GlobalModuleIndex in ROOT and Cling

Arpitha Raghunandan

Google Summer of Code 2019: CERN-HSF

Mentors

- Vassil Vassilev
- Oksana Shadura
- Yuka Takahashi

Contents

Overview

Most software require libraries. Each software needs to access both the interface (API) and the implementation for each library. The C family of languages use appropriate header files to access the interface to a library. The implementation is then linked to the relevant library. C++ Modules are more sophisticated and are an alternative method to use these software libraries. This method eliminates several issues involved in using the C preprocessor to access the APIs of libraries. It further helps improve compile-time scalability.

ROOT, originating in CERN, is at the heart of research in high-energy physics (HEP). It is a modular scientific software toolkit or a framework used for data processing. HEP physicists use ROOT to analyze their data and to perform simulations. ROOT allows the storing, accessing and mining of data. ROOT uses Cling C++ interpreter as its backend. The Clang C++ compiler implements industry-standard C++ modules API. ROOT calls the Cling API to load C++ modules, where Cling is a C++ interpreter which originated at CERN.

ROOT requires implicit header inclusion as it has several features that interact with libraries. These headers are usually immutable, and reparsing is redundant. ROOT uses C++ modules which are designed to minimize this reparsing of same header content by providing an efficient on-disk representation of C++ code. Implementation of GlobalModuleIndex is a performance enhancer for this module method of using software libraries. GlobalModuleIndex is a mechanism to create a table of symbols and PCM names so that ROOT can load a corresponding library when a symbol lookup fails.

During this project, I understood the current working of ROOT, where libraries are accessed through Pre-Compiled Headers (PCH), as well as the module implementation, where libraries are accessed through Pre-Compiled Modules (PCM). I further worked with the implementation of GlobalModuleIndex in ROOT and Cling to improve the performance of ROOT by speeding up start-up time.

GlobalModuleIndex Design

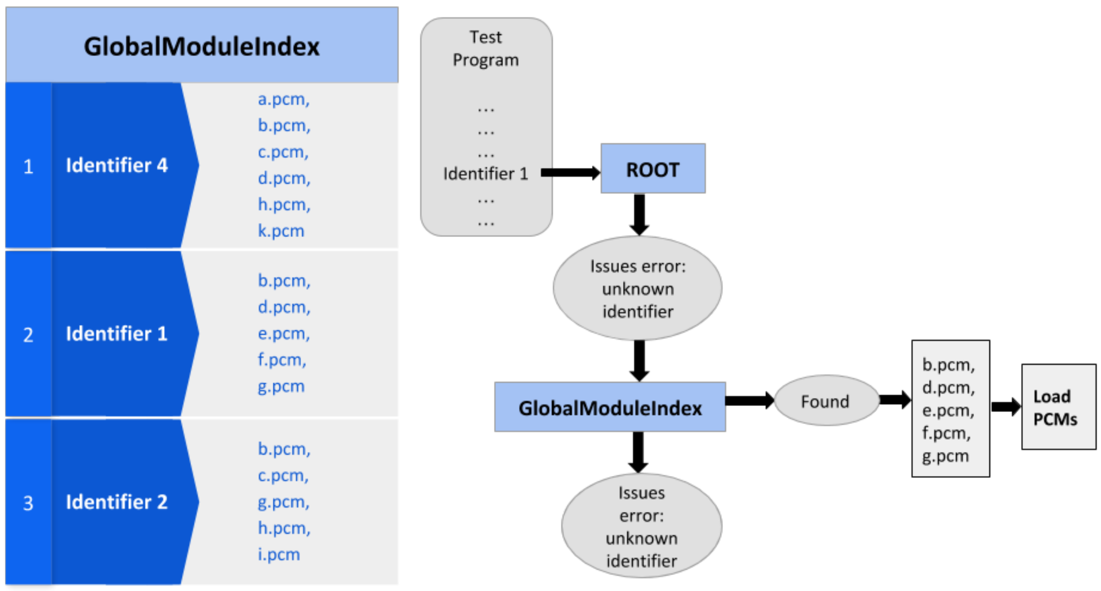
GlobalModuleIndex
1
Identifier 4
a.pcm, b.pcm, c.pcm, d.pcm, h.pcm, k.pcm
2
Identifier 1
b.pcm, d.pcm, e.pcm, f.pcm, g.pcm
3
Identifier 2
b.pcm, c.pcm, g.pcm, h.pcm, i.pcm
Test Program
Identifier 1
ROOT
Issues error: unknown identifier
GlobalModuleIndex
Found
b.pcm, d.pcm, e.pcm, f.pcm, g.pcm
Load PCMs
Issues error: unknown identifier

