Future of Scheduling in Athena

Paolo Calafiura, Julien Esseiva, Xiangyang Ju, Charles Leggett, Beojan Stanislaus, Vakhtang Tsulaia

on behalf of the ATLAS Computing Activity

BERKELEY LAB

CHEP 2024

21st October 2024

ATLAS EXPERIMENT

The HPC Challenge

Traditional Grid (HTC)

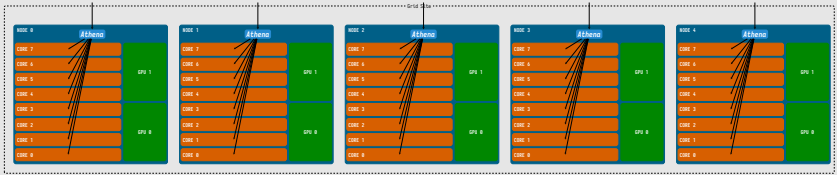
- Each job runs on single node
- Every node has direct outside network access
- Accelerators generally not available

HPC

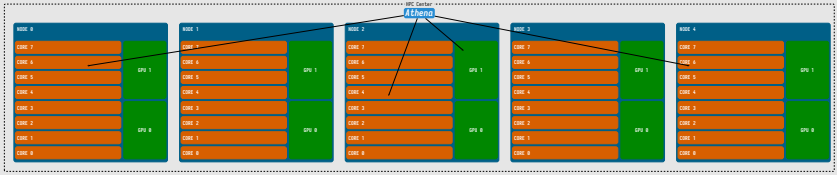
- HPC jobs generally need to run on many nodes
- On many HPCs, nodes can't access outside network
- HPCs often rely on accelerators for majority of FLOPS
  - Can't waste CPU time blocking on accelerator

General Concept

Traditional Grid Sites

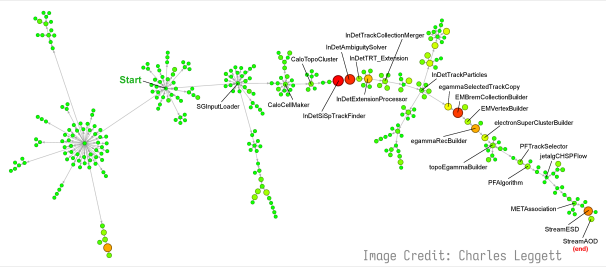


HPC Concept



Athena Scheduling Primer

- Processing represented by Directed Acyclic Graph of *Algorithms* repeated for each event
  - Derived from `Gaudi::Algorithm` (Gaudi)
  - `execute` member function
- AthenaMT exhibits both inter- and intra-event parallelism
  - Two threads may run algorithms from the same, or different events



Implementation

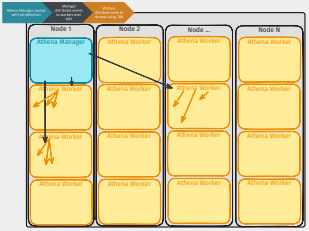
- Can split challenge into two components
  - Distribute work across multiple nodes
    - Avoid starving nodes
  - GPU-aware scheduling
    - Avoid blocking CPU

