GSoC Project proposal

Superbuilds for ROOT

Mentors:
Vassil Vassilev (Vassil.Vassilev@cern.ch)
Danilo Piparo (Danilo.Piparo@cern.ch)

Contact

Name: Pavlo Svirin
Email: paul.svirin@gmail.com
Github: pavlo-svirin

Timezone: EEST (GMT+3 hours)

Abstract

ROOT is a framework for data processing, born at CERN, at the heart of the research on high-energy, molecular and laser physics, as well as in astronomy. Every day, thousands of physicists use ROOT applications to analyze their data or to perform simulations.
ROOT has plenty of built-in components which allow to do:

- Histogramming and graphing to view and analyze distributions and functions,
- curve fitting (regression analysis) and minimization of functionals,
- statistics tools used for data analysis,
- matrix algebra,
- four-vector computations, as used in high energy physics,
- standard mathematical functions,
- multivariate data analysis, e.g. using neural networks,
- image manipulation, used, for instance, to analyze astronomical pictures,
- access to distributed data (in the context of the Grid),
- distributed computing, to parallelize data analyses,
- persistence and serialization of objects, which can cope with changes in class definitions of persistent data,
- access to databases,
- 3D visualizations (geometry),
- creating files in various graphics formats, like PDF, PostScript, PNG, SVG, LaTeX, etc.
- interfacing Python code in both directions,
- interfacing Monte Carlo event generators.

Because of such a variety of features, ROOT is a very large software which takes a very long time to compile. Currently ROOT compiles all of the components available within the source code distribution.
The goal of the project is to speed up the compilation process by letting users specify which components of the ROOT will be needed. This can be done by converting ROOT's CMake configuration into a set of "CMake External Projects" (superbuilds). Superbuilds can remove all of this cruft from the project's source repository, and enable you to more directly use the upstream project's build system as an independently built software component. It is basically a simple package manager that you make consistently work across your target build platforms, and if your target platforms have a common package manager you might consider using that instead.

Also, during the configuration process it could be possible to identify if ROOT is already installed in the destination folder, the components which are already installed and offer an option to skip their compilation and use these already installed components for current compilation.
The goal of this project is not to change the build system of ROOT, but to optimize it and offer users an option to select only the parts of the ROOT to be built. Partial builds for ROOT can allow the creation of “edition” builds if necessary. There is also an option to create an Ncurses menu-based configuration application (TUI, text user interface) which will simplify the configuration process and will let users not to remember names of ROOT subprojects, thus making build process easier.

Personal Information:

Name	Pavlo Svirin
Email	paul.svirin@gmail.com
Mobile Number	+380 63 5780001
Time zone	EEST (UTC + 2:00)
Github	pavlo-svirin
University	Kyiv University of Market Relations
Major	Management
Degree	Bachelor of Management
Current year	3rd year (Expected graduation 2025)
Availability	I am available for the entire project duration

Programming Experience:

I participated as a member of a joint project in CERN’s experiment ALICE. The goal of this work was to integrate different types of remote computing resources into Grid Computing Environment of ALICE. For these developments I was using PERL, Java and Python.
Also, I was integrating websockets into the AliROOT framework. AliROOT is an extension of the ROOT framework, which includes specific features required for computations of the data received by the ALICE experiment. For this project I was using C++.

Development Environment

I'm using CentOS 8 as the base operating system for development. Testing is done in a Docker container where all of the necessary packages for ROOT development is preinstalled.
ROOT configuration is developed for CMake 3.28, compilation is performed using GCC 10.
For the development I will be using ROOT v.6.30.04 source code taken from the official GitHub repository.

Qualification task

This task involved the following steps:

- **selecting the minimum list of ROOT components which produce a viable build; testing this list of components with reduced build; identification of dependencies among the ROOT components:**

ROOT contains around 120 components inside. Some of these components are essential for ROOT, these are "core", "interpreter" and "math". The rest of the components are optional.

The "interpreter" contains LLVM-based CLing C++ interpreter inside, this component is absolutely necessary for the rest of the components to be built. The "interpreter" component does not depend on any other component. Most of the components have dependencies based on other ROOT components, and, thus, to be built require other components automatically enabled.

- **modules.modulemap bug resolution**

There existed for years a ticket open in ROOT's Jira dedicated to a bug in ROOT build scripts. This issue prevented building ROOT on a system where a system-wide ROOT was already installed. The solution was to rename ROOT's module.modulemap file since its name was identical to the file name required by LLVM during the build process. The solution was successfully tested and included as a patch into the official source code of ROOT.

