Lessons Learned from CMS Framework Development

Christopher Jones
JLab Software and Computing Round Table
10 March 2020
Introduction

• These ‘lessons’ are my own opinion, not necessarily those of CMS management

• Will cover the following areas
  • CMS’s problem domain
  • Framework structure
  • Multi-threading
  • Job configuration
  • Miscellaneous
Problem Domain
Know Your Problem Domain

• The lessons here are based on what CMS needs
  • Other experiments will often have different needs
CMS Problem Domain

• Have billions of statistically independent Events to process
  • Events give perfect parallelization
  • More interested in time it takes to process all Events rather than time it takes to process one Event
    • CMS cares more about Event throughput rather than latency

• Algorithms used to process Events have limited concurrency
  • Concurrency scales better between Events compared to within an Event

• Constrained on memory per CPU core
  • Running N single-threaded jobs on N cores often exceeds memory of a node
  • Lots of memory in a job can be shared across events
    • Memory exclusively needed to process event is less than memory per core
  • Having multiple Events processing concurrently sharing memory is sufficient for memory constraint
    • High concurrency within an Event has not been needed
Structure
Encapsulate Algorithms into Modules

- Frameworks have done this for 30 years
  - still a good idea

- Break computations into ‘small’ algorithms -> Modules
- Modules only communicate via data products
- Framework responsible for
  - scheduling when a Module runs and
  - manages access and lifetime of data products

- CMS uses ~1800 Module instances in reconstruction job
- CMS has ~ 4000 separate Module C++ classes
**Macro Data Processing Data Model**

- CMS breaks its ‘Processing’ data into a hierarchy
  - These concepts directly match how data is taken in the experiment

- Run
  - A time period defined by online data taking
  - Contains LuminosityBlocks

- LuminosityBlock
  - 23 second time period during data taking
  - Contains Events
  - Atomic unit of processing
    - Guarantee that all or none of the Events in a LuminosityBlock are processed in a job

- Event
  - Triggered beam crossing during data taking

- Modules can get callbacks when Run, LuminosityBlock or Event changes
+ Micro Data Processing Data Model

• Data products are stored in either Run, LuminosityBlock or Event
  • Any C++ type can be used

• Data product retrieval done in a type safe manner
  • Same API used for Run, LuminosityBlock and Event

• Trivial to match which data products came from which Modules
  • Retrieval requires using the unique string label assigned to the producing Module
Unified Conditions/Geometry System

- Based on Interval of Validity (IoV)
  - Range of Runs, LuminosityBlocks or Events for which the conditions are valid

- Data Model
  - Conditions with same IoV are placed in the same Record
  - A Record can hold any C++ data type

- Modules are used to update conditions when IoVs change
  - Modules are allowed to depend upon conditions from other Modules

- Framework manages lifetime of conditions
  - Modules always get proper conditions for given Run, LuminosityBlock or Event transition

- See
  - https://indico.cern.ch/event/408139/contributions/979797/attachments/815718/1117725/
- Different Data Processing and Conditions Data Models

• The CLEO e+e- experiment framework used a unified data model
  • Runs, LuminosityBlocks and Events were just types of Records

• By learning 1 data model you could do any kind of processing
  • Calibration and analysis used exact same ideas

• See
Multi-Threading
Use Multi-threading Instead of Multi-process

- A single CMS process can need more memory than the average mem/core
- Initially CMS tried forking process
  - *Copy On Write* from Unix allowed sharing of unchanging memory across processes
  - Found consolidating results from multiple processes very difficult
  - Load balancing across processes was not trivial

- CMS very happy with results we get with our fully threaded framework
  - Use Intel’s Threading Building Blocks (TBB) library to do scheduling of work on threads
  - The TBB task model is a very good fit for CMS’s framework

- See
+ Avoid Singletons

- Singletons are globally accessible stageful objects in a program

- Singletons cause hidden dependencies between Modules
  - Makes it difficult to properly schedule run order of Modules

- Singletons make thread-safety more difficult
  - One thread could be updating the value while another is reading it

- Singletons impose a single instance even if logically there could be multiple
  - E.g. If a singleton was used to deliver a calibration then could not have concurrent processing across a calibration change

- Handled by CMS’s policy of Modules only communicating via data products
Data Products/Conditions are Immutable

- Modules only communicate via data products/conditions
  - Once published by a Module the data products/conditions are not allowed to change

- Provenance of the data product/conditions easy to understand
  - Just care about where it was produced, not where it was read

- Easier to understand debugging of problems

- Easier to make thread safe
  - const thread-safety is supported by C++ standard
- Avoid User Written Caches in Modules

• Avoid using member data as temporary state
  • Often seen when member data used rather than a calling argument of a member function
  • Prohibits using a Module instances concurrently by multiple events

• Avoid using member data to cache IoV based information
  • E.g. on each Run update an algorithm specific structure
  • These hinder concurrent IoV processing
  • Better to have framework do caching
    • Could be part of conditions system
    • Module API could offer customizable caching internal to Module
- Limit Synchronization Points

• Events are perfectly parallelizable

• Any arbitrary segregation of events can lead to synchronization
  • E.g. a Module assuming only 1 Run being processed at a time
    • Could not process Events from different Runs concurrently
  • CMS’s Module API for single threaded framework assumed
    • Gave callbacks on begin and end of Files, Runs and LuminosityBlocks
      • Implied only 1 file, Run and LuminosityBlock being processed at a time
    • Made threaded migration harder
Avoid Blocking Threads

• Using mutex for long running algorithms limits thread scaling

• Better to allow framework to schedule Modules which can not run at same time
  • Tell framework about Modules using a shared resource
  • E.g. 2 Modules using same thread unsafe 3rd party library
Job Configuration
Use Standard Scripting Language for Configuration

• CMS originally (2005) had a custom configuration language
  • Hard for users to get documentation
  • Was not very good at allowing extension/modification of an existing configuration

• CMS switched to using python for job configuration in 2007
- Validate Configuration Options

• The names and types of configuration information are not validated in CMS
  • If set incorrectly, either ignored or leads to runtime failure
  • Have to look at C++ code to find what configuration info is required for each Module

• CMS has optional validation
  • Requires additional user written code
  • If written, does provide documentation in addition to validation

• In hindsight, wished we had required validation
- **Require Module Data Dependencies Specified in Configuration**

  - CMS requires Modules to register in their constructor
    - What data products they produce
    - What data products they consume
  - What is consumed can optionally be obtained from the configuration
    - Makes it difficult to modify an existing configuration to change data dependencies
  - Would be better to always require consumes come from configuration
Miscellaneous
- Better Integration with others Tools

• CMS physicists typically want to do analysis outside of the CMS framework

• CMS standard file format
  • Is readable from a ROOT executable with the addition of a few libraries for dictionaries
  • CMS also has a *nano* data format which is usable from ROOT directly

• Python is often used to write analysis scripts
  • PyROOT is often used to read the data

• Framework Modules cannot easily be used outside of the Framework
  • Have had numerous requests to use some from python
Questions?