C++ runtime modules

Raphael Isemann

ROOT
Data Analysis Framework

https://root.cern

Overview C++ modules

- clang’s C++ Modules optimize header parsing
  - C++ module = precompiled headers
  - clang can load on-demand code from modules
- Developed by Google, Apple in clang
  - They want a faster compiler
  - Code is open source and they collaborate with us

Overview C++ modules

- Work similar to precompiled headers (PCHs)
  - We already use a PCH in ROOT
  - But only one PCH is allowed at a time
  - Multiple PCHs at a time ➔ C++ modules
- ROOT’s interpreter uses clang
  - We can make use of C++ modules in ROOT
  - Faster compilation times in clang ➔ faster ROOT runtime when interpreting

Requirements for C++ modules

- clang’s C++ modules work with C++11/14/17
  - No module specific C++ code necessary
- Few new requirements:
  - Header need to be standalone
    - Need to contain all required includes
    - Shouldn’t rely on macros defined outside their visibility
    - It’s easy to test for this, so please do!
  - No cyclic dependencies between C++ modules

C++ modules

- Module configuration happens via modulemaps
  - Textual files containing mostly just a list of headers
  - Need to be placed in the specific include directory
- If clang sees an include to a module header, it builds the module if necessary and attaches it.
- Module files are stored in a cache directory
  - For ROOT that’s the `lib/` directory for now.

Moving CMS/ROOT to C++ modules

Workplan:

1. Compile ROOT with C++ modules
2. Generate C++ modules with rootcling
3. Use C++ modules during ROOT’s runtime
4. Compile CMS with C++ modules
5. Enable modules for CMS runtime

1. Compile ROOT with C++ modules

- New ROOT build option `-Dcxxmodules=On`
- Compiles ROOT with clang's C++ modules
- Allows fast compatibility testing with modules
  - nightly builds of clang check for module regressions
- Status: Completed

System modulemaps

- We need C++ modules for the system (STL, libc)
  - More efficient than copying them into all modules
  - Also fixes bugs because we avoid merging contents
- We ship system modulemap files
  - Only Apple ships some (broken) modulemaps.
- System modulemap files are placed via VFS
  - VFS = clang’s virtual file system overlay feature
- Will be important when we go out to users

Possible future items

- Making build system more aware of module dependencies ➔ compilation speedup
  - CMake doesn’t know about module header dependencies yet ➔ no good scheduling
  - If multiple clang instances try to build same module, they all wait just wait on the first clang build

2. Generate C++ modules with rootcling

- rootcling also generates C++ modules now
- Activated by setting env variable `ROOT_MODULES=1`
- rootcling now needs to respect dependencies
  a. If dict A depends on B, then B needs to be generated before A.
- Status: Completed

On demand building

- Clang can build modules on its own when it encounters them
  a. Used for the system C++ modules
- Should NOT be used for dict C++ modules
  a. Comments etc. will not be stored then
  b. We will see the corresponding errors during runtime
  c. At the moment NOT yet a rootcling error.

System modules

- System modules like STL/libc/boost/Geant4 have no specific rootcling invocation
  a. They currently get built as a side product by clang's implicit build mechanism
  b. Not as efficient as explicitly building them (nested module build take a lot of memory).
  c. Requires that all dependencies are used from a rootcling header.

Possible future work items

- Generating the module from the interpreter brings in a lot of clutter into the module file
- The dependency requirement isn't very user friendly (but seems hard to avoid)
- See root evolution proposal about rootcling refactor [RE-0003]

3. C++ modules during ROOT’s runtime

- ROOT runtime uses the generated modules
- Allows mixing non-module/module dicts
  a. Only if a dict has a module we load it.
- Still using rootmaps for autoloading
  a. But behind the scenes we use modules now
- Status: Completed (1610/1650 tests pass)

Runtime performance

- ~25% speedup on startup in normal tutorials
- ~35% speedup on parsing-heavy tutorials
  a. e.g. when using boost
- Same speed for ROOT PCH modules
  a. They already use the PCH which already is a module
- Runtime should be in general always equal or better than without modules.
- Tracking page: https://teemperor.de/root-bench/benchmarks.html

Performance optimizations

- Lots of chances to optimize speed/memory.
- Most optimizations will also help PCH.
- No more iterating over the whole AST
- We should finish the template specialization patch.
    a. Hurts PCH, really hurts the C++ modules.
- We should keep an eye on the benchmarks.

Fixing autoloading

- Currently preloading modules/PCH is fixing some autoloading issues.
    a. E.g. Decls in namespaces seem to be broken
- Maybe we should attempt to fix that
    a. Reduced performance because we (correctly) load more things now that we didn’t do before.
    b. Improves C++ modules memory a lot.
    c. CMS seems to have already fixed this.

Thanks for your attention!

Also thanks a lot to:

Vassil for supervising, Guilherme, Enrico, Javier, Xavier, Axel, Martin(not here), Enric, Bertrand, Danilo, Kim, Oksana, Pere, Phillippe and many more...

Runtime memory

- Modules call `mmap` on module files
  a. RSS memory therefore depends a lot on the kernel and how much it loads the files into memory
- Measured changes to alloc. memory are +/-20%.
- Memory consumption depends on user code:
  a. Many sparsely used includes ➔ Good improvements
  b. Already parsing-optimized code (e.g. forward decls instead of includes) ➔ No improvements