
  - Intel GPUs are used even less at the moment, but will get center stage in Aurora
  - FPGAs are getting more and more attention, but they come with even more questionmarks…

ATLAS's Priorities

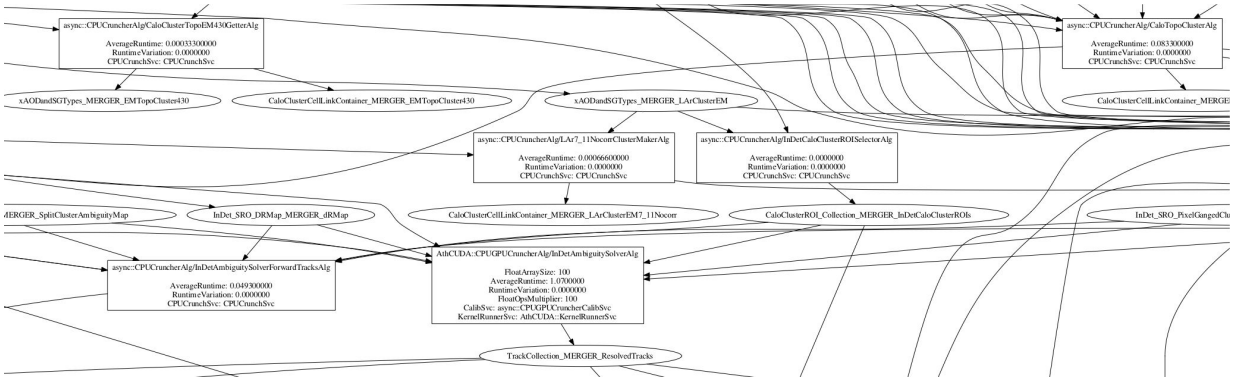
- We "mostly" need to write custom code
  - Machine learning is used very successfully in identifying and calibrating reconstructed objects since a long time. But the inference used there is not a big CPU user in our data processing.
- We want to be able to write our code once
  - And run that single source on as many different hardware backends as possible
  - This is necessary because of the large size of our code (O(4M) lines of C++)
    - We **really** don't want to introduce any code duplication…
- The single source should be able to run "reasonably" on CPU-only hosts as well
  - For the foreseeable future most of our nodes will still not have any accelerators attached to them
- Be able to use "as high level C++" as possible in the code
  - Most of our algorithms are implemented on top of quite high level concepts / objects. The more this can be kept for the accelerated code, the better.

Task Scheduling in AthenaMT



- Athena (Gaudi) uses TBB to execute algorithms on multiple CPU threads in parallel
  - The framework's scheduler takes care of creating TBB tasks that execute algorithms, at the "right times"
- The goal, of course, is to fully utilise all CPU cores assigned to the job, but not to use more
  - So any offloading needs to thoughtfully integrate into this infrastructure

Reconstruction Emulation



- During the development of GaudiHive snapshots were taken of the behaviour of ATLAS reconstruction jobs
  - Recording how algorithms depended on each others' data products, and how long each of them took to run on a reference host
  - The data is still kept in GaudiHive/data/atlas in GraphML + JSON files
- This information was used extensively in the development of the algorithm scheduling code of Gaudi not that long ago
  - And now I taught my project how to construct asynchronous test jobs using it

Reconstruction Emulation Results

Setup	Time [s]
50 events, 8 threads, CPU-only algorithms	68.3 ± 0.47
50 events, 8 threads, 3 “critical-path” CPU/GPU algorithms, run only on CPUs	68.1 ± 0.66
50 events, 8 threads, 3 “critical-path” algorithms offloaded with ideal FPOPS	54.5 ± 0.47
50 events, 8 threads, 3 “critical path” algorithms offloaded with 10x FPOPS	151.2 ± 27.2
50 event, 8 threads, 4 “heavy non-critical-path” algorithms offloaded with ideal FPOPS	49.5 ± 1.51
50 events, 8 threads, 4 “heavy non-critical-path” algorithms offloaded with 3x FPOPS	70.3 ± 10.0

- Did a number of tests…
  - As reference ran jobs with only using the sort of CPU crunching that was developed previously
  - As a validation I exchanged some of the algorithms to run my CPU/GPU crunching code, but running only on the CPU
    - Checking that I’d get the same results as in the first case
  - Finally configured 3 of the CPU intensive reconstruction algorithms to run on the (NVidia) GPU instead
    - Applying also an additional multiplier to the number of FPOPS that they’d have to execute on the GPU

Reconstruction Emulation Results

Setup	Time [s]
50 events, 8 thread algorithms	
50 events, 8 thread "critical-path" CPU/ run only on CPUs	
50 events, 8 thread "critical-path" algori with ideal FPOPS	
50 events, 8 thread path" algorithms off FPOPS	
50 event, 8 threads non-critical-path" al offloaded with ideal	
50 events, 8 threads, 4 "heavy non-critical-path" algorithms offloaded with 3x FPOPS	70.3 ± 10.0

- Did a number of tests

Some takeaways:

- One has to be very careful with offloading algorithms that many other algorithms depend on
  - Making these too slow can cause big issues for the job
- Algorithms off of the "critical path" can handle being executed less efficiently on an accelerator, but not by much
- My ASync::SchedulerSvc code is clearly not scheduling asynchronous algorithms as efficiently as it should at the moment
  - As it turns out, that is **very** important to do, otherwise the job is not able to fill its CPU/GPU resources efficiently.

ly using the
as developed

some of the
PU crunching
CPU
e same results

PU intensive
un on the

onal multiplier
S that they'd
have to execute on the GPU

CUDA, oneAPI, ...

- All performance results shown previously are using CUDA
- We implemented the same tests using oneAPI, and the built-in “Gen 9” GPU of a test machine as well
  - Unfortunately, as expected, it provides significantly lower performance in these synthetic examples ☹️
- We are providing feedback to the oneAPI development team about the issues that we encountered

ACTS, FastCaloSim, etc.

- Investigations are going on in multiple other areas as well
  - In exactly the ones that were discussed yesterday to some extent, using GPUs in Monte-Carlo event generation and fast-simulations
- People are also looking at converting multiple real-life algorithms to run on GPUs
  - Algorithms developed during LS1 for the trigger are being ported to our current Athena/CUDA setup
    - Using simple techniques for now, as used in our Control/AthenaExamples/AthExCUDA "package"
  - The two other examples the most work is going into recently is certain operations in ACTS, and much of the ATLAS FastCaloSim code
  - Hopefully we will be able to provide more information about these to the wider audience in the coming months…

Summary

- ATLAS is now putting effort into heterogeneous computing as well
  - We created new sub-groups in the offline and trigger software areas specifically for this
- Recently most of the developments / investigations went into framework- / core-level code
  - Trying to evaluate multiple programming methods for implementing custom algorithms on accelerators
- Hopefully soon we will be able to show many more results 😉

http://home.cern