

The ATLAS multithreaded offline framework

Scott Snyder

On behalf of the ATLAS Collaboration

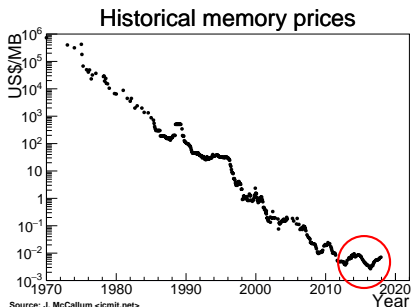
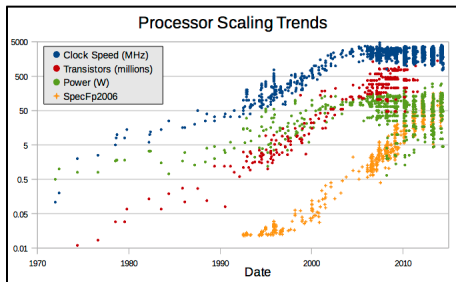
Brookhaven National Laboratory, Upton, NY, USA

July 10, 2018

CHEP 2018

Introduction

- LHC Run 3 will start in 2021, producing more data than ever before.
- Clock speeds have plateaued. Processors are getting wider vector units and more cores.
- Memory prices not been decreasing much recently.
- Expect ratio of memory to cores to decrease.
- Can't continue to keep cores occupied simply by running multiple jobs on one machine. Need to reduce memory required per core.



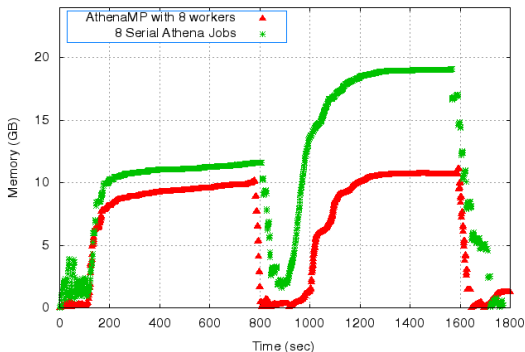
Source: J. McCallum <jcmit.net>

AthenaMP

For Run 2, ATLAS reduced memory requirements via *multiprocessing*.

- A job forks subprocesses to process events in parallel. Memory is shared automatically via copy-on-write.

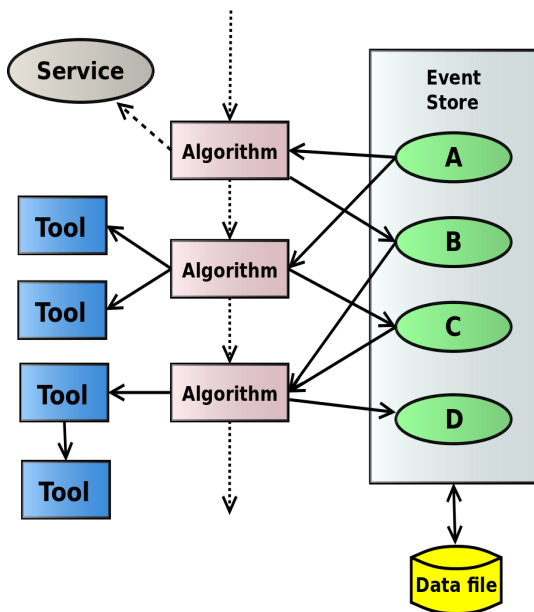
ATLAS Preliminary. Memory Profile of MC Reconstruction



Yields significant memory savings but not sufficient for Run 3.

Go to a fully multithreaded solution.

Athena framework



Based on Gaudi package shared with LHCb and others.

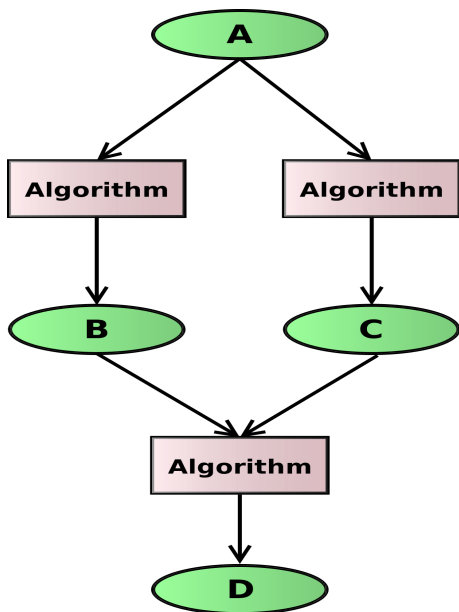
“Whiteboard” pattern

- Sequence of Algorithms communicating via event store.

