Implementation of the ATLAS trigger within the multi-threaded AthenaMT framework

Rafał Bielski (CERN)

on behalf of the ATLAS Collaboration

XXVII International Symposium on Nuclear Electronics and Computing (NEC'2019)

Introduction

- We are 1/3 through LHC Long Shutdown 2
- ATLAS is undergoing **Phase-1** upgrade in hardware and software
- All Phase-1 upgrades are designed to target beyond Run 3 into HL-LHC era
- The upgrade includes major replacements in Trigger hardware and software
- This talk covers the **ATLAS Trigger software** upgrade

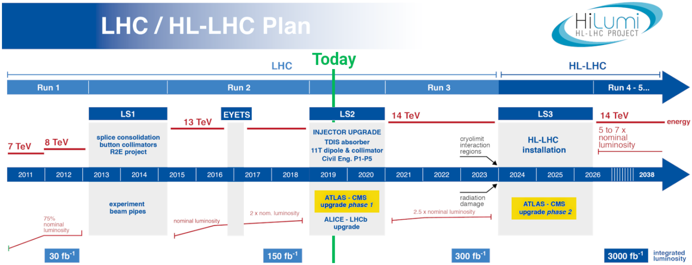


Image ref [a]

ATLAS TDAQ system – Run 2

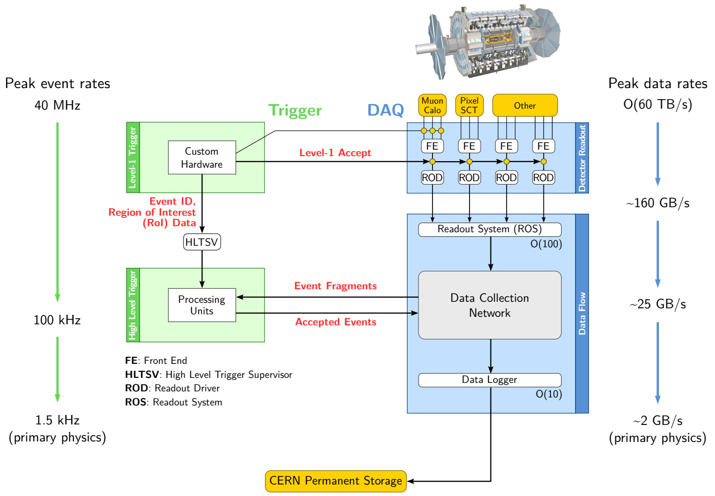
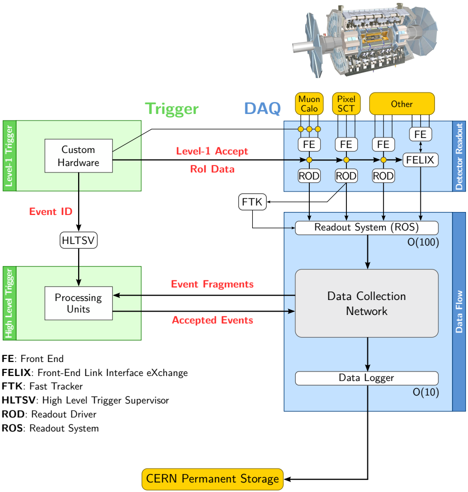


Image ref [b]

ATLAS TDAQ system – Run 3

Level-1 Trigger Hardware Upgrade

High Level Trigger Software Upgrade – focus of this talk

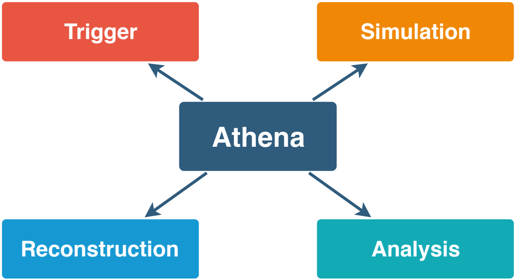


New detectors and Level-1 systems use new readout

No major changes in Data Flow infrastructure

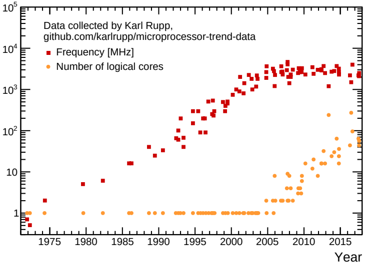
Image ref [b]

