Limits calculation with HistFitter in TLA

Impact of build types on Geant4 execution time

Lund - Doktoranddag

Caterina Marcon

12 December, 2019

Trigger object Level Analysis

- **Purpose:** TLA searches for **low-mass dijet** resonances (450-1800 GeV) using ATLAS detector at LHC.

- **Issue**: LHC searches for lighter resonances with small cross-sections have been hampered by **restrictions in data-taking rate** -> dijet events with an invariant mass below 1 TeV are **largely discarded** by the trigger system.

- **Solution:** implementation of a novel data technique -> data analysis uses only a fraction of the full event that is saved in a dedicated data stream and reconstructed within the software trigger system. These are the "**trigger objects**".

Validate HistFitter framework for setting limits

- **Task**: switch to using **HistFitter** [1] (frequentist stats framework) for the limit calculation (instead of the **Bayesian** approach used until now).
- Bayesian (**BAT**) and Frequentist (HistFitter) approaches to limit setting are a bit different:
  - limits obtained in the Frequentist approach are tighter than the Bayesian one;
  - HistFitter requires less resources to run.

[1] https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/HistFitterTutorial

Methods

- In absence of any excess, the observed data and predicted background are used to **set model-independent limits** on new phenomena.

- The analyzed dataset has an integrated luminosity of up to 29.3 fb^{-1} and a centre of mass of 13 TeV.

- For this **validation**, **limits on the cross section** are set on a generic model where the signal is modeled as a **gaussian** contribution to the observed mjj distribution:
  - width = 7%;
  - peak 800 GeV;
  - mass range: from 531.0 to 2081.0 GeV.

Methods

- To validate BAT and HistFitter, a **95% credibility-level upper limit** on the cross section has been considered.
- The expected limits are calculated including systematic uncertainties on both signal and background model:
  - **Background systematic uncertainties**:
    - Uncertainty for **choice of fit function**;
    - Uncertainty for **fit parameter values**.
  - **Signal systematic uncertainties**:
    - Uncertainties on the **jet energy scale (JES)**;
    - Uncertainties on the **luminosity**.

Results

- **Background systematic uncertainties**:

Configuration	Limits	
	Bayesian	HistFitter
No Systematic	6118.52	6117.71
Fit Function choice only	6118.54	6117.51
Fit parameter values only	6257.82	6257.71

- The results from BAT and HistFitter are compatible within 1%.

Results

- **Signal systematic uncertainties**:

Configuration	Limits	
	Bayesian	HistFitter
Jet Energy Scale only	6333.07	6127.04
Luminosity only	6145.35	6117.71

