EPJ Web of Conferences **245**, 05037 (2020) https://doi.org/10.1051/epjconf/202024505037
CHEP 2019

Impact of different compilers and build types on Geant4 simulation execution time

Caterina Marcon[1,*], *Oxana* Smirnova[1], and *Servesh* Muralidharan[2]

[1]Lunds Universitet, Fysiska Institutionen, Box 118, SE - 221 00 Lund, Sweden
[2]CERN, CH-1211 Geneva 23, Switzerland

Abstract. Experimental observations and advanced computer simulations in High Energy Physics (HEP) paved the way for the recent discoveries at the Large Hadron Collider (LHC) at CERN. Currently, Monte Carlo simulations account for a very significant amount of computational resources of the Worldwide LHC Computing Grid (WLCG). The current growth in available computing performance will not be enough to fulfill the expected demand for the forthcoming High Luminosity run (HL-LHC). More efficient simulation codes are therefore required.
This study focuses on evaluating the impact of different build methods on the simulation execution time. The Geant4 toolkit, the standard simulation code for the LHC experiments, consists of a set of libraries which can be either dynamically or statically linked to the simulation executable. Dynamic libraries are currently the preferred build method.
In this work, three versions of the GCC compiler, namely 4.8.5, 6.2.0 and 8.2.0 have been used. In addition, a comparison between four optimization levels (Os, O1, O2 and O3) has also been performed.
Static builds for all the GCC versions considered, exhibit a reduction in execution times of about 10%. Switching to newer GCC version results in an average of 30% improvement in the execution time regardless of the build type. In particular, a static build with GCC 8.2.0 leads to an improvement of about 34% with respect to the default configuration (GCC 4.8.5, dynamic, O2). The different GCC optimization flags do not affect the execution times.

1 Introduction

The Large Hadron Collider (LHC) at CERN is the largest particle collider in the world, reaching the highest energies in particle collisions. Results obtained from LHC measurements are reproduced by means of computer simulations, which allow to investigate the relevant physical interactions and play a key role in interpreting the collected data.

Currently, Monte Carlo simulations occupy up to 40% of the World Wide LHC Computing Grid (WLCG) computational resources. This percentage will rise when the LHC luminosity (a parameter proportional to the collision rate) will be further increased: during the forthcoming Run 3, the luminosity will grow (up to 5×10^{34} $cm^{-2}s^{-1}$) with respect to the previous two Runs, and it will be more than doubled with respect to Run 3 during the high-luminosity phase (HL-LHC) planned for 2026 and beyond [1].

*e-mail: caterina.marcon@hep.lu.se

Future requirements for LHC computing resources already exceed the foreseeable growth in computing power described by Moore's law [2, 3]. Thus, increasing the available computing capacity is no longer sufficient to deal with the growing demand. In addition to the optimization of the build process discussed in this work, computing models will also need to be improved for efficiency in order to fully exploit the existing processor architectures.

The aim of this preliminary study is to investigate different methods to reduce the simulation execution time without sacrificing the quality of the simulated data and without altering the existing source code.

The simulation software for LHC experiments relies predominantly on the Geant4 simulation toolkit [4] and in this work, the attention will be focused on the following aspects:

- evaluate the impact of different build types, static and dynamic, on the Geant4 execution time. Dynamic (or shared) libraries allow for a modular structure of the executables: they are loaded at runtime and can be shared among several instances of the same program; they are not embedded into the compiled code and therefore they can be modified individually without need of complete rebuilds. Smaller executables and easier maintenance represent a strong advantage of dynamic libraries. Static libraries, on the other hand, are compiled into the application and they cannot be shared. This produces larger executable files, each containing a copy of the library; complete rebuild of the source code is required to update or replace static libraries. Access to static libraries is faster, therefore static linking is expected to result in execution times shorter than those obtained with the currently widespread dynamically linked code;
- estimate the effects on the execution time of different GCC compilers (4.8.5, 6.2.0 and 8.2.0) and optimization flags (Os, O1, O2 and O3) [5].