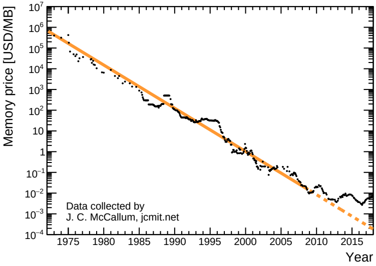
ATLAS Athena software framework



- Athena [1] is a multi-purpose data processing framework of the ATLAS Experiment
- Based on Gaudi [2] – core framework shared with LHCb
- Designed in early 2000s without multi-threading in mind
- Used successfully throughout Runs 1 and 2

Towards AthenaMT

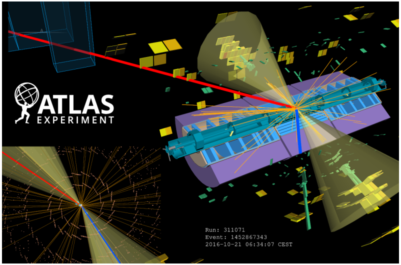




- Modern computing hardware provides limited memory per core
- Already in Run 2, ATLAS struggled to use its offline processing resources efficiently with Athena
- Forking Athena processes was a stopgap solution to reduce memory needs (thanks to copy-on-write)
- Ultimate solution: redesign the core framework for native, efficient and user-friendly multi-threading support → **AthenaMT**
- Core developments started in Gaudi around 2015 (Gaudi Hive) [3]
- Parallel task execution exploiting Intel Thread Building Blocks (TBB) [4]

ATLAS High Level Trigger

ATLAS HLT Software



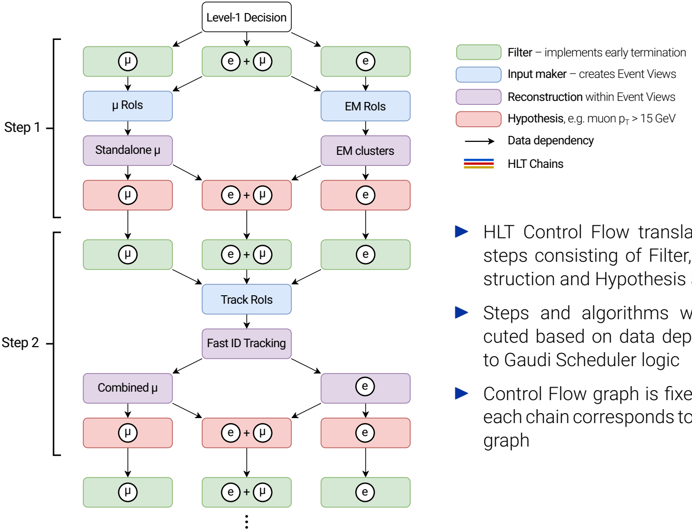
- Needs to reconstruct physics objects and take decision within ∼0.5 s
- Part of Athena, sharing some code with offline reconstruction, but cannot afford to reconstruct full event (up to 30 s)
- Achieves the goal with **partial event reconstruction** in Regions of Interest (RoI) and **early rejection**
- RoI are defined as geometrical cones around a track or cluster starting in collision point
- In Run 2, HLT implemented custom trigger algorithm scheduling and data caching system

HLT in AthenaMT

- Major rewrite of HLT software for deep integration with Gaudi Hive and AthenaMT
- HLT requirements considered during design of AthenaMT from the beginning
- Replacing own scheduling and caching by native Gaudi Scheduler, but still aided by HLT-specific **Control Flow** logic to ensure early termination
- Partial event reconstruction provided by **Event Views**

