Accelerating PYTHIA8

Leif Lönnblad
On behalf of the PYTHIA8 collaboration

Department Physics
Lund University

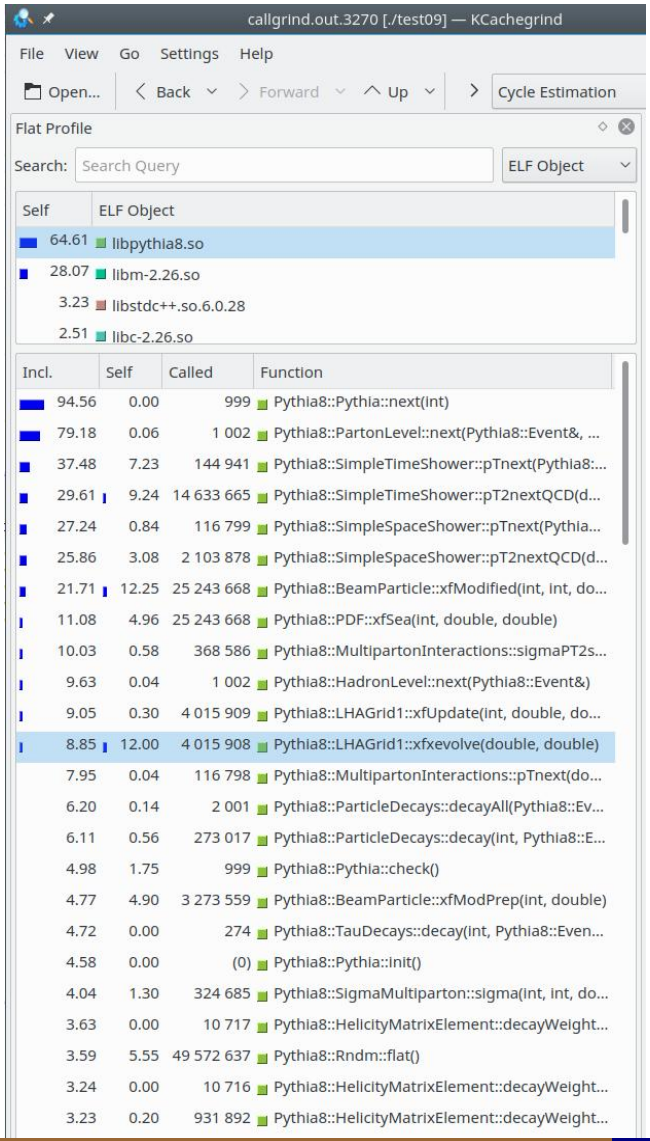
CERN, 2023-11-13

Outline

- Introduction
- Recent developments
  - `ParallelPythia`, Shower variations, biasing
- New developments
  - Hadronisation weights, Onium showers
- Future developments
  - Sectorised merging histories, GPUs,

[arXiv:2203.11601, A comprehensive guide to the physics and usage of PYTHIA8]

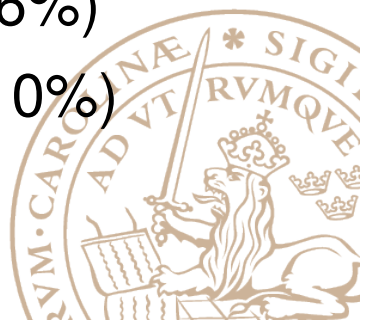
Is PYTHIA8 a bottleneck?