Unlike other performance studies [6], to avoid the dependency on control frameworks and other libraries, a standalone simulation has been considered as benchmark.

2 Methods

Geant4 version 10.5 has been used for this study. To evaluate the effects of different build types on the execution time, both static and dynamic Geant4 libraries have been produced. A standalone Geant4 simulation [7], also compiled both statically and dinamically against the aforementioned Geant4 builds, has been used as benchmark. This simulation is based on GDML geometries[1] loaded at runtime, rather than hard-coded detector definitions.

The execution time of the simulation has been defined as the time interval between the sampling of the first primary particle and the conclusion of the last event.

For this preliminary study, in order to test different geometrical complexities, two detector configurations have been considered: a complete geometry (referenced to as *full* in the text), comprising the main components from beamline to muon detectors, and a subset of complete geometry (*inner* in the text) extending only to the inner detectors.

In both cases, 50 GeV pions have been used as source particles. In order to maintain the execution time constant, the number of simulated primaries has been varied according to the detector complexity: 5000 initial pions for full detector and 50000 pions for the inner case.

Three versions of the GCC compiler, namely 4.8.5, 6.2.0 and 8.2.0, have been used for these investigations and a comparison between four GCC optimization flags (Os, O1, O2 and O3) has also been performed. The default optimization level used by most build systems is O2, hence in this study it has been considered as a reference.

[1]The Geometry Description Markup Language (GDML) is an application-indepedent, XML-based geometry description format, which can be used as the primary geometry implementation language as well as an exchange format for existing applications [8].

For each of the studied configurations the benchmark simulation has been run 5 times. Average values are presented and for all the studied configurations, standard deviations are of the order of 1%.

The computations have been carried out on a standalone machine at CERN and on a university cluster in Lund (see Tab. 1). On both machines, CPU and memory resources have been exclusively allocated for these simulations to prevent uncontrolled overhead from concurrent processes (other than the minimum operating system's tasks). CPU multithreading on the university cluster nodes is disabled.

Table 1. Computing resources

	CERN standalone machine	**Compute node on Lund University cluster**
CPU	2× Intel Xeon E5-2630 v3 2.40GHz	2× Intel Xeon E5-2650 v3 2.30GHz
Architecture	64 bit Haswell x86_64	64 bit Haswell x86_64
N. of cores	16	20
Threads per core	2	1
Cache	20 MB (L1: 64 KB, L2: 256 KB, L3: 20 MB)	25 MB (L1: 64 KB, L2: 256 KB, L3: 25 MB)
RAM	64 GB	128 GB
Filesystem	XFS	IBM General Parallel File System (GPFS)
Operating system	CentOS 7	CentOS 7

3 Results and discussion

Figs. 1 and 2 show the comparison between static and dynamic performance considering the full detector geometry. The results presented in Fig. 1 are relative to the CERN standalone machine on which the static builds can reduce the execution time by more than 10%. In this case, the compiler version has a remarkable effect: the execution time can be reduced by up to 30% by switching from GCC 4.8.5 to newer GCC versions. The static build compiled with GCC 8.2.0 leads to an improvement of about 34% with respect to the default configuration. In Fig. 2 the computations performed on the cluster are shown. Also in this case the static approach allows a performance gain: the execution time is reduced by more than 10%; however, the impact of the three different compilers is not as relevant as in the previous case. For both configurations with full detector geometry the GCC optimizations do not result in a visible improvement.

In Figs. 3 and 4 the comparison between static and dynamic performance using the inner detector geometry is given. Fig. 3 shows the results for the calculations performed on the standalone CERN machine. The static approach, for all GCC versions, reduces the execution time by more than 9%. In Fig. 4, the results of the computations carried out on the cluster are presented: with the static approach the execution time can be scaled down by 10%. Here, the impact of the compiler versions is less significant than in the full geometry case. Also with this geometry the compiler optimizations do not have visible effects on the execution time.

