Updates on ROOT C++ modules

Yuka Takahashi - Princeton University, CERN
Vasil Geogiev Vasilev - Princeton University
Oksana Shadura - University of Nebraska Lincoln
News!

C++ Modules released as a technology preview in ROOT 6.16

- All related tests are passing
- CI in ROOT Pull requests, hourly build in incremental build
Agenda

1. Our goal
2. Status - Stability and tests
3. Status - Correctness
4. Status - Performance
5. Steps to migrate to C++ Modules
6. Future work
Our goal

1. Improve correctness of ROOT
2. Avoid parsing header files at ROOT’s runtime
3. Optimize performance of ROOT for third-party code (most notably ALICE, ATLAS, CMS and LHCb)
Our goal

1. Improve correctness of ROOT ✓
2. Avoid parsing header files at ROOT’s runtime ✓
3. Optimize performance of ROOT for third-party code (most notably ALICE, ATLAS, CMS and LHCb) WIP!
Status - Stability and tests

1. Stable tests
   - Vetoing 11 false positive tests out of 1865 tests

2. Continues integration in PR and incrementals
Status - correctness

Correctness benefit

Without Modules

```
$ root -l
root [0] gMinuit // Cannot load variable
IncrementalExecutor::executeFunction:
symbol 'gMinuit' unresolved while linking [cling interface function]!
```
Status - correctness

Correctness benefit

With Modules

```
$ root -l
root[0] gMinuit // Could load libMinuit
(TMinuit *) nullptr
```
Status - correctness

Correctness benefit

Without Modules

```
[yuka@yuka-arch root-release]$ root -l
root [0] #include <m17n-core.h> // System header
root [1] m17n_init_core() // Cannot autoload
system libraries!
IncrementalExecutor::executeFunction:
symbol 'm17n_init_core' unresolved while linking
[cling interface function]!
```
Status - correctness

Correctness benefit

With Modules

[yuka@yuka-arch module-release]$ root -l
root [0] #include <m17n-core.h>
root [1] m17n_init_core() // Could autoload libm17.so!
root [2]
Status - performance

Long tests (30s~)

Master With Cxx Modules ON/OFF
RSS memory of ROOT

<table>
<thead>
<tr>
<th>Test</th>
<th>CxxModules On</th>
<th>CxxModules Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-stressgeometry</td>
<td>392.3905</td>
<td>305.5525</td>
</tr>
<tr>
<td>test-stresstmva</td>
<td>598.144</td>
<td>526.212</td>
</tr>
<tr>
<td>tutorial-roostats-HybridInstructional</td>
<td>286.119</td>
<td>203.275</td>
</tr>
<tr>
<td>meta-MakeProject-runcms310</td>
<td>1,656.072</td>
<td>1,656.072</td>
</tr>
</tbody>
</table>

Master With Cxx Modules ON/OFF
CPU time of ROOT

<table>
<thead>
<tr>
<th>Test</th>
<th>CxxModules On</th>
<th>CxxModules Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-stressgeometry</td>
<td>61.68</td>
<td>63.12</td>
</tr>
<tr>
<td>test-stresstmva</td>
<td>35.08</td>
<td>35.72</td>
</tr>
<tr>
<td>tutorial-roostats-HybridInstructional</td>
<td>30.74</td>
<td>31.09</td>
</tr>
<tr>
<td>meta-MakeProject-runcms310</td>
<td>81.72</td>
<td>82.9</td>
</tr>
</tbody>
</table>
Status - performance

Non-PCH tests (~1s)
Status - performance

Startup & Hsimple (~0.4s)

Master With Cxx Modules ON/OFF
RSS memory of ROOT

- Startup: 119.441 MB
- hSimple: 168.289 MB

Master With Cxx Modules ON/OFF
CPU time of ROOT

- Startup: 0.078452 s
- hSimple: 0.263353 s
Status - performance

Memory - hSimple

Updates on ROOT C++ modules

Preloading all modules

Autoloading libraries

Implement Bloom filter

Check fCXXRecordDecl Cache

Enable Cling modulemap

Yellow line is PCH
Green line is Modules

https://rootbnch-grafana-test.cern.ch/

… And More!!
Status - performance

Real time - hSimple

Yellow line is PCH
Green line is Modules

https://rootbnch-grafana-test.cern.ch/

Hunting down bottlenecks!
Steps to migrate to C++ Modules

-Druntime_cxxmodules=ON to your CMake :)