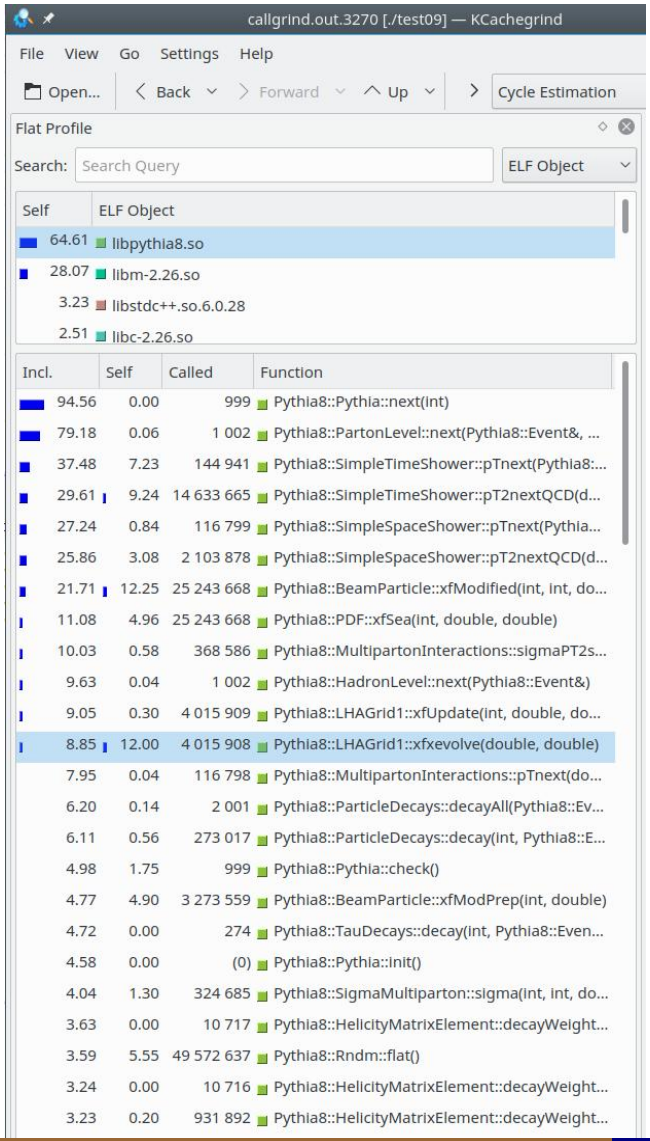
In its *default* setup PYTHIA8 is fast.

One (13 TeV $t\bar{t}$) event takes ∼10 ms

- ∼ 40% Final state shower
- ∼ 30% Initial state shower
- ∼ 10% Multi parton interactions
- ∼ 10% Hadronisation + decays
- ∼ 50k random numbers (∼6%)
- ∼ 4k PDF evaluations (∼10%)



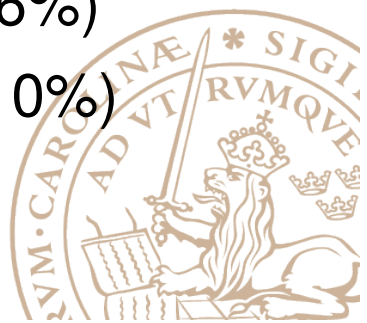
Is PYTHIA8 a bottleneck?



In its *default* setup PYTHIA8 is fast.

One (13 TeV $t\bar{t}$) event takes $\sim$10 ms

- $\sim$ 40% Final state shower
- $\sim$ 30% Initial state shower
- $\sim$ 10% Multi parton interactions
- $\sim$ 10% Hadronisation + decays
- $\sim$ 50k random numbers ($\sim$6%)
- $\sim$ 4k PDF evaluations ($\sim$10%)
- $<$ 2% ME generation



Every step in PYTHIA8 depends on what happened in the previous step, so parallelisation is in general not possible.

Even where parallelisation is possible, the steps typically involves a lot of admin code that is not suited for e.g. GPUs.

But default PYTHIA8 is fast so no problem!

Paralellisation

Every event is independent, so PYTHIA8 itself can be parallelised

- Everything is perfectly thread safe
- OpenMP and HDF5 LHEF supported
- New wrapper class using `std::thread`

ParallelPythia

```
# from examples/main161.cc

#include "Pythia8/Pythia.h"
#include "Pythia8/PythiaParallel.h"
using namespace Pythia8;
int main() {
  // Use the PythiaParallel class for parallel generation.
  PythiaParallel pythia;
  pythia.readString("Beams:eCM = 8000.");
  pythia.readString("HardQCD:all = on");
  pythia.readString("PhaseSpace:pTHatMin = 20.");
  pythia.init();
  Hist mult("charged multiplicity", 100, -0.5, 799.5);
  // Use PythiaParallel::run to generate the specified number of events.
  pythia.run(10000, [&](Pythia* pythiaPtr) {
    // Find number of all final charged particles and fill histogram.
    int nCharged = 0;
    for (int i = 0; i < pythiaPtr->event.size(); ++i)
      if (pythiaPtr->event[i].isFinal() && pythiaPtr->event[i].isCharged())
        ++nCharged;
    mult.fill( nCharged );
    // End of event loop. Statistics. Histogram. Done.
  });
  pythia.stat();
  cout << mult;
  return 0;
}
```

Shower variations

With multiple weights per event we can save a lot of CPU time for scale variations etc.

