Profiling NNLO+PS simulations: GENEVA as a case study

Simone Alioli
MILANO-BICOCCA UNIVERSITY & INFN

EVENT GENERATORS' AND N(N)LO CODES' ACCELERATION WORKSHOP

ICSC Italian Research Center on High-Performance Computing, Big Data and Quantum Computing — Missione 4 • Istruzione e Ricerca

CERN 14/11/2023

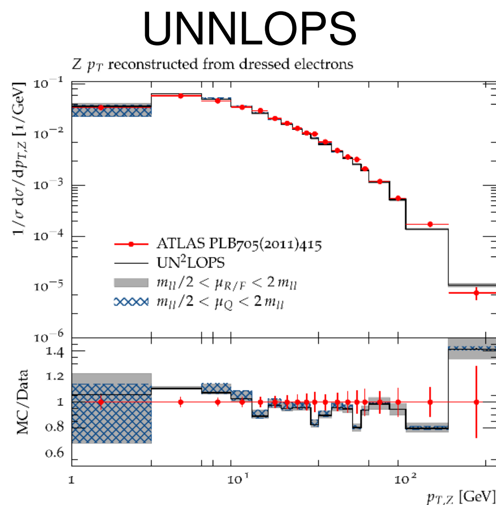
Outline

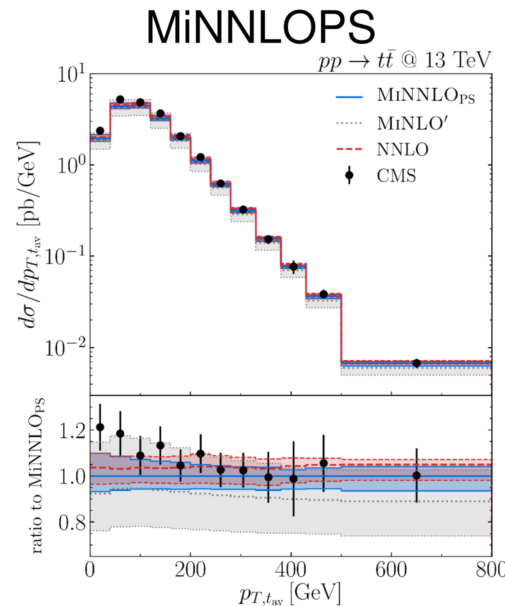
- Introduction and statement of exercise goals
- NNLO+PS: GENEVA and similarities/differences with other codes
- Discussions of NNLO+PS simulation stages and timing
- Profiling GENEVA ggH, DY and ZZ production
- Discussion of bottlenecks and ideas for improvements
- Outlook and conclusions

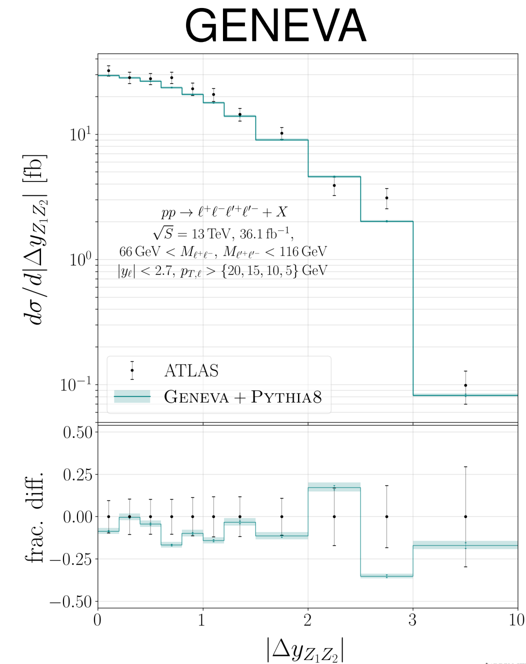
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Introduction

- The increasing experimental precision of LHC measurements challenges existing generators, pushing the request for higher accuracy
- The state-of-the-art is the inclusion of NNLO corrections into parton-shower Monte Carlo
- Three main approach to the problem:







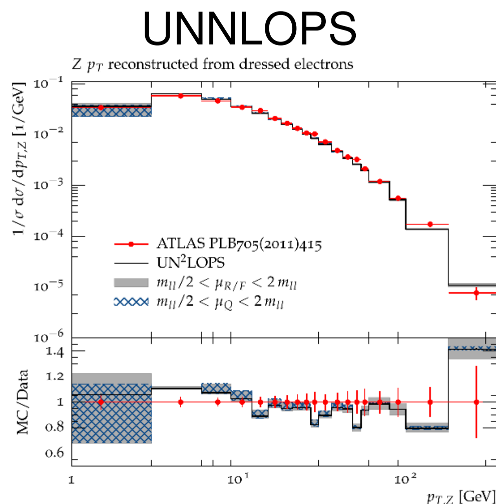
Also NNLO+PS with sector showers available for e^+e^- and $H \to b\bar{b}$

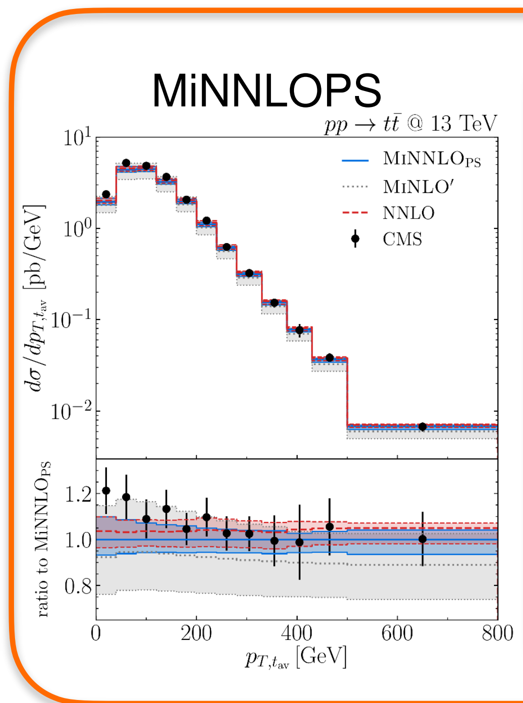
[Campbell et al. 2108.07133]

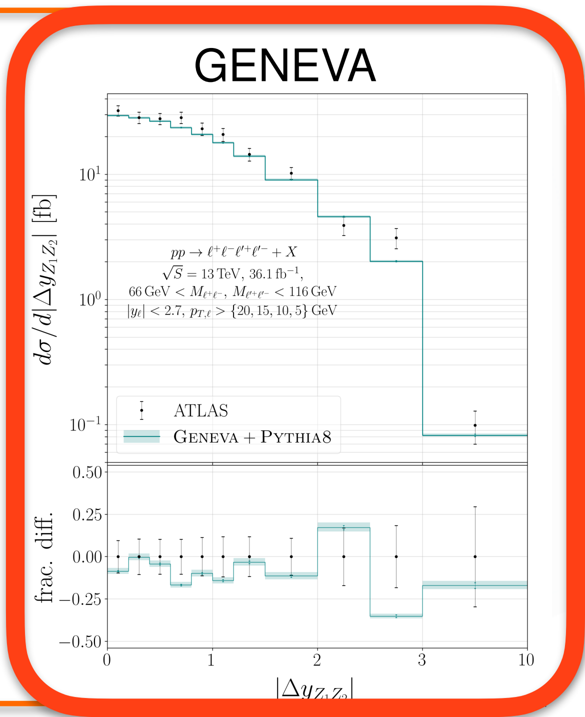
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Introduction

- The increasing experimental precision of LHC measurements challenges existing generators, pushing the request for higher accuracy
- The state-of-the-art is the inclusion of NNLO corrections into parton-shower Monte Carlo
- Three main approach to the problem:







Also NNLO+PS with sector showers available for e^+e^- and $H \to b\bar{b}$

[Campbell et al. 2108.07133]

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Requests from the organizers

- Perform an exercise in "Profiling of the different stages of a NNLO+PS simulation (amplitude evaluation, phase space integration, PDF evaluation, showering, hadronization and MPI, etc)"
- Consider three benchmark processes : neutral Drell-Yan (Z), Higgs via gluon fusion (ggH) and diboson production (ZZ) at the LHC
- Give overall view on how "alternative technologies" (Machine-learning or GPUs) could improve the current approach in view of HL-LHC needs
- Personal opinion on where the interaction with computer scientists would be more beneficial and/or needed.

UNIVERSITA' DEGLI STUDI DI MILANO BICOCCA

Disclaimer about differences among codes

- Using GENEVA as an example because is the code I am more familiar with

- MiNNLOPS simulations share many similarities: it is organized in roughly the same “stages”, it requires identical fixed-order calculations and similar ingredients for resummation (B2 instead of H2, on-the-fly integration over “splitting functions”, etc.)