The difference is about 3%

- The two methods are compatible

Limits calculation with HistFitter in TLA

Impact of build types on Geant4 execution time

Lund - Doktoranddag

Caterina Marcon

12 December, 2019

Motivation

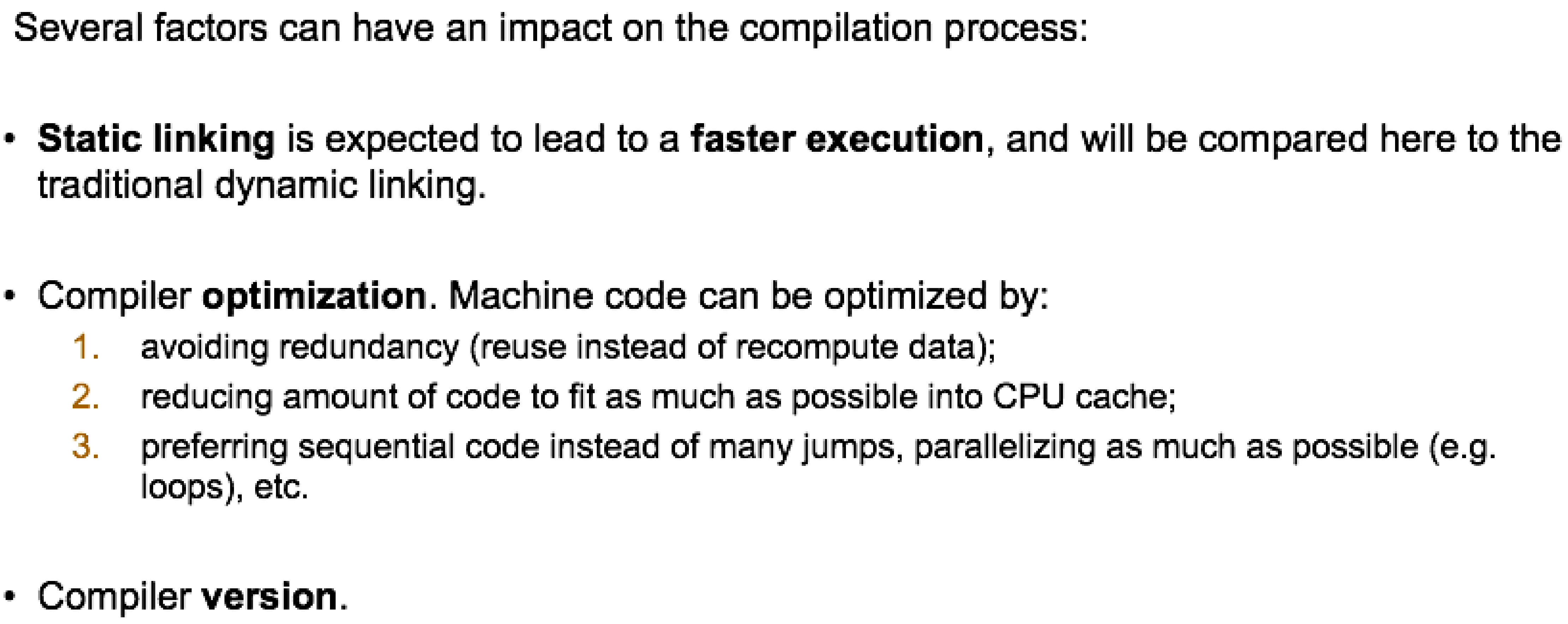
- Currently**, Monte Carlo detector simulation at LHC** can occupy up to **40 %** of World Wide LHC computing grid's resources. This percentage is set to grow when LHC luminosity will be further increased.

- It is necessary to find a new approach for improving the execution time of simulations without sacrificing the quality of simulated data.

- The purpose of this preliminary study is to investigate how to **reduce the Geant4 simulation execution time**.

- This is achieved by running **standalone Geant4 simulations**, whose performance can then be evaluated independently from other libraries and control frameworks.

Method



Several factors can have an impact on the compilation process:

- **Static linking** is expected to lead to a **faster execution**, and will be compared here to the traditional dynamic linking.

- Compiler **optimization**. Machine code can be optimized by:
  1. avoiding redundancy (reuse instead of recompute data);
  2. reducing amount of code to fit as much as possible into CPU cache;
  3. preferring sequential code instead of many jumps, parallelizing as much as possible (e.g. loops), etc.