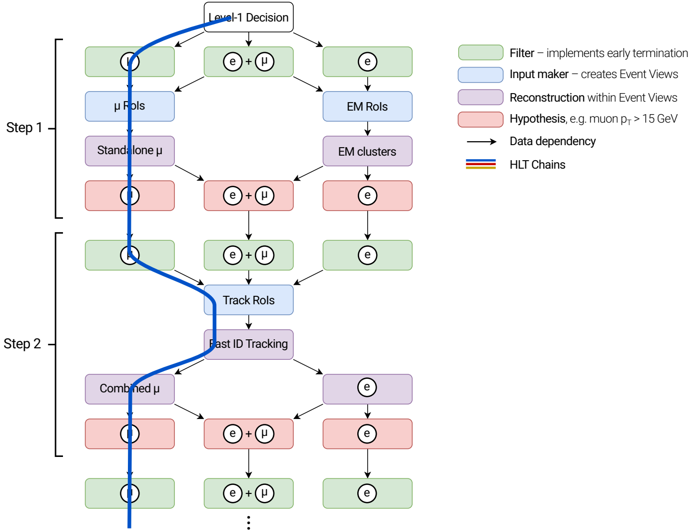
Image ref [c]

HLT Control Flow

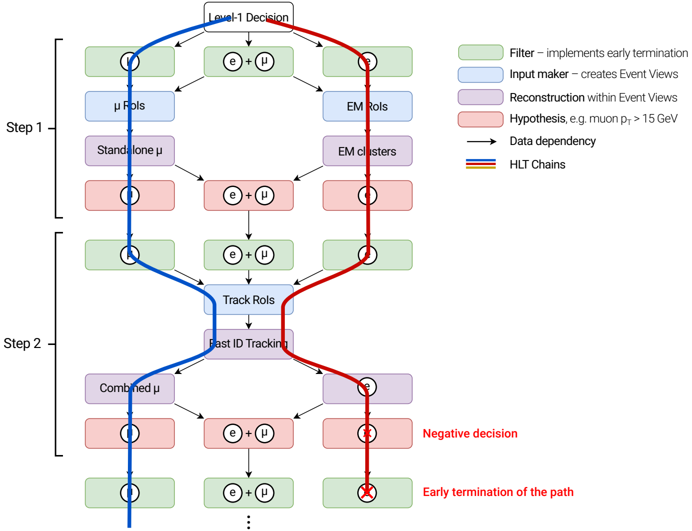


- HLT Control Flow translates HLT Chains into steps consisting of Filter, Input Maker, Reconstruction and Hypothesis algorithms
- Steps and algorithms within steps are executed based on data dependencies according to Gaudi Scheduler logic
- Control Flow graph is fixed in initialisation and each chain corresponds to one path through the graph