Scale variations is easy for MEs. Showers are more tricky.

- We have splitting functions and no emission functions

$$P(p_\perp^2) \times \exp\left(- \int_{p_\perp^2}^{p_{\perp,\mathrm{prev}}^2} dk_\perp^2 \, P(k_\perp^2) \right)$$

- Using the veto algorithm we oversample with a simple overestimate function and throw to get the correct distribution.

[arXiv:1605.08352]

Every time we accept/reject we can get weights for different variations,

$$P(p_\perp^2, \alpha_s(\mu_R)) \to P(p_\perp^2, \alpha_s(C\mu_R)) \quad \Rightarrow \quad p_{\text{acc}} \to p'_{\text{acc}}, p_{\text{rej}} \to p'_{\text{rej}},$$

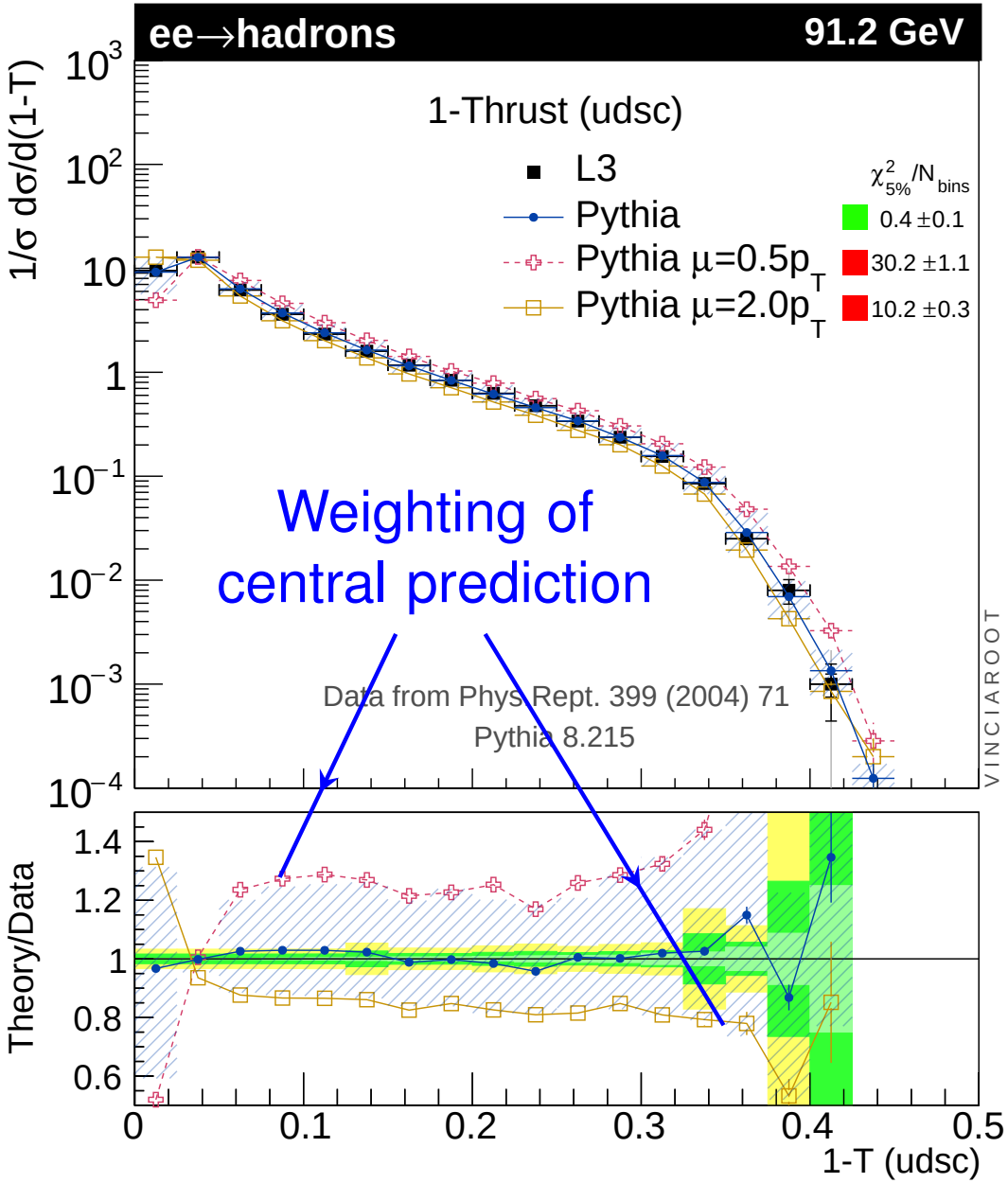
accept/reject according p_{acc} and p_{rej}, and get an event weight

