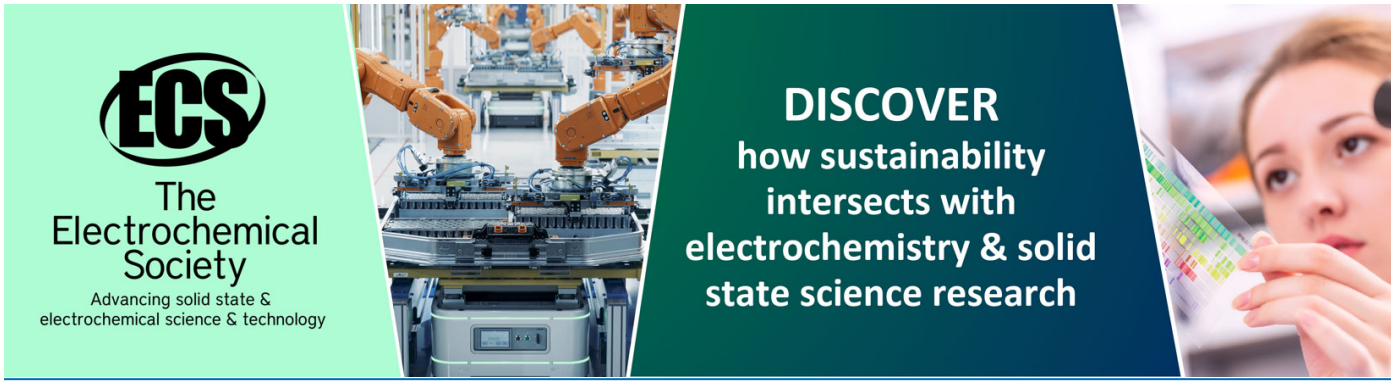
Implementation of the ATLAS trigger within the multi-threaded software framework AthenaMT



IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 032002 doi:10.1088/1742-6596/898/3/032002

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

Ben Wynne on behalf of the ATLAS Collaboration.
James Clerk Maxwell Building, Peter Guthrie Tait Rd., Edinburgh, EH9 3FD, UK
E-mail: bwynne@cern.ch

Abstract. We present an implementation of the ATLAS High Level Trigger, HLT, that provides parallel execution of trigger algorithms within the ATLAS multithreaded software framework, AthenaMT. This development will enable the ATLAS HLT to meet future challenges due to the evolution of computing hardware and upgrades of the Large Hadron Collider, LHC, and ATLAS Detector. During the LHC data-taking period starting in 2021, luminosity will reach up to three times the original design value. Luminosity will increase further, to up to 7.5 times the design value, in 2026 following LHC and ATLAS upgrades. This includes an upgrade of the ATLAS trigger architecture that will result in an increase in the HLT input rate by a factor of 4 to 10 compared to the current maximum rate of 100 kHz. The current ATLAS multiprocess framework, AthenaMP, manages a number of processes that each execute algorithms sequentially for different events. AthenaMT will provide a fully multi-threaded environment that will additionally enable concurrent execution of algorithms within an event. This has the potential to significantly reduce the memory footprint on future manycore devices. An additional benefit of the HLT implementation within AthenaMT is that it facilitates the integration of offline code into the HLT. The trigger must retain high rejection in the face of increasing numbers of pileup collisions. This will be achieved by greater use of offline algorithms that are designed to maximize the discrimination of signal from background. Therefore a unification of the HLT and offline reconstruction software environment is required. This has been achieved while at the same time retaining important HLT-specific optimisations that minimize the computation performed to reach a trigger decision. Such optimizations include early event rejection and reconstruction within restricted geometrical regions. We report on an HLT prototype in which the need for HLT-specific components has been reduced to a minimum. Promising results have been obtained with a prototype that includes the key elements of trigger functionality including regional reconstruction and early event rejection. We report on the first experience of migrating trigger selections to this new framework and present the next steps towards a full implementation of the ATLAS trigger.

1. Introduction

Two trends are driving the development of software for the ATLAS [1] experiment at the LHC, particularly for the ATLAS High Level Trigger (HLT). Firstly, the planned LHC upgrades will have the instantaneous luminosity increase to 3 times the design value (of $10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ [2]) by 2021, and 7.5 times by 2026. This implies that the HLT must handle between 4 and 10 times its current input rate. Secondly, the evolution of computer processing hardware is broadly trending towards more cores per device with less memory per core and little change in clock frequency. As a result the HLT will need to utilise multi-threading with efficient memory sharing, and perform stringent background rejection.