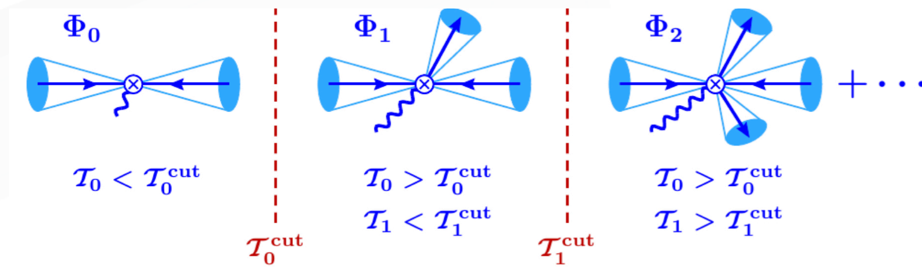
- I don’t know enough the inner working of other codes to comment on them, suggestions or criticisms to what I say from respective authors are welcome.

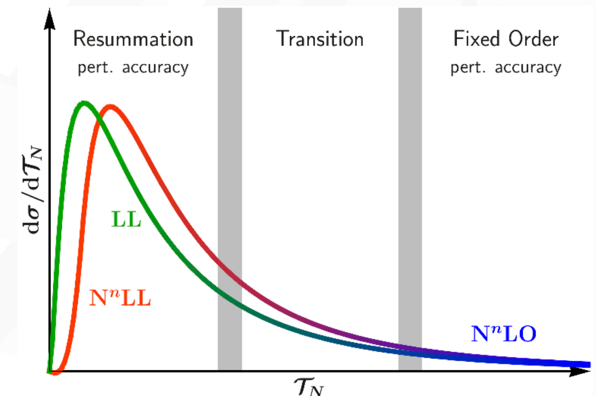
In particular codes more based on the shower like VINCIA/SectorShower might have very different organizational and numerical challenges

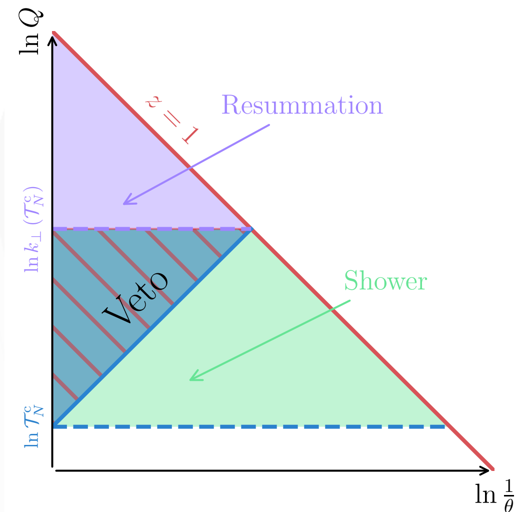
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

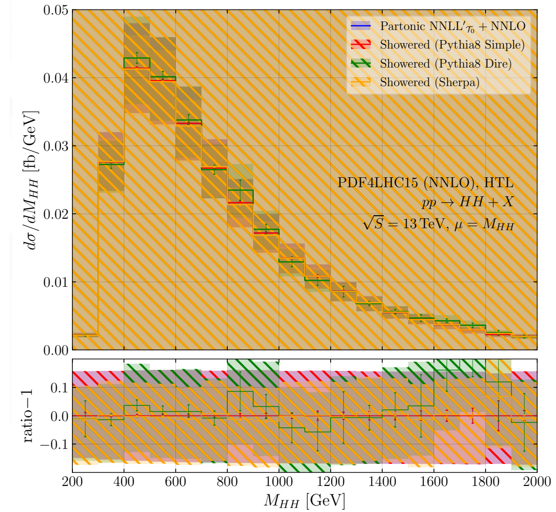
The Geneva method

- Monte Carlo fully-differential event generation at higher-orders (NNLO)
- Resummation plays a key role in the defining the events in a physically sensible way (NNLL')
- Results at partonic level can be further evolved by different shower matching and hadronization models









Stages of a GENEVA NNLO + PS simulation

- Setup stage:

  Performs one or more warmup runs and optimization of integrator (VEGAS grids, MUNICH multi-channel weights, hit-or-miss upper bounds, etc.).

- Main calculation - event generation stage:

  Performs the main calculation and writes out LHEF event files with multiple weights to account for theory uncertainties (PDFs, scales, parameters variations).
  Caveat: not all PS points used for the calculation are written on file (unweighting)

- Reweighting stage

  Adjust the event weights to account for the unwritten events. Given external input can be used to include higher-order effects, power-suppressed corrections …

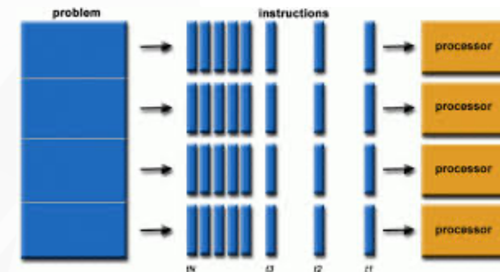
- Showering stage

  Performs the showering, including hadronization, MPI, QED etc.
  GENEVA restrictions require re-showering of same event multiple times.

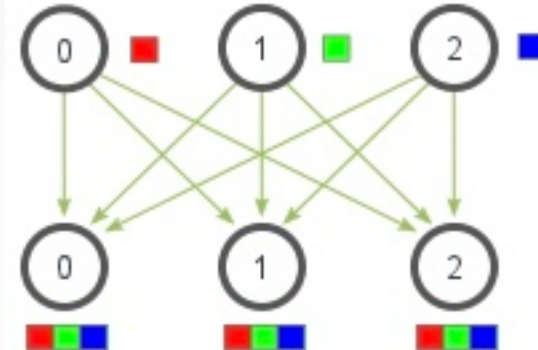
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Current parallelization strategies

- Parallelization is achieved via MPI:



Each run is executed locally on a subset of the requested phase space points generated starting from a given random seed, intermediate results are shared across all cores via MPI all-to-all communications. To achieve <u>scalability, each run must do the same</u>!



Synchronization usually happens at the end of each main stage. During the setup, it is beneficial to gather the largest possible statistics before each optimization step during each iteration, so synchronization is triggered also at the end of each sub-stage iteration, before performing the optimization for next iteration .

- I/O operations are only executed locally using MPI read/write

Each run is in charge of reading a single input and writing a single output files, after reading results are shared via all-to-all MPI

- Scalability tested in real-world scenarios up to O(10k) cores

Requires HPC cluster with fast connections (IB) and distributed FS.

Parameters and runtimes for production runs

- Simulations performed on local small-size cluster (13 nodes, 832 AMD EPYC cores w HT, 56 Gb/s IB interconnect, distributed GlusterFS)
- Events at NNLO+NNLL'+PS accuracy, up to 25 scale, no PDF variations
- Using OpenLoops, Mint and MUNICH,LHAPDF, PYTHIA8, vvamp for ZZ

$gg \to H$ (and $pp \to \ell^+\ell^-$)

setup: 512 runs, 20K points each = 10 M total, 4 iterations
Runtime ~ 5 minutes per run per iteration
Combination ~ 2 minutes per iteration
Total runtime ~ 30 minutes

generate: 512 runs, ~670K points each = 345 M total points
Runtime 4h per run
CPU effective hours 2048
Target 1 per mille stat error on tot xsec, scale but no PDF unc.
Total number of events on files ~40M

reweight: 512 runs, ~80K events each
Runtime 3 minutes per run

shower no QED and no MPI: 512 runs, ~80K events each
Runtime 2h per run (~30m for DY, ggH higher maxRetry)

shower with QED and MPI: 512 runs, ~80K events each
Runtime 2h per run

$pp \to ZZ \to \ell^+\ell^-\ell^+\ell^-$

setup: 512 runs, 30K points each = 15 M total, 3 iterations
Runtime from 1h30m to 2h30m per run per iteration
Combination ~ 3 minutes per iteration
Total runtime ~ 7h20m

generate: 512 runs, ~100K points each = 51.2 M total points
Runtime from 8h to 12h per run
CPU effective hours 5541
Target 2 per mille stat error on tot xsec, scale but no PDF unc.
Total number of events on files ~13M