Algorithms may own Tools and use (singleton) Services.

Fixed Algorithm sequence, set in job configuration.

AthenaMT: Intra-event parallelism



Task scheduling based on the Intel Thread Building Blocks library with a custom graph scheduler.

Algorithms declare their inputs and outputs.

- Scheduler finds an algorithm with all inputs available and runs it as a task.
- “Data flow.”

Flexible parallelism within an event.

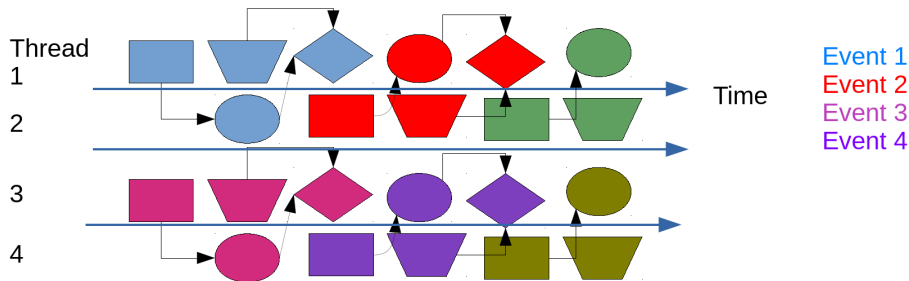
Can still declare sequences of algorithms that must execute in fixed order (“control flow”).

AthenaMT: Inter-event parallelism

Allow *multiple* event stores (“slots”).

Allows parallelism both within and event and between events.

Number of simultaneous events in flight is configurable.



Different shapes: different algorithms; different colors: different events.

Algorithm dependency declarations

Algorithm data dependencies declared via special properties (HandleKey).

```
SG::WriteHandleKey<X> m_xKey { this, "XKey", "x" };
```

Used together with an *event context* that identifies the particular event slot being used:

```
SG::WriteHandle<X> x (m_xKey, ctx);  
ATH_CHECK( x.record (std::make_unique<X>()) );  
// Can modify the object until the WriteHandle is deleted.  
x->set (something);
```

Context argument may be omitted — will then be read from a thread-local global.

Algorithm types

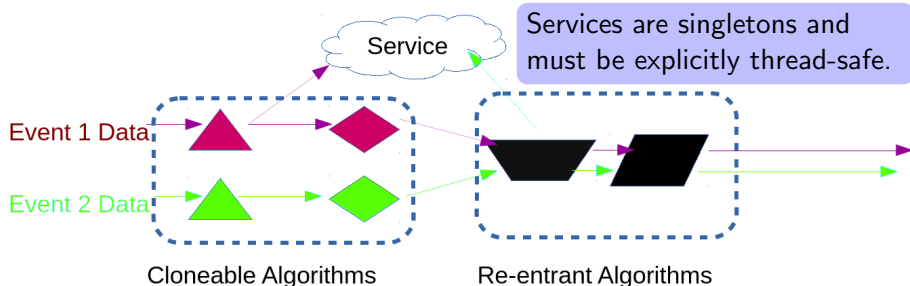
By default Algorithms are singletons, and a given algorithm cannot be executing simultaneously in more than one thread.

Algorithms may be declared *clonable*. Multiple copies of the Algorithm are made and can be executing simultaneously.

```
virtual StatusCode execute();
```

Algorithms declared *reentrant* are singletons but may execute in multiple threads. Any internal mutable state must be thread-safe.

```
virtual StatusCode execute_r(const EventContext&) const;
```



Conditions

Calibrations, etc. depending on event number or time.

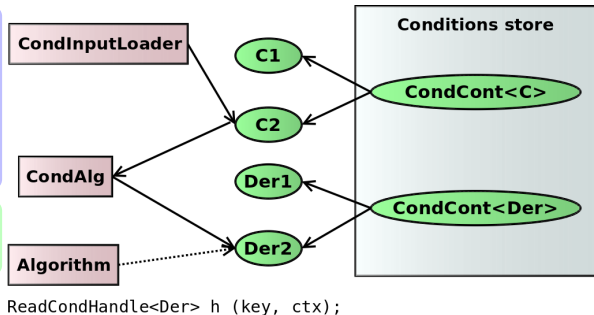
Different events may use different conditions versions.

Conditions store holds potentially multiple versions of conditions objects.

CondInputLoader algorithm loads needed conditions for each event.

To apply a transformation to conditions data, use a 'conditions algorithm' acting on data in the conditions store.

Algorithms access conditions data via handles.



Scheduler knows about dependencies and schedules them accordingly.

