DUNE Software Framework Requirements
HSF Review - Appendix

Review panelists: Marco Clemencic[a], Giulio Eulisse[a], Christian Haack[b], Matti Kortelainen[c], Charles Leggett[d], Marc Paterno[c], Simon Patton[d]
HSF frameworks WG conveners: Chris Jones[c], Kyle Knoepfel[c], Attila Krasznahorkay[a]
HSF coordinators: Benedikt Hegner[a], Liz Sexton-Kennedy[c], Graeme Stewart[a]

This appendix is a companion to the HSF review findings document that was submitted to DUNE on 12 July 2021.

Prior experience

The following stories are meant to provide perspective regarding what types of migrations HEP experiments have undertaken in the last 5-10 years.

LHCb experience

To simplify the transition to multithreading in Gaudi based applications, LHCb introduced the `Gaudi::Functional` components with the goal of constraining the kind of code users can write to modern multithread-friendly best practices. Although the `Gaudi::Functional` components are not mandatory to write thread-safe and reentrant Gaudi algorithms, they make it much easier by, for example, enforcing reentrant functions for the main processing method of algorithms.

During the CERN LHC Long Shutdown separating the so-called Run 2 from Run 3, LHCb migrated the vast majority of the High Level Trigger algorithms from the old model to `Gaudi::Functional`. The migration was performed initially by a few dedicated expert developers for the core algorithms, later joined by detector experts for the detector specific code. Apart from allowing the conversion to thread-safe code, the migration gave the occasion of a critical review of old code to rewrite it, dramatically improving its efficiency.

Quantifying the effort that was required for this migration is not straightforward as many of the contributors could not work continuously on it. The best estimate, therefore, is that the migration took 5 FTE years to complete.

a European Organization for Nuclear Research (CERN)
b Technical University of Munich
c Fermi National Accelerator Laboratory (FNAL)
d Lawrence Berkeley National Laboratory (LBNL)

CMS experience

In January of 2014 the CMS' multithreaded framework was ready for use by the collaboration.[1] The core team used about 20 different modules/algorithms as a demonstration of readiness. A key strategy in this transition was the ability to declare framework components as either legacy, or thread-friendly, allowing the framework to orchestrate the running of all of the modules together in a thread-safe manner. The CI system was then used to gradually migrate from single threaded execution to multi-threaded execution independently because the inputs to an algorithm and its outputs were always available to the major reconstruction and simulation modules independently of it's migration status. In June of 2014, there were 200 modules converted and by August 230. This represented a concerted push both from the handful of core developers as well as responsables for components in the rest of the collaboration. Having been through a redesign of the framework in 2005, the code's starting point was in a better position for the transition to concurrent execution. Rules established in 2005 like modules only communicating through the event and providing only immutable access to the data products therein, made the transition easier. There were about 65 people contributing to CMSSW in this period from January through August 2014.[2]

About half of the work done during that time was related to the transition. Being the first experiment to require thread safety, CMS blazed the trail with third-party products. CMS had to make sure that all of the third-party software that our applications used was thread-safe. They used git repositories forked from the official repositories of products like ROOT, CLHEP and fastjet to provide minimal thread-safety for use in the multithreaded framework. Another key element is stated in the paper: "In order to meet the challenge of migrating a large body of code like CMSSW into thread-safe implementations, we have developed tools to automate the discovery of non-conforming code". In addition, the use of runtime tools for finding errors has been very important. The automated tools used in the transition as well as performance measurements are documented in the paper.

ATLAS experience

Similar to LHCb, ATLAS migrated its reconstruction and trigger software to the multi-threaded version of Gaudi during LHC's Long Shutdown 2. While the core framework components are shared between LHCb and ATLAS, the exact implementations of the reconstruction algorithms differ greatly.

ATLAS decided to stay with the infrastructure that it created during LHC's Run-1 and Run-2, implementing all of the components required by reconstruction on top of experiment specific base classes `AthAlgorithm`, `AthAlgTool` and `AthService`, which themselves derive from the `Algorithm`, `AlgTool` and `Service` base classes provided by Gaudi. During the migration the access to the data stores had to be completely re-written. Moving from an unrestricted access to

[1] E Sexton-Kennedy *et al* J.Phys.Conf.Ser. 608 (2015) **1**, 012034 [https://inspirehep.net/literature/1372982, doi:10.1088/1742-6596/608/1/012034]
[2] https://github.com/cms-sw/cmssw/graphs/contributors?from=2014-01-01&to=2014-08-31&type=c

the event store/metadata store/conditions store in any type of component, to an access that is declared explicitly to the framework, and can only happen in algorithms and tools. It was also necessary to completely rewrite how metadata and conditions data would be stored in memory, to allow events with different Intervals of Validity to be processed in parallel.

