How to review 4 million lines of ATLAS code

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 072013 doi:10.1088/1742-6596/898/7/072013

How to review 4 million lines of ATLAS code

Graeme A Stewart[1], Walter Lampl[2] for the ATLAS collaboration

[1] University of Glasgow, School of Physics and Astronomy, University Avenue, Glasgow G12 8QQ, Scotland

[2] University of Arizona, Department of Physics, Tucson, AZ 85721, USA

E-mail: graeme.andrew.stewart@cern.ch

Abstract. As the ATLAS Experiment prepares to move to a multi-threaded framework (AthenaMT) for Run3, we are faced with the problem of how to migrate 4 million lines of C++ source code. This code has been written over the past 15 years and has often been adapted, re-written or extended to the changing requirements and circumstances of LHC data taking. The code was developed by different authors, many of whom are no longer active, and under the deep assumption that processing ATLAS data would be done in a serial fashion.

In order to understand the scale of the problem faced by the ATLAS software community, and to plan appropriately the significant efforts posed by the new AthenaMT framework, ATLAS embarked on a wide ranging review of our offline code, covering all areas of activity: event generation, simulation, trigger, reconstruction. We discuss the difficulties in even logistically organising such reviews in an already busy community, how to examine areas in sufficient depth to learn key areas in need of upgrade, yet also to finish the reviews in a timely fashion.

We show how the reviews were organised and how the ouptuts were captured in a way that the sub-system communities could then tackle the problems uncovered on a realistic timeline. Further, we discuss how the review has influenced the overall planning for the Run 3 ATLAS offline code.

1. Introduction

The ATLAS Experiment [1] designed and developed its offline software framework based on the Gaudi Framework [2] in the early 2000s. The Athena Framework [3] has been a tremendous success for the experiment, processing many billions of ATLAS real and simulated events throught LHC Run 1 and Run 2.

However, certain design limitations in Athena are now becoming evident. In particular the framework and algorithmic code were designed for a serial processing mode, where only one event was processed at a time by a well defined sequence of algorithims, also running serially. In the era when CPU dies had single cores and clock speeds were steadily rising this model worked very well. That situation came to an end in the mid-2000s, as shown in Figure 1, when the rise in clock speed stalled, due to thermal limitations. Transistor density continued to rise (commonly know as *Moore's Law*), but instead of faster cores CPUs started to gain more and more cores and the *multicore* era had begun.

In order to exploit multi-core resources during LHC Run 1, ATLAS ran an independent job on each CPU core, exploiting trivial event level independence in its processing. However, running independent jobs on each core makes poor use of other resources, particularly of memory. To alleviate this issue ATLAS introduced a multi-processing workflow in Run 2. In this model

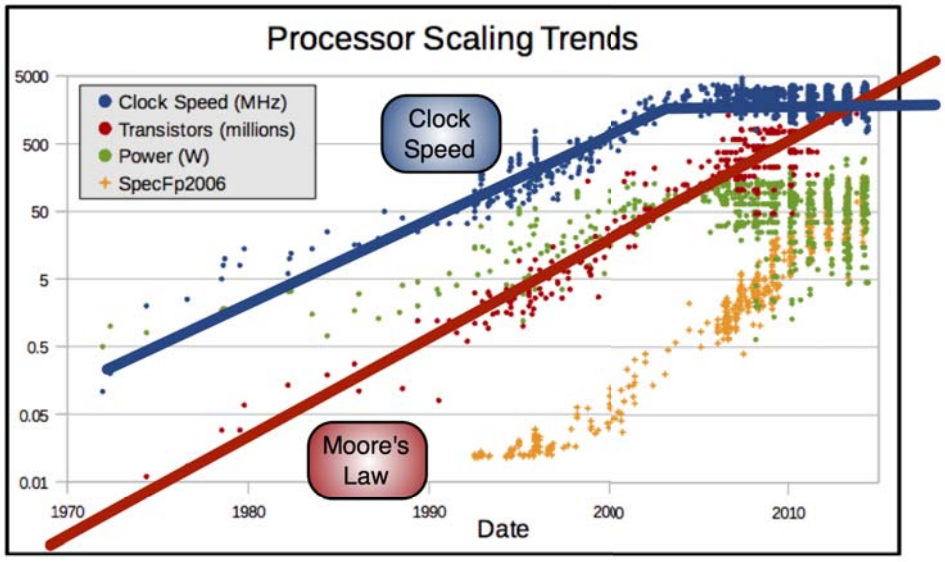

Figure 1. CPU transistor densities, clock speeds power and performance from 1970-2015. Courtesy of Charles Leggett, LBNL.

configuration and initialisation are done in a single threaded job, but just before event processing begins, a number of worker processes are forked from the main Athena job and these perform event processing in parallel. This AthenaMP [4] [5] processing takes advantage of the Linux kernel's *copy on write* feature for memory pages after a fork, so that only one physical copy of memory pages that are shared with the parent process are required. This allows significant memory savings as ATLAS jobs allocate large static memory structures during initialisation that then do not change (e.g., detector geometry and magnetic field). Notwithstanding these impressive savings, memory consumption in ATLAS reconstruction is still very high, compared to the 'standard' memory allocation per core in WLCG, of 2GB and further improvements are required by Run 3.