Project tasks

1. External projects for ROOT.
    1.1. Split ROOT into a set of external projects.
    1.2. Implement dependencies among external projects

1.3. Implement installation of external projects into the destination directory
1.4. Implement lookup of already built and installed external projects in the destination directory which will allow to skip long compilation of already present components.

2. Implement and test distributed module.modulemap
3. User interface
   3.1. Implement command-line selection of the projects to be built
   3.2. Implement correct processing of command-line arguments to “make” call
   3.3. Implement TUI tool for configuration of build process

Timeline

Community Bonding Period	
May 1 - May 26	● Engage with the community. ● Discuss goals which are specified for this project and introduce corrections if necessary. ● Establish regular meetings with the mentors. ● Set up the development environment. ● Identify all of the necessary third-party libraries which have to be installed and configured in order to proceed with the development.
Coding period, stage 1 begins	
Week 1 27.05.2024-02.06.2024	Define separate CMake external projects for ROOT components (task 1.1) **Deliverables**: ● description of each ROOT component in the form of CMake external project; such description will include source directory, configuration options, build and install commands.
Week 2 03.06.2024-09.06.2024	Implement and test dependencies among external projects. (task 1.2) **Deliverables**: ● External project descriptions with dependencies to other external projects
Week 3 10.06.2024-16.06.2024	Implement and test selection of target components from parameters passed to CMake (task 3.1)

	Deliverables: ● a CMake script which parses the list of required external projects received through a “cmake” argument and enables specified external projects together with their dependencies
Week 4 17.06.2024-23.06.2024	Implement and test correctness of installation of the ROOT components into the destination directory (Task 1.3) **Deliverables**: ● CMake installation script which copies all of the enabled and compiled external projects into the destination directory
Week 5 24.06.2024-30.06.2024	Check that arguments from make call (like “-j” and others) are passed and processed correctly by all of the external projects (Task 3.2) **Deliverables**: ● Updated code for external projects which takes into account command-line arguments and tunes building process accordingly
Week 6 24.06.2024-30.06.2024	Implement distributed modulemap file for ROOT (Task2) **Deliverables**: ● installation script which writes modulemap data into separate files and then adds “include” statements into the main modulemap file
Week 7 01.07.2024-07.07.2024	Perform integration of all of the deliverables from week 1-6 and run integration tests to check conformance of the system. **Deliverables**: ● stable and well-tested under different condition build system for ROOT which includes the developments of the weeks 1-6
Week 8 08.07.2024-14.07.2024	Buffer Week for any unresolved tasks of stage 1
Midterm Evaluations, stage 2 begins	
Week 8 15.07.2024-21.07.2024 Week 9 22.07.2024-28.07.2024	Design and implementation of ncurses-based TUI for ROOT compilation in which selection of components to be compiled can be made (Task 3.3) **Deliverables**: ● A TUI tool which will allow selection of the components to be built, will form the arguments for cmake call properly and will call cmake.
Week 10 29.07.2024-04.08.2024	Development of the CMake modules which identify already installed ROOT components and tune configuration variables accordingly (Task 1.4)

Week 11 05.08.2024-11.08.2024	**Deliverables**: ● CMake modules which look for the components in in the CMAKE_INSTALL_PREFIX directory for the known ROOT components which take lot of time to build (like Interpreter component);
Week 12 11.08.2024-18.08.2024	Integration and testing of all of the deliverables from weeks 1-11 into the single build system **Deliverables**: ● stable and well-tested under different condition build system for ROOT which includes the developments of the weeks 1-11
Week 13 19.08.2024-25.08.2024	Buffer Week for any unresolved tasks of stage 2
Week 14 26.08.2024-01.09.2024	Extended testing, developing documentation, presenting the work. **Deliverables**: ● test cases; ● demonstrated reduction of build time; ● demonstrated flexibility of the new approach to build ROOT according to user's needs; ● blog post about the achieved results; ● presentation at the compiler-research.org team meeting.

Other time commitments during the project period

During the time of the project I will be also helping at the National Academy of Sciences of Ukraine with the project, which is concerned with databases and web service (around 10 hours per week).
The classes that I would like to take in summer time are dedicated to unmanned aerial vehicle engineering (6 weeks, 60 hours). Most of the classes are taught online and held on weekends. However, this depends whether these classes will be scheduled for summer months.

References

1. ROOT website: https://root.cern
2. CMake Superbuilds and Git Submodules: https://www.kitware.com/cmake-superbuilds-git-submodules/
3. CMake external projects documentation: https://cmake.org/cmake/help/latest/module/ExternalProject.html