Not counting the effort required to update all of the core framework components to serve this new setup--which often happened in parallel with the reconstruction code updates-- O(20) developers had to spend O(3) years with an O(50%) commitment to make the ATLAS offline reconstruction run "reasonably efficiently" with 8 parallel threads. With further effort still necessary to reduce the crash rate of the reconstruction software even more, and to achieve better performance with 16+ threads than what we can get at the moment.

ALICE experience

ALICE has been undergoing a major upgrade of it's detector and readout during the long shutdown between Run 2 and Run 3. A number of subdetectors have been completely replaced and the others received a major upgrade. Associated with this, a complete rewrite of its software framework and reconstruction software is being performed, from the readout software, all the way to the actual analysis written by physicists which have to be ported to the new framework. The effort started in 2015, with the publication of the O^2 Upgrade TDR[3] and it's well undergoing nowadays with commissioning of the new detector and simulation / analysis data challenges being successfully performed as this note is being written. Given the amount of changes, the large scope of the transition, and the fact that the old system has been running in parallel, maintained by the same manpower, it's difficult to account for actual personpower. That said, our best estimate is that one should not count less than O(20) FTEs over 6 years to perform the whole migration, with a core group of 10 developers largely dedicated to the transition over the whole period of time.

NOvA experience

The following experience (with minor edits by Kyle Knoepfel) was provided by Gavin Davies, Assistant Professor of Physics at The University of Mississippi.

NOvA migrated to art in 2010 and since then has performed a further two framework upgrades to additional major art versions. Prior to 2010, the experiment utilized a third-party framework called FMWK[4,5], maintained at Indiana University. FMWK had its origins with the MIPP experiment which needed a C++-based analysis framework. The reasons for the change to art included external support from Fermilab's Scientific Computing Division (SCD) in maintaining and developing a framework, improved synchronicity with ROOT development, and consistency with additional HEP experiments making transitioning across experiments easier. In addition, the art framework offered better type safety and memory management, desirable features such

[3] ALICE-TDR-019, 2015. CERN-LHCC-2015-006 [https://cds.cern.ch/record/2011297]
[4] https://cdcvs.fnal.gov/redmine/projects/nusoft/wiki/FMWK
[5] https://indico.fnal.gov/event/2414/contributions/75412/attachments/47000/56450/nova_overview.pdf

as configurable services, the ability to navigate output ROOT files and draw distributions from data members, and future potential multi-thread running. The initial migration to art 1.0 was handled by approximately 3 FTE on NOvA with additional, invested interest of roughly 3 FTE support from SCD.

The transition was completed within an approximately 4 month timeframe; at the time NOvA had approximately 20 active users.

In 2017, NOvA initiated a migration from art 1 to art 2. This process took approximately 1.5 years to complete--in part due to the experiment actively taking data, but also due to NOvA falling far behind the art development. Additionally, the validation of this migration was further complicated because it involved a major version change from ROOT 5 to 6. Despite being far behind art development, SCD provided around 0.5 FTE to assist with the necessary interface and dictionary-generation changes.

The NOvA migration from art 2 to art 3 was the most trivial migration, with only comparatively limited support required from SCD; primarily in providing the required external products with correct versioning, and builds against SLF6 and SLF7. This was all despite this migration occurring in tandem with the Scientific Linux version change and the move from Python 2 to Python 3.

NOvA also supports two build systems, which added a further burden to the migration effort: the primary SoftRelTools (SRT) build system[6,7] and a secondary CMake-based one called cetbuildtools/MRB. With a much larger active user base (around 75 active users) and active data-taking and physics analysis, the timing of migrations has been very difficult.

A significant factor that would reduce the effort required during migrations is an ability to keep in tandem with framework minor version updates. This is a non-trivial maintenance load for an actively running experiment that now supports over 100 active users. The proliferation of supported external products that require consistent machine, framework, Python, and ROOT version builds further increases the maintenance difficulty.

[6] G.S. Davies *et al*, J.Phys.Conf.Ser. 664 (2015) **6**, 062011[https://inspirehep.net/literature/1408129, doi:10.1088/1742-6596/664/6/062011]
[7] J. Amundson, Contribution to CHEP 2000, 731-732 [https://inspirehep.net/literature/538395]