This continued pressure on memory is also critical to address as continued development of CPU technology has seen the additional rise of *many-core* processors (e.g., the Intel Xeon Phi) and of low power processors (e.g., ARM64 architecture). With these architectures even 2GB of memory per core is usually unfeasible and certainly uneconomic.

Thus ATLAS already identified in 2012 that a significant multi-threaded upgrade to its framework would be necessary [6] [7]. However, the question of the problems that would be encountered in the migration of the ATLAS algorithmic code was largely left unanswered at that point, badged simply as a 'significant effort'. To address that question, ATLAS decided to launch a wide ranging offline software design review in 2016, the results of which we report on here.

2. ATLAS software design review

2.1. Scope

An ATLAS offline software release consists of approximately 3.8M lines of C++ source code and headers and about 1.4M lines of python. The ATLAS code is sub-divided in a 'package' organisation, where code is grouped into about 2200 package sub-directories in our source code management system.

There is a strong concept of package ownership by software domains, where domains can

correspond to a detector subsystem (e.g., the pixel subsystem of the ATLAS inner detector) or to a reconstruction subdomain (e.g., the egamma software group). In addition, groups such as Simulation, High Level Trigger, Generators, etc. are responsible for their particular packages in the release. Low level common software components are managed by the Core Software Group.

As the amount of code to be covered was enormous, it would just not have been feasible to conduct a line by line review of all of the source files. Instead it was decided early on to focus on the software *design*: the key elements of data flow, the interactions with other pieces of algorithmic code and the use of framework services. A design focus allowed for quite rapid conclusions about deprecated code to be made before any time was wasted on details. Indeed, even before the review had begun, it had become clear that certain pieces of code had reached the end of their useful life in serial mode and that a full rewrite was far preferable to continued patching and tinkering.

Although the case for a review of this type was largely accepted by the community, there was considerable reluctance to undertake it in the near term from many quarters. It was perceived as an extra burden on a community already pressed by many short term goals (release preparation, detector operations and reprocessing). However, ATLAS software management decided that the review should happen, even if some short term items were delayed – too often in HEP long term software health is sacrificed to short term expediency.

2.2. Organisation

It was decided to organise the review by software domain. This allowed for all of the relevant software packages to be treated together, with review material for related code areas being coherently presented. Each domain was asked to nominate between 2 and 3 experts to prepare the review material.

After further discussion in the development community we decided to conduct the review internally to ATLAS. Thus each of the software experts in charge of their domain were also asked to contribute to the process by reviewing 2 or 3 other domains. As our own developers are familiar with the software engineering practice required to develop in the Gaudi/Athena framework this meant that much of the common boiler plate would be understood without explanation and also allowed knowledge of domain specific solutions to propagate horizontally during the review.

2.3. Review inputs

As the primary goal of the review was to establish the state of readiness of the ATLAS offline software for a multi-threaded framework, considerable guidance was provided to the groups as to what material they should provide for the review. Groups were asked to prepare input with:

- A high level description of the code design (single page or slide)
- Links to documentation of the code and/or its design, including comments on if it is complete and up to date
- A workflow diagram with algorithm and tools, highlighting key algorithms and how they interact through data objects
- How the transient event store is used
- How the code obtains any non-event data values, such as detector conditions
- A description of the procedure for testing and validating the code
- Identification of any areas of actively used code that are poorly maintained
- Current numbers on resource consumption from standard test jobs (memory and cpu consumption as a fraction of total event resource consumption)

The final item allowed the review to balance resource consumption considerations in the context of the overall resources used in event processing.

Further input was solicited regarding specific patterns (or anti-patterns) that would not work well in a multi-threaded environment:

- Any use of code constructs that would fail in a multi-threaded environment, such as non-const statics or `const_cast`
- Communication that circumvents the transient event store (e.g., using a member variables in shared class instances)
- The use of public tools[1] where simple conversion to a private tool is not possible
- Anywhere non-thread safe resources might be accessed (e.g., SQLite connections or other resource handle where you are not sure about the safety of the underlying library)
- Anywhere python is used in the event loop
- Where any incidents[2] are used and how they might be replaced
- Where algorithm or tool state is updated in the event loop

The workflow of the algorithm was also analysed to see where parallelisation opportunities might be found.