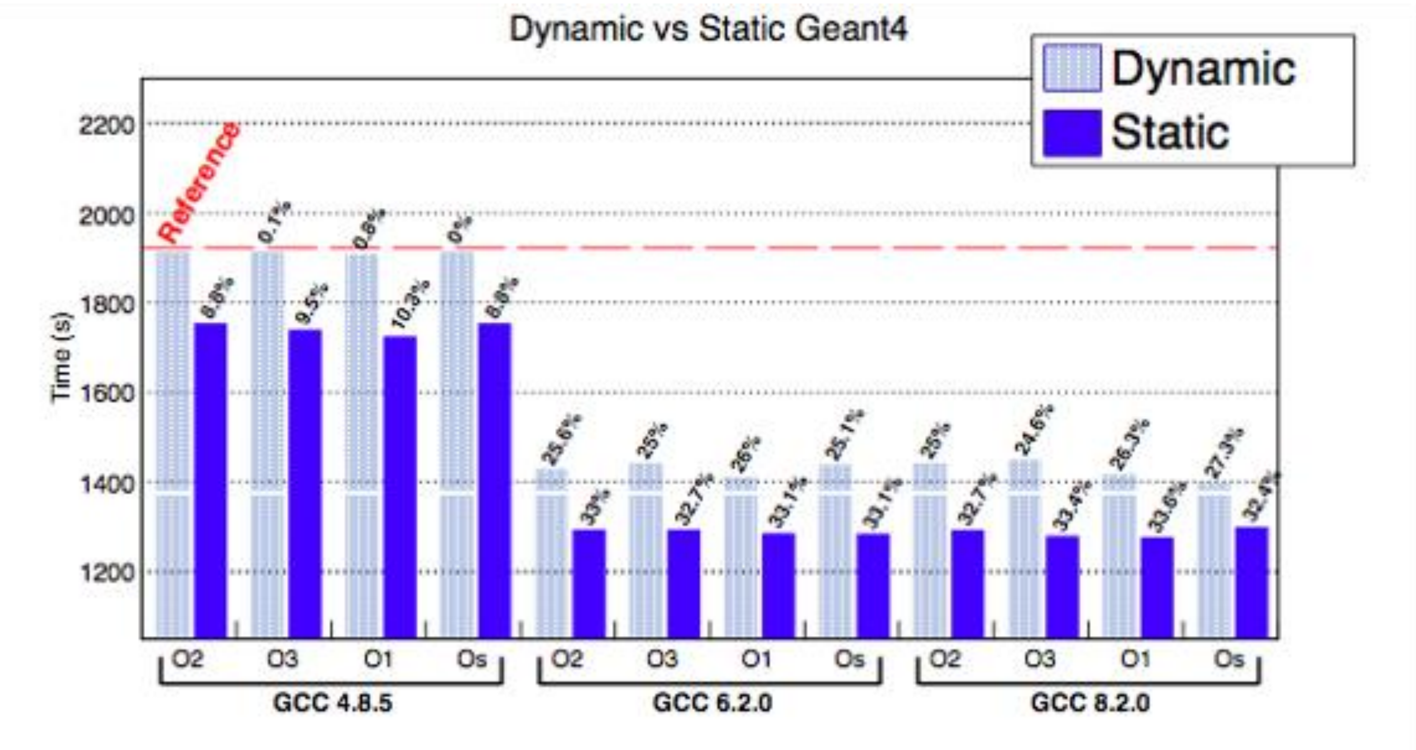
- Compiler **version**.

Method

- As a benchmark, **standalone G4 simulation** with two different geometries (from A. Dotti [1]) has been used. **50 GeV pions** are used as source particles. The number of simulated primaries varies according to the detector geometry.
- Compiled G4 (version 10.5) both **statically** and **dynamically**.
- Three versions of the GCC compiler, namely **4.8.5, 6.2.0 and 8.2.0**, have been used for these investigations.
- A comparison between four GCC optimization levels (**Os, O1, O2 and O3**) have also been performed. The default level used by most build systems is -O2 and it will be used as reference.
- The computations were carried out on **a standalone machine at CERN IT** and on **a university cluster** in Lund.
- CPU and memory resources on both machines (standalone and cluster) were **exclusively allocated** to the simulations and not shared with any concurrent process other than the minimum OS tasks.

