Multi-threaded Object Streaming

Conditions in the CMS experiment

Non-event-related information (“conditions”) in the CMS experiment at the Large Hadron Collider (LHC) at CERN, Geneva, Switzerland, encompasses crucial parameters for the simulation, reconstruction and analysis software. With the CMS software framework moving to a multi-threaded execution model, and profiting from the experience gained during the data taking in Run-1, a major re-design of the CMS conditions software was done. During this work, a study was done to look into possible gains by using multi-threaded handling of the conditions.

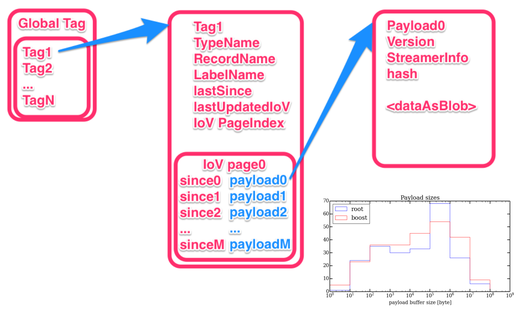
Conditions Structure

“Global Tag”
- coherent collection of “Tags” used for running

“Tag”
- identified by “Record” (and optionally “Label”)
- contains a list of “IoV”s

“IoV” - “Interval of Validity”
- “time” (or run, run-lumi) range (a.k.a. “since”)
- each interval has a unique “since” from which it is valid and contains a pointer to the payload object (conditions) associated with that time-interval (until the next “since”)

“Payload”
- Conditions object stored in DB as a “blob” in the new structure, serialised using either ROOT-5 or the Serialization package from BOOST



1 GT ≈ 300 Tags | *1 Tag ≈ 100-10000 IoVs* | *Payload sizes: 100 B - O(100) MB*

An initial study to use ***Boost serialization*** (and iostreams) packages, with a third-party product for the “Archive” (“EOS”(*)), showed promising results to store CMS Conditions Objects as “BLOB”s in the database. Using existing tools (libclang and its Python bindings) and a helper script, single-threaded performance was found to be comparable to (de-)serialisation with ROOT-5.

Performance testing was then done by comparing parallel, multi-threaded loading of the conditions from the DB, followed by parallel, multi-threaded deserialisation of the BLOBs into C++ objects. Write (serialisation) performance of the payloads was not measured as this is a one-time event and not time-critical.

(*) EOS: Portable Binary Archive - http://epa.codeplex.com

Conditions Serialisation

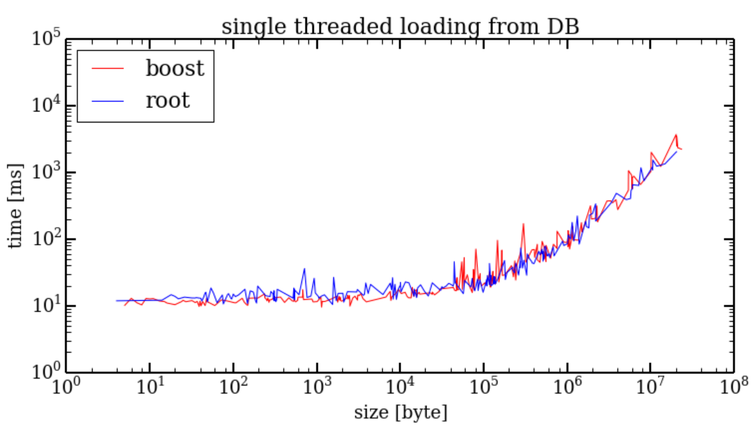
Tools already available from (llvm) compiler: *libclang* and it’s Python binding (***pyclang***) give access to the ***full*** **A**bstract **S**ource **T**ree (**AST**) of the code inspected.

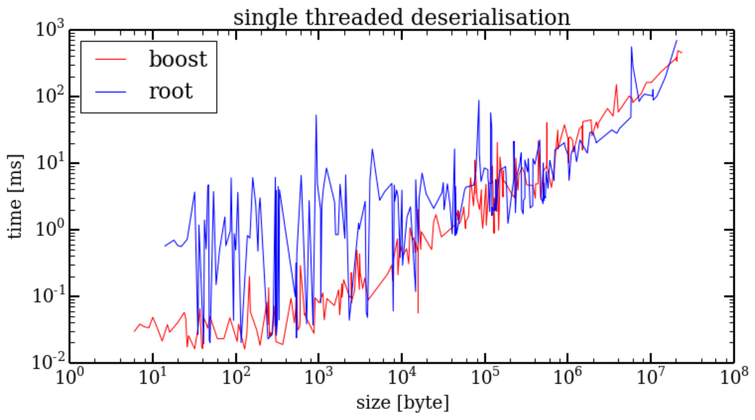
Small (about 500 lines of code overall) Python script automatically generates the (de-)serialisation code from the C++ header files of the user-defined payloads. This script is called by the CMS code configuration and build tool (SCRAM).