Figure 1

ROOT requires many libraries; textual includes are expensive and fragile. The current use of PCH, although optimal, stores the complete header information in one big file, i.e. it is monolithic. These issues are solved using C++ modules. Here, each PCM file (ex: a.pcm) corresponds to a library (ex: liba.so). As a result, modules can greatly help in improving performance.

GlobalModuleIndex is an on-disk hash table containing all identifiers present in all PCMs. It can help improve ROOT's performance with modules technology by speeding up its start-up time. Currently, ROOT at start-up loads all the PCM files. However, with GlobalModuleIndex, only the required modules can be loaded at start-up time.

On the first run of ROOT, if no modules.idx is found, the GlobalModuleIndex is created. All modules are loaded, and their lookup table is serialized. When an identifier is encountered, the GlobalModuleIndex is checked if the identifier is unknown to ROOT/Cling/Clang. If the GlobalModuleIndex knows the identifier, it returns a set of modules containing it, which in turn get loaded. If not, an Unknown Identifier error is issued. This can be seen in Figure 1.

Implementation

I first started working on understanding the initial implementation of GlobalModuleIndex, developed by my mentor Vassil. I found and compared the (number of) modules loaded both with and without the GlobalModuleIndex implementation for:

- ROOT start-up time
- tutorials/hsimple.C test
- tutorials/geom/geometry.C test

I also found the identifiers which cause the PCMs to be loaded in the above 3 cases, which can be seen here, here and here. I further found the methods which cause the above PCMs to be loaded in the 3 cases. I also identified a few ROOT identifiers (not in stl and libc) which require these modules to be loaded.

Rdict PCMs were generated by Rootcling to store information needed for serialization efficiently. The GlobalModuleIndex showed better performance without these Rdict PCMs. I found the *rdict.pcms that get loaded for the above three cases to understand why this happens.

One of my significant contributions over the summer was working on fixing failing tests. Initially, over 450 tests were failing, which was reduced to 50 tests by cleaning the code. Adding this change to the code when running with modules on helped fix some of the failing tests. Another temporary solution to fix the failing tests was introducing FIXMEModules. It is a list of PCM file names which should not be loaded at start-up time, but loading them fixes a large number of failing tests. Only 23 tests now fail with the current state of GlobalModuleIndex.

I presented my work at every C++ Modules Biweekly Meetings. This is a link to my presentation, which contains more details about my contribution. I further gave a 15-minute presentation at the IRIS-HEP meeting explaining my work over the summer.

Building and Running ROOT

To use the C++ module technology in ROOT, ROOT must be built using the **`-Druntime_cxxmodules=On`** flag.

Without the GlobalModuleIndex implementation, running *`root.exe -l -q tutorials/hsimple.C`* will load over 80 modules. The GlobalModuleIndex implementation reduces the number of modules loaded to around 50.

Hence, the GlobalModuleIndex implementation considerably reduces the number of modules loaded by ROOT.

Patches

As part of Google Summer of Code 2019, I worked on GlobalModuleIndex implementation in ROOT, that can be found in the following patch: https://github.com/root-project/root/pull/4016. The patch is still under review and will be landed to the ROOT repository soon.

My other contributions during the Google Summer of Code 2019 include:

- https://github.com/root-project/roottest/pull/324: This is to reduce the scope of the exclusion of a failing test.
- https://github.com/root-project/root/pull/3635: This is to reduce ExcludeModules to decrease dependencies on rootmap files.
- https://github.com/root-project/root/pull/3531: This is to add additional stl headers to stl.modulemap.
- I am working on fixing a bug in HeaderSearch.cpp in Clang. The bug arises when PrebuiltModulePath and ModuleCachePath both point to the same folder. I am trying to come up with a reproducer test for this bug and will open a Phabricator review soon.

