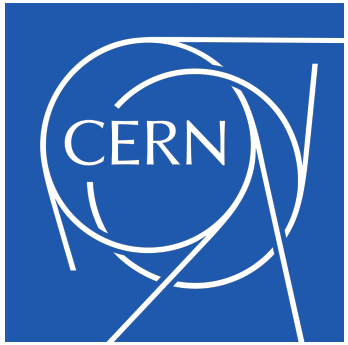


Profile-guided optimization

Speeding up LHCb software through compilation optimization

Oscar Buon

June 19, 2023

Supervisor: Sebastien Ponce

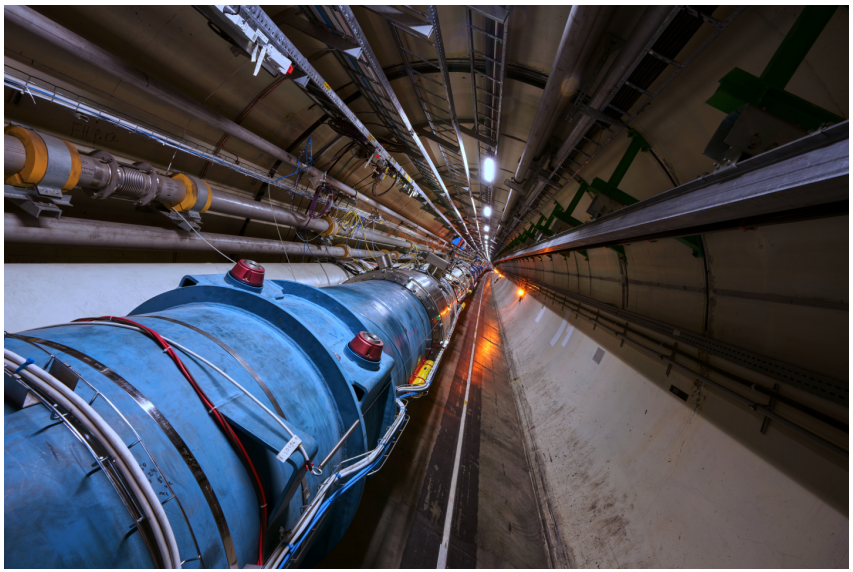
CERN

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

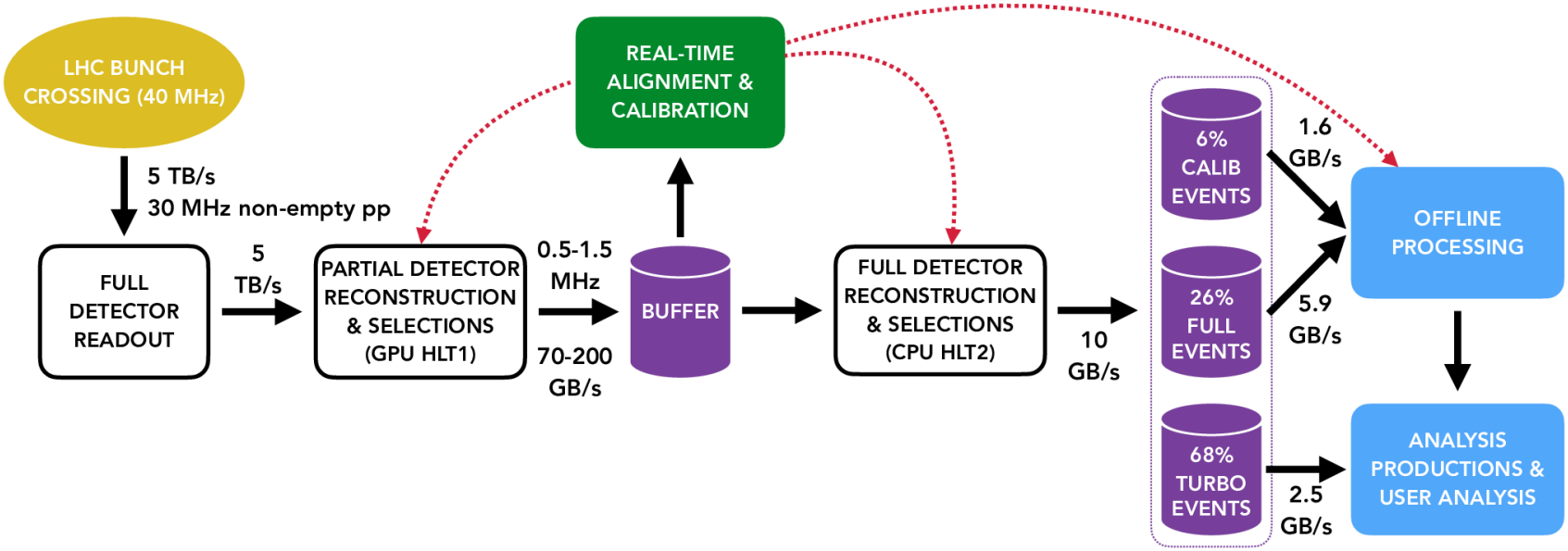


LHCb

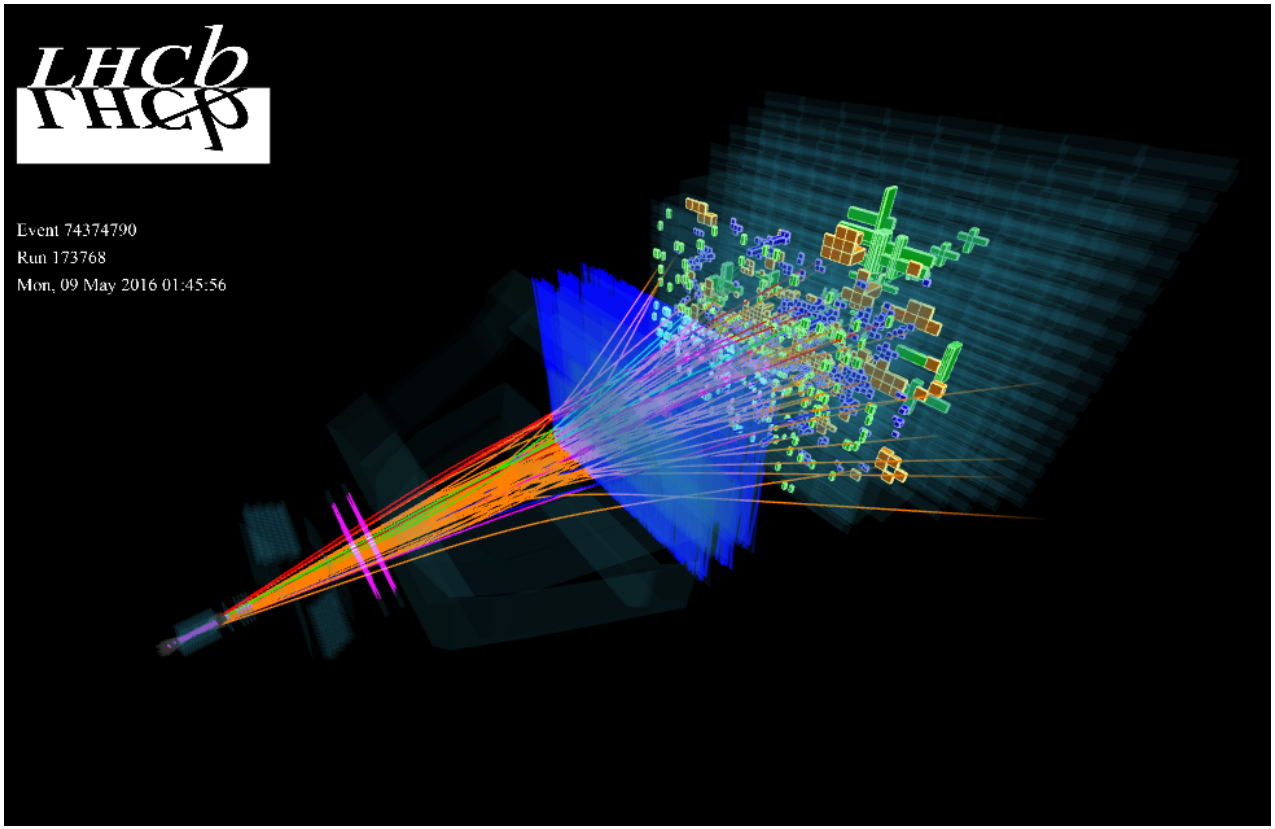
- One of the 4 main experiments installed on the LHC.
- More than 1,200 people working for the collaboration.
- Studies asymmetry between matter and antimatter via b-physics. Collision of hadrons (heavy particles).

Physics Computing

Important computing infrastructure.



Computing group



The computing group has to maintain this infrastructure for LHCb.

Software

The software is a stack of programms:

- Common base shared with other experiments.
- LHCb specific programms.
- Reconstruction, simultation...

Millions lines of code which some are 30 years old. Code 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)
gsecorf
(Gaudi::gsecorf)
Intcomponents
(Gaudi::Intcomponents)
GaudiKernel
(Gaudi::GaudiKernel)
GaudiPluginService
(Gaudi::GaudiPluginService)

Library types

- STATIC: 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. Creating a new function for static libraries by copying the one for shared libraries:

`gaudi_add_library` $\Rightarrow$ `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` because without the linker removes what seems to be unused code.
- `-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.

These flags seem to be Hacks and are not recommended to be used. There may still be bugs, especially with functors.

The final executable goes from $\sim 20kB$ to $2.5GB$ (opt+g).
Standard method to run test with python 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

Makes programms running several times faster. But a few times these heuristics are wrong.

A better way for the compiler should be knowing running behavior. 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

C programm composed of translation units (1 translation unit $\approx$ 1 .c file).
Compiler cannot optimize accross them by default.
LTO 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`

Conclusion

Test: `hlt2_pp_thor` (6×1000 events)

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

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

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`