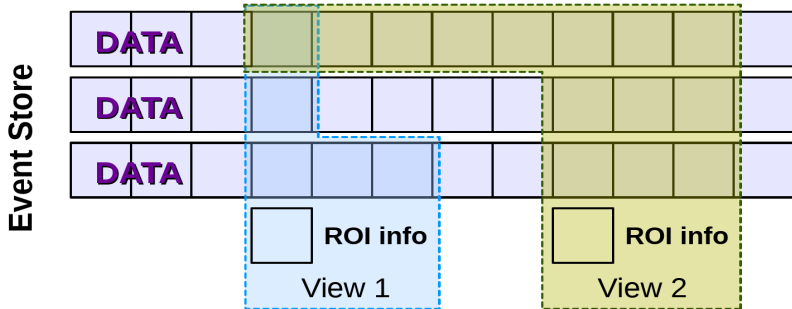
Trigger

Want high-level trigger to use the same algorithms as the offline code.

For performance, trigger does reconstruction only within geometrically limited regions of interest (ROI).

EventView: Stores data for one ROI and implements the same interface as the full event store.

Algorithms that access data via handles can transparently run in an EventView rather than the full store.



Algorithm/tool/service migration

- Algorithms/tools need to change to handles to access event/conditions data.
- Change to using conditions algorithms rather than callbacks.
- Event data must not be modified once recorded in the event store.
- Avoid thread-unfriendly code: use of statics, const-correctness violations.
- Services need to be explicitly thread-safe.
- Reentrant algorithms that have mutable data must be explicitly thread-safe.

Thread-safety static checker

- Have a static checker available to flag some thread-safety problems.
 - ▶ Mostly relating to const-correctness and use of static data.
- Set of checks similar to that done by the CMS checker.
 - ▶ But implemented within gcc rather than clang, so they can run as part of the default build.
- Gives warnings like:

```
ArenaSharedHeapSTLAllocator.icc:499:10: warning: Static
expression 'SG::ArenaSharedHeapSTLAllocator<Payload>::s_index'
passed to pointer or reference function argument of
'SG::ArenaHeapAllocator* SG::ArenaSharedHeapSTLHeader::get_pool(si
within function 'void SG::ArenaSharedHeapSTLAllocator<T>::get_pool
may not be thread-safe.
    m_pool = m_header->get_pool<T> (s_index);
```

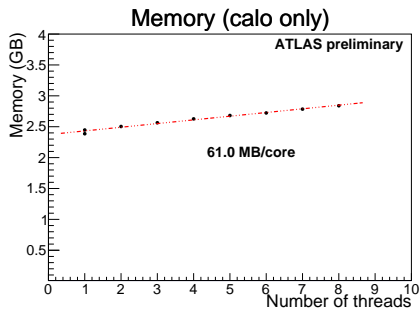
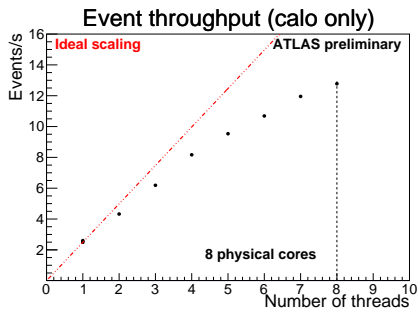
- Also has checks related to naming conventions and other coding style issues.

Status/schedule

- Q4 2017: Migrate all algorithms to use handles.
 - ▶ Largely done. A few stragglers still to fix.
- Q2 2018: Use conditions handles; replace callbacks with conditions algorithms.
 - ▶ Done for many conditions which are expected to actually change during a run for inner detector/calorimeter.
 - ▶ Have successfully run fully MT jobs for calorimeter system reconstruction.
- Q4 2018: Tools are const-correct; services are thread-safe. Raw-data unpacking works in MT jobs.
- Q1 2019: Remove use of incidents.
- 2019: Thread-safety debugging; preliminary validation and performance fixes.
- 2020: Detailed validation and integration. Further performance improvements.
- Q1 2021: Run 3 starts.

Scaling behavior

- Preliminary CPU/memory scaling for calorimeter-only reconstruction with 8 physical cores.
- Writing output is a known bottleneck — disabled for this test.
- CPU scaling promising, but still some instances of lock contention to resolve. Memory increases very modestly with number of threads.



Summary

- ATLAS is redesigning its offline and trigger software to cope with increasing amounts of data and expected future trends in hardware.
- Reconstruction is becoming fully multithreaded.
- Gaudi development reinvigorated with contributions from multiple experiments.
- Migration is well underway.
- Some selected pieces of the reconstruction already work in a fully multithreaded job.
- Expect to have full reconstruction working in 2019 to be ready for Run 3 in early 2021.
- Forms a basis for further evolution for the HL-LHC era and beyond.