Profile-guided optimization

Speeding up LHCb software through compilation optimization

Oscar Buon

June 19, 2023

Supervisor: Sebastien Ponce

CERN

Intergovernmental particle physics on France-Switzerland border.
About 15,000 people working at CERN.
Made discoveries that led to Nobel Prizes.
World Wide Web was invented at CERN.


WHERE THE
WEB
WAS BORN
In the offices of this corridor, all the fundamental technologies of the World Wide Web were developed.
Started in 1990 from a proposal made by Tim Berners-Lee in 1989, the effort was first divided between an office in building 31 of the Computing and Networking Division (CN) and one in building 2 of the Electronics and Computing for Physics Division (ECP).
In 1991 the team came together in these offices, then belonging to ECP. It was composed of two CERN staff members, Tim Berners-Lee (GB) and Robert Cailliau (BE), aided by a number of Fellows, Technical Students, a Coopérant and Summer Students.
At the end of 1994 Tim Berners-Lee left CERN to direct the WWW Consortium (W3C), a world-wide organization devoted to leading the Web to its full potential. The W3C was founded with the help of CERN, the European Commission, the Massachusetts Institute of Technology (MIT), the Institut National pour la Recherche en Informatique et en Automatique (INRIA), and the Advanced Research Projects Agency (ARPA).
In 1995 Tim Berners-Lee and Robert Cailliau received the ACM Software System Award for the World Wide Web. In 2004, Tim Berners-Lee was awarded the first Millenium Technology Prize by the Finnish Technology Award Foundation.
The CERN Library
June 2004

Large Hadron Collider (LHC)

- Large Hadron Collider: particle collider
- $27km$ (biggest in the world)
- $\sim 100m$ underground

LHCb

- One of the 4 main experiments installed on the LHC.
- More than 1,200 people working for the collaboration.
- Studies asymmetry between matter and antimatter via b-physics. Collision of hadrons (heavy particles).

Physics Computing

The detector needs an important computing infrastructure to work properly.
The detector produces about TB/s of data that are analyzed to reconstruct the trajectories and the properties of the particles.
Then statistics are made from these reconstructed particles.
Moreover simulations are made through the software. Then statistics are comapred to the simulations.
Some computing analysis are dispatched in servers arround the world (grid).

Computing group

A first level software will filter data to keep interresting tracks.
Then data are stored in a buffer for hours to be analyzed in a second level software.
Some computing analysis are dispatched in servers arround the world (grid).
The computing group has to maintain this infrastructure for LHCb.

Software

The software is a stack programms:

- The base of the stack is composed of software like ROOT or Gaudi that are shared with other experiments.
- On top of them there are Detector, LHCb or Lbcom that modelise the experiment.
- And then there are Rec that reconstruct the tracks and the simulation programms.
- Some other programms are to use or to test the others, like Moore.

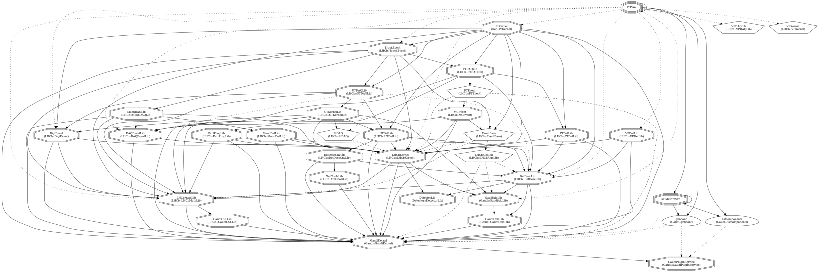
There are several million lines of code which some are 30 years old. Moreover the code is mainly written by non-software engineers. All of this is running on thousands of multithreaded Linux servers.

Table of contents

Graphviz for printing graphs

Printing graphs was usefull to have an idea of the dependencies.
Graphviz in CMake to create dependency graphs:

- CMake arg: `--graphviz=path/to/files.dot`
- Convert to SVG:
  `dot -Tsvg -o path/to/file.svg path/to/file.dot`
- Make arg: `CMAKEFLAGS="--graphviz=path/to/files.dot"`
- Setting `GRAPHVIZ_EXTERNAL_LIBS` to `FALSE` is usefull.


