ROOT "on" C++ Modules

The feature "C++ modules" is expected to become part of the C++17 standard. A "C++ modules"-aware build system could reduce build times up to 50%. ROOT can use the feature further - to optimize the execution speed and reduce the memory footprint at runtime.

In this talk, I give a brief introduction of clang's implementation of the C++ modules. I present the experimental results in modularizing ROOT's build system and steps towards using the feature at runtime. I describe some of the encountered challenges during the conducted work.

Vassil Vassilev

USCMS, FNAL

ROOT Workshop 2015, Saas-Fee, Switzerland, 15.09.2015

Key Objectives

1. Improve correctness in some cases* (see Autoloading);
2. Avoid shipping header files with the binaries;
3. Speed up ROOT*;
4. Decrease ROOT's memory footprint*;
5. Enable future simplification of ROOT's dictionaries;
6. Enable future on-demand IO description generation;
7. Decrease ROOT and experiments' software stack compilation times.

---

* mainly when working with third party code. From ROOT's standpoint third party code means also experiment's software stack.

C++ Modules*

Main goal is to enable scalable compilation of C++ code.

---

*according to n4047: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4047.pdf

C++ Compilation Model

Translation Units

```cpp
// A.cpp

int pow2(int x) {
  return x*x;
}
```

```cpp
// B.cpp

extern int pow2(int x);
int main() {
  return pow2(42);
}
```

Each translation unit (TU) is independent. The communication problem is resolved via *name linkage*.

Linking and Compiler Copy-Paste

TU communication is done via external names. In order to minimize errors header files are introduced.

```
// A.h

int pow2(int);
```

```
// A.cpp

#include "A.h" // expanded textually to "int pow2(int);" and recompiled.
int pow2(int x) {
  return x*x;
}
```

```
// B.cpp

#include "A.h" // expanded textually to "int pow2(int);" and recompiled.
//#include <string> expands to 19290 LOC on OSX.
int main() {
  return pow2(42);
}
```

Translation Units to Module Units

```
// A.h

int pow2(int);
```

```
// A.h module interface aka modulemap
module A {
  int pow2(int); // The compiler will export pow2 as part of module A.
}
```

```
// A.cpp

#import A; // compiles A.h once (per config macro) and reuses it.
int pow2(int x) {
  return x*x;
}
```

```
// B.cpp

#include "A.h" // if module A is present behaves as #import A;
//#include <string> doesn't recompile 19290 LOC over and over again.
int main() {
  return pow2(42);
}
```

C++ Modules are planned to appear in the next C++ Standard in 2017.

Clang's Implementation of C++ Modules

- Very close to the proposal;
- Add some additional features which ROOT can use further (eg. autolink);

Potential Uses of C++ Modules in ROOT

- Compile time - compile ROOT with clang with -fmodules enabled and reduce up to 50% the build times*;
- Runtime - use the compile time modules for other purposes at ROOT's runtime (through cling);

---

* according to clang C++ Modules experts.

Potential Impact on the Experiments (in particular CMS)

Once the feature is implemented (with some extra work on the experiment's side):

- Avoid having to ship the header files with the binaries;
- Speed up the interaction with non-ROOT libraries, no header parsing or extra forward declarations, resulting in less memory consumption.

Work Plan

1. Compile time C++ Modules - make ROOT compile with -fmodules enabled;
   - Hunt down issues in clang (slow: hard to reproduce, hard to fix, slow patch review procedure);
   - Hunt down issues in ROOT.
2. Runtime C++ Modules - incorporate the compile time C++ modules deep in ROOT (to be elaborated once we 1. is done);
   - Simplify ROOT's autoloading;
   - Reuse the module maps instead of rootmap files;
   - Remove auto parsing;
   - Remove the dependency on header files to be installed with the binaries.

Experimental Results

```
make -j4 CXXMODULES=1 CXX=/recent/clang/bin/clang++ lib/libGpad.so
```

Compile libGpad.so (with its dependencies) with C++ modules enabled shows ~18% compilation speedup, from 6m20s to 5m10s on my machine for a debug build.

A few issues with the current module organization:

- Giant modules (1 module per library);
- Implicit modules* - 50% longer module build times;
- Better build system integration is needed.

---

* https://llvm.org/bugs/show_bug.cgi?id=20794

Thank you!

???