2. AthenaMT

The AthenaMT framework is a new development based on the GaudiHive prototype [3] in much the same way as Athena [4] was based on Gaudi [5]. It was developed to fulfill two major goals [6]: to introduce multi-threading to ATLAS software at a high level, and to unify the online and offline processing environments, facilitating code sharing. Consequently the HLT can achieve greater background rejection by making use of the (typically) more detailed offline reconstruction algorithms. When offline code is used in the HLT at present, a 'wrapper' class must be added for compatibility, since the framework itself has been modified to support HLT behaviour. In the future the framework will be identical for online and offline workflows, with HLT-specific behaviour implemented using configurable algorithms. A more detailed comparison of the old and new approaches is shown in Table 1.

Table 1. Athena vs AthenaMT feature comparison

Athena	AthenaMT
Single-threaded • Algorithm sequence	Multi-threaded • Algorithm dependency graph
HLT-specific steering layer • Schedules algorithms • Makes trigger decisions	Common scheduler for HLT and offline Trigger decisions made by algorithms
HLT-specific algorithm class • Enables RoI-based reconstruction	Common algorithm class for HLT and offline • Facilitates code-sharing
	RoI data stored using EventViews (see Section 3) • Accessed or manipulated by any algorithm • HLT-specific information stored as event data

The crucial difference between online and offline processing for the ATLAS experiment is the use of Regions of Interest (RoIs). These are small areas of the detector (in pseudorapidity η, azimuthal angle ϕ, and beam-axis z) where the Level-1 (L1) hardware trigger system has detected a feature for the HLT to examine. By limiting HLT processing to these areas, CPU time and read-out bandwidth requirements are minimised for the majority of events, which will be rejected. Accepted events also require relatively little CPU time, since the accept decision can be made without fully reconstructing the event. The HLT-specific 'steering' layer in the current framework is used to implement the processing of RoIs. In AthenaMT, RoI processing will be achieved using a new component called EventViews.

3. EventViews

AthenaMT executes algorithms in parallel by constructing a directed, acyclic graph[1] of data dependencies, and executing all algorithms that have their input requirements satisfied. It does this by requiring that all data objects are produced and consumed via smart pointers called DataHandles. The scheduler can then query the DataHandles of an algorithm in order to find its dependencies. An additional effect is to remove the need for an algorithm to request a piece

[1] A graph where each edge has a direction, and where navigating the graph cannot lead to the same vertex twice.

of data *from* somewhere in particular: it is simply the responsibility of the framework to provide it.

This allows the re-introduction of RoI processing, by encapsulating all the data for an RoI in a container called an EventView. The EventView interface is compatible with DataHandles, and the framework can use an EventView to provide data to an algorithm in place of the default EventStore object that contains all event data. So, an algorithm using DataHandles requires no modification to perform RoI processing: it will receive the appropriate data via its handles.

EventViews are created dynamically during event processing, by HLT algorithms. The majority of views are expected to correspond to RoIs created by the L1 trigger, and alongside the regional data will also contain an object that describes the RoI itself. There are other potential use-cases — reflected in the existing HLT — where multiple RoIs from L1 might be merged together, for example to reconstruct a B-meson or W/Z-boson. Again, such an EventView will contain a descriptive object as well as the relevant physics data.

EventViews and the algorithms that create them will replace one of the key features of the current HLT-specific framework layer: associating algorithms with RoIs. The other feature needed is the ability to make trigger decisions.

4. Trigger decisions and the menu

When processing an RoI, physics objects are reconstructed by Feature EXtraction (FEX) algorithms, and then selections made on the reconstructed objects by HYPOthesis (HYPO) algorithms. The final result of processing an RoI is a set of booleans representing whether different hypotheses have passed or failed. For example, a FEX algorithm might attempt to reconstruct a jet in an RoI produced by the ATLAS hadronic calorimeter, and the result might then be compared to the hypothesis "this RoI contains a jet with $p_T > 20\,\text{GeV}$." Once all existing RoIs have been processed, and various hypotheses evaluated, the results are compared with the trigger 'menu' — a list of possible criteria for accepting an event. If one or more menu criteria are satified, additional reconstruction of the event might be scheduled or the event might be accepted and stored for offline analysis. The majority of events will not satisfy any menu criteria, and so will be rejected: discarded with no further data processing.

The HLT steering is currently responsible for making trigger decisions based on the menu. In AthenaMT this task will instead be performed by multiple decision-making algorithms, that will each be responsible for a part of the menu. A decision algorithm will take as input all the merged HYPO results from the EventViews that have been created. If the event is not rejected (or accepted for offline analysis) the decision algorithm may be followed by further rounds of view creation and decision-making, as defined by the menu.