Figs. 5 and 6 show the trend of the execution time as a function of the number of threads used for the computations. The number of primaries per thread has been kept constant (1250 primaries / thread) for all the cases. Therefore, for a doubled number of threads, the number of primaries is also doubled; the effect of parallelization ensures that the resulting execution time is not doubled, despite a minor increase of the order of 10% between 1 and 16/20 threads, which can be ascribed to CPU and I/O overhead. In Fig. 5, the full detector geometry has been considered whereas in Fig. 6 the inner configuration has been used. The improvement between static and dynamic linking is confirmed in all the analyzed cases. CPU multithreading, i.e. the virtualization of CPU threads, is responsible for the higher increase in execution

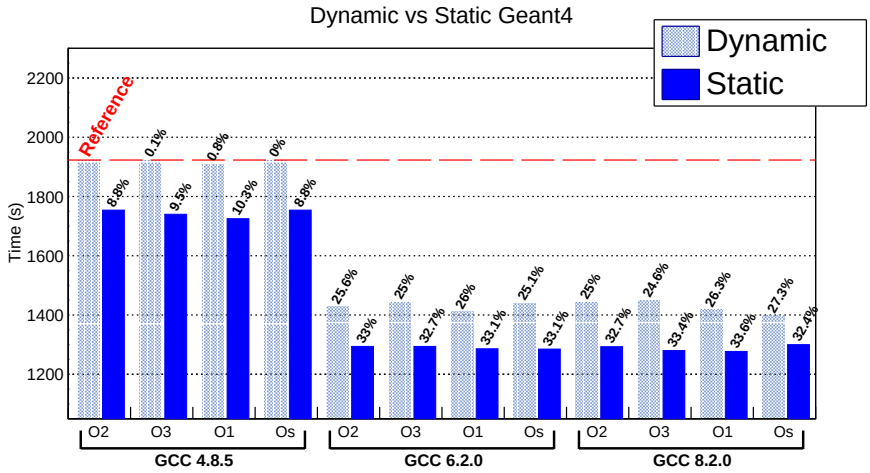

Figure 1. Comparison between dynamic and static performance with full geometry (percentages indicate the speedup with respect to the reference case). The computations have been carried out on CERN standalone machine. For the computations 5000 primary particles have been considered and 4 threads have been used. Note that the time axis starts at 1000 s.

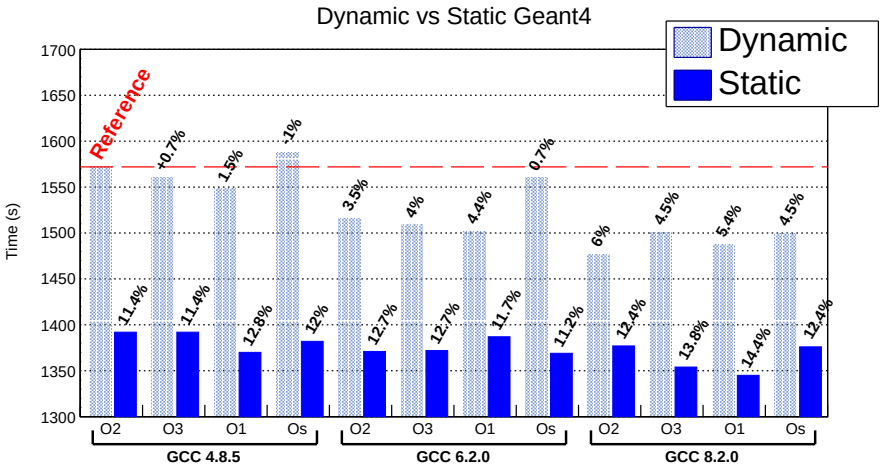

Figure 2. Comparison between dynamic and static performance with full geometry (percentages indicate the speedup with respect to the reference case). The computations have been carried out on Lund cluster. For the computations 5000 primary particles have been considered and 4 threads have been used. Note that the time axis starts at 1300 s.

