Usage and Requirements in LHCb

Sébastien Ponce
`sebastien.ponce@cern.ch`

with input from :

Gerhard Raven
Marco Clemencic
Hadrien Grasland
Benjamin Couturier
Christoph Hasse
Monir Hadji

Context

- LHCb is working on Run 3 upgrade
- This includes software upgrade
- And a full software trigger at 40MHz
- TDR due by the end of the year
  - so it's writing time
  - main design is already known

Outline

Framework

Gaudi::Functional - goals reminder

- Many algorithms look like “data in $\rightarrow$ data out”
- Standardize this pattern
  factor out “getting” and “putting” the data
    - less code to write
    - more uniform code, easier to understand
    - get rid of boiler plate code on user side
    - fix bottlenecks once and for all
- Patterns available
    - Consumer, Producer, Filter, Transformer, MultiTransformer, ScalarTransformer

Gaudi::Functional practical code

```
class MySum: public Transformer
    <OutputData(const Input1&, const Input2&)> {
  MySum(const std::string& name, ISvcLocator* pSvc)
  : Transformer(name, pSvc,
                     { KeyValue("Input1Loc", "Data1"),
                       KeyValue("Input2Loc", "Data2") },
                     KeyValue("OutputLoc", "Output/Data") )
  {}
  // ...
  OutputData operator()(const Input1& in1,
                        const Input2& in2) const override {
    return in1 + in2;
  }
  // ...
}
```

Gaudi::Functional usage in LHCb

- The main way of writing algorithms
- Ensures thread safety
- Checking it at compile time
  - thanks to usage of constness
  - and to the use of ToolHandles
- as a consequence, TES objects are immutable

Handles in the back

- DataHandles
  - allow to build directed acyclic data dependency graph
  - needed for proper algorithm scheduling
    - producer is scheduled prior to its consumers
- ToolHandles
  - allow to know which tools will be used by algorithms
  - and what data those tools request

Handles on anything - AnyDataHandle

- that is DataHandle on any object
  - not necessarily inheriting from DataObject
- hides completely the TES from users
- thus allows to modify it deeply (drop it ?)
  - without touching user's code

```
AnyDataHandle<std::vector<int>> ids
  ("/Event/Test/Ids", Writer, this);
ids.put(vector<int>({42,84}));
```

LHCb Req 1 : Fix DataHandles

- define the final interface
- consistently use references
- be compatible with Ranges

A word on counters

usage

- extensively used in the LHCb framework

```
++counter("Number of tracks");
```

- very useful for results validation

Problem

- StatEntity in the back
- "++" translates to 5 double operations !
- plus a lock around them

LHCb Req 2 : Have efficient counters

- template existing ones
  - many do not need doubles (e.g. nb tracks)
- define a real "counter"
  - where only $++$ is defined
  - and single int member is needed
  - can even be lock free

I'm volunteering on that topic.

Timing

- currently several ways to time code in Gaudi
  - GaudiSequencer
  - Auditors
  - TimelineSvc
- they all work and give same result
- but do we want to keep duplication ?
- Most importantly : do they work in multithreaded environment ?
  - you guessed it, they mostly do not

Parallel I/O

- Root parallel I/O missing
  - copying on reads by using MDF format
  - copying on writes by not writing out results
    - as we are only testing trigger efficiency
- Needs to be sorted out at one stage
  - good topic for a hackathon ?

Event Model

Consequences of functionnal approach

- no direct access to TES anymore (no get/put)
- objects stored in TES are unmodifiable
- so cannot be modified/extended
- $\rightarrow$ need for object composition ?

Composition in practice

Read-only TES example

- algo A stores `vector<Tracks>` in /daq/tracks
- algo B reads /daq/tracks and stores `vector<double>` in /daq/tracksIPs
- the 2 vectors are aligned
  - anciently, Tracks would have been modified
- can algo C deal with `vector<TrackWithIPs>` ?

Composition, the SoA view

- Say a Track has 5 doubles (x, y, z, tx, ty)
- `vector<Tracks>` could be stored as `array<vector<double>, 5>`
  - optimizes SIMD instructions usage
- but we would like to see it as `vector<Tracks>`
- something as an AoS view on the SoA storage
- same idea of "merging" aligned vectors in a view

SOAContainer & SOAView

```
// AOS - style object
struct Hit {
  float m_x;
  float x() const noexcept { return m_x; }
};

// SOA - style
struct HitFields {  // fields defined as types
  typedef struct : public SOATypelist::wrap_type<float> {} f_x;
}
// Skin decorating HitFields
template ... struct HitSkin : ... , HitFields {
  auto & x() const noexcept { return this->template get<f_x> (); }
}
SOAContainer <std::vector, HitSkin, HitFields::f_x> hits;
hits.reserve(...);
hits.emplace_back(...);
```

Composition at TES level

- we now have composition at data level
- could the TES automatize this ?
  - write `vector<A>` in /.../A
  - write `vector<B>` in /.../B
  - read back `vector<AandB>` from /.../AandB
- Probably feasible :
  - create a new Algo filling AandB from A and B
  - algo stores a proxy to SOAContainer, no actual data
  - use the composition path transparently
    - the proxy will spit out an SOAContainer
- to be tried during the hackathon ?

Condition Data

Conditions usage in LHCb

- condition access need to be thread safe
- conditions for different IOVs may be used in parallel
  - but not many (actually, max 2)
  - and this is seldom (every many 1000s events)
- so we do not need an optimized solution
- high expectations on the work of Hadrien
- LHCb backend for condition storage is Git

Detector Description

Thinking about changing our geometry

- Current geometry in production for 15 years
  - but too detailed/slow for tracking and simulation
- Simplified geometries implemented by hand with no support from the framework

Current evolution

- studying DD4HEP as a replacement
  - Gaudi integration done for FCC
- However
  - Difficult to map LHCb Detector description to DD4HEP
  - Direct mapping may not even be what we want...
- currently, geometry has been converted but still being debugged
  - some bugs already corrected in the conversion tools

Summary

Conclusion

- Fully happy of the functionnal approach
  - the key to thread safety
- Few items to be worked on in the framework
  - DataHandles
  - Counters
  - Parallel I/O
- Interested in attempting generic composition