AthenaMPI

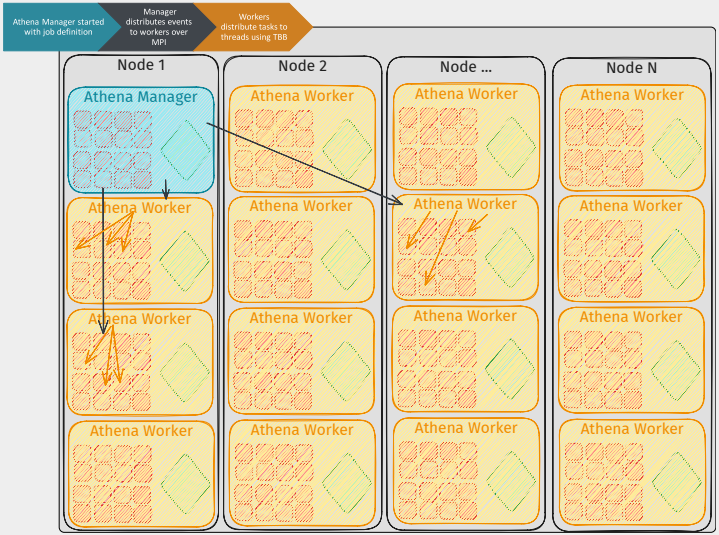
ACAT 2024 Poster

AthenaMPI

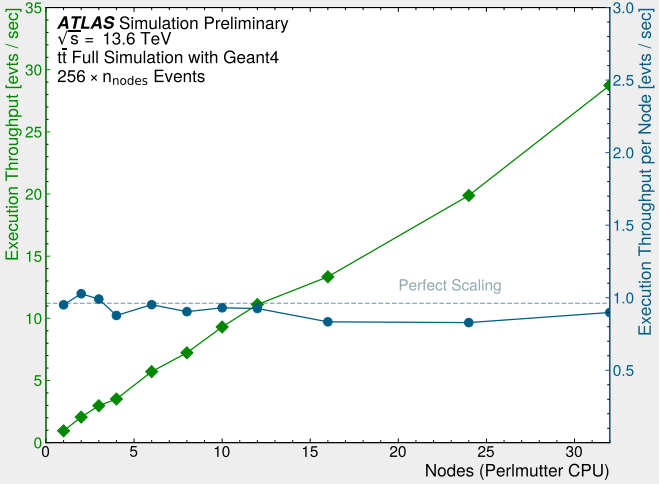
- Solves the problem of distributing work across multiple nodes
- Manager – Worker paradigm with MPI (Message Passing Interface) for communications
- Each worker requests event assignment from manager as needed
  - Pull architecture automatically balances load across workers
  - Avoids trying to schedule onto a busy node
    - Lesson from HPX experience: Not just inefficient, but *slow*



Architecture



Performance



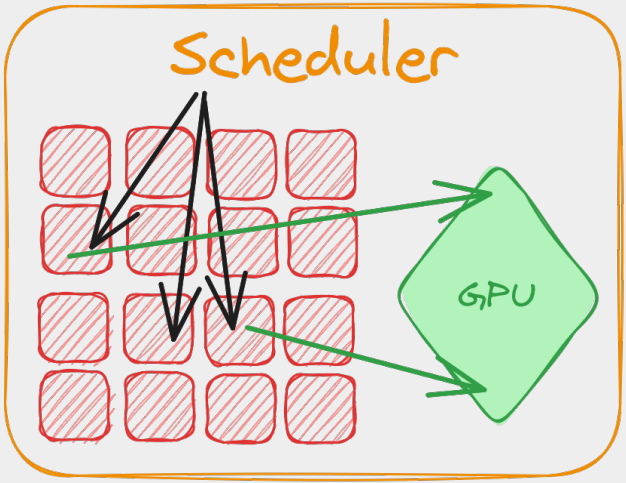
Ideal Scaling Behaviour

Next step: Grid trials

AsynchronousAlgorithm

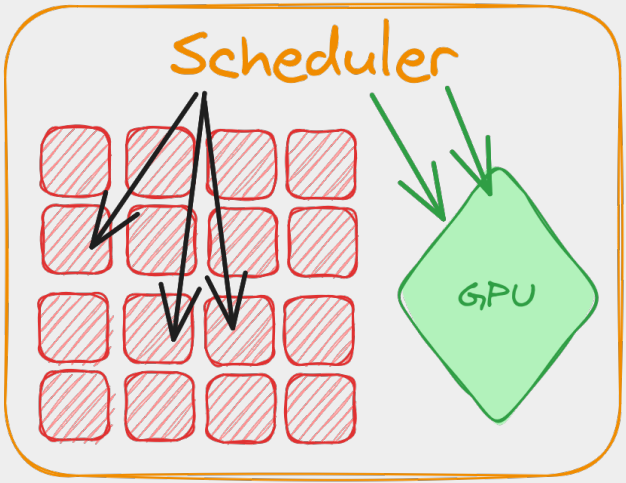
ACAT 2024 Talk

Classic Architecture



Algorithms offload to GPU and block CPU

GPU-Aware Architecture



Algorithms offloading to GPU don’t block

Implementation

- Strictly following GPU-Aware Architecture restricts GPU hardware that can be supported
- Instead implemented a general framework for `AsynchronousAlgorithms`
  - i.e. `Algorithms` with an asynchronous `execute` member function
- Also implemented CUDA support using this framework

Asynchronicity through Fibers

- Chose to use 

  - Fibers are lightweight (user mode) threads
  - Stackful coroutines + scheduler + convenience features

Why not C++20 Coroutines?