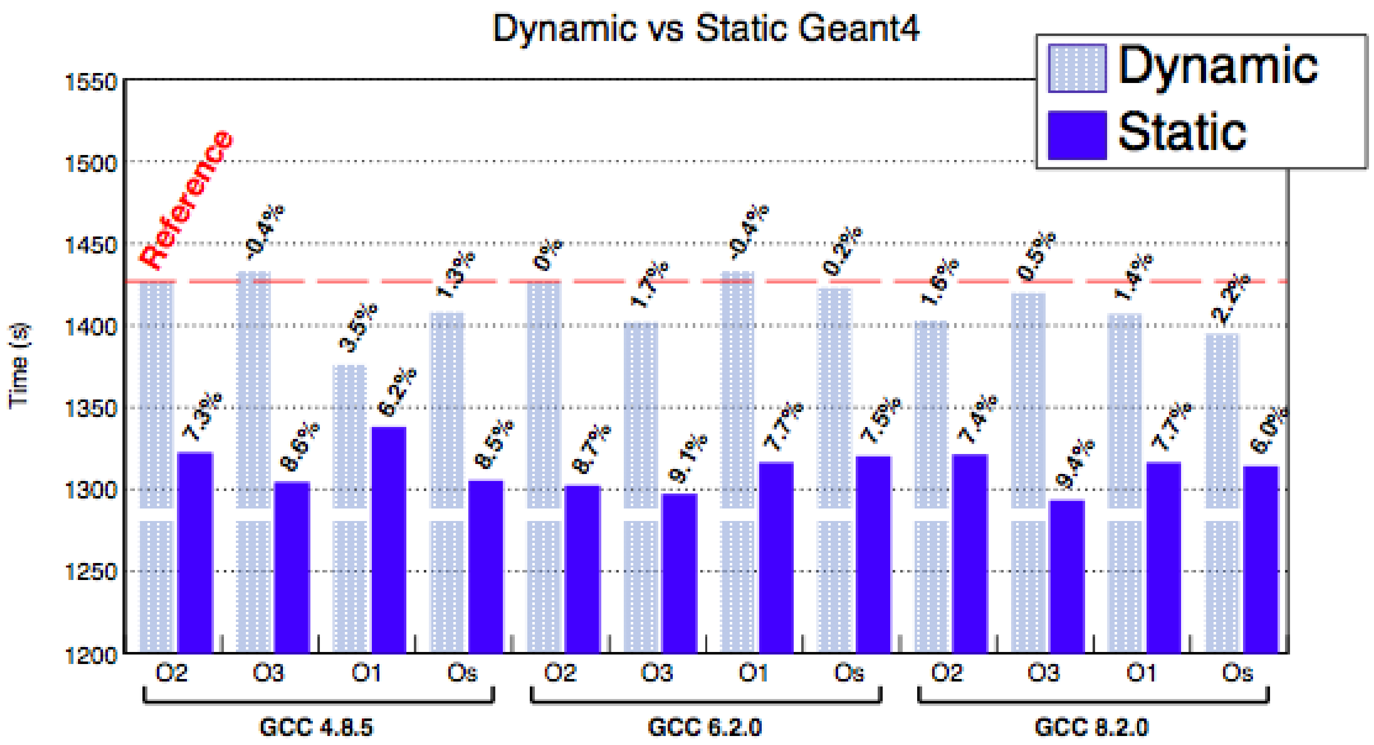
[1] https://gitlab.cern.ch/adotti/Geant4HepExpMTBenchmark

Static vs dynamic performance with **full detector geometry**



- The computations were carried out **on CERN machine** considering 5000 initial events and using 4 threads. The computation was repeated 3 times for each configuration.
- The static approach, for all the GCC versions, reduces the execution time by more than **10%** in some cases.
- Regardless of the build approach, switching from GCC 4.8.5 to GCC 6.2.0 and GCC 8.2.0 results in an average of **30%** improvement in the execution time.
- A static build with GCC 8.2.0 leads to an improvement of almost **34%** with respect to the default configuration (GCC 4.8.5, dynamic, O2).
- The different GCC optimizations do not seem to have visible effects on the execution time.

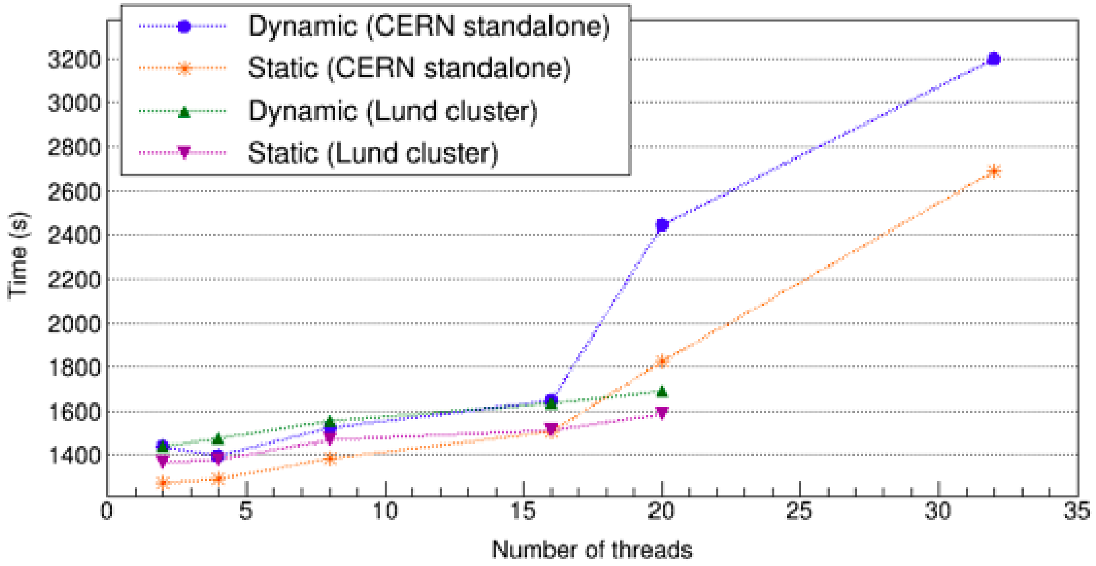
Static vs dynamic performance with **Inner Detector geometry**