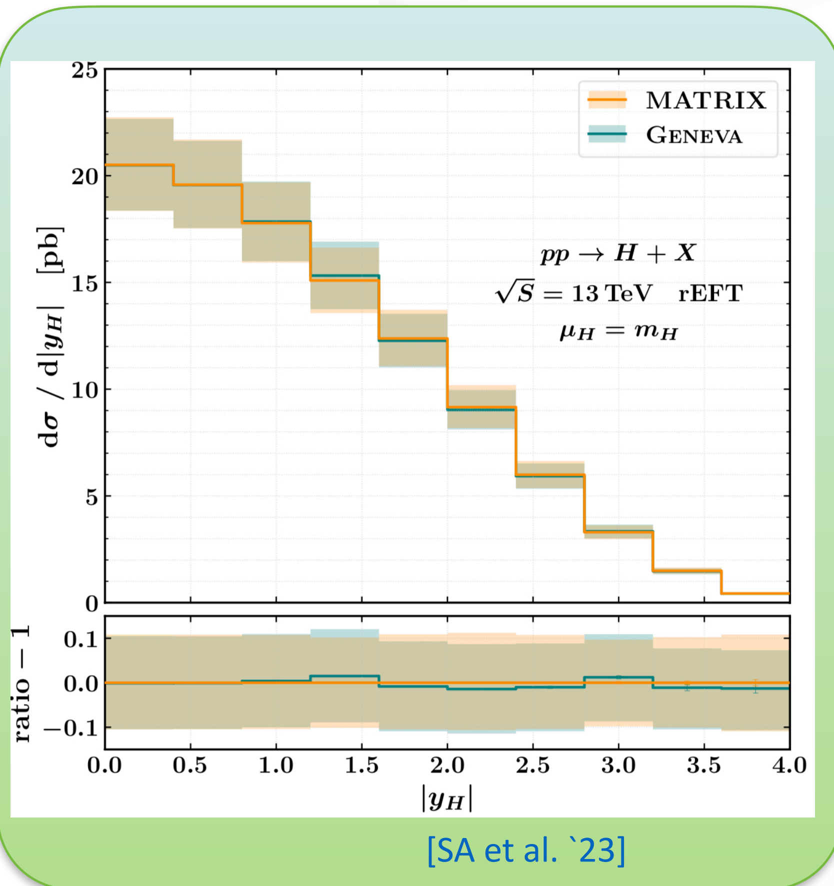
reweight: 512 runs, ~26K events each
Runtime 3 minutes per run

shower no QED and no MPI: 512 runs, ~26K events each
Runtime 20m per run

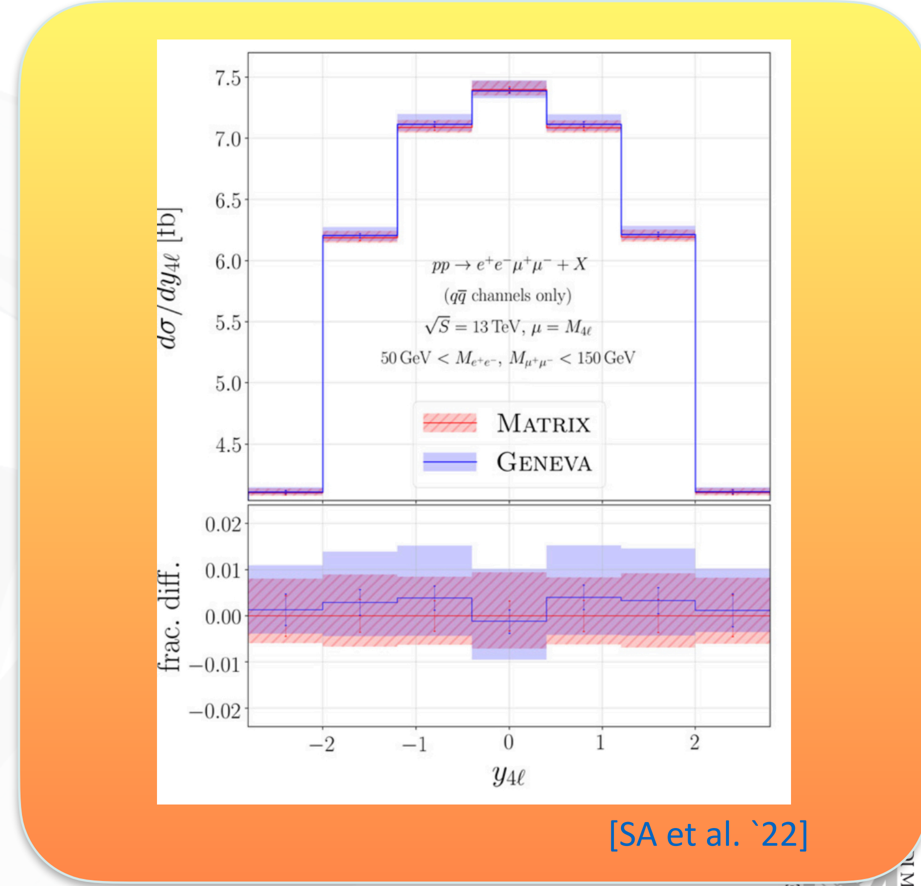
47 events/sec to 3 ev/s (x1.8 if no HT)

Parameters and runtimes for production runs

- Simulations performed on local small-size cluster (13 nodes, 832 AMD EPYC CPUs w HT, 56 Gb/s IB interconnect, distributed GlusterFS)
- Using OpenLoops, Mint and MUNICH,PYTHIA8 and qqVVamp for ZZ
- Events at NNLO+NNLL'+PS accuracy, up to 25 scale, no PDF variations



[SA et al. `23]



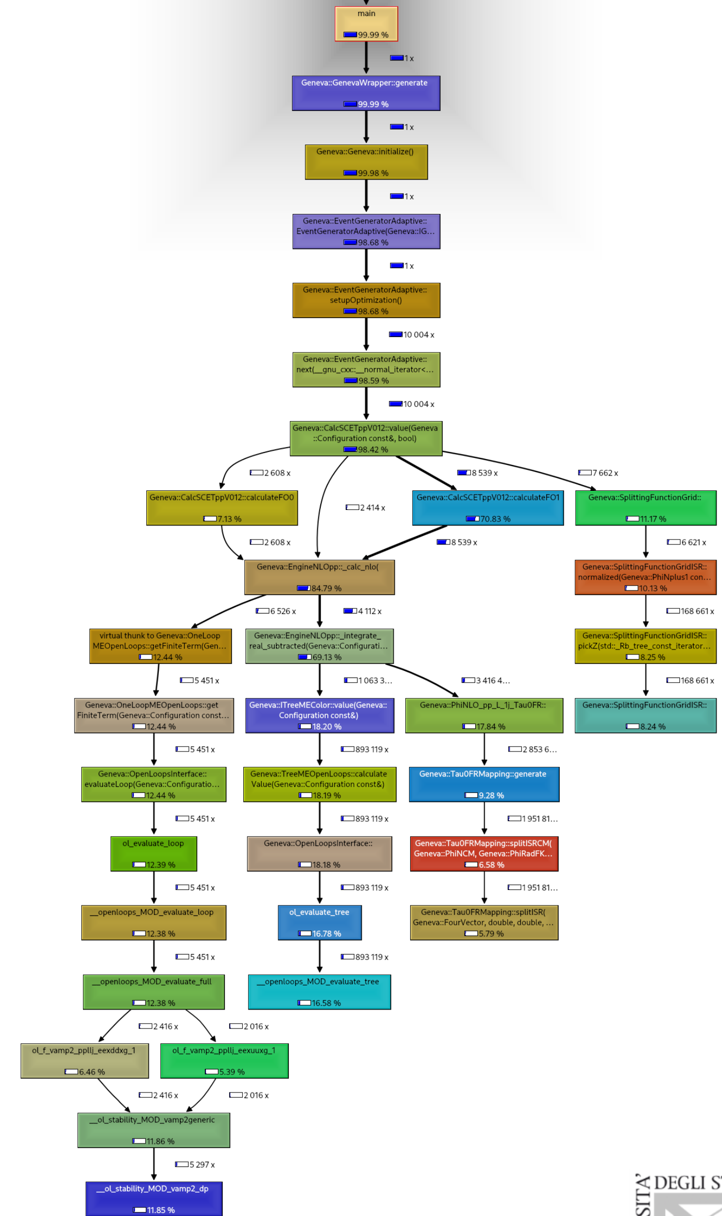
[SA et al. `22]

- $\lesssim 5$ per mille stat. error on distributions

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the setup stage

- Profiling on laptop with Intel Xeon W-10885M CPU, gcc-13 compiler suite
- Using **valgrind —tool=callgrind** cross-checking results with **perf**
- Run with 10K and 100K points to avoid initialization bias
- Drell-Yan case @ NNLL' +NNLO:



