C++ Modules in ROOT and Beyond

Vasil Georgiev Vasilev (Princeton University (US))
Yuka Takahashi (Princeton University (US))
David Lange (Princeton University (US))
Malik Shahzad Muzaffar (CERN)
Mircho Rodozov (CERN)
Oksana Shadura (University of Nebraska Lincoln (US))
#include <vector>

**Textual Include**

- **Expensive**
  - Reparse the same header
- **Fragile**
  - Name collisions

**Precompiled Headers (PCH)**

- **Inseparable**
  - Storing precompiled header information (same as modules)
  - Stored in one big file
    - Inseparable

**User Code**

```cpp
Rcpp
#include <header.h>
...
double PI = 3.14;
// => double 3.14 = 3.14;
```

**Modules**

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

- ROOT sometimes requires parsing of sizeable chunks of C++ code to aid user actions at the prompt, dictionary setup, and various interpreter services.
- C++ Modules are designed to minimize the reparsing of the same header content via an efficient on-disk representation of C++ Code.
Dictionaries in ROOT

- ROOT provides **automatic generation of serialization information (I/O) and automatic header and dependency resolution (partially) via dictionaries**
  - Dictionary is an extra file C++ needs to be synthesized and compiled into the defining library

```
UserLibrary
- inc/a.h
- inc/…
- src/A.cxx
- src/…

UserLibrary selection file
[LinkDef.h or selection.xml]

rootcling

G__UserLibrary.cxx
libUserLibrary.rootmap
libUserLibrary_rdict.pcm

libUserLibrary.so

ROOT

g++
```
Dictionaries in ROOT

UserLibrary selection file
[LinkDef.h or selection.xml]

UserLibrary
- inc/a.h
- inc/…
- src/A.cxx
- src/…

rootcling

G__UserLibrary.cxx
UserLibrary.pcm

g++

libUserLibrary.so

ROOT
Dictionaries in ROOT

_**G__UserLibrary.cxx**_ has several sections which require reprocessing immutable headers in runtime

They are replaced by the much more efficient C++ Modules

```cpp
const char* foo = R"RAW(
#include "TEveTrack.h"
#include "TEveTrackEditor.h"
#include "TEveTrackGL.h"
#include "TEveTrackProjected.h"
#include "TEveTrackProjectedGL.h"
#include "TEveTrackPropagator.h"
#include "TEveTrackPropagatorEditor.h"
#include "TEveTriangleSet.h"
#include "TEveTriangleSetEditor.h"
#include "TEveTriangleSetGL.h"
...
-"TEveGedNameTextButton", payloadCode, "@",
-"TEveGeoManagerHolder", payloadCode, "@",
-"TEveGeoNode", payloadCode, "@",
-"TEveGeoNodeEditor", payloadCode, "@",
...
-class __attribute__((annotate(R"ATTRDUMP(An arbitrary polyline with fixed line and marker attributes.)ATTRDUMP"))) ..
)"
```

V. Vassilev, Y.Takahashi, D.Lange, O. Shadura  
C++ Modules in ROOT and Beyond  
CHEP 2019
C++ Modules in ROOT

- Technology Preview released in ROOT 6.16
- Default on UNIX in ROOT 6.20
C++ Modules in ROOT. Loading strategies

- **Eager loading** of *pcm files at startup in ROOT 6.20
  - Small linear performance overhead depending on the number of modules
- **Delayed loading based on global module indexing (GMI)** planned for ROOT 6.22
  - No overhead, pay only for what you use!

Functionality is there, now optimizing!

V. Vassilev, Y. Takahashi, D. Lange, O. Shadura  
C++ Modules in ROOT and Beyond  
CHEP 2019
Global Module Index (GMI)

- Global Module Index (\texttt{clang::GlobalModuleIndex}) is an aid for name lookup into modules
- Global Module Index returns a set of modules where a given identifier is present
  - This allows us to avoid eager loading of all modules and load only necessary ones on demand

- It is using key-value store concept -> on-disk hash table
- Populated by scanning on demand all identifiers in all modules
- Created at build time (it is a on-disk file)
GMI measurements

![Graphs showing GMI, C++ Modules, and PCH run time and RSS memory usage for root.exe and hsimple.C.](image-url)
C++ Modules in CMSSW

- Available in CXXMODULE_IB
C++ Modularization in CMSSW

Library without modulemap
- Header file pollute the rest of the library dependencies
  - The header duplication is problematic for performance but also triggers a lot of bugs when resolving the duplicate content

Library with modulemap
- Headers are persisted in the PCM
  - Header duplication is reduced by referencing the dependent module

Top-down approach

Bottom-up approach

Used in CMSSW
C++ Modules-aware dictionary generation in CMSSW

[module.modulemap]

- Definition file of headers to build a PCM in Clang
- Contain all “interface” headers, which are used by libraries

```cpp
module "MathCore" {
  module "TComplex name" { header "TComplex.h" export * }
  module <name of the file> {
    header <relative path to the header file location> }
  export libMathCore.so
}
```

modulemap will contain all interface header files

→ Automatic generation of modulemap
Dictionaries Using C++ Modules in CMSSW

[Automatic generation of modulemap]

- CMSSW has “interface” headers
  - Exposed to libraries outside
- Automatically generate the modulemap by adding interface headers
  - Modulemap needs to be generated before the execution of genreflex
- Design features
  - Headers can be easily vetoed from generation of modulemap via SCRAM settings
C++ Modules-aware dictionary generation in CMSSW

Just add flag `-cxxmodule`

SCRAM Build system → Modulemap → C++ Compiler

Genreflex (rootcling) → CMS Dictionaries

CMS C++ Modules PCMs → CMS Runtime

CMS Libraries

CMS C++ Modules

PCMs

V. Vassilev, Y. Takahashi, D. Lange, O. Shadura

C++ Modules in ROOT and Beyond

CHEP 2019
The largest part of the speedup is in the initialization phase
C++ Modules in ROOT. Roadmap
C++ Modules in ROOT. Roadmap

- Modular ROOT and non-modular software stacks should work seamlessly but at no performance benefits (and costs)
- **ROOT 6.20 (January 2020):** C++ Modules will be default for all UNIX platforms (implementation based on eager module loading)
- **ROOT 6.22 (mid 2020):** C++ Modules by default for OS X (implementation based on global module indexing (GMI))

We are here to help to migrate your stack!
Thank you for your attention!

For more questions please contact: vvasilev@cern.ch
root-cxxmodules@cern.ch
Backup
Motivation of C++ Modules

Yuka Takahashi 13.03.2019

Migrating large codebases to C++ Modules, ACAT 2019
C++ Modules in CMSSW. Mechanism of the modulemap

[modulemap, modulemap overlay file, virtual modulemap overlay]

```
CMake
directory

/ $build_dir

usr/

some directory/

*.h (headers)

include/

module.modulemap

Virtual Modulemap Overlay File (VMOF)

name : "usr/include"

contents : {
    name : "module.modulemap"
    external-contents :
        "path/to/stl.modulemap"
}

Location of system headers
(Generated from CMake)

Relative path to stl.modulemap

module "stl"
    module "algorithm"
        header "algorithm"

module "header.h"
    header "some/directory/headers.h"

directories

files

22