Yuka Takahashi 19.11.2018
Steps to migrate to C++ Modules

Why to migrate?
- For ROOT end users
- Correctness benefits

Without Modules

[yuka@yuka-arch root-release]$ root -l
root [0] #include <m17n-core.h> // System header
root [1] m17n_init_core()
IncrementalExecutor::executeFunction:
symbol 'm17n_init_core' unresolved while linking
[cling interface function]!

With Modules

[yuka@yuka-arch module-release]$ root -l
root [0] #include <m17n-core.h>
root [1] m17n_init_core()
root [2]

Without Modules

$ root -l
root [0] gMinuit // Cannot load variable
IncrementalExecutor::executeFunction:
symbol 'gMinuit' unresolved while linking [cling interface function]!

With Modules

$ root -l
root [0] gMinuit // Could load libMinuit
(TMinuit *) nullptr
Steps to migrate to C++ Modules

Why to migrate?
- For experiments
  - Correctness benefits
- Performance!

Experiments are parsing hundreds of headers at the startup time at the moment
  - Experiments are still using textual include
  - PCH can’t be used because it’s too big

Modules can do this as it’s separable
Steps to migrate to C++ Modules

Why to migrate?
- For experiments

Current Status
Working closely with CMSSW
Getting a lot of feedback from CMS usage
- Leads to fix bugs in Clang & Cling (fixed 3 bugs in Clang)
Future work

1. Test & Stability
   - Increase the coverage of all possible options (-D*)
2. Performance
   - Measure the performance numbers in CMSSW
   - Keep optimizing the performance!
3. Migration of codebase
   - Help with the migration process of the third-party code, and in particular the major LHC experiments (ALICE, ATLAS, CMS, LHCb)
Thank you for your attention!
Backup Slides
<table>
<thead>
<tr>
<th>Allocations</th>
<th>Temporary</th>
<th>Peak</th>
<th>Leaked</th>
<th>Allocated</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>419197</td>
<td>416570</td>
<td>1.4 MB</td>
<td>1.4 MB</td>
<td>110.5 MB</td>
<td>clang::serialization::MultiOnDiskHashTable&lt;clang::serialization::reader::ASDeclContext...</td>
</tr>
<tr>
<td>419197</td>
<td>416570</td>
<td>1.4 MB</td>
<td>1.4 MB</td>
<td>110.5 MB</td>
<td>clang::ASTReader::FindExternalVisibleDecsByDeclName(clang::DeclContext const*, clang::D...</td>
</tr>
<tr>
<td>419197</td>
<td>416570</td>
<td>1.4 MB</td>
<td>1.4 MB</td>
<td>110.5 MB</td>
<td>clang::MultipleExternalSemaSource::FindExternalVisibleDecsByDeclName(clang::DeclContext...</td>
</tr>
<tr>
<td>30386</td>
<td>307484</td>
<td>917.6 kB</td>
<td>917.6 kB</td>
<td>80.8 MB</td>
<td>LookupDirect(clang::Sema&amp;, clang::LookupResult&amp;, clang::DeclContext const*) in ?? (libCling.so)</td>
</tr>
<tr>
<td>102430</td>
<td>1011014</td>
<td>229.4 kB</td>
<td>229.4 kB</td>
<td>26.9 MB</td>
<td>clang::DeclContext::using_directives() const in ?? (libCling.so)</td>
</tr>
<tr>
<td>101101</td>
<td>101014</td>
<td>229.4 kB</td>
<td>229.4 kB</td>
<td>26.9 MB</td>
<td>(anonymous namespace)::UnqualUsingDirectiveSet::addUsingDirectives(clang::DeclCont...</td>
</tr>
<tr>
<td>101101</td>
<td>101014</td>
<td>229.4 kB</td>
<td>229.4 kB</td>
<td>26.9 MB</td>
<td>(anonymous namespace)::UnqualUsingDirectiveSet::visitScopeChain(clang::Scope*, clan...</td>
</tr>
<tr>
<td>101101</td>
<td>101014</td>
<td>229.4 kB</td>
<td>229.4 kB</td>
<td>26.9 MB</td>
<td>clang::Sema::CppLookupName(clang::LookupResult&amp;, clang::Scope*) in ?? (libCling.so)</td>
</tr>
<tr>
<td>101101</td>
<td>101014</td>
<td>229.4 kB</td>
<td>229.4 kB</td>
<td>26.9 MB</td>
<td>clang::Sema::LookupName(clang::LookupResult&amp;, clang::Scope*, bool) in ?? (libCling.so)</td>
</tr>
<tr>
<td>57480</td>
<td>5489</td>
<td>71.8 kB</td>
<td>71.8 kB</td>
<td>1.6 MB</td>
<td>clang::DeclContext::lookup(clang::DeclarationName) const in ?? (libCling.so)</td>
</tr>
<tr>
<td>25910</td>
<td>2583</td>
<td>229.4 kB</td>
<td>229.4 kB</td>
<td>1.1 MB</td>
<td>bool checkGlobalOrExternalConflict<a href="">clang::FunctionDecl</a>(clang::Sema&amp;, clang::Function...</td>
</tr>
<tr>
<td>15690</td>
<td>1137</td>
<td>1.6 MB</td>
<td>1.6 MB</td>
<td>6.6 MB</td>
<td>&lt;unresolved function&gt; in ?? ()</td>
</tr>
<tr>
<td>12890</td>
<td>6571</td>
<td>223.0 kB</td>
<td>222.0 kB</td>
<td>509.8 kB</td>
<td>TString::Init(int, int) in TString.hxx:242 (libCore.so)</td>
</tr>
</tbody>
</table>