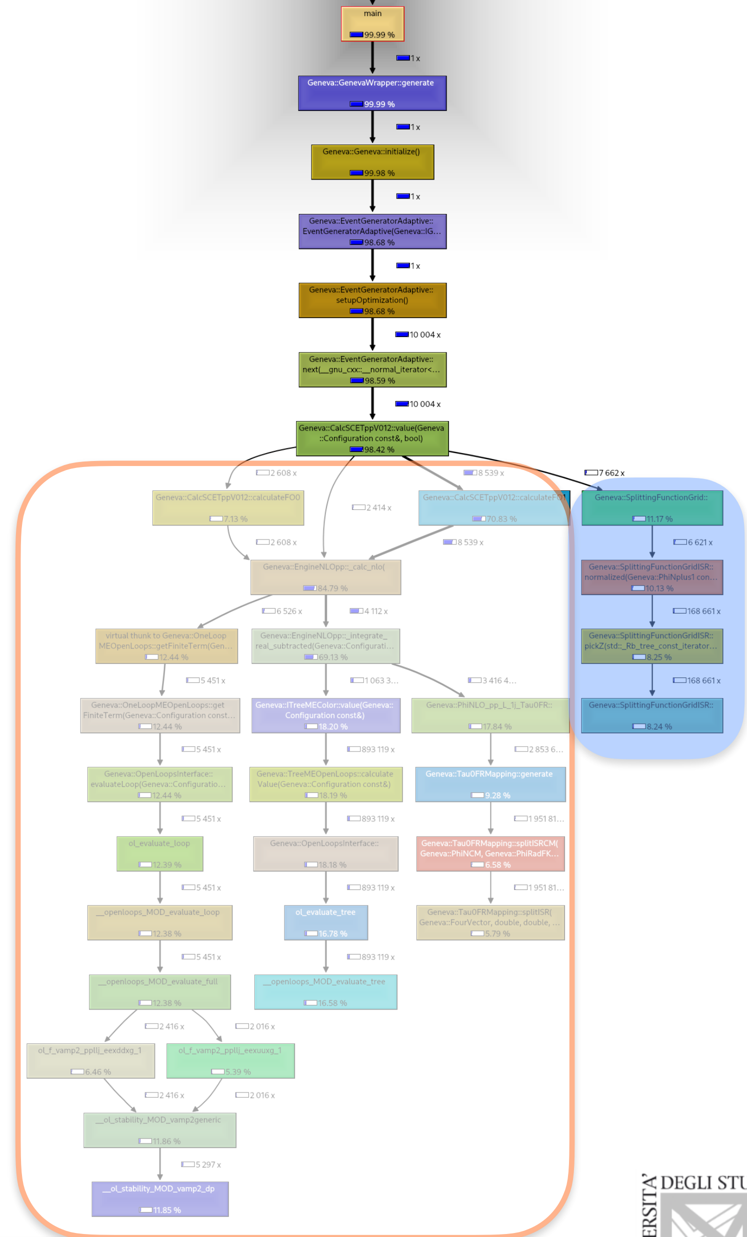
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the setup stage

- Profiling on laptop with Intel Xeon W-10885M CPU, gcc-13 compiler suite
- Using **valgrind —tool=callgrind** cross-checking results with **perf**
- Run with 10K and 100K points to avoid initialization bias
- Drell-Yan case at NNLL’+NNLO:

85% spent in NLO calculations

12% spent in splitting functions



UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

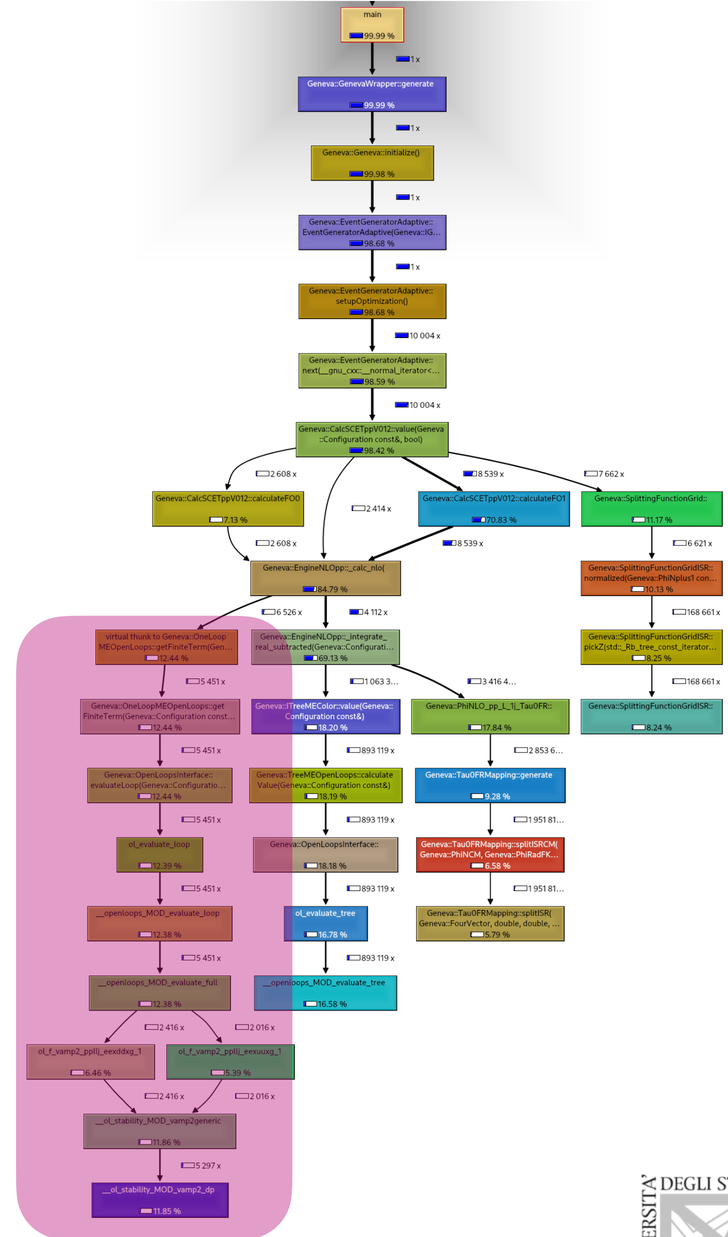
Profiling the setup stage

- Profiling on laptop with Intel Xeon W-10885M CPU, gcc-13 compiler suite
- Using **valgrind —tool=callgrind** cross-checking results with **perf**
- Run with 10K and 100K points to avoid initialization bias
- Drell-Yan case @ NNLL’+NNLO:

85% spent in NLO calculations

12% virtual ME

12% spent in splitting functions



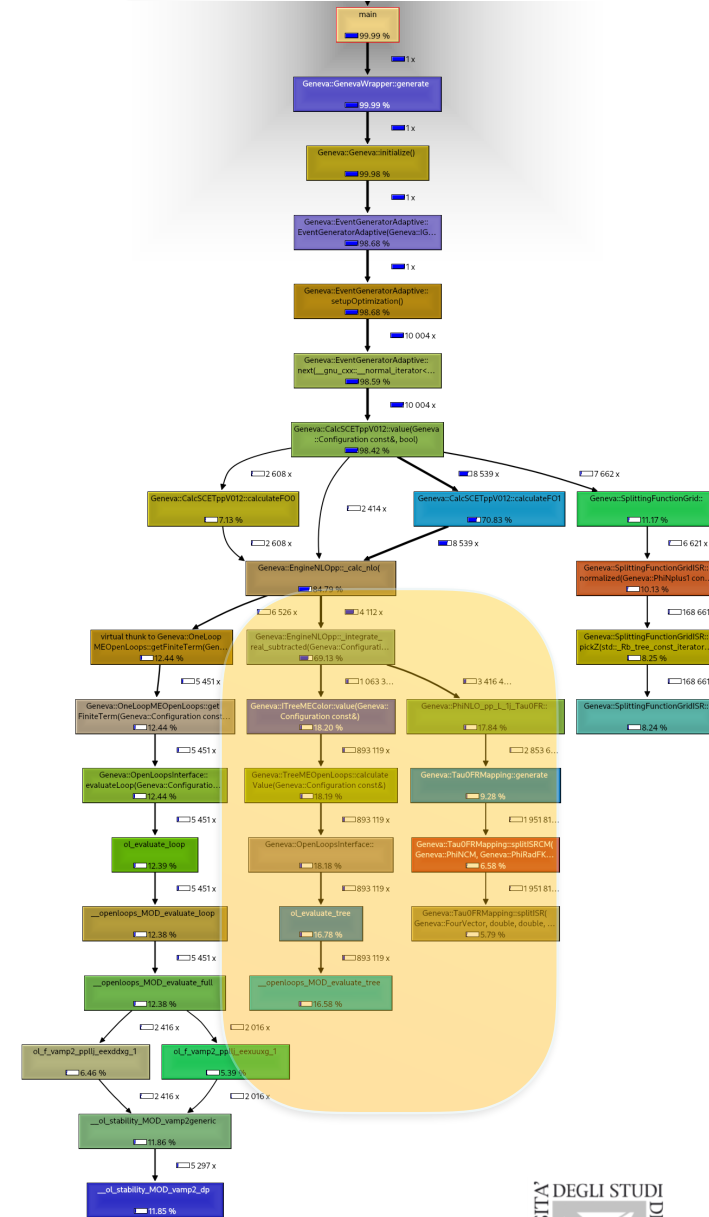
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the setup stage