As the favoured design pattern for new algorithms is to try and make them stateless, groups were asked to comment on the feasibility of making an algorithm's `execute()` stateless and reentrant, which would mean it could be made `const` and run on multiple events in parallel. This also requires that algorithms would also only call other `const` methods during their own execution.

2.4. Review process

In common between the reviewers and the reviewed, a schedule of meetings was setup that would allow the reviewed domains to prepare review material well in advance. The intention of the review meeting was to concentrate on difficult or unclear points, not to walk through the material piece by piece. Thus it was important that the review material was available to reviewers before the meetings. Our intention was to allow one week in advance for 'pre-review'. In practice this goal was rarely achieved, but in the vast majority cases the material was available with 48 hours notice and this was sufficient for the reviewers, who had by and large cleared a space in their schedules.

The presentation format for the review documentation was left open to groups, but we quickly converged on the superiority of collaborative document tools, such as Google Docs or Google Slides. Not only did this allow the reviewees to collaborate on the material effectively, it allowed the reviewers to use the comment functionality of such tools to highlight points of concern in advance of the review meeting and for the domain to respond with clarifications. Figure 2 shows an example of this.

With the review inputs well prepared, the vast majority of the reviews were able to be conducted within a 1 hour time slot. For each review one of the ATLAS software coordinators (who were *ex-officio* members of all the review panels) took notes that would later be used to draw conclusions (see Section 2.5).

[1] In Gaudi public tools are singletons and shared, which is now deprecated as it is inherently unsuitable for multi-threading; the singleton pattern is still used for framework services, which then have to be programmed to be thread safe.

[2] Incidents are a Gaudi framework concept utilising the Observer design pattern to invoke a series of callbacks to client code when a particular condition is encountered. It is known in ATLAS that the use of incidents is particularly coupled to thread hostile practices, e.g., clearing an internal cache at the end of the processing of an event.

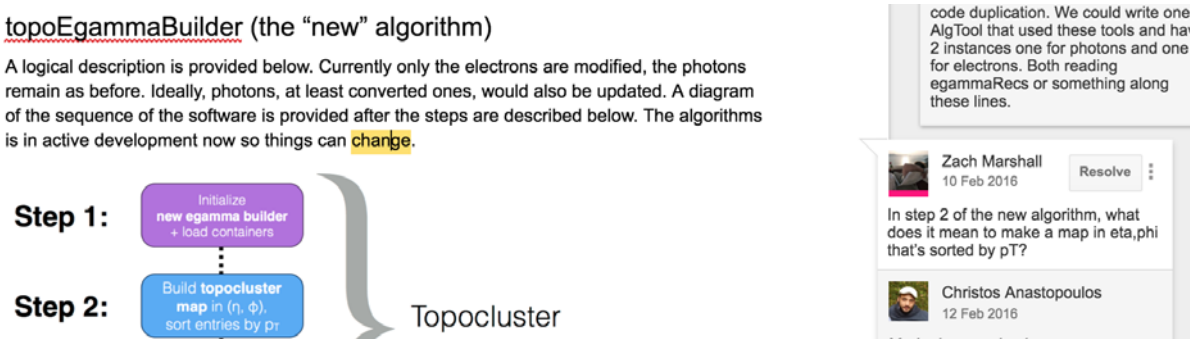

Figure 2. A snippet of the review material, with comments, presented by the eGamma group.

Table 1. Domain areas reviewed in 2016

eGamma	Core Simulation
Tracking	Tau
Silicon Strip Tracker (SCT)	Muon
Flavour Tagging	Pixel
Transition Radiation Tracker (TRT)	Trigger
Jet/MET/pFlow	Calorimetry
Digitisation	Generators

The reviews started in February 2016 and ran until July 2016 with the domain areas listed in Table 1 taking part.

Although the reviews only required the participation of the review panel and the reviewees, they were open to all. Attendance was frequently more than double the core membership and it was clear that the software community found the review process itself useful and educational, providing insights into other domains' approach to software design problems.

It was also noticeable that although the focus of the review was clearly on the code design, reviewers not infrequently looked at the implementation of code in the most important packages and made astute comments on the current implementation and on strategies for future improvements.

2.5. Review outputs

The intention of the review was to identify concrete changes to ATLAS code for Run 3. It was therefore decided to capture the outputs of the review in the issue tracker that the ATLAS software project uses, the Atlassian Jira tracker, managed by CERN IT.

An Epic ticket was assigned to each domain to capture the state of their progress towards multi-threading. Within that Epic, individual tickets were assigned to capture issues that needed to be addressed. It was then left to the domain itself to breakdown issues into whatever digestible sub-issues were necessary to organise and manage the necessary improvements.

2.5.1. The egamma issues It is illustrative to look at the tickets that were assigned to the egamma domain, which were quite typical of the issues that were uncovered during the review:

Lock egamma containers after egamma reco In the new framework it is essential that data objects are locked in the transient event store to signal that they can be used as immutable inputs for downstream algorithms.

