Modularizing ROOT

Vassil Vassilev

C++ Modules

- In essence they are a compilation optimization. Using C++ Modules is meant to reduce the compilation times
- ‘Pre-Parsed form’ of header files

C++ Modules. Concepts.

- Define a mapping between set of header files and libraries, via so called module maps.
  Read more on: http://clang.llvm.org/docs/Modules.html
- An example module map:

```
module ROOT {
  module core {
    header “TROOT.h”
    header “Tobject.h”
    //autolink libCore
    export *
  }
  export *
}
```

C++ Modules. Concepts.

- Differences from the PCH
  C++ modules can be attached/used in the middle of compilation, whereas the PCH needs to be present before the compiler starts compilation. PCH is configured with a predefined set of macros, C++ modules are flexible and macro-configurable.
- If 2 Modules can overlap, semantic merging must happen.
  The compiler must merge the declarations coming from both modules (sometimes violating the ODR). Very complex process. Hard to implement.

```
module ROOT {
  module core {
    header “TROOT.h”
    header “TObject.h”
    ...
    export *
  }
  module fictional {
    ...
    header “TObject.h” // or anything else #including TObject.h
  }
  export *
}
```

C++ Modules. Compile-time Modules.

- Compile with clang -fmodules flag. It will pickup any module maps on the include path and build reusable modules. Next time one uses a header file, its corresponding module will be picked up, reducing compilation times.

C++ Modules. Runtime Modules.

Use compile-time C++ modules for other purposes at ROOT's runtime (through cling). Eg:

- As a source of reflection information;
- As a much, much better and efficient replacement for the autoloading
- As a better replacement for the dictionaries.

Plan

We thought modules are very broken. Wrong!

- I decided to start the initial work on them, continuing Cristina's progress.
- Build LLVM and Clang standalone with compile-time C++ modules enabled.
- Build ROOT with compile-time modules enabled.
- Turn compile-time C++ modules into runtime C++ modules.

Building LLVM & Clang with C++ modules

- Got recent clang.
- Created modulemaps for libc and libc++
- Ran LLVM's build
- Report and fix a few issues.
- All building fine now. A few patches on cfe-commits waiting for my action.

Building ROOT with C++ modules

- Got recent clang.
- Created modulemaps for libc and libc++
- Ran LLVM's build
- Report and fix a few issues.
- All building fine now. A few patches on cfe-commits waiting for my action.

Issues with ROOT's (non-modular) design

- Core must be one module.
- Dependencies of math_core to hist
- Funny configuration macros in RooFit
- Interesting implementation of truetype fonts header files
- Dictionaries are walking beyond the edge of the C++ standard.

ROOT builds and runs with compile-time C++ modules

All N M SLOC builds except:

- libhist dictionary
- Vc
- TLinearFitter (a bug waiting to be reduced and fixed)

ROOT build times with C++ modules on

Mode		1	2	3
Modules	real usr sys	10m41.357s 28m27.412s 1m21.774s	10m32.134s 28m12.473s 1m24.633s	10m31.730s 28m14.331s 1m24.166s
No Modules	real usr sys	12m44.682s 36m40.274s 1m19.758s	12m37.036s 36m33.107s 1m21.848s	12m35.165s 36m31.370s 1m20.773s

Resulting in ~16% compilation speedup.

I expect with better organized modulemap, even bigger speedups.

ROOT compile-time C++ modules performance

Only ~16% because:

- ~610 modules build, which is much more than it should be for ~90 libs
- A better modulemap/build system integration is needed
- I believe that clang 3.6 or 3.7 will fully support C++ modules