- The computations were carried out **on CERN machine** considering 50000 initial events and using 4 threads. The computation was repeated 3 times for each configuration.
- The static approach, for all the GCC versions, reduces the execution time by more than **9%** in some cases.
- The impact of different compilers is not relevant as in the full geometry case.
- The different GCC optimizations do not seem to have visible effects on the execution time.

Execution time vs number of threads

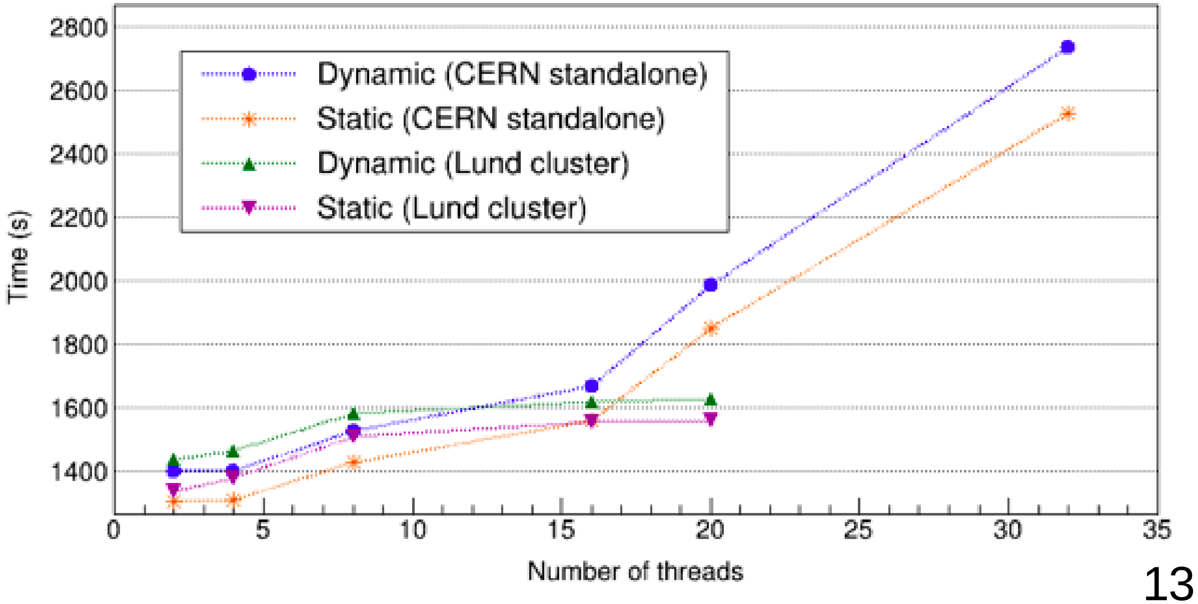
Full detector geometry



- The improvement between static and dynamic linking is confirmed in all cases on both machines.

- Code was built using GCC 8.2.0.
- Simulations were run keeping the number of events per thread constant.

Inner detector geometry



Conclusions

- Execution time for simulations based on Geant4 can be significantly improved by changing the default build method: linking Geant4 and its associated libraries statically can produce binaries that run even **10% faster**.
- Switching from gcc 4.8.5 to 8.2.0 results in a reduction of the execution time up to **25%**.
- Static libraries are embedded into the executable, resulting in a much larger size (~700 MB) than the corresponding dynamically-linked code (~ 2.5 MB).
- The different GCC optimizations do not seem to have visible effects on the execution time.