[yuka@yuka-arch root-release]$ root -l root [0] gMinuit
IncrementalExecutor::executeFunction: symbol 'gMinuit' unresolved while linking [cling interface function]!

[yuka@yuka-arch module-release]$ root -l root [0] gMinuit (TMinuit *) nullptr
Status - performance

Preloading Modules

All correctness benefit over PCH is due to this
- Preloading of all modules
- Replace old infrastructure
  - rootmap
C++ Modules in a Nutshell
C++ Modules in a Nutshell

```cpp
#include <vector>
```
C++ Modules in a Nutshell

#include <vector>

- Textual Include
  - Expensive
  - Fragile

- PCH
  - Inseparable

- Modules
#include "TVirtualPad.h"
#include <vector>
#include <set>

int main() {
...
}

original code

---

Yuka Takahashi 19.11.2018

√ Updates on ROOT C++ modules
C++ Modules in a Nutshell

Textual Include

1. Expensive
   Reparse the same header

2. Fragile
   Name collisions

Rcpp library

```cpp
#define PI 3.14
...
```

Users’ code

```cpp
#include <header.h>
... double PI = 3.14;
// => double 3.14 = 3.14;
```
C++ Modules in a Nutshell

PCH (Pre Compiled Header)

1. Storing pre compiled header information (same as modules)
2. Stored in one big file
C++ Modules in a Nutshell

Modules

- Pre compiled **PCM files** contain header information
- PCMs are **separated**

Each PCM file (a.pcm) corresponds to a library (liba.so)
C++ Modules in a Nutshell

Modules

- Pre compiled **PCM files** contain header information
- PCMs are **separated**

- Compile-time scalability
- Fragility
- Separable
Implementation
Implementation

User Interface

ROOT
Integration with ROOT libraries and provide ROOT specific features

rootcling
Generate pems and libraries

Cling
C++ interpreter interface

Clang
Provide fundamental implementation of C++ modules
Implementation

Clang

- External project under LLVM
- Bi-weekly meeting with C++ Modules community
- Reporting & fixing bugs
- ROOT is the largest user of Modules outside industry