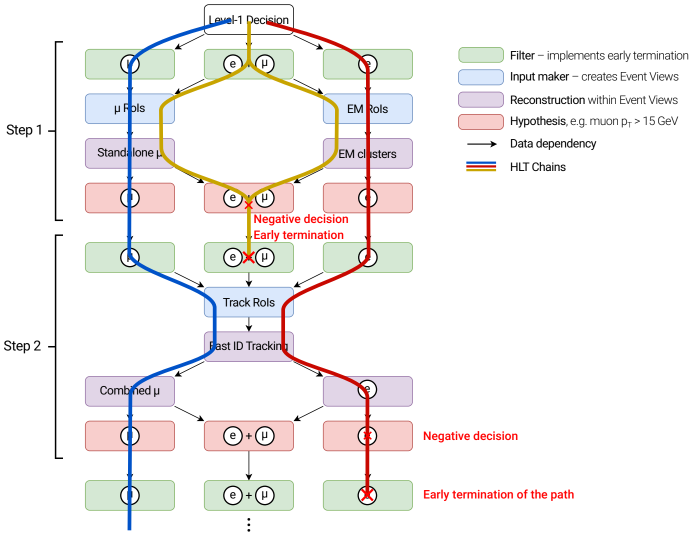
HLT Control Flow



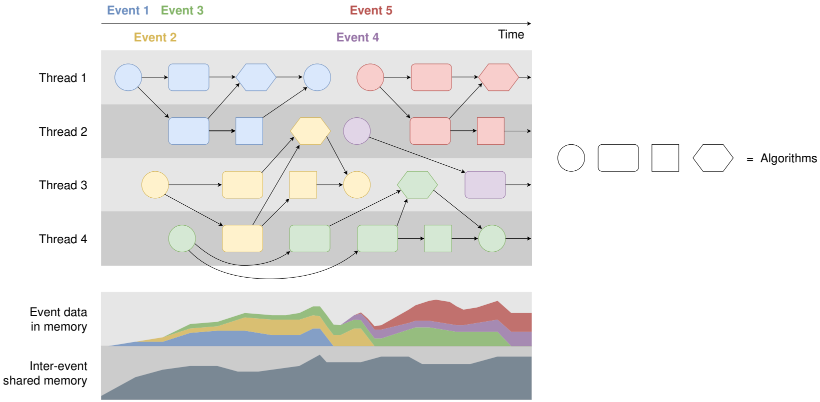
HLT Control Flow



HLT Control Flow



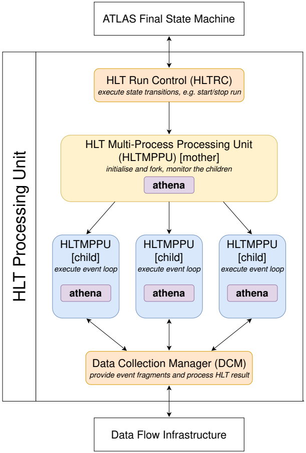
Parallel processing

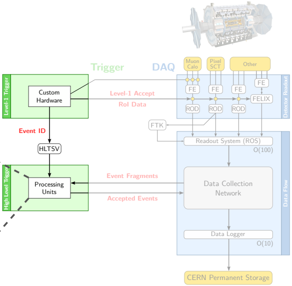


AthenaMT offers three kinds of parallelism

- inter-event: multiple events are processed in parallel
- intra-event: multiple algorithms can run in parallel for an event
- in-algorithm: algorithms can utilize multi-threading and vectorisation

Parallel processing

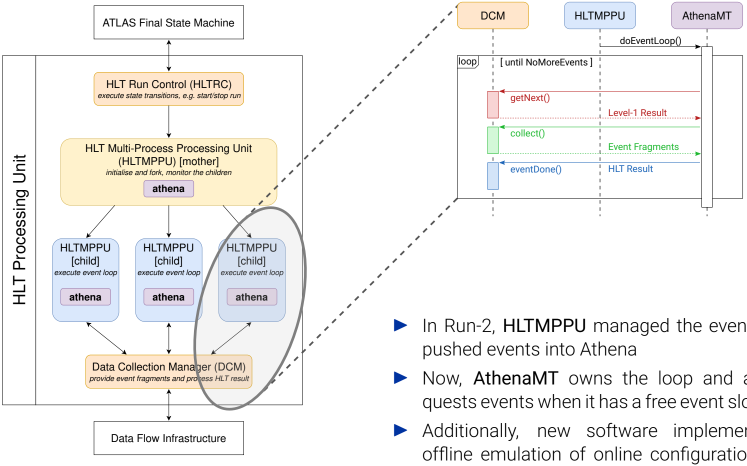




HLT within TDAQ software offers one more level

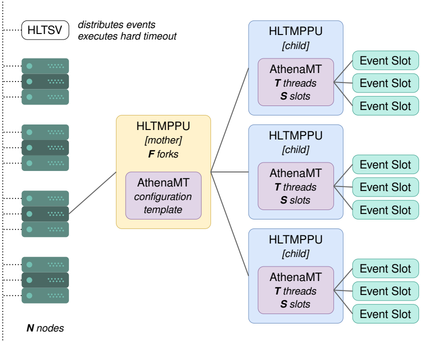
- In Run-2, forking HLT Athena processes was implemented by **HLTMPPU**
- The infrastructure will not change for Run-3, but interfaces between Athena, HLTMPPU and DCM need to change
- Varying number of **forks**, **parallel events** per fork and **threads** per fork, provides large flexibility in optimising the stability and performance of the system

HLT online data flow



- In Run-2, **HLTMPPU** managed the event loop and pushed events into Athena
- Now, **AthenaMT** owns the loop and actively requests events when it has a free event slot available
- Additionally, new software implements better offline emulation of online configuration → facilitates development and testing

Operational parameters and error handling



- The HLT farm consists of **N** $\sim$ 2500 nodes, each running one HLTMPPU mother process
- Total number of events the farm can process simultaneously is **N**odes$\times$**F**orks$\times$**S**lots $\sim$ 50k
- Parameters may differ between racks, depending on CPU configuration
- Configuration has to be optimised not only for performance, but also for stability