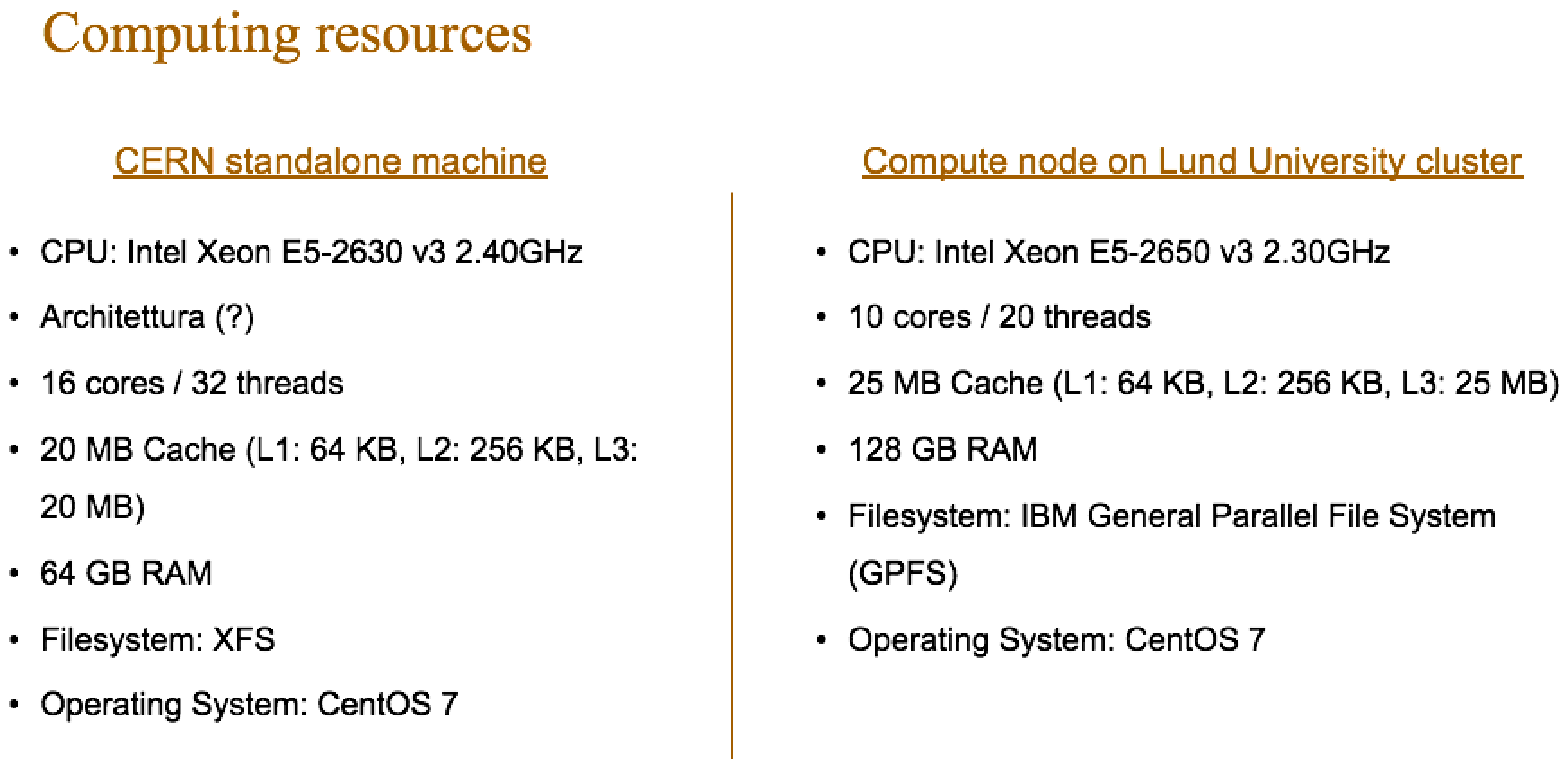
Courses & conference

- Computational Programming with Python
- Scientific Computing with Python and Fortran

- CHEP 2019 - Adelaide

Thank you for your attention!