- Profiling on laptop with Intel Xeon W-10885M CPU, gcc-13 compiler suite
- Using **valgrind —tool=callgrind** cross-checking results with **perf**
- Run with 10K and 100K points to avoid initialization bias
- Drell-Yan @ NNLL'+NNLO:

85% spent in NLO calculations
12% virtual ME
70% real MEs, mappings and sub
(evaluated many times for each Born PS point)

12% spent in splitting functions



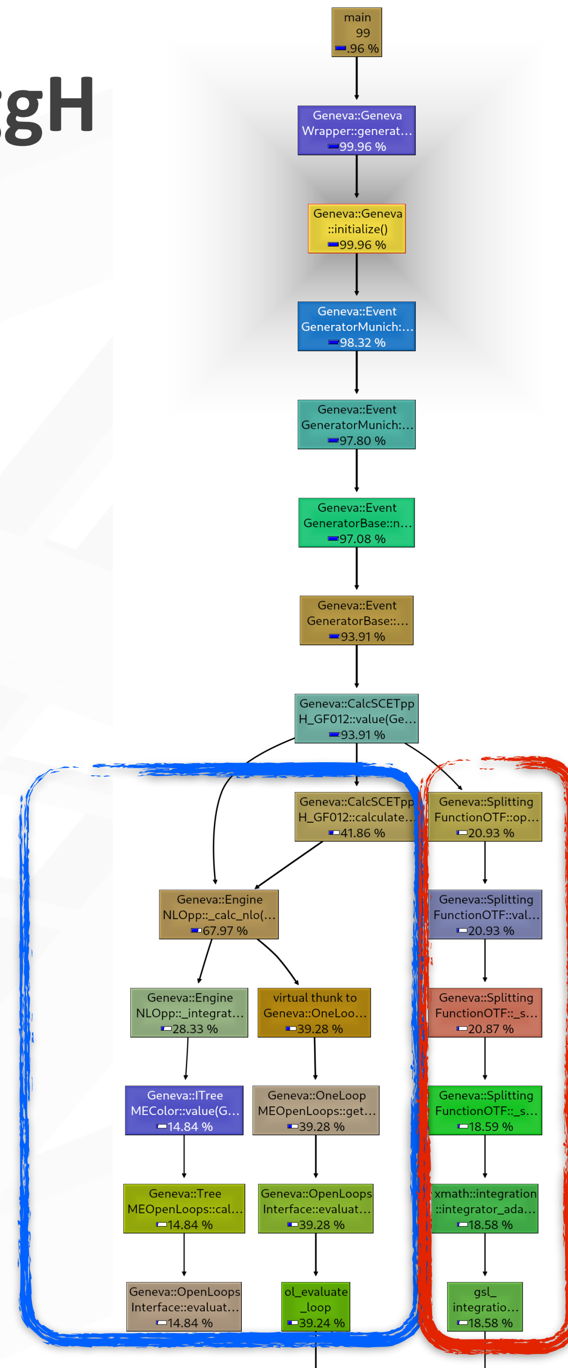
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the setup stage - ggH

- ggH case @ NNLL'+NNLO: MUNICH, on-the-fly splitting functions, OpenLoops ME

70% spent in NLO calculations
40% virtual ME
30% tree ME, mappings and subs.

20% spent in on-the-fly integration of splitting functions



Profiling the setup stage

- **ZZ case @ NNLL’+NNLO**:
  MUNICH, on-the-fly splitting functions, OpenLoops ME, qqVVamp

- Valgrind can only handle 100 events, results slightly biased by initialization times (perf could do 1K, similar values)

96% spent in 2-loop hard function evaluation

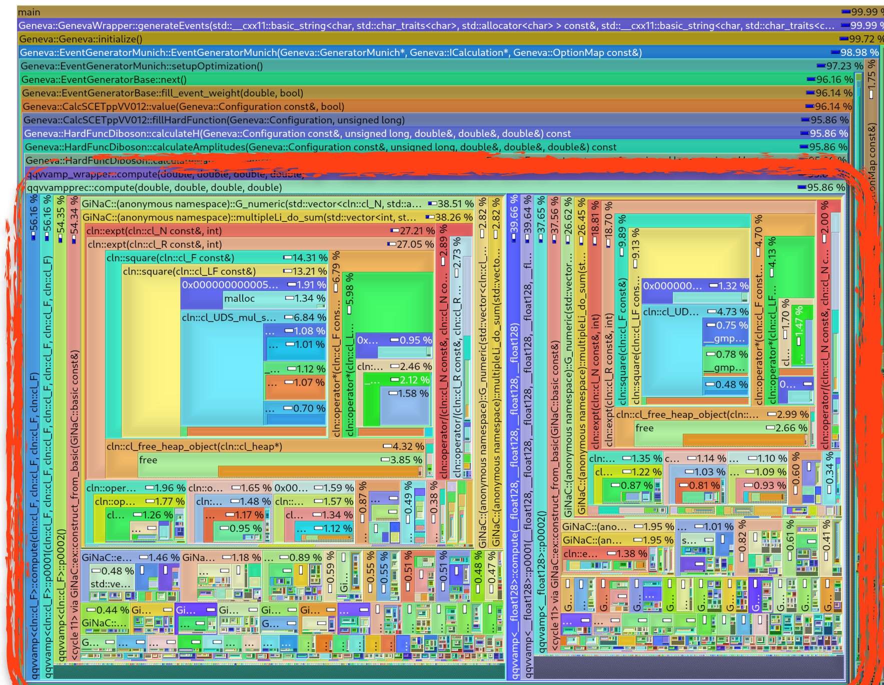
Optimizing anything else seems pointless !

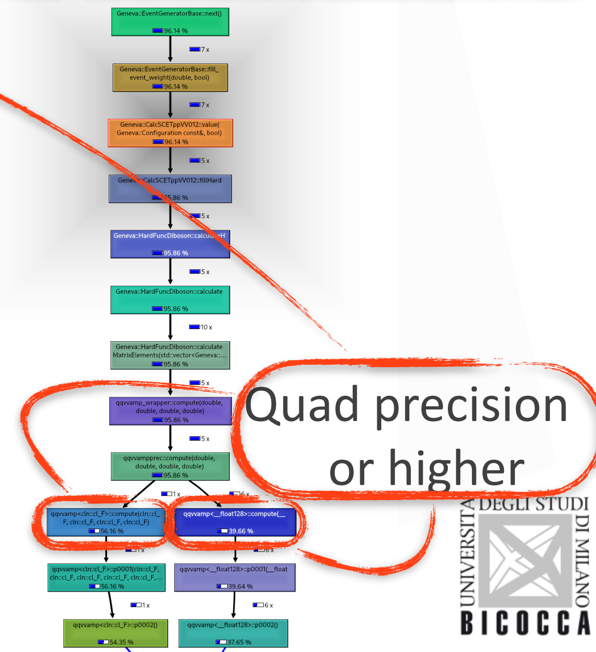
  - If parametric dependence is not too large, fitting is a viable option

    For WW 20 coeffs: 4-dim real functions

    [Wiesemann et al. `21]

  - ML techniques can help, neural network multi-variable integration with M_{Z_1}, M_{Z_2} parametric dependence [Maitre, Santos-Mateos `23]





UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Other possible improvements