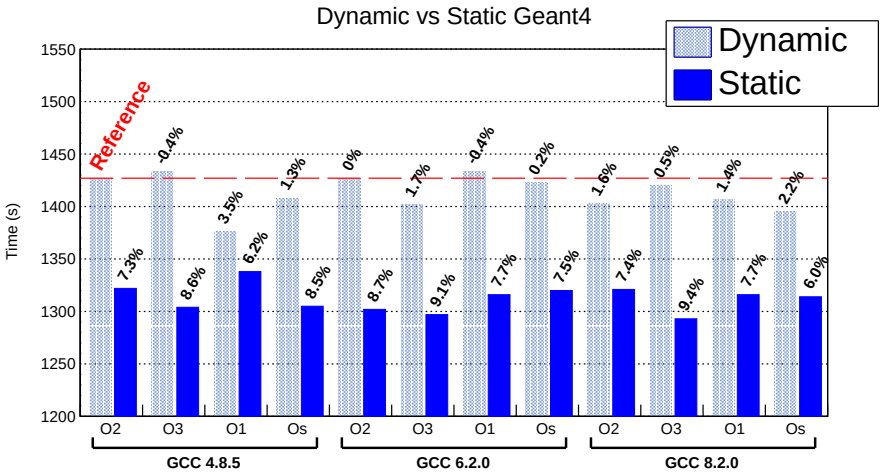

Figure 3. Comparison between dynamic and static performance with inner geometry (percentages indicate the speedup with respect to the reference case). The computations have been carried out on CERN standalone machine. For the computations 50000 primary particles have been considered and 4 threads have been used. Note that the time axis starts at 1200 s.

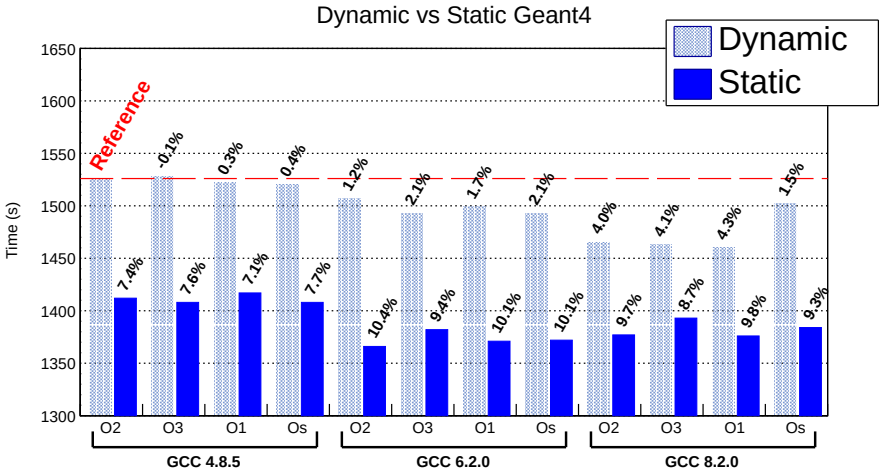

Figure 4. Comparison between dynamic and static performance with inner geometry (percentages indicate the speedup with respect to the reference case). The computations have been carried out on Lund cluster. For the computations 50000 primary particles have been considered and 4 threads have been used. Note that the time axis starts at 1300 s.

time observed for 20 and 32 threads on the standalone machine, which has only 16 physical cores. The same effect is not observed on the Lund cluster because no multithreading is available and only 20 physical cores can be used for the computations.