- In case of an error in an algorithm, the event is **force-accepted into a special "debug" stream** and the processing continues with the next event
- In case of a crash or hard timeout in one event, **S** events from a child are force-accepted
- In case of communication errors, **F**$\times$**S** events belonging to one mother may be force-accepted
- Size of the per-fork thread pool **T** affects performance but not error handling

Image ref [d]

Status and conclusions

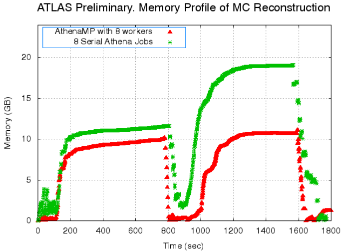
- ATLAS High Level Trigger is undergoing a major update integrating it with AthenaMT
- Most core functionalities are already in place and effort is now put into adapting reconstruction and selection algorithms
- Large-scale tests on HLT farm with AthenaMT already ongoing and more planned
  - Completed tests with multiple nodes using replayed 2018 data
  - Proved the ability to run simple algorithms in multiple forks/slots/threads configuration
  - Running substantial set of physics selections planned in the coming weeks
  - Full-detector tests planned in 2020 including old and new Level-1 hardware
- Performance measurements and parameter optimisation needed at start of Run 3
- ATLAS is on a good path towards running multi-threaded HLT in 2021

back-up

Memory sharing

Multi-process approach

- Memory is shared by forked (cloned) processes thanks to copy-on-write technique implemented by Linux kernel
- All memory is shared as long as it is **read-only**
- First memory write operation of a forked process clones a given memory page
- Athena processes cloned after initialisation share large read-only data like magnetic field maps



Multi-threaded approach

- Single process running several threads has to manage the memory usage between threads itself
- All memory can be shared, including **writeable** memory
- Concurrent write operations have to be carefully managed to avoid corruption of data in memory
- Locking too large blocks of code may lead to inefficient CPU usage
- More efficient use of memory if the software supports it
- AthenaMT can implement common caches between concurrently processed events to optimise memory usage
- Example of writeable shared memory: time-dependent conditions like luminosity value or beam spot position

Image ref [e]

Implementation details

Parallelism

- Algorithms in AthenaMT can be declared as:
  - **reentrant** – one instance can be executed simultaneously with different inputs by multiple threads
  - **clonable** – one instance can be executed only once, but multiple instances can run in parallel
  - non-reentrant and non-clonable (should be avoided)
- Data sharing in memory is done via explicitly thread-safe singleton services, in particular “stores” – event store, conditions store, detector store

EventView

- Stores data for one RoI and implements the same interface as the full event store
- Algorithms that access data via DataHandles can transparently run in an EventView rather than the full store

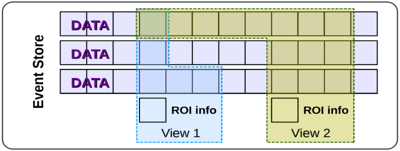


Image ref [f]

References

Software citations

1. ATLAS Collaboration, Athena [software] Release 22.0.1, 2019, `doi.org/10.5281/zenodo.2641997`
2. Barrand G and others, *GAUDI – A software architecture and framework for building HEP data processing applications*, 2001, Comput.Phys.Commun. 14045–55. See also Gaudi [software] Release v31r0 `gitlab.cern.ch/gaudi/Gaudi/tags/v31r0`
3. Clemencic M, Funke D, Hegner B, Mato P, Piparo D and Shapoval I, *Gaudi components for concurrency: Concurrency for existing and future experiments*, 2015, J.Phys.Conf.Ser. 608 012021
4. Reinders J, *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism* 2007, O'Reilly Media. See also TBB [software] Release 2019_U1 `github.com/intel/tbb/tree/2019_U1`

Image sources

a Based on image from `project-hl-lhc-industry.web.cern.ch/content/project-schedule` [access 2019-09-06]

b Image and its variants in further slides based on image from `twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ`

c Image from `twiki.cern.ch/twiki/bin/view/AtlasPublic/EventDisplayRun2Physics`

d Original image, makes use of an icon made by Smashicons from `flaticon.com`

e Image from `https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults`

f Snyder S (ATLAS Collaboration), *The ATLAS multithreaded offline framework*, 2018, ATL-SOFT-SLIDE-2018-430