Performance Results

My next major contribution was evaluating the performance of the GlobalModuleIndex implementation. The performance measurements were carried out for three cases:

- When running ROOT master with **`-Druntime_cxxmodules=Off`**
- When running ROOT master with **`-Druntime_cxxmodules=On`**
- When running ROOT master with **`-Druntime_cxxmodules=On`** and the **`GlobalModuleIndex`** implementation

Links to the final performance results:

- https://tinyurl.com/Performance-Measurements-1
- https://tinyurl.com/Performance-Measurements-2

The second link further contains three sheets indicating the name of the libraries/modules loaded under the three conditions.

Graphical representation of results:

Figure 2 shows a plot of the User CPU time (in seconds), Figure 3 shows the System CPU time (in seconds) and Figure 4 shows the Maximum Resident Set Size (in kbytes) for the three cases mentioned above. As can be seen, the ROOT master with `-Druntime_cxxmodules=On` performs a lot better with the GlobalModuleIndex implementation. However, the current implementation is not an optimal one, and hence, its performance is still not better than ROOT master with `-Druntime_cxxmodules=Off.` There is a work in progress to help further improve its performance.

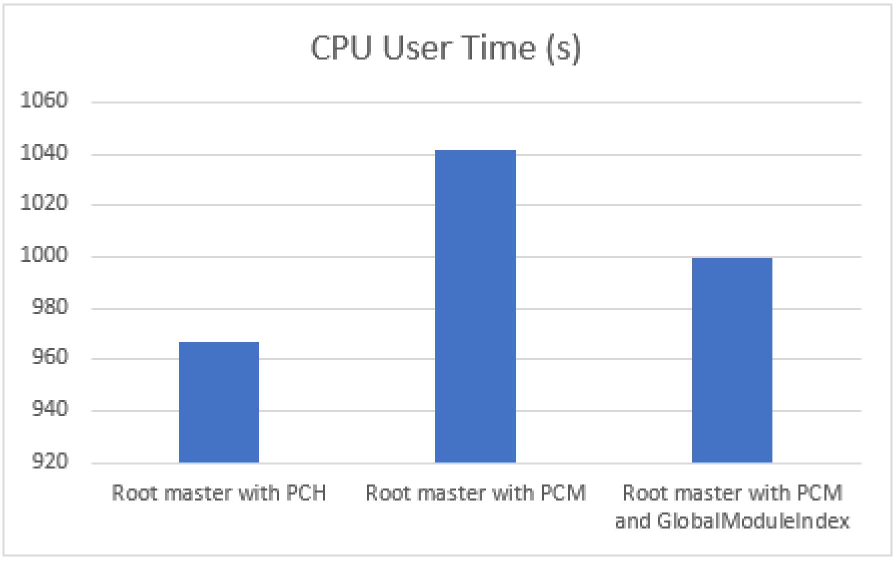

Figure 2

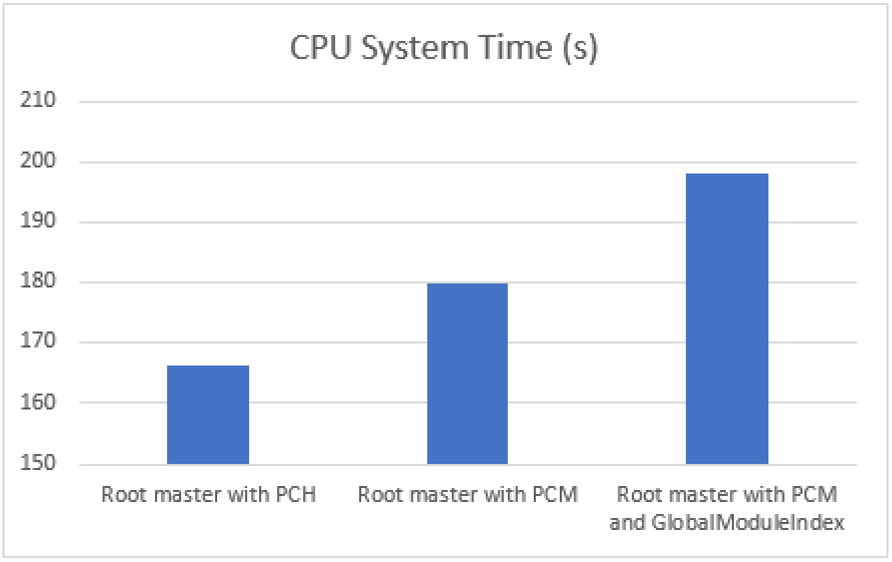

Figure 3

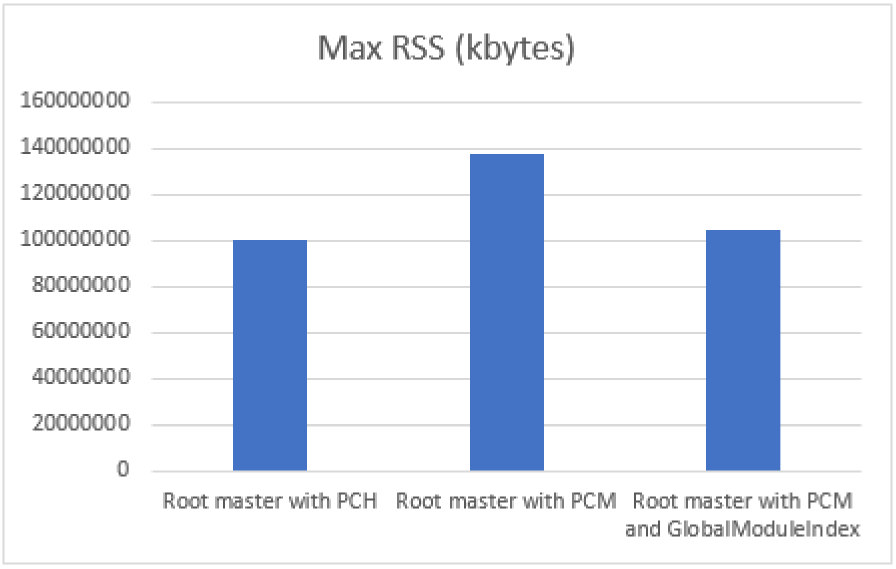

Figure 4

Two scripts were developed to help with performance measurements:

- https://gist.github.com/arpi-r/6d80d6c026d23ad7017706716de15786
  This link consists of three scripts which must be placed in the build directory. The scripts must be named the same as in the gist. The scripts can be run using the following command in the build directory:
  ```
  bash ./runctest1.sh
  ```