$$w_{ev} = \prod_i \frac{p'_i}{p_i}$$

Note that large reweighting factors may result in fluctuating weights, so the statistical uncertainty will increase:

$$N_{\text{ev, eff}} = \frac{(\sum w)^2}{\sum w^2}$$

- μ_R scale for QCD emissions in FSR
- μ_R scale for QCD emissions in ISR
- inclusion of non-singular terms in QCD emissions in FSR
- inclusion of non-singular terms in QCD emissions in ISR
- PDF members of a PDF family in LHAPDF6
- individual PDF members of a PDF family in LHAPDF6


ee→hadrons
91.2 GeV
1-Thrust (udsc)
L3
Pythia
Pythia μ=0.5p_T
Pythia μ=2.0p_T
χ²_5%/N_bins
0.4 ±0.1
30.2 ±1.1
10.2 ±0.3
1/σ dσ/d(1-T)
Weighting of central prediction
Data from Phys.Rept. 399 (2004) 71
Pythia 8.215
VINCIAROOT
Theory/Data
1-T (udsc)

Biasing

The same procedure may be used to bias your events, e.g. to get more $g \to b\bar{b}$

$$P'_{b\bar{b}}(p_\perp^2) = CP_{b\bar{b}}(p_\perp^2)$$

but now the rôle of variation and standard setting are reversed, so we accept/reject according to p'_{acc} and p'_{rej}, and get an event weight

$$w_{ev} = \prod_i \frac{p_i}{p'_i}$$

Biasing works well for low probability processes, but for more common processes we are hit by large weight fluctuations.

The statistical uncertainty of an observable scales like a power of the inverse no-emission probability, Δ^{-h}

For an n-emission observable we get an uncertainty

$$\delta\mathcal{O}_n \propto \frac{\Delta^{-h}}{\sqrt{C^n}}, \quad \text{with} \quad h \approx C$$

You cannot bias MPI.