- Started before C++20 supported in Athena
- Reduced burden on users
  - No need to “coroutinify” user code
- Reduced development burden
  - Boost Fiber provides a scheduler, and fiber-local storage classes
  - Boost Fiber provides usable CUDA (and HIP) support
- Avoids pitfalls
  - e.g. Running CPU parts on scheduler thread

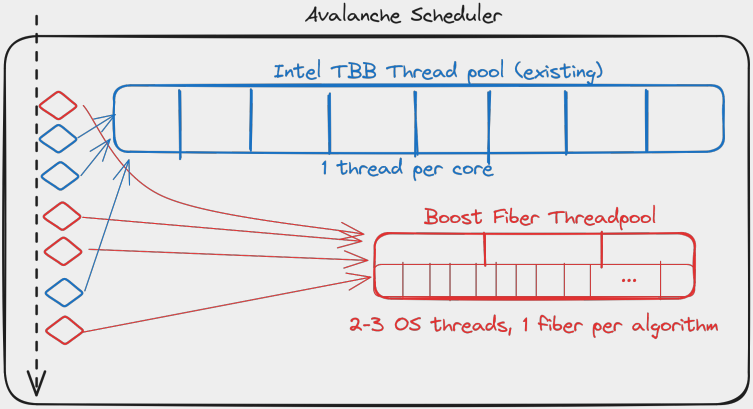
Usage

- Derive from `Gaudi::AsynchronousAlgorithm` instead of `Gaudi::Algorithm`
  - NB for ATLAS folk: Actually `AthAsynchronousAlgorithm` instead of `AthReentrantAlgorithm`
- `yield` after submitting work
- Using provided CUDA support:
  - Use provided support to create a CUDA stream
  - Use CUDA `Async` functions with this stream
  - Call `cuda_stream_await` on stream to yield

Under the Hood

- `AsynchronousAlgorithm`s scheduled on separate small thread pool
- OS thread freed for next fiber whenever you yield
- If CUDA support used, Boost Fiber can handle checking if CUDA is done
  - Fiber not woken to carry out this check
- Use Fiber-specific storage to restore necessary data when fiber resumed
  - `EventContext` and whiteboard information telling us where event data is stored

Scheduling



queue of CPU and GPU tasks

Synchronous → TBB Asynchronous → Fiber

Current Status

- Gaudi prototype demonstrated at ACAT 2024
- Now complete and merged into Gaudi
- Athena interface added and ready
  - Awaiting move to Gaudi v39
- Plan to add support for other hardware (AMD, Intel)
  - Boost Fiber includes HIP (AMD) support analogous to CUDA support
  - SYCL (Intel) support would need to be added using callback support
- Work ongoing to use NVidia's Triton in Athena. `AsynchronousAlgorithms` could be used here too.

Summary

- Athena *must* support HPCs as a first-class citizen
- Unique challenges due to multi-node jobs and accelerator use requirements
- AthenaMPI handles multi-node scheduling
  - About to start grid trials
- `AsynchronousAlgorithm`s handle accelerators
  - Awaiting final merge
  - Further feature development planned

Backup

Perlmutter CPU Node

```
OS: SUSE Linux 15 SP3 x86_64
Host: HPE_CRAY_EX425 1.6.3
Kernel: 5.3.18-150300.59.87_11.0.78-cray_shasta_c
Uptime: 8 days, 23 hours, 30 mins
Packages: 1238 (rpm)
Shell: zsh 5.6
Terminal: /dev/pts/8
CPU: AMD EPYC 7763 (256) @ 2.450GHz
Memory: 24482MiB / 515316MiB
```

History

Athena is ATLAS's offline data processing framework

- Classic Athena is single-process and single-threaded
  - Events are processed in order, one-at-a-time, by the event loop
- AthenaMP developed to control memory use
  - Exploits the `fork` syscall and Linux's copy-on-write behaviour.
    Allows sharing of memory used for conditions *at start*
  - Data doesn't stay de-duplicated
- AthenaMT brought true multithreaded scheduling
  - Single process, single event loop, multiple events in parallel on multiple threads
  - Data shared between events stays de-duplicated
  - Accelerators also on the scene
    - Handled by ad-hoc offloading outside central scheduler
  - Central technology is Intel's TBB (Threaded Building Blocks)

Previous Talks and Posters

Previous Work on This Project

- ACAT 2022 talk
- CHEP 2023 talk
- ACAT 2024 talk and poster

Previous Related Work

- Raythena proceedings from CHEP 2019
- GPU Usage in ATLAS proceedings from CHEP 2019

Other Talks at this Conference

- Machine Learning Inference in Athena with ONNXRuntime (Poster Session)
- AthenaTriton (Earlier Today)