Make EMBremCollectionBuilder a separate algorithm Sometimes algorithms encapsulate multiple parts of reconstruction, even where there is no data dependency between them. Breaking apart such algorithms will allow the framework scheduler to more effectively exploit concurrency.

Optimise EMBremCollectionBuilder It was noted that this is an expensive algorithm in the egamma chain and scales quadratically as $tracks \times clusters$. It is believed that a new algorithm, with much better pileup scaling can be developed.

Remove `const_cast` *and non-const statics in egamma code* Casting away constness violates data immutability and thread safety. Non-const statics updated during the event loop fail when multiple events are in flight. These issues need to be fixed for egamma reconstruction to run correctly on multiple events.

Convert egamma algorithms and tools to data handles To abstract data access a new method of accessing the event store has been introduced, which uses handle semantics. Conversion to this access pattern also manages data input and output declarations, required by the scheduler, automatically.

Check for deprecated egamma code Some deprecated pieces of code were already identified during the review, but further checks are warranted.

Update egamma documentation In the egamma domain, as in many others, documentation of the code has fallen behind what is actually implemented. Thus some time needs to be spent improving this (ATLAS recommends that code documentation be maintained with the code, using doxygen).

Code quality and ATLAS coding conventions ATLAS maintains code quality guidelines and conventions. However, not all of these are adhered to. Especially when writing new code developers should make sure they do so.

Make tools stateless/reentrant where possible As explained above in Section 2.3, stateless event loop execution allows algorithms to be executed as `const` methods, that allows the compiler to do more extensive checks on thread safety (albeit that these are necessary, but not sufficient). Developers should review the code with that in mind.

Revisit configuration flags A general review point that was identified is the inconsistent use of configuration flags between domains. Each domain needs to review their flags and ensure that they are applied consistently with regard to global option settings.

3. Conclusions

Across the 14 domains that were reviewed there were significant differences seen in the level and preparedness of developers for a multi-threaded framework. Many of the original developers of ATLAS's offline software are no longer active in the ATLAS software project, indeed many have left the field all together. Thus in many domains the existing documentation had fallen far behind the actual implementation of the software. Even when documentation did exist, or the behaviour of an implementation could be discerned, the design choices could not be. Thus while some of the *how* could be understood, the *why* was often lost.

However, for many groups the review input material has become the basis for modern and up to date documentation and the process of preparing the documentation was itself an excellent learning experience for the current maintainers.

The review was also a didactic experience for developers and revealed some shortcomings in understanding. One frequent error made was that developers confused the issues around `const_cast` and `static` variables to believe that `static_cast` was problematic for threading, which it is not. Although such confusion was unfortunate, the fact that it arose pointed to groups that would need extra help and support in the future, as well as to possible improvements that were needed in documentation. As the review was implemented by the community there was much cross fertilisation of ideas between domains; we will continue to foster such horizontal information sharing.

Overall the review was success, with most developers praising the process and what was learned. This was despite the fact that it was an additional burden on an already busy set of people. The fact that issues will be followed in the standard software tracker is considered a boon for proper follow up.

ATLAS has now learned a huge amount regarding the work that will be required to have proper multi-threading in place for Run 3. Many of the core framework pieces are now in place for the actual algorithmic code migration, which is scheduled to start in 2017 and a series of workshops and training events are now planned to help with the design and implementation of the next generation of ATLAS offline code.

References

[1] The ATLAS Collaboration 2008 *Journal of Instrumentation* **3** S08003
[2] Barrand G *et al* 2001 *Comput.Phys.Commun.* **140** 45–55
[3] Calafiura, P *et al* 2005 *Computing in High Energy Physics 2004,* Interlaken
[4] S Binet *et al* 2012 *Journal of Physics: Conference Series* **368** 012018 URL `http://stacks.iop.org/1742-6596/368/i=1/a=012018`
[5] Binet S, Calafiura P, Snyder S, Wiedenmann W and Winklmeier F 2010 *Journal of Physics: Conference Series* **219** 042002 URL `http://stacks.iop.org/1742-6596/219/i=4/a=042002`
[6] Bains J, Bold T, Calafiura P, Kama S, Leggett C, Malon D, Stewart G A and Wynne B W (ATLAS Collaboration) 2016 ATLAS Future Framework Requirements Group Report Tech. Rep. ATL-SOFT-PUB-2016-001 CERN Geneva URL `https://cds.cern.ch/record/2196967`
[7] Calafiura P, Lampl W, Leggett C, Malon D, Stewart G and Wynne B 2015 *Journal of Physics: Conference Series* **664** 072031 URL `http://stacks.iop.org/1742-6596/664/i=7/a=072031`