PrPixel
PrKernel
(Rec::PrKernel)
VPDAQLib
(LHCb::VPDAQLib)
VPKernel
(LHCb::VPKernel)
TrackEvent
(LHCb::TrackEvent)
FTDAQLib
(LHCb::FTDAQLib)
UTDAQLib
(LHCb::UTDAQLib)
FTEvent
(LHCb::FTEvent)
MuonDAQLib
(LHCb::MuonDAQLib)
UTKernelLib
(LHCb::UTKernelLib)
MCEvent
(LHCb::MCEvent)
DigiEvent
(LHCb::DigiEvent)
DAQEventLib
(LHCb::DAQEventLib)
PartPropLib
(LHCb::PartPropLib)
MuonDetLib
(LHCb::MuonDetLib)
SiDAQ
(LHCb::SiDAQ)
UTDetLib
(LHCb::UTDetLib)
EventBase
(LHCb::EventBase)
FTDetLib
(LHCb::FTDetLib)
VPDetLib
(LHCb::VPDetLib)
DetDescCnvLib
(LHCb::DetDescCnvLib)
LHCbKernel
(LHCb::LHCbKernel)
LHCbAlgsLib
(LHCb::LHCbAlgsLib)
XmlToolsLib
(LHCb::XmlToolsLib)
DetDescLib
(LHCb::DetDescLib)
LHCbMathLib
(LHCb::LHCbMathLib)
DetectorLib
(Detector::DetectorLib)
GaudiAlgLib
(Gaudi::GaudiAlgLib)
GaudiCoreSvc
GaudiGSLLib
(LHCb::GaudiGSLLib)
GaudiUtilsLib
(Gaudi::GaudiUtilsLib)
genconf
(Gaudi::genconf)
listcomponents
(Gaudi::listcomponents)
GaudiKernel
(Gaudi::GaudiKernel)
GaudiPluginService
(Gaudi::GaudiPluginService)

Context

Worked on a stack with one main CMake:

`https://gitlab.cern.ch/clemenci/lhcb-super-project-template/`

Target: `x86_64_v3-centos7-gcc11+detdesc-opt+g`

Library types

- STATIC: an archive of object files that are merged to the executable at link time.
- SHARED: .so (Linux) file that is automatically linked to the executable at runtime (when starting it). Allows several executables to link with a library installed on the system.
- MODULE: same as shared but the module is loaded on demand via the `dlopen` function, so not automatically.

Static libraries

Gaudi provides CMake functions to create libraries, modules or executables. Then we just have to create a new function for static libraries by copying the one for shared libraries:
`gaudi_add_library` ⇒ `gaudi_add_static_library`
And inside remplacing `SHARED` by `STATIC` in `add_library` function call.

Static modules

By using previous static libraries. Need to add:

- `-Wl,--whole-archive` link option while linking to the final executable.
- `-Wl,--whole-archive` and `-Wl,--allow-multiple-definition` link option to the executable.
- the global link option `-Wl,--export-dynamic`

Explanation of flags

- `-Wl,--whole-archive` because without it the linker removes the code in the module that registers it. Note that this could be an error in the linker because that behavior was not the same when using object files instead of static libaries.
- `-Wl,--allow-multiple-definition` because `-Wl,--whole-archive` causes symbols to be included several times.
- `-Wl,--export-dynamic` to allow functors to access to symbols.

We have to notice that some of these flags seem to be Hacks and are not recommended to be used. Moreover there may still be bugs, especially with functors.

The final executable goes from $\sim 20kB$ to $2.5GB$.
The standard method to run test with python is not working. Need to run directly the executable with a json options file:

```
build.x86_64_v3-centos7-gcc11+detdesc-opt+g/run
    ./build.x86_64_v3-centos7-gcc11+detdesc-opt+
    g/Gaudi/Gaudi/Gaudi_static_options.json
```

Profile-guided optimization

The compiler uses some heuristics to optimize some elements:

- Inlining
- Block ordering
- Register allocation
- Virtual call speculation
- Dead code separation

But sometimes these heuristics are wrong.

A better way for the compiler should having running data. The the principle of PGO is:

- Compiling the programm with instrumentation.
- Running it to create profiles (counters).
- Recompiling the programm with the profiles.

Link-time optimization

Allows the linker to perform optimizations that take account of all translation units.

Final building pipeline

- Compiling the programm with `-fprofile-generate`
- Running it to create profiles
- Recompiling the programm with `-flto -fprofile-use -fprofile-correction`

Results

Test: `hlt2_pp_thor` (6×1000 events)

Optimization	Acceleration	Confidence interval (2σ)
LTO	0.17%	±1.12%
LTO & PGO	6.74%	±1.44%
Static LTO	0.87%	±0.60%
Static LTO & PGO	6.88%	±0.83%

Conclusion

- Using LTO & PGO makes the programm running faster.
- Using static libraries and modules doesn't seem to be usefull and leads to some bugs.