  The script creates a folder named loadedlibraries in the build directory. It produces a file in this directory for each test/tutorial run by ctest. Each file is named after the test/tutorial name and contains a list of library/module names which are loaded when it runs. Once all the tests/tutorials are run by ctest, the script also generates a graph indicating the number of modules/libraries loaded for the hsimple.C tutorial as well as the test/tutorial for which the maximum and the minimum number of libraries/modules are loaded.
- https://gist.github.com/arpi-r/fc0f7e691f72217e296a0fe627fc8565
  This link consists of two scripts which must be placed in the build directory. The scripts must be named the same as in the gist. The scripts can be run using the following command in the build directory:
  ```
  bash ./runctest2.sh
  ```
  The script creates a folder named perfmeasurements in the build directory. It produces a file in this directory for each test/tutorial run by ctest. Each file is named after the test/tutorial name and contains the user CPU time, system CPU time and maximum RSS details when it is run. These details for each test/tutorial for the three cases mentioned above were then extracted using another python script.

Future Work

The current implementation of GlobalModuleIndex returns a superset of the required modules. Hence, it is underperforming, and the implementation can be further fine-tuned to improve performance.

Currently, there is no distinction between the declaration and the definition of entities. If the **`identifier`** → **`module`** mapping can be made such that the definition is stored first, the number of modules loaded can be further reduced; this will, in turn, enhance the performance.

Despite not having achieved optimal performance improvement, the current GlobalModuleIndex implementation shows promising results. This technique can be further optimized to achieve higher performance with module technology in ROOT and Cling.

Acknowledgement

I would like to acknowledge my mentors for their support throughout this project.