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 better compilation times
  - 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
  - No cyclic dependencies between C++ modules

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

Compile ROOT with C++ modules

- New ROOT build option `-Dcxxmodules=On`
- Compiles ROOT with clang’s C++ modules
- Allows fast testing if ROOT codebase compatible with modules
    a. nightly builds of clang check for module regressions
- Status: Completed

Generate C++ modules with rootcling

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

Use 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

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

Compile CMS with C++ modules

- Same strategy as for ROOT
- Mostly adding missing includes and fixing typos.
- Also creating modules for external libraries
  a. boost, Geant4, HepMC, Pythia, ...
  b. Millions LOC of C++ code now available as modules
- Status: Completed

Enable CMS modules for CMS runtime

- Start building modules for CMS dictionaries
- We can partly start modularizing bottom-up
- Needs planning how we ship system modules
  a. We have a script that automates that, needs to be integrated into SCRAM.
- Status: Not started

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 ✓ (95%)
4. Compile CMS with C++ modules ✓
5. Enable modules for CMS runtime

Thanks for your attention