Backup



Computing resources

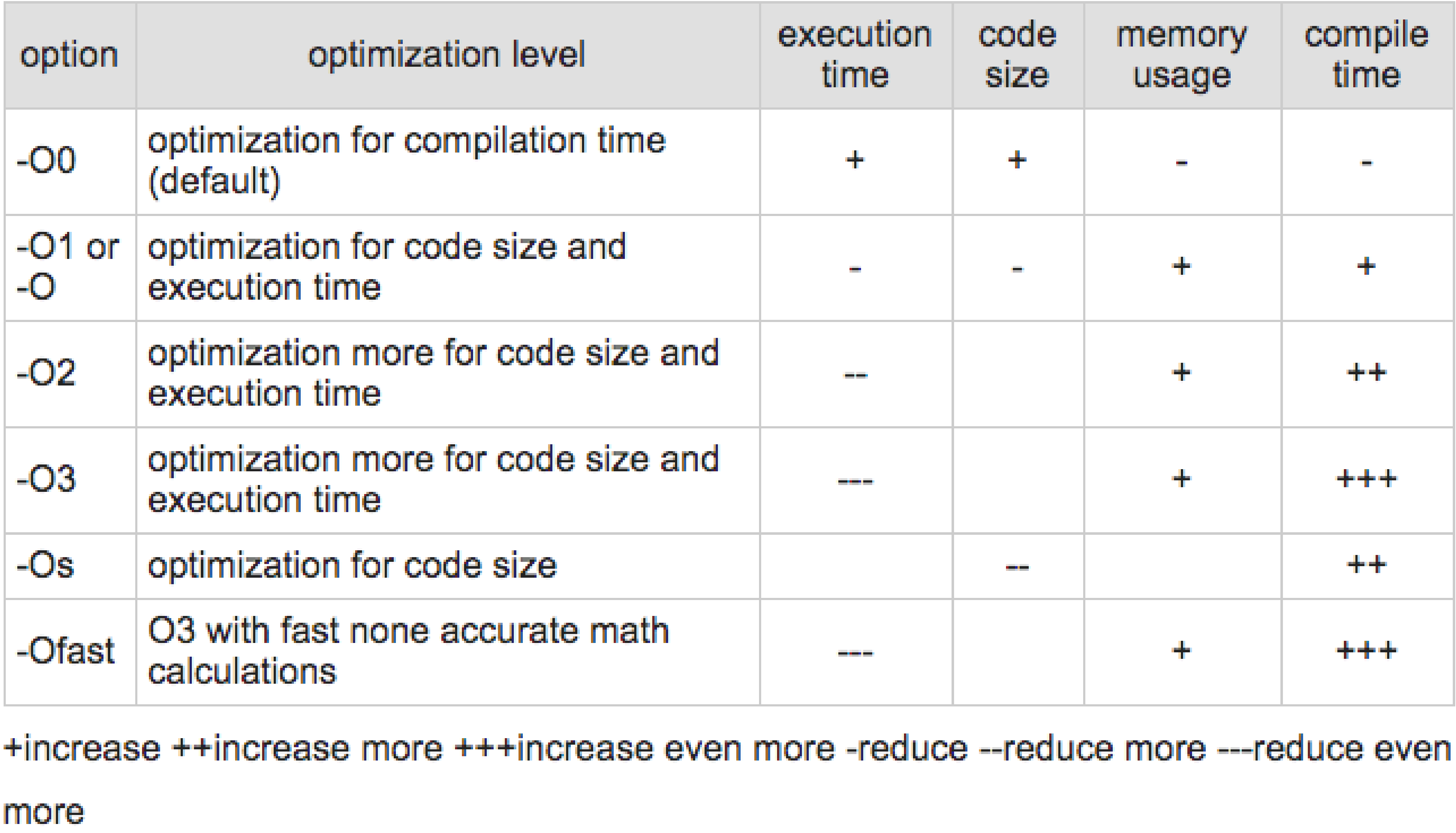
CERN standalone machine

- CPU: Intel Xeon E5-2630 v3 2.40GHz
- Architettura (?)
- 16 cores / 32 threads
- 20 MB Cache (L1: 64 KB, L2: 256 KB, L3: 20 MB)
- 64 GB RAM
- Filesystem: XFS
- Operating System: CentOS 7

Compute node on Lund University cluster

- CPU: Intel Xeon E5-2650 v3 2.30GHz
- 10 cores / 20 threads
- 25 MB Cache (L1: 64 KB, L2: 256 KB, L3: 25 MB)
- 128 GB RAM
- Filesystem: IBM General Parallel File System (GPFS)
- Operating System: CentOS 7

Backup



option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

+increase ++increase more +++increase even more -reduce --reduce more ---reduce even more