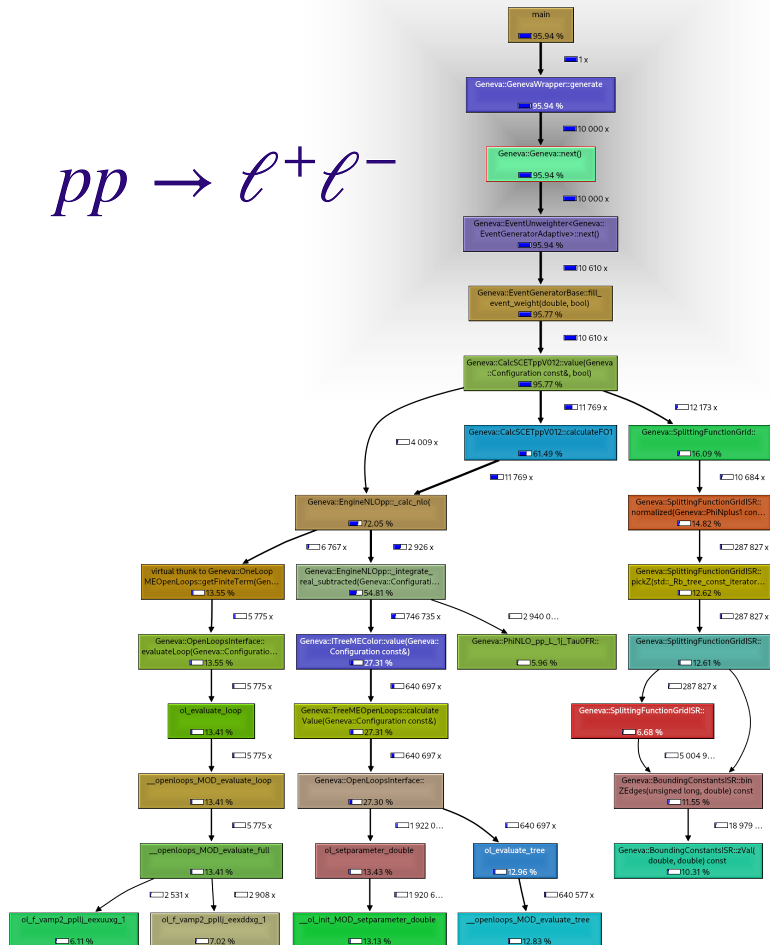
- Replacing (some parts of) the setup stage with machine-learned importance sampling with normalizing flows (trained on GPUs) [MadNIS, Heimel et al. `23]
- Caching events and evaluating ME on GPUs can achieve impressive speedup
- Replacing lower-dimensional integrals (e.g. 2D integral over splitting functions) with cubature methods (done on GPUs) rather than MC ones will reduce the number of points needed [DYturbo, Camarda et al. `19]
- The real-subtraction integrals in NLO calculations can also be reduced to 2D or 3D integrals (doable with cubature methods on GPUs)
- These integrands are complicated functions (matrix-elements, PDFs), depending on external libraries, i.e. not immediately portable to GPUs

It is crucial that developers of ML-improved phase-space integrators and GPU-ready Matrix Element libraries distribute these as standalone packages, using well-defined interfaces (e.g. updated BLHA accords), without the generator-specific unnecessary overhead.

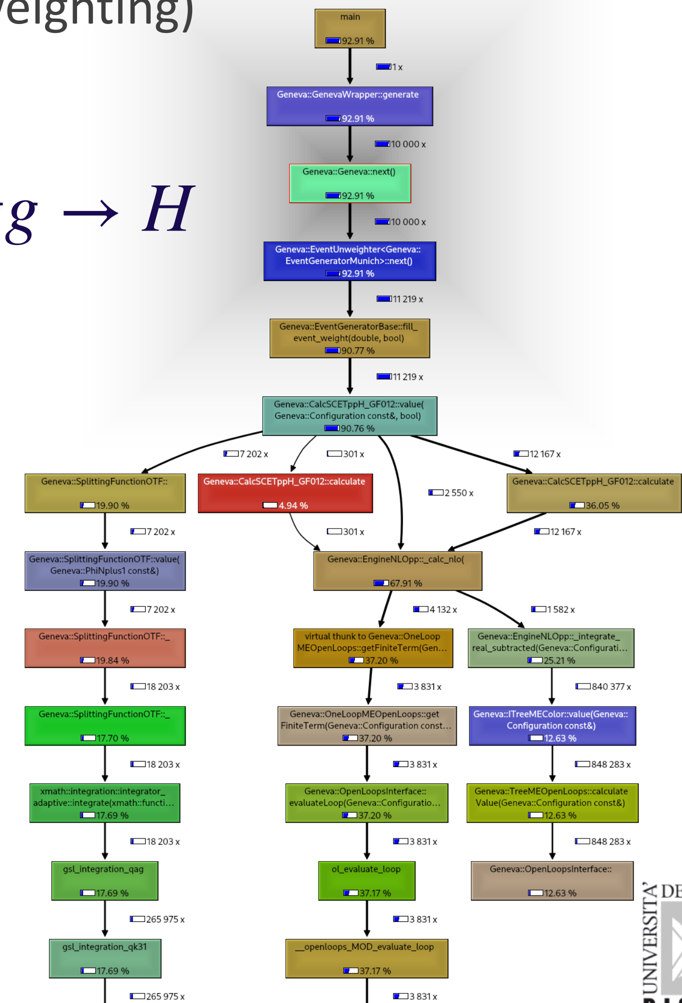
Profiling the generation stage - DY and ggH

- Generation stage is exactly as the setup stage, minus the optimization part.
- Profiling does not show significant variations from last iteration of setup
- Event write-out never a limiting factor (unweighting)

$pp \to \ell^+\ell^-$



$gg \to H$

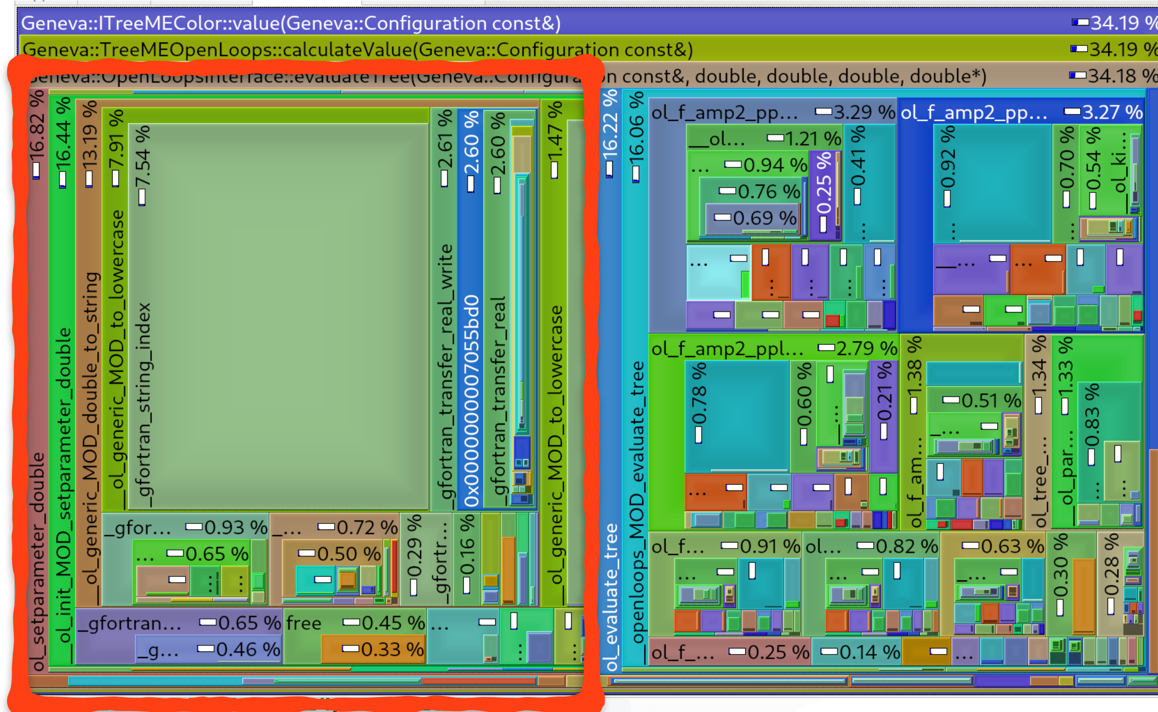


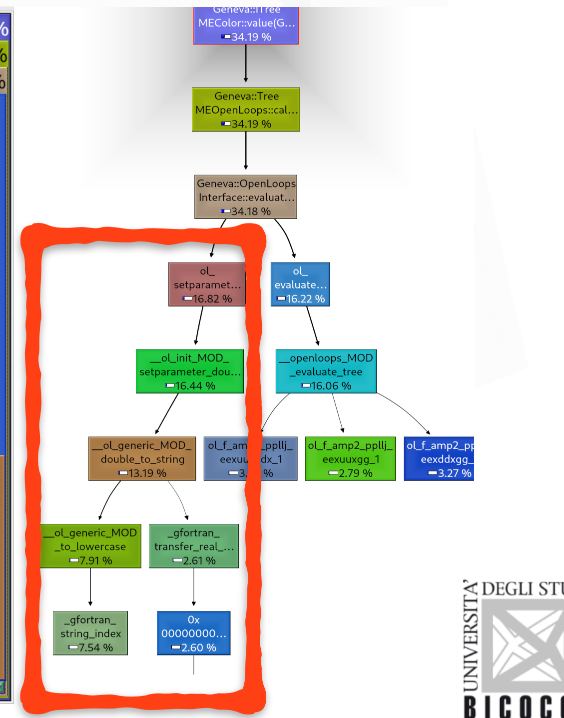
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the generation stage