Example payload code:

```
                                                    #ifndef
CondFormats_PhysicsToolsObjects_Histogram_h
#define CondFormats_PhysicsToolsObjects_Histogram_h

#include "CondFormats/Serialization/interface/Serializable.h"

template<typename Value_t, typename Axis_t = Value_t>
class Histogram {
      // …
      // transient cache variables
      mutable Value_t total COND_TRANSIENT;       //CMS-THREADING
#if !defined(__CINT__) && !defined(__MAKECINT__) && !defined(__REFLEX__)
      mutable std::atomic<bool> totalValid COND_TRANSIENT;
#else
      mutable bool totalValid COND_TRANSIENT;
#endif

      COND_SERIALIZABLE;
}; // end class<> Histogram
```

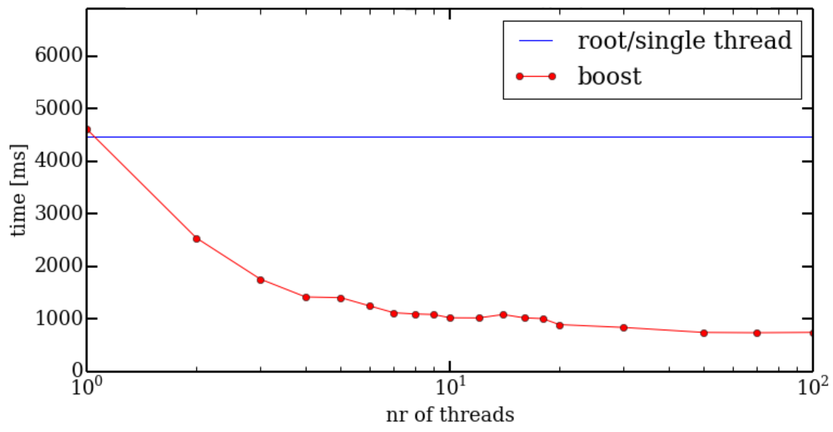
Single-threaded Loading and Deserialisation

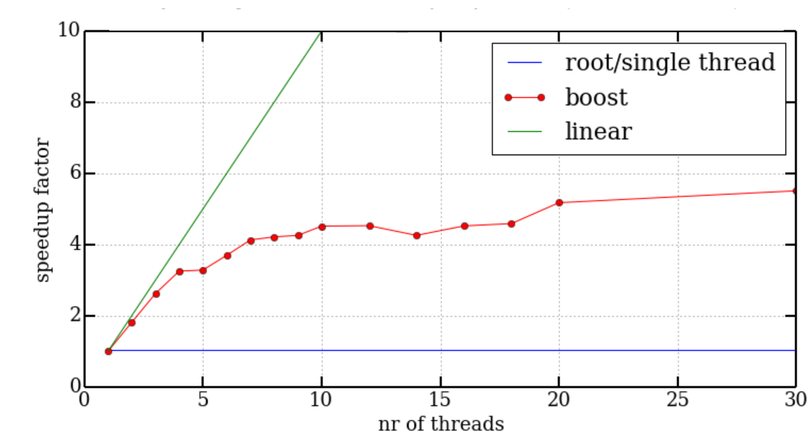




Single-threaded loading and de-serialisation of payloads from the DB - loading and de-serialisation time vs. size of payload. Database connection overhead dominates loading for payloads smaller than ~100kB. For de-serialisation with boost, only at very small payloads the system overhead (e.g. memory allocation) becomes visible.

Multi-threaded Deserialisation





Parallel, multi-threaded deserialisation time (above) and speed-up factor (below), as a function of the number of parallel threads. The tests were done on a 24 core machine, monitoring the environment to ensure that no significant load was running in parallel on the machine and the database.

Speed-up Factors

nTreads (fetch&deserial)	1	8	16	100	gain 8/1	gain 16/1	gain 100/1
loading IOVs	2723	2557	2570	2612	1.1	1.1	1.0
loading payloads	25409	10737	7173	5862	**2.4**	**3.5**	**4.3**
deserialising payloads	4233	1190	1134	648	**3.6**	**3.7**	**6.5**
overall time elapsed	32366	14485	10878	9122	2.2	3.0	3.5

Summary

A performance study for new CMS conditions software shows that ***multi-threaded*** loading and de-serialisation of payloads results in a **speedup** of about a **factor of 3-4** for about 8-10 threads, both in ***loading*** payloads from DB (8s/25s) and in ***deserialisation*** (1.1s/4.2s)

Both (de-)serialisation technologies (ROOT(5) and BOOST/pyclang) have a similar level of “user-friendliness” when using automatically generated “streamer code”.

Download this poster:

Dr. Andreas Pfeiffer - CERN - CH-1211 Geneve 23 - Switzerland
andreas.pfeiffer@cern.ch