Using the objects described above, the design for the ATLAS HLT is as follows. Menu algorithms run in the context of an entire event, creating EventViews, collecting results from them, and using these results to make trigger decisions. Within the event views, unmodified offline algorithms can run as FEX algorithms to reconstruct physics objects, which are then compared to trigger selection criteria by HYPO algorithms. Multiple stages of view creation and trigger decision-making are possible, although a decision to reject the event will interrupt this process. A simplified example workflow is depicted in Figure 1.

5. Summary

The ATLAS collaboration is adopting a new, multi-threaded framework — AthenaMT — in response to the evolution of computing hardware towards greater parallelism. This framework also aims to provide a common environment for online and offline processing, allowing the ATLAS HLT to make greater use of high-precision offline reconstruction algorithms. However, the HLT presently relies on a customised framework layer to implement RoI-based reconstruction, which minimises processing and readout requirements for rejected events.

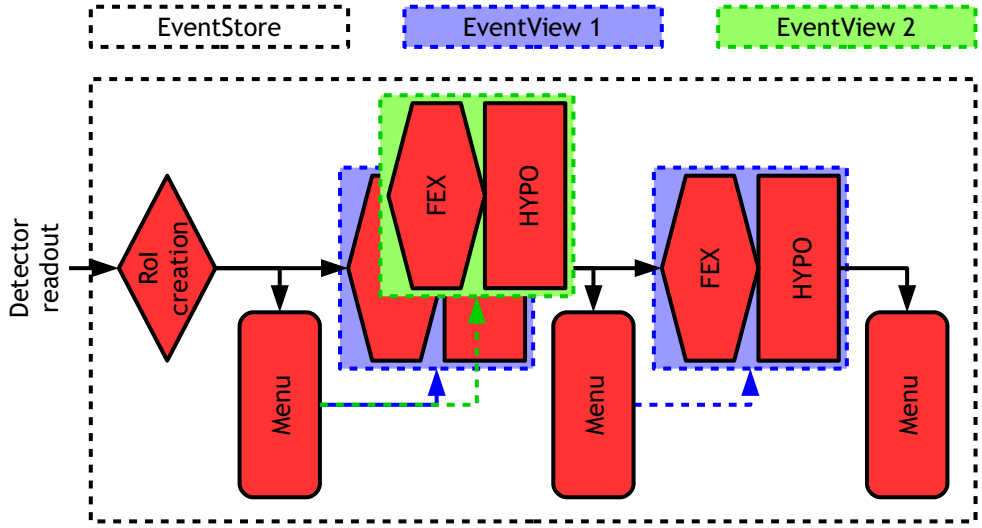


Figure 1. Simplified block-diagram showing the proposed interaction between HLT algorithms and EventViews. Once L1 information is read out, algorithms running on the entire event will create and consume EventViews in order to make trigger decisions based on the menu. Within each EventView, FEX algorithms reconstruct physics objects and HYPO algorithms create elements of the trigger decision.

This report gives a brief overview of how the HLT will be re-implemented in AthenaMT, with new algorithms to manage RoI-based reconstruction. Subsets of event data corresponding to an RoI can be passed to reconstruction algorithms using the DataHandle interface, which allows algorithms to process data without knowledge of its source. The EventView component will be used to contain this data, and EventViews will be created during event processing in response to RoIs read out from the L1 trigger, or more complex reconstruction tasks requiring the merging of RoIs.

EventViews will be created in groups by algorithms that populate them with data, then collect the outputs of the algorithms that run within the view. In particular, the combined results of all the hypothesis algorithms within EventViews will be used to make a trigger decision based on the menu. The result of this decision might be to store the event for offline processing, to reject it entirely, or to perform additional processing with another round of view creation and consumption.

Prototypes of all of these components have been developed, and are being assembled into a complete workflow demonstrator. Additional code will be migrated over the coming years, with the aim of having the full ATLAS HLT running in AthenaMT by the start of LHC Run 3.

References

[1] ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider *JINST* **3** S08003
[2] Bryant P, Evans L 2008 LHC Machine *JINST* **3** S08001
[3] Hegner B, Mato P, Piparo D 2012 Evolving LHC Data Processing Frameworks for Efficient Exploitation of New CPU Architectures *Proc. IEEE-NSS (Anaheim)*
[4] Calafiura P, Lavrijsen W, Leggett C, Marino M, Quarrie D 2005 The athena control framework in production, new developments and lessons learned *Proc. CHEP (Interlaken)* pp 456–458
[5] Barrand G et. al. 2000 GAUDI - A software architecture and framework for building LHCb data processing applications *Proc. CHEP (Padova)*
[6] ATLAS Collaboration 2016 ATLAS Future Framework Requirements Group Report *CERN Document Server* ATLAS-SOFT-PUB-2016-001