Optimize ROOT use of modules for large codebases

Jun Zhang

Intro to “Modules”

The term ***modules*** can have many meanings, but in this specific context, we’re talking about ***Clang Modules***. (Different from C++ standard modules)

Clang modules is actually a really misleading name, perhaps we should better call it ***better precompiled compiled headers.***

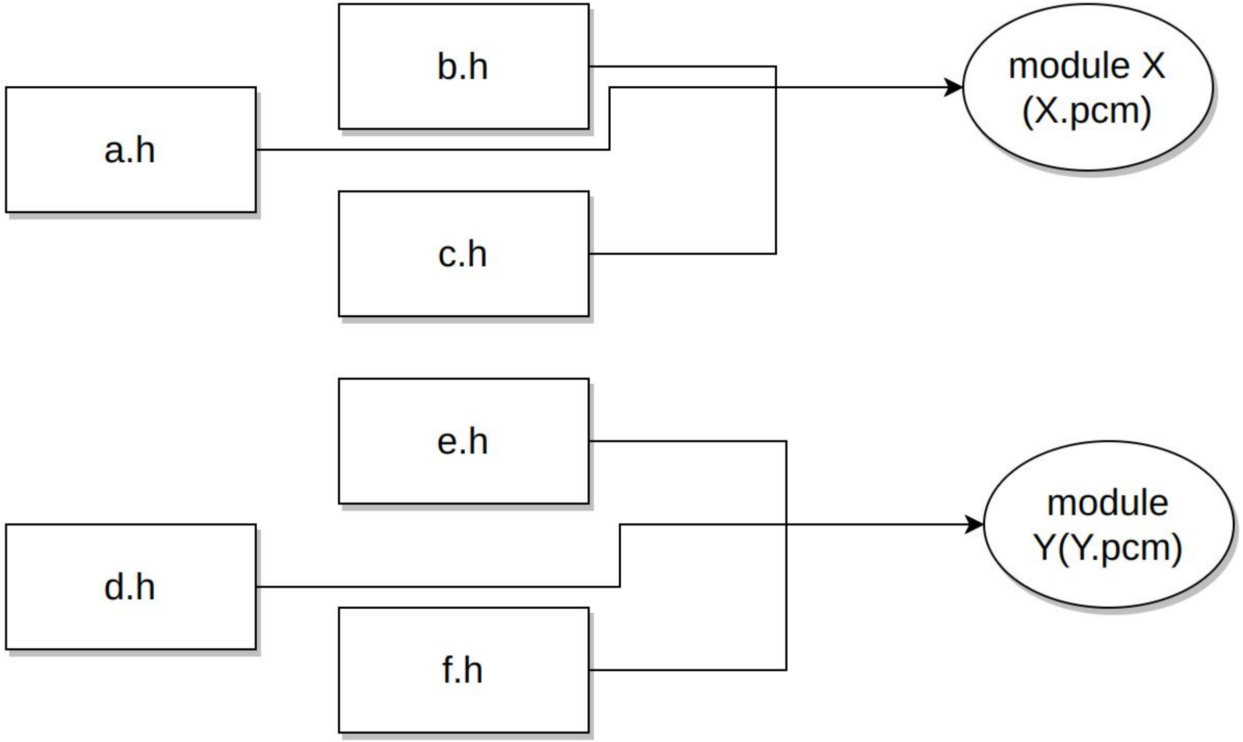
Why use modules?

What problem Clang modules trying to solve is that ***redundant header parsing*** in C++. Just imagining that you have some really large headers, and you need to parse them every time you do `#include`. This is inefficient.

In the static C++ compilation, everything is *fine* since the compilation and execution are in the different preiods. But in the interactive C++ (like cling or clang-repl), we're not that lucky as we have to parse headers at runtime. So that's why Clang modules comes to the play.

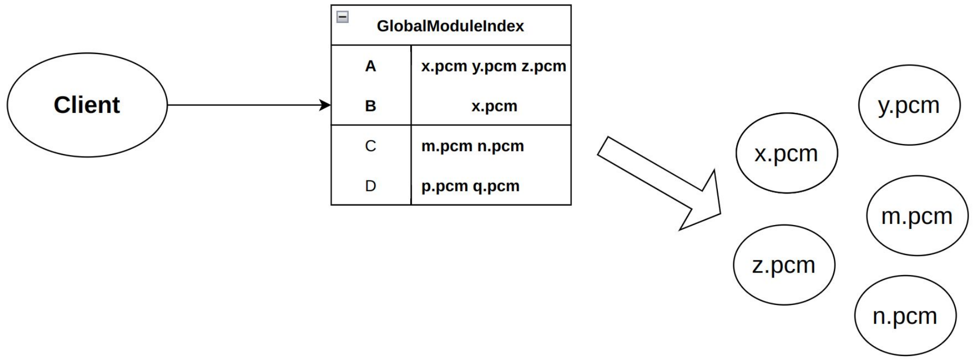
How modules work?

The basic idea of Clang modules is very simple. We divide the headers into Modules according to their connection to each other, then parse the headers into AST states and serialize them to the disk. Then we use a modulemap to establish the relationship between the module and the headers, so the next time we see a *#include* we don't need to parse it again but deserialize the AST from the disk.


b.h
a.h
c.h
module X (X.pcm)
e.h
d.h
f.h
module Y(Y.pcm)

What’s GlobalModuleIndex?

In the interactive C++, it’s not enough to just use modulemap to make everything work, we also introduced GlobalModuleIndex, which acts as a database of Modules and identifier names.



The limitation of current GlobalModuleIndex

Currently GlobalModuleIndex works really good, but it also has limitation. one source of performance loss is the need for symbol lookups across the very large set of modules.

```
root [0]: edm::X
```

So when we have input edm, ROOT will try to load all modules that contain the identifier edm. But because edm is a NameSpaceDecl and if it contains many modules, the performance will suffer. After all, all we want is just X!

Current Work

The fix for the problem is fairly simple:

Add another flag in the GlobalModuleIndex to indicate if it’s a namespace, and don’t load corresponding modules when see an identifier that matches it.

Ref: PR10910

The issues we ran into

- We can just ignore top level namespaces queries, which not brings down huge memory footprint.

  ```
  auto X = A::B::C(42);
  ```

- We can confirm that current patch works with ROOT itself, but we haven't measure the improvement with CMSSW.

Dark corners under the hood

The root problem is that we're storing those identifers in a lexical way, we don't have the ability to know where the identifier exactly come from.

```
namespace A { namespace M{}} (module X)

namespace B { namespace M{}} (module Y)
```

That said, we can never really ignore those unnecessary module loadings…

Plan B

- How?
  After trying many times, we came up with a new approach. Maybe we can create a new module that forward declared all namespaces, then always load it first everytime.

- What's the result?

  The performance improvement is not bad (max memory preesure decreased from 1.1GB to 600 MB), but we're still loading some huge modules (boost.pcm).

  Still a few tests failing.

Hack in Clang itself?

If we can’t really reduce those module loadings, can we reduce their size instead?

- But where the major memory usage comes from?

  ASTReader.cpp#L3340

- How?

  Sparse Vector?

Oh no, GSoC!

Consider the difficulties we ran into and incoming deadline of GSoC, we can try to achieve our final goal step by step, and focus on merging the current patch first.

Thank you!