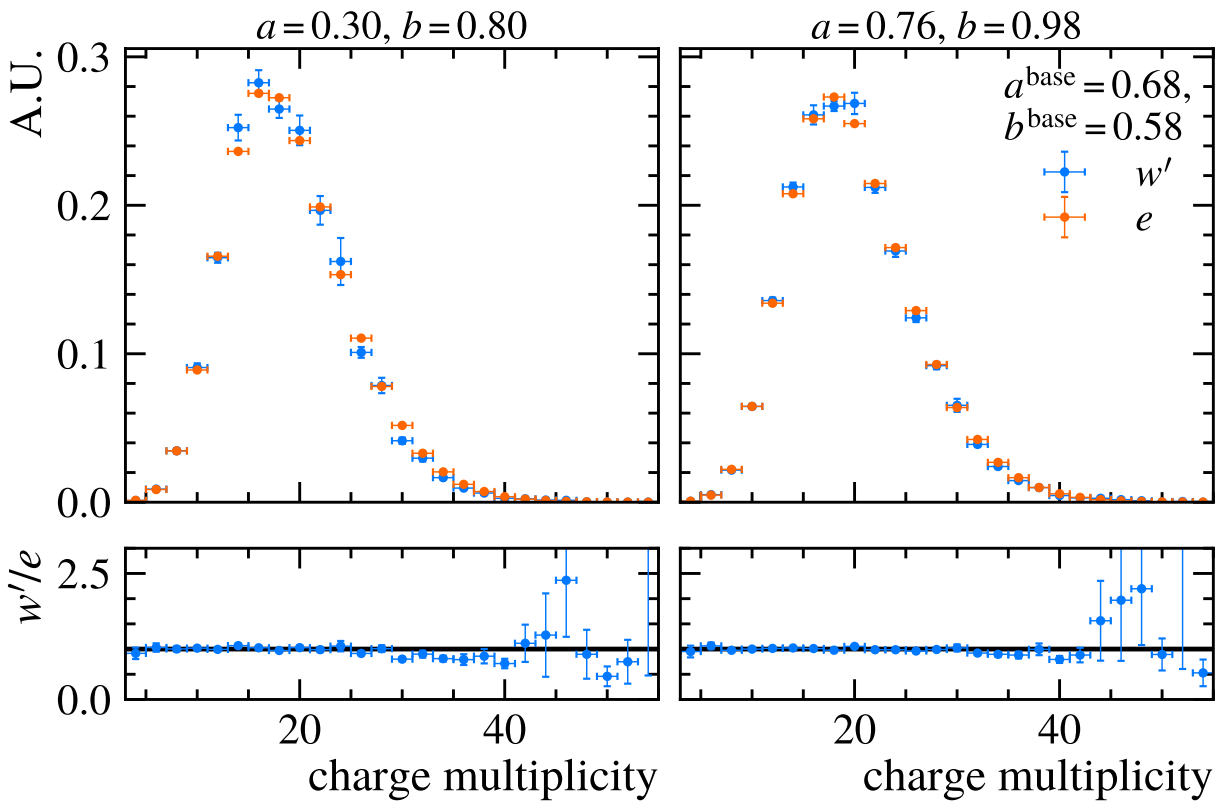
Hadronisation weights

In addition, weight variations are soon also available for hadronisation parameters.

Works on the same principle as shower variations (with the same caveats), but this time modifying hit-and-miss algorithms for e.g. the Lund symmetric fragmentation function.

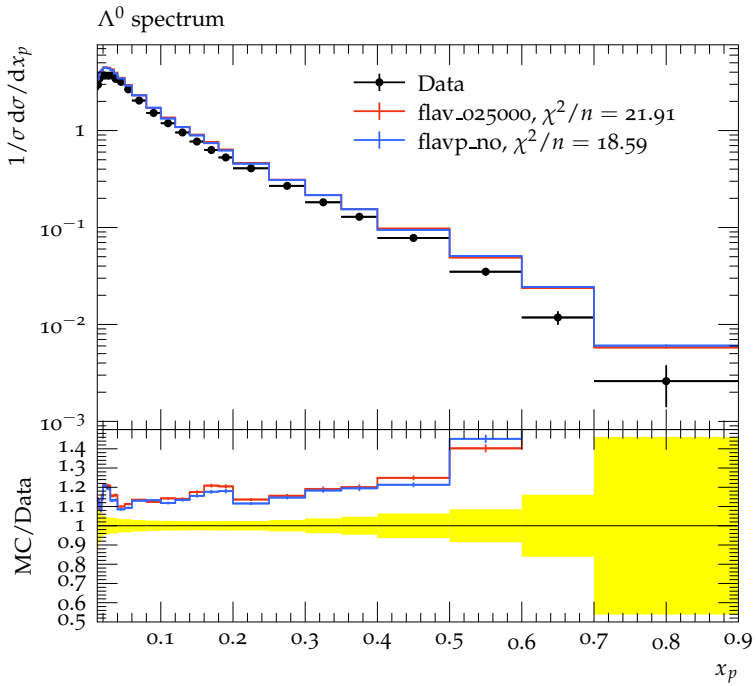
$$p(z) \propto \frac{(1-z)^a}{z} e^{-bm_\perp^2/z}$$