- Important to carefully look at interface to codes that have already been optimized on their own and pay attention to optimize those interfaces too.
- Extreme example in GENEVA dynamical parameter settings α_s, μ_R in OpenLoops Iface where 50% of tree-level ME time (~10% of total time) spent on GFORTRAN string look-up
- With a simple caching system this is easily fixed

Incl.	Self	Called	Function	Location
9.70	9.70	78 489 499	_gfortran_string_index	libgfortran.so.5.0.0





DEGLI STUDI DI MILANO
UNIVERSITÀ
BICOCCA

Profiling the generation stage - ZZ diboson

- Using **valgrind** not possible (>80GB RAM needed), results limited to **perf**
- Run 1K points to avoid initialization bias
- ZZ case: as for the setup stage 93% of time spent on 2-loop hardfunc evaluation
- Same improvements using grids employed in setup could speedup generation too.

Samples: 740K of event 'cycles:Pu', Event c

Overhead	Shared Object
44.39%	libginac.so.5.0.3
26.60%	libcln.so.6.0.6
11.77%	libc.so.6
5.97%	libgmp.so.10.5.0
4.46%	libGeneva-hardfunc_qqvv.so.1.1.0
3.94%	libgcc_s.so.1
1.68%	libstdc++.so.6.0.32
0.54%	libopenloops.so
0.21%	libmunich-core.so
0.07%	libGenevaCore.so
0.06%	libpythia8.so
0.06%	[unknown]
0.05%	libquadmath.so.0.0.0
0.05%	libopenloops_pplllj_lt.so
0.03%	libgfortran.so.5.0.0
0.03%	ld-linux-x86-64.so.2
0.03%	libLHAPDF.so
0.02%	libm.so.6
0.01%	libGenevaComponents.so
0.01%	libmunich-ppemexmx04.so
0.01%	libGenevaCalculations.so
0.00%	libGenevaDriver.so
0.00%	libopenloops_pplll_lt.so

Possible improvements for generate stage

- The generation stage is the most resource-hungry. This is where the optimization efforts should focus. It shares the same problems of the setup stage, and it could benefit from the same improvements in integration [MadNIS, Heimel et al. `23] [Gao et al. '20]
- At this stage the events are written out, so strategies to unweight the events and minimize the amount of data that needs to be stored <u>while maintaining the same statistical properties and correlations</u> of the weighted samples. This is a statistics and computer-science problem where interactions with experts could prove useful! Already some ideas using ML techniques [ML in SHERPA, Danziger et al. `22] [GAN Backes et al. `21] [NN with Fact aware ME, Janssen et al. `23]
- On-the-fly parameter variations (like scale or PDFs) only need to be calculated on events which are retained, achieving sizable speed-ups.
- Embarrassingly parallel approach should ensure best scaling, but require GPU-readiness of inner tools: one can easily vectorize over batch of events but needs GPU-ready MEs, PDFs, integrators …

Reweighting stage

- In GENEVA this is done by a mpi4py python script, completely parallelized. Each job reads one file, extracts the required info, share it to other cores via MPI.reduce all-to-all comm. and calculates the re-weighting factor.

- Each cores re-writes its own file adjusting the weight according to the newly calculated re-weight factor

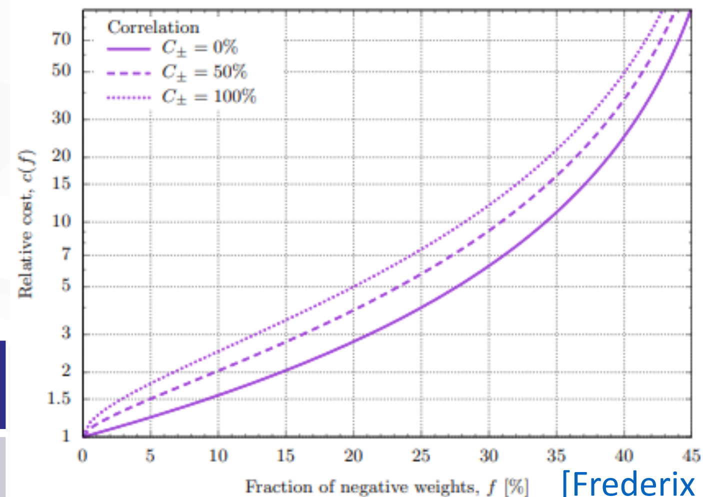
- This is currently limited by I/O read/write of gzip compressed files.

- Cell Resampling or alternative methods to reduce the fraction of negative weights are being explored

$$N(f_) = \frac{N(f_ = 0)}{(1 - 2f_)^2}$$

[Andersen et al. `23]

	ggH	DY	ZZ
$f_$	20 %	17 %	13 %



[Frederix et al. `20]

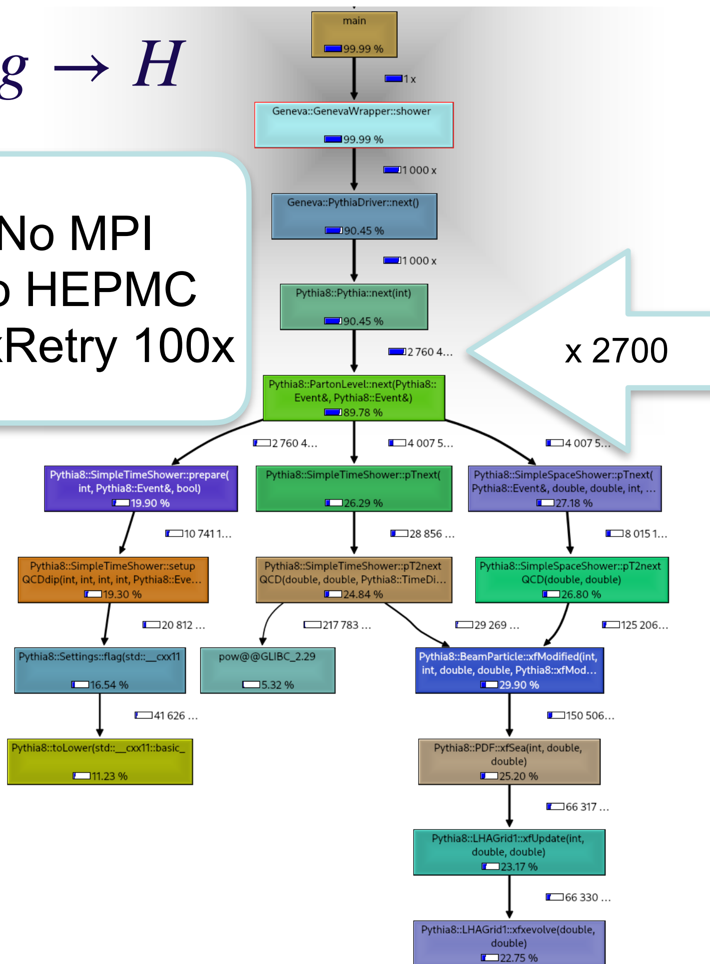
- New ideas and algorithms for efficient unweighting to positive weights could be beneficial: again a computer-science problem!

