CHEP 2018, Sofia, Bulgaria

Optimizing Frameworks' Performance Using C++ Modules-Aware ROOT

Yuka Takahashi, Vassil Vassilev, Raphael Isemann
{yuka.takahashi, vvasilev, raphael.isemann}@cern.ch

1. Introduction

We will present our results and challenges with C++ modules in ROOT. ROOT was extended with experimental support for using C++ modules during runtime, with aims to reduce it's **memory usage and improve its correctness**.

2. C++ Modules in a Nutshell

- Header information is stored in precompiled PCM files
- No more header parsing during ROOT's runtime

In C/C++, the interface of a library is accessed by including the appropriate header files:

```
#include <stdio.h>
```

These textual includes cause well-known problems:

- **Compile-time scalability:** #include is copying the contents to the includer's code, so the parser has to reparse the same common header files multiple times, which is expensive.
- **Fragility:** Textual includes are influenced by previously defined macros. For example, macro PI is #defined in Rcpp library. Including this library while using local variable PI will end up in a redefinition error.

C++ modules is a mechanism to boost compilation time by precompiling headers into PCM files, where AST information can be lazily loaded.

3. From C++ Modules to Runtime C++ Modules

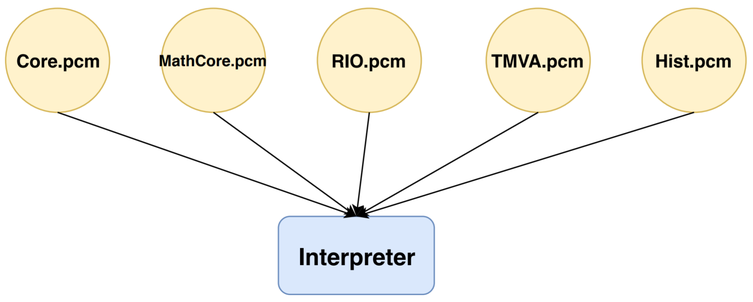

Figure 1: Runtime Modules (pcms). Each PCM file (E.g. Core.pcm) corresponds to a library (E.g. libCore.so).

C++ modules are able to reduce compilation times. However, **the compilation scalability issues in C/C++ becomes runtime issues** for an interpretative environment which ROOT provides. They span from slow prompt to slow IO.

4. Advantages over the Status Quo

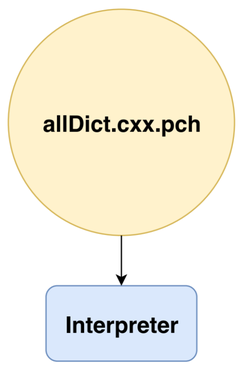

Figure 2: Precompiled Headers (PCH). Information of the header is stored in one file.

PCH files are precompiled header files and work similar to C++ modules. **The advantage of modules over PCH is that they can be used by experiments.** Experiments are still using textual includes as PCH only covers ROOT. PCH cannot be exported to experiments because of various technical limitations.

5. Results

5.1 Perfomance

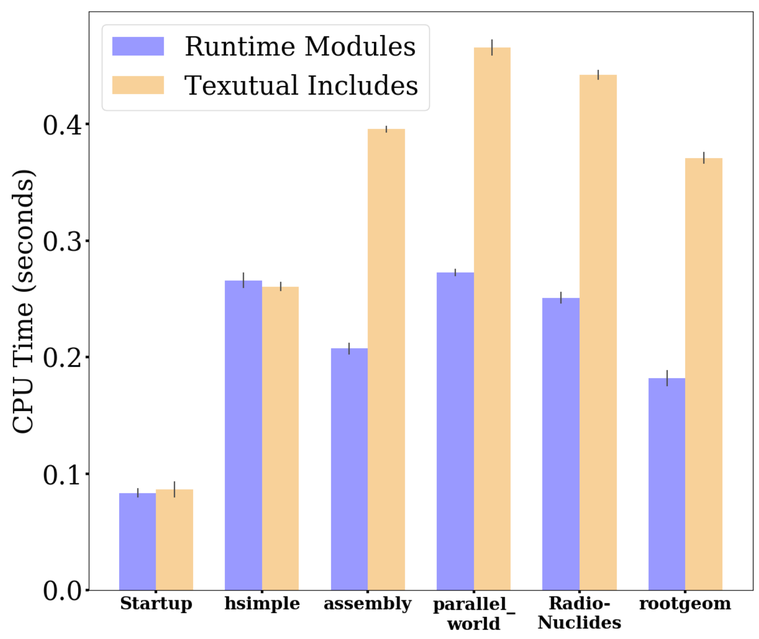

Figure 3: CPU Time required to run selected tutorials. The first column is displaying ROOT's time to start into an empty shell.

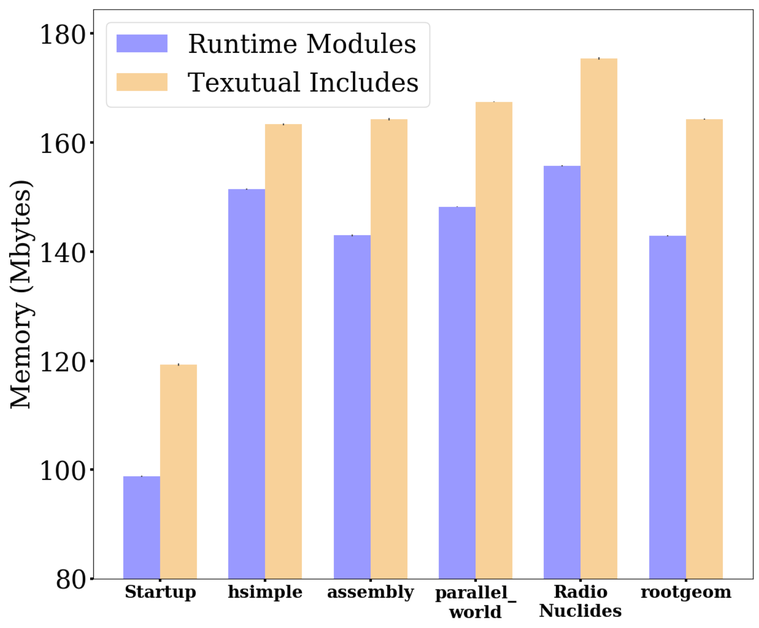

Figure 4: Residential memory used to run tutorials.

Fig.3 and Fig.4 are the performance results we receive from modules, compared to textual headers. The results are coming from synthetic benchmarks close to the experiment software stacks and in particular CMSSW.

5.2 Correctness

PCH:

```
$ bin/root.exe -l
root [0] gMinuit
IncrementalExecutor::executeFunction:
symbol 'gMinuit' unresolved while
linking [cling interface function]!
```

Runtime Modules:

```
$ bin/root.exe -l
root [0] gMinuit
(TMinuit *) nullptr
```

Runtime Modules are supporting more features than PCH. For example, gMinuit is an extern variable which cannot be autoloaded by ROOT at the moment. However, with modules, we can automatically resolve symbols and cases like those are now correctly handled.

6. Implementation

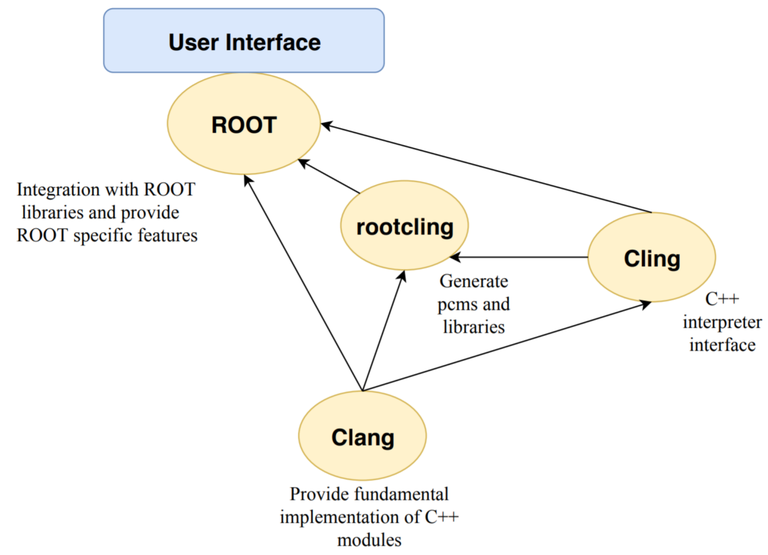

Figure 5: Visualization of ROOT interpreter core.

As shown in Fig.5, we are developing and using LLVM/Clang implementation of C++ modules, collaborating with developers from Google and Apple. Cling is a C++ interpreter developed by CERN, and rootcling is a dictionary generator for ROOT. We are implementing runtime modules in these parts while integrating ROOT with them.

7. Roadmap

- Compile ROOT with C++ modules
  Status: **Completed**
- Compile CMSSW with C++ modules
  Status: **Work in progress**
- Use runtime C++ modules in ROOT
  Status: **Mostly Done**
- Use runtime C++ modules in experiments
  Status: **Work in progress**

To summarize, runtime modules are mostly working, but need work to get better performance.

8. Conclusion

Here we briefly introduced our experimental runtime C++ modules support in ROOT and how it will affect experiments' software stacks. Modules are not yet competent compared to PCH, but are more flexible and have clear advantage over texual includes.

9. Future work

Runtime modules are still an experimental feature. Our ultimate goal is to make it default in ROOT and in experiments:

- Stabilize modules behavior and tests
- Adoption by experiments and other ROOT users
- Improve performance of loading modules

Further Information

- https://clang.llvm.org/docs/Modules.html
- https://root.cern.ch/
- https://root.cern.ch/cling