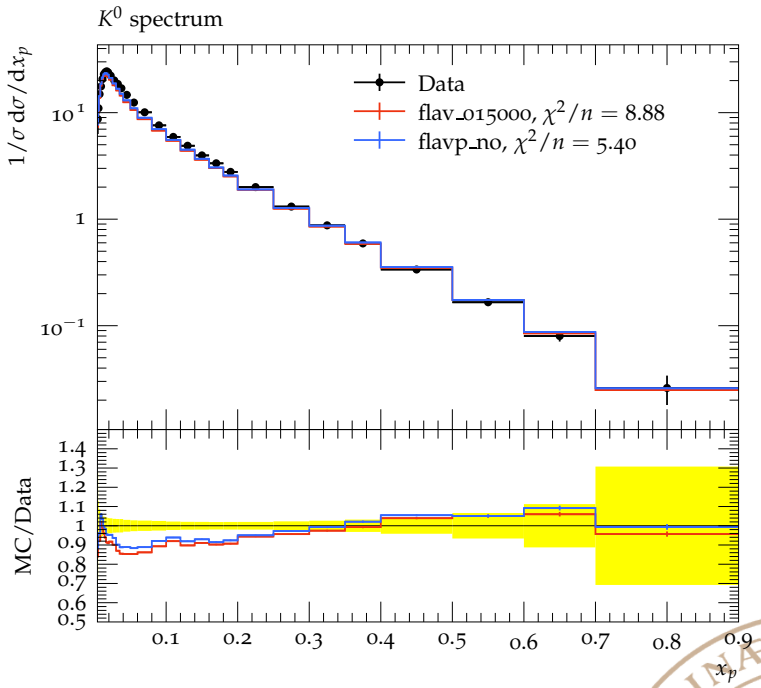
[arXiv:2308.13459]


$a = 0.30,\ b = 0.80$
$a = 0.76,\ b = 0.98$
A.U.
0.3
0.2
0.1
0.0
$a^{\text{base}} = 0.68,$
$b^{\text{base}} = 0.58$
w'
e
w'/e
2.5
0.0
20
40
charge multiplicity
20
40
charge multiplicity