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the showering stage - PYTHIA8

$gg \to H$

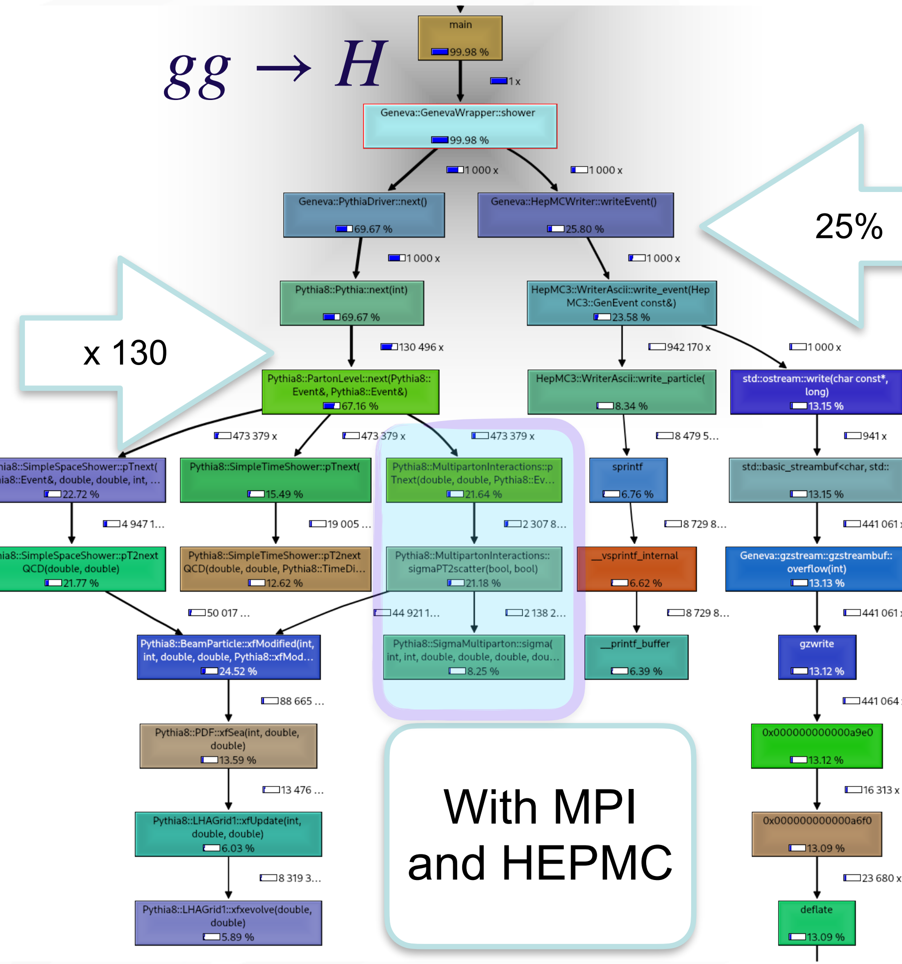
No MPI
No HEPMC
MaxRetry 100x



x 2700

x 130

$gg \to H$



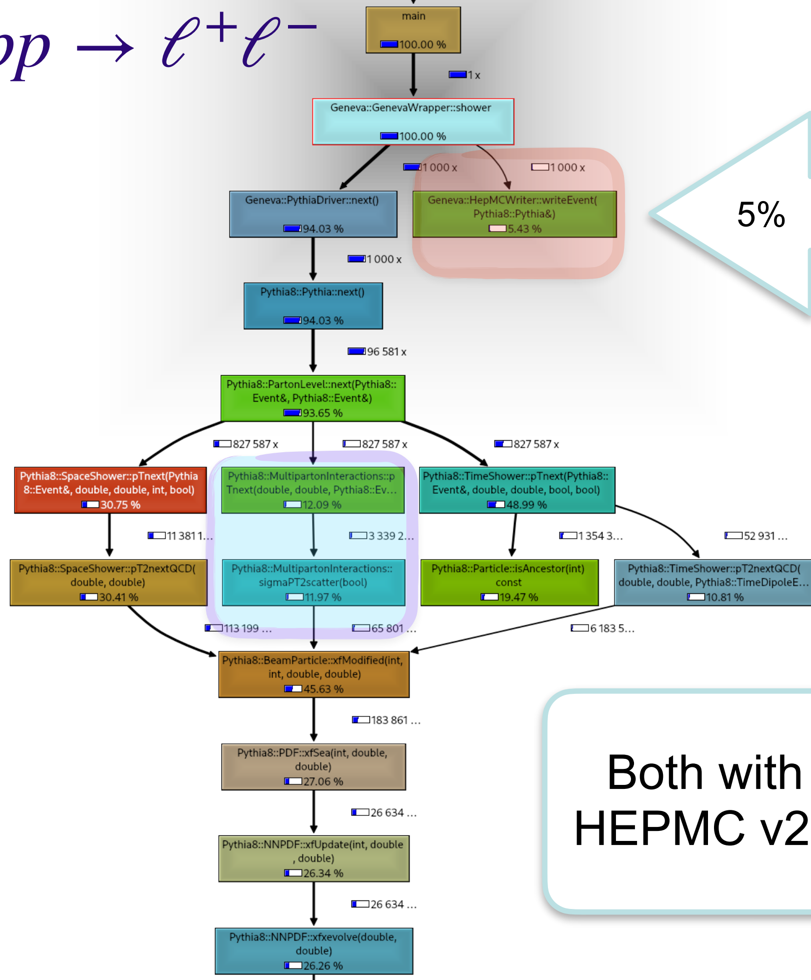
25%

With MPI
and HEPMC

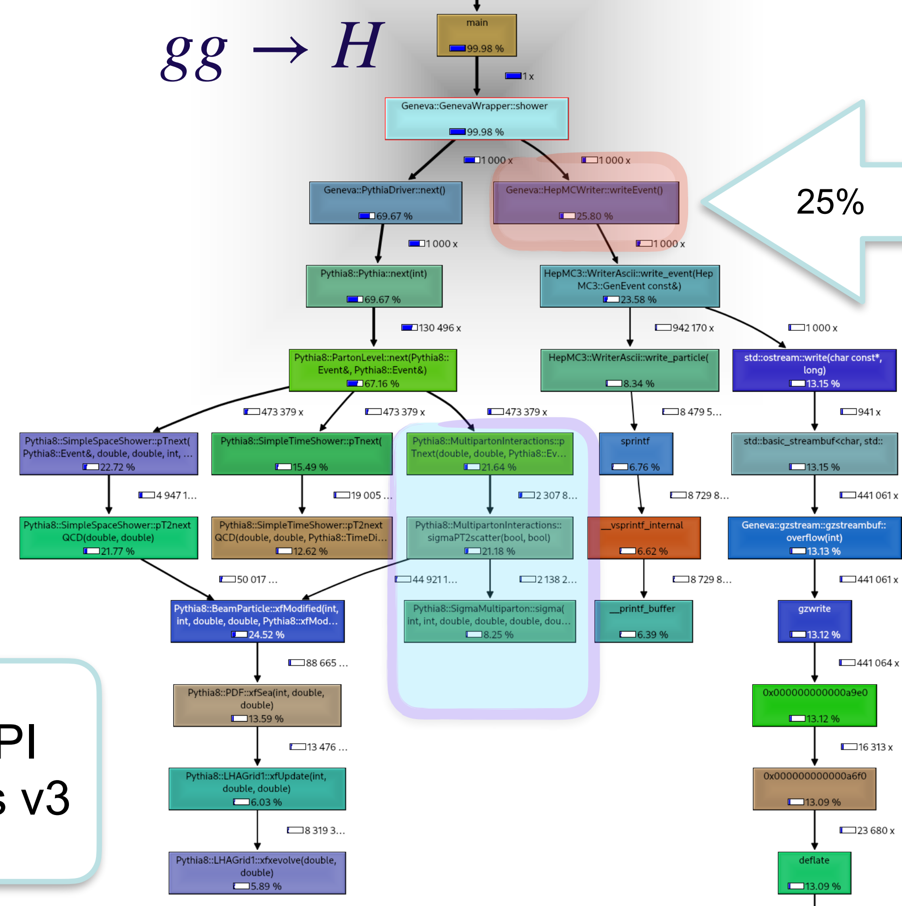
- Pythia is fast (cit. L. Lonnblad) but if you need it to re-run it multiple times (up ~2.7K times per event in ggH) it can always be faster!
- MPI is on-par with normal showering

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Profiling the showering stage - PYTHIA8



5%



Both with MPI
HEPMC v2 vs v3

- HepMC I/O takes its toll, but it is much better when properly optimized
    - MPI is on-par with normal showering

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Conclusion and outlook

- Reported profiling exercise to figure out the production runtimes and bottlenecks of NNLO+PS simulations
- For simple processes (ggH and DY) matrix elements, both loop and tree-level are the heavy hitters. NLO subtractions might also become expensive if done multiple times for each Born point.
- For ZZ production slowdown is entirely due to 2-loop calculation. Replace it with (ML-inspired) fits if authors do not provided fast codes.
- There are possibilities to gain considerable speedups by moving to vectorization/GPUs but need libraries with GPU-ready ME and PDFs GPUs can also be employed to replace low-dimensional MC integrations with deterministic methods
- Showering stage more expensive with GENEVA, but not the bottleneck.

- NNLO+PS are tools that make the most accurate theory predictions available in an easy-to-use event format. Experimental collaborations should try their best to exploit them! We are are to help!
- The HPC community has moved to GPUs, either we rapidly adapt our codes or are left without machines to run on!

Thank you for your attention.

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA