ROOT6: The Quest For Performance

Danilo Piparo – CERN, PH-SFT
For the ROOT Team
• Illustrate strategies adopted to increase ROOT performance
• Review design choices and lessons learned

Successful collaboration of the ROOT team and the LHC experiments. **Without their contribution, ROOT6 would not be as good as it is now.**
Problem: ROOT5 interpreter Cint

- C parser, with some C++ capabilities
- Reflection, I/O: no support for new C++ standards, e.g. C++11
- Cracks in the infrastructure: e.g. support for gccxml on OSX

Solution: Replace Cint with Cling

- Cling: a C++ interpreter based on Clang/LLVM technology

Side effect: a lot of work!
- We believe the benefits outweigh the costs

Investments are needed for future sustainability
Push forward software technology

- Cling: first of its kind (JIT of C++)
- Re-write of entire ROOT Core components
- Including layer between ROOT and its interpreter

Existing features to support, rich set of new ones

- Many users: $O(10^4)$ – Backward compatibility guarantees
- Experiment setups: multi-MLOC software systems

A quest but an opportunity

- Such a radical change rarely happens in core software

Improve strategies to evolve our sw, e.g. with agile techniques

Compared with CINT, optimised during 20y!
C++ entities in Clang: Abstract Syntax Tree (AST)

- Classes, functions, templates, statements …
- Exists in memory and can be **persisted on disk in two forms**
  1) **Pre-Compiled Header**: can load only one, file granularity
  2) **Pre-Compiled Modules**: can load many, AST node granularity
- Both queried lazily by the compiler
- Dictionaries: a thin layer around portions of AST

Original ROOT6 design: **AST source of information for**

- Reflection and I/O
- Interactive function calls
Interlude: ROOT, Clang and the AST

ROOT

- Provides implementations

Type System

AST

Cling

Establish connection with Cling (static initialisation)

PCH (Interfaces of ROOT classes + STL)

Lib1

Provides interfaces

Lib2

Libn

PCM1

PCM2

PCMn

16/4/2015

CHEP2015, Okinawa – Track 4
Interlude: ROOT, Clang and the AST

**Provides implementations**

ROOT

Type System

AST

Cling

PCH (Interfaces of ROOT classes + STL)

**Provides interfaces**

Lib1

Lib2

Libn

Establish connection with Cling (static initialisation)

C++ PCMs not delivered on time by Clang

16/4/2015

CHEP2015, Okinawa – Track 4
Provides implementations

ROOT

Type System

Query

Reflection info

AST

Cling

Lib1

Provide interfaces

Lib2

Libn

Parsed at load time
(static initialisation)

Use header files. Parsing costs memory & runtime

PCH
(Interfaces of ROOT classes + STL)

 hdrs1

 hdrs2

 hrdsn
• Issue solved already in Autumn 2014
  – 6.02, 6.04 series not affected!

• Consequences of absent PCMs at the time:
  – Good for analysis and single users
  – Too much memory when integrated with LHC experiments’ software stacks: \(~1\) GB RSS extra 😞
  – Runtime penalty associated to these allocations

Adapt quickly to changing reality
Improve memory consumption: **Reduce parsing**

1) **I/O operations**
   - I/O info for selected classes in “ROOT-PCMs” (ROOT files)
   - Optimise file format for those
   - Information forwarded directly to ROOT type system

2) **Interactive usage**
   - Parse “on demand” (or “Autoparsing”)
   Trigger parsing of headers related to library only when needed
   a. To call functions and methods
   b. To get I/O info when not provided by ROOT-PCMS

Iterative, incremental, evolutionary
A Change In the Design

 ROOT

 Type System

 Query

 Reflection info

 AST

 Cling

 PCH
 (Interfaces of ROOT classes + STL)

 Lib1

 Lib2

 Libn

 Parsed at load time (static initialisation)

 hdrs1

 hdrs2

 hdrsn

 (Interfaces of ROOT classes + STL)
A Change In the Design

Forward information to ROOT type system directly

ROOT
Type System

Query
Reflection info

Cling

PCH
(Interfaces of ROOT classes + STL)

Lib1

Parse when needed

Lib2

ROOT PCM1

hdrs1

Libn

ROOT PCM2

hdrs2

Read at load time

ROOT PCMn

hdrsn

16/4/2015
CHEP2015, Okinawa – Track 4
Memory

• pp→ttbar events @ 13 TeV (event loop):
  – Generation & Simulation: \(-6\%\) RSS wrt ROOT5
    → Yes, better than ROOT5 😊
  – Reconstruction: \(+4\%\) MB RSS wrt ROOT5

• RSS variations: depend on amount of interpreted functions
  – E.g. cuts specified in job configuration

Runtime: ~Identical in the event loop

Also thanks to experiments’ flexibility and willingness to make this happen – thank you!
Why? Profiling & improvements: meaningless w/o correctness!

- Significant expansion of ROOT test suite
  - Target test-driven development
  - All plugins and externals tested (e.g. Davix, xRootd)

- In addition: increase of test platforms (~8 -> ~17)
  - And planning to add more, also non-x86_64*

* See:
  493: Future Computing Platforms for Science in a Power Constrained Era
  500: Building a Tier-3 Based on ARMv8 64-bit Server-on-Chip for the WLCG
### Public link to status of tests:

cdash.cern.ch/index.php?project=ROOT

Impossible to optimise w/o continuous & automatized correctness checks
• Very first feature seen by the user
  – Baseline: ROOT5, ~100 ms (Python 2.7 ~20 ms)

Solution:
• Leverage PCH to store I/O information of ROOT most used classes (Hist, RooFit, …)
• Optimise data structures and algorithms holding/manipulating autoloading info: e.g. use STL!
• Optimise reading of ROOTmap files

Strive for technical excellence in all corners
IgProf

- Both for memory and runtime studies
- “Make diffs” of counters’ snapshots, e.g.:
  - Given a symbol: Event-by-event differences in memory

Valgrind family

- **Callgrind**: very short runs (e.g. startup times)
- **Massif**: complement IgProf information and information display

Kernel Data Structures

- “Poor Man’s Solution” `TSystem::GetProcInfo`
  - e.g.: memory before/after method invocation
  - Use as “tracing bullet”

**Crucial to choose the right ones**
ROOT6: many new features, backward compatible

• We now look towards an exciting future!

Lessons learned:

• **Agile principles**: asset when betting on cutting edge sw technologies
• Close collaboration with “clients”: clear benefit for big sw projects
• **Ruthless, automated & ubiquitous testing**: requirement of ambitious performance improvement campaigns
• “Right” mix of profiling tools can make the difference

• LHC, SuperKEKB, Intensity Frontier: Challenging scenarios!
  → The quest will continue
  – Leverage even more STL in the ROOT Core
  – More vectorisation (and het. platforms): Tree analysis and Math
  – Better integration with profilers
  – Exploit many cores architectures even more
ROOT6: written in C++11. **Need recent compiler** (e.g. GCC ≥ 4.8)
- **Idea:** leverage compilers’ optimisations ("-Ofast")
  - Optimise FP treatment (e.g. operands re-ordering)
  - More inlining
- **Improve routines fragile wrt changes in FP behaviour**
- **Optional, not the default**

“Technical” (non algorithmic) optimisations do matter.

Enabled with -D CMAKE_BUILD_TYPE=Optimized

2016/4/2015 CHEP2015, Okinawa – Track 4