Flavour variations

$\rho =$`StringFlav:probStoUD`

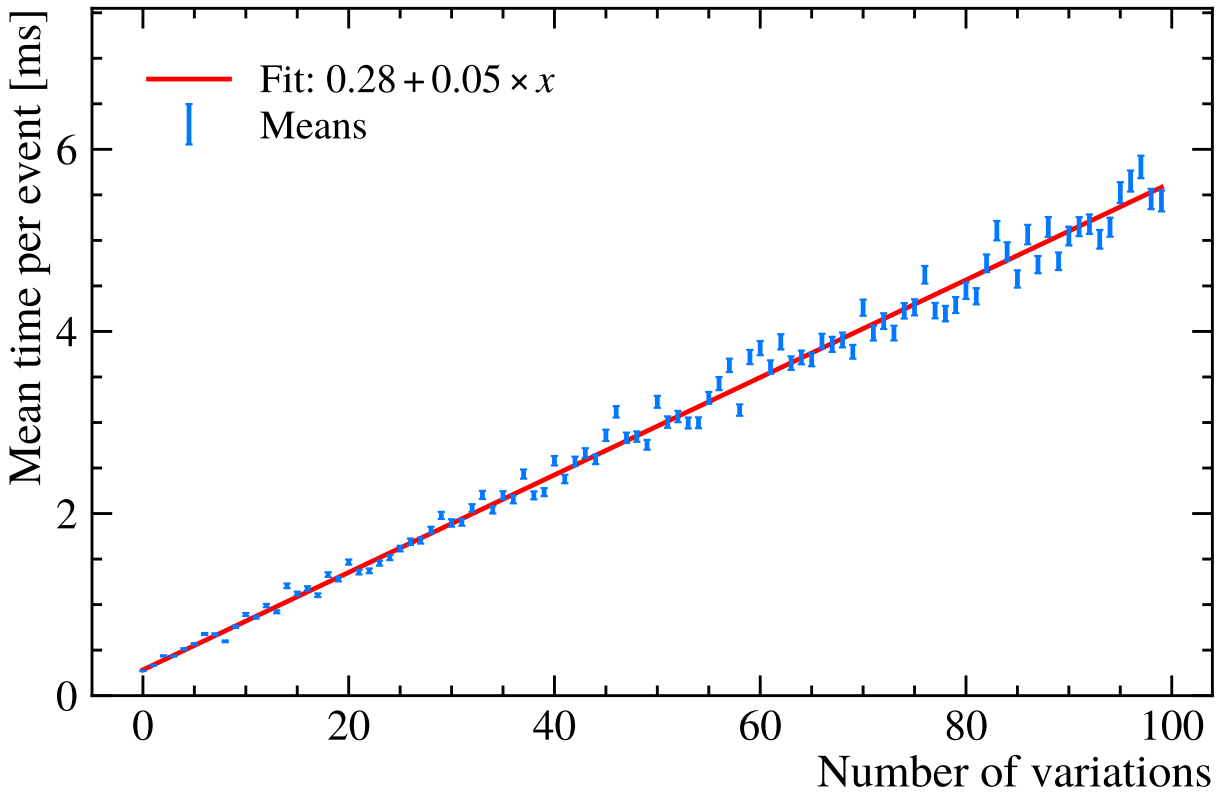




$\rho = 0.217 \rightarrow \rho = 0.25$

$\rho = 0.217 \rightarrow \rho = 0.15$

Timing


Fit: 0.28 + 0.05 × x
Means
Mean time per event [ms]
Number of variations
0
2
4
6
0
20
40
60
80
100

This is mainly useful for tuning

But again, if you can do weight variations you can do biasing (caveats still applies)

Onia showers

PYTHIA8 now includes shower splittings into charmonium and bottomonium splittings, e.g. $c \to J/\psi + c$ and $g \to O(^3P_J) + g$

(it's unfortunately a bit slow)

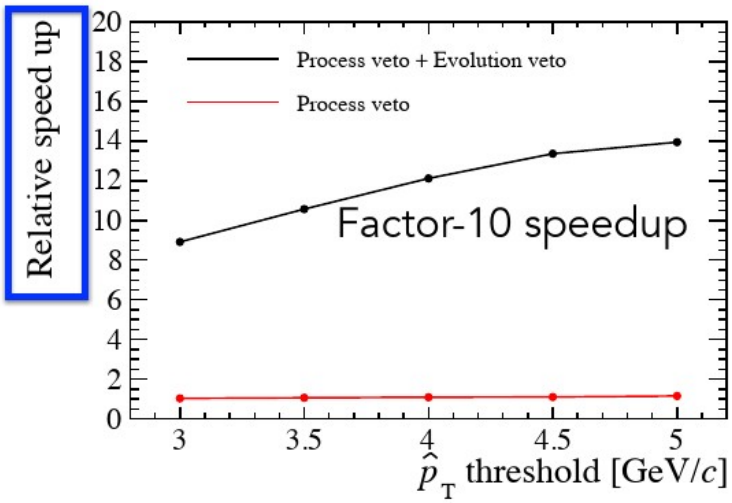
Biasing with a constant factor available.

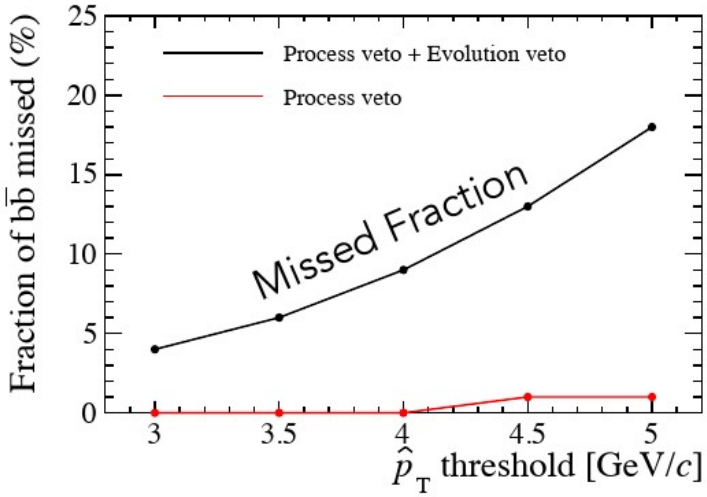
[available since version 8.310]

Biasing heavy flavours

An alternative to biasing is to simply try to veto uninteresting events at a very early stage.

