Asynchronous Offloading in Gaudi

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

ACAT 2024

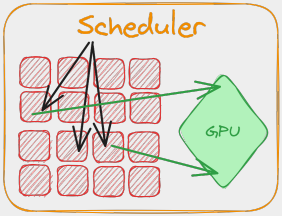
13th March 2024

The Gaudi Framework

- Event processing framework used by ATLAS, LHCb, and others
- Experiment-specific frameworks are layered on top of Gaudi
  - Athena for ATLAS
- Handles basic tasks including work scheduling

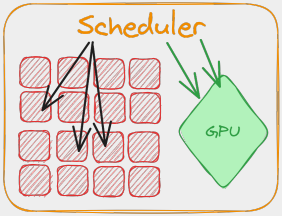
Gaudi on HPCs

- Gaudi designed for High Throughput Computing on CPUs
- Two challenges
  - Efficiently running on multiple nodes – See AthenaMPI poster
  - Handling offloading to GPUs
    - Primary (not sole) purpose of asynchronous offloading



Gaudi on HPCs

- Gaudi designed for High Throughput Computing on CPUs
- Two challenges
  - Efficiently running on multiple nodes – See AthenaMPI poster
  - Handling offloading to GPUs
    - Primary (not sole) purpose of asynchronous offloading



The Avalanche Scheduler

- Gaudi scheduler used by ATLAS
- Schedules work units called `Algorithms`
  - Each algorithm takes *m* inputs and produces *n* outputs
  - No crossing of event boundaries
- Algorithms organized in DAG
  - Defined primarily by data dependencies
    - Some explicit control flow
  - DAG repeated for every event
- Each algorithm-event pair produces TBB task on `task_arena`

Filling the CPU

- Avalanche scheduler keeps CPU busy
- *Assuming algorithms are CPU bound*
- Offloading mostly ad-hoc
  - CPU thread stays busy while algorithm waits for results from CPU
- Need way to free up CPU while waiting for GPU

`AsynchronousAlgorithms`

- Add new type of `Algorithm` – `AsynchronousAlgorithm`
- Informs the scheduler that algorithm offloads to GPU
  - Should not consume significant CPU time
- Scheduled separately
- Functionality to asynchronously wait for GPU

Possible Option: Overcommiting the CPU

- Could run many more CPU threads than cores
  - Threads would yield if still waiting
- Syscalls for context switches are expensive
- No control over spurious wakeups
- Number of concurrent GPU algorithms limited by number of threads

Chosen Option: Fibers (/ lightweight threads / stackful coroutines)

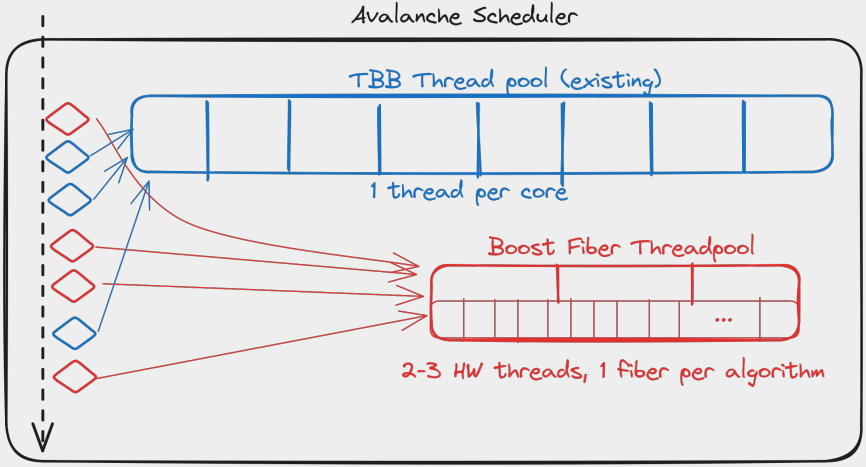
- Similar to threads but entirely in user space
  - Cooperative multitasking – CPU work never interupted to check on GPU
- Usermode context switches are faster

  A context switch between threads costs usually thousands of CPU cycles on x86 compared to a fiber switch with less than 100 cycles.

  — Boost Fiber Documentation

- Runtime can check if GPU is done – No spurious wakeups
- Cheap to create fibers – Create as many as needed

Design



GPU algorithms mapped to fibers

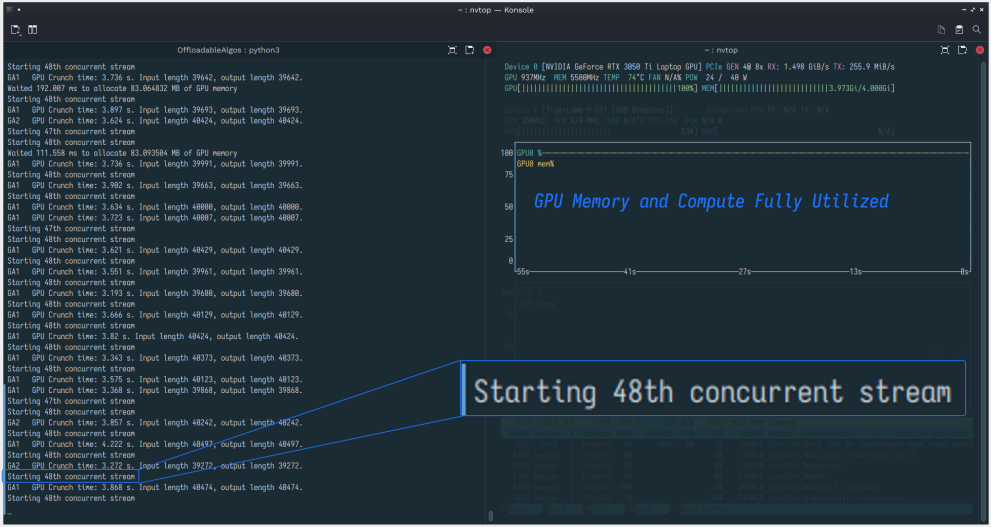
Design Details

- Distinguish CPU and GPU algorithms in scheduler
- Synchronous (CPU) algorithms routed to TBB thread pool
- Asynchronous (GPU) algorithms launch new fibers on small Boost Fiber thread pool
- Asynchronous algorithms suspend (yield CPU) while waiting for GPU
- Boost Fiber only re-schedules suspended fiber if GPU has returned data

CUDA Support

- Not inherently CUDA specific, but implemented support
- Able to await completion of CUDA stream
  - Fiber only awoken if stream is complete
  - Stream creation and reuse is managed
- Utilities to manage memory
  - Portal for allocating GPU memory
    - Throttle tasks if memory is full
  - Use vecmem to wrap pinned memory allocation

Performance



48 simultaneous GPU algorithms on 2 CPU threads

Summary

- `AsynchronousAlgorithm`s allow GPU work to run asynchronously
- Implemented using Boost Fiber
  - Reduces cognitive workload on users
- Allows many simultaneous GPU algorithms on small number of CPU threads

Backup

Why Not Standard (stackless) Coroutines?

- This implementation uses Boost Fiber, not C++ 20 standard coroutines
- Coroutines would require manual implementation of re-scheduling etc.
  - Get this all from Boost Fiber
- Get GPU monitoring “for free”
- Fibers fit well into existing design
  - Pretty similar to TBB tasks
- Easier for users to write code to run in a fiber
  - Can write an ordinary function, not a coroutine

Why Not oneTBB Tasks?

- With oneTBB, can suspend tasks, however:
- Still need to support TBB 2020
- Boost Fiber has built-in support for CUDA (and HIP)
- Useful to separate out CPU work