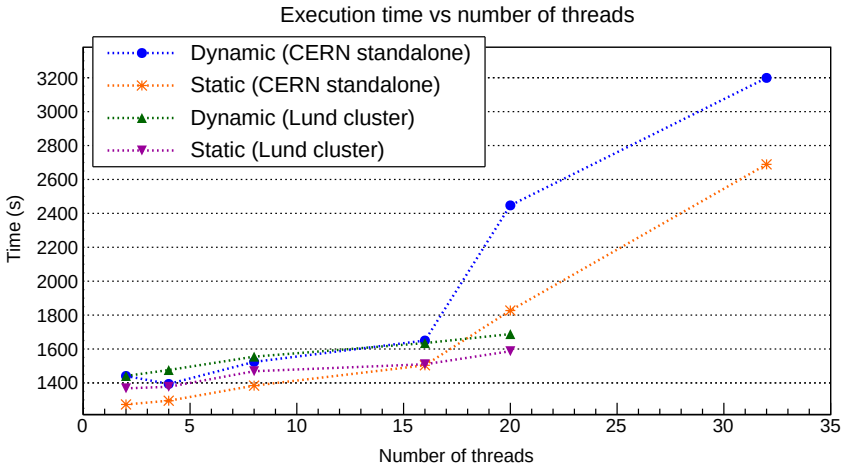

Figure 5. Execution time as a function of number of threads with full detector geometry. The number of primaries per thread has been set equal to 1250. GCC 8.2.0 compiler has been used for the computations. Note that the time axis starts at 1200 s.

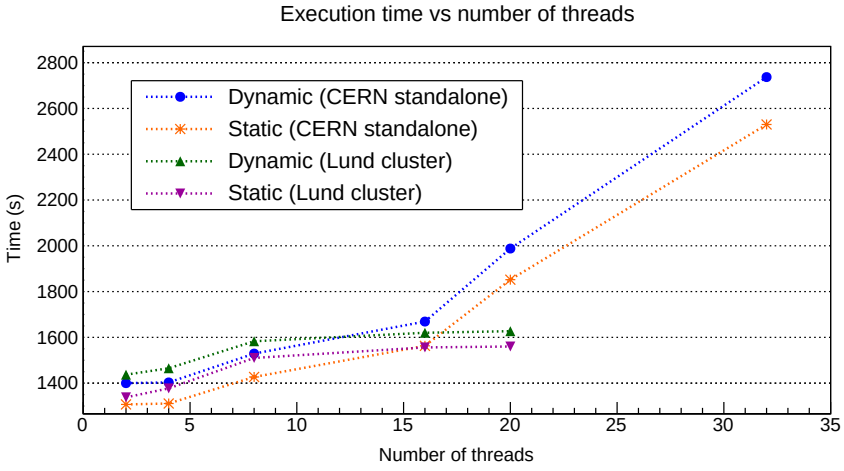

Figure 6. Execution time as a function of number of threads with inner detector geometry. The number of primaries per thread has been set equal to 12500. GCC 8.2.0 compiler has been used for the computations. Note that the time axis starts at 1200 s.

4 Conclusions and outlook

Execution time for simulations based on Geant4 can be significantly improved by changing the default build method. This study shows that linking Geant4 statically instead of dynamically allows to produce binaries that run about 10% faster. However, since static libraries are embedded into the executables, their size is much larger than the corresponding dynamically linked code (700 MB instead of 2.5 MB).

The compiler versions can also have a remarkable impact: execution times can be reduced by 25% by switching from GCC 4.8.5 to GCC 8.2.0. This effect is more pronounced when more complex detector geometries are considered. For all the studied configurations, the different GCC optimizations do not affect the execution time.

To extend the scope of the present work, the following future steps will be undertaken:

- test other detector configurations in order to test a wider range of geometrical complexity;
- investigate the impact of different physics models and different primary particles;
- evaluate the performance of other compilers such as clang, Intel's icc and Portland's pgi;
- consider more advanced compile-time optimizations, such as link-time optimization (LTO).

References

[1] *LHC Commissioning with beam*, https://lhc-commissioning.web.cern.ch/lhc-commissioning/

[2] G.E. Moore, Electronics **38** (1965)

[3] *Projected resource requirements for ATLAS*, https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults/

[4] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

[5] *GCC Optimization options*, https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[6] C. Marcon, *Investigating possible CPU utilisation improvement in simulation job*, https://indico.cern.ch/event/738396/contributions/3047012/attachments/1677036/2692778/Presentation_SCWeek.pdf

[7] A. Dotti, *Geant4 simulation benchmark*, https://gitlab.cern.ch/adotti/HepExpMT

[8] R. Chytracek, J. Mccormick, W. Pokorski, G. Santin, IEEE Transactions on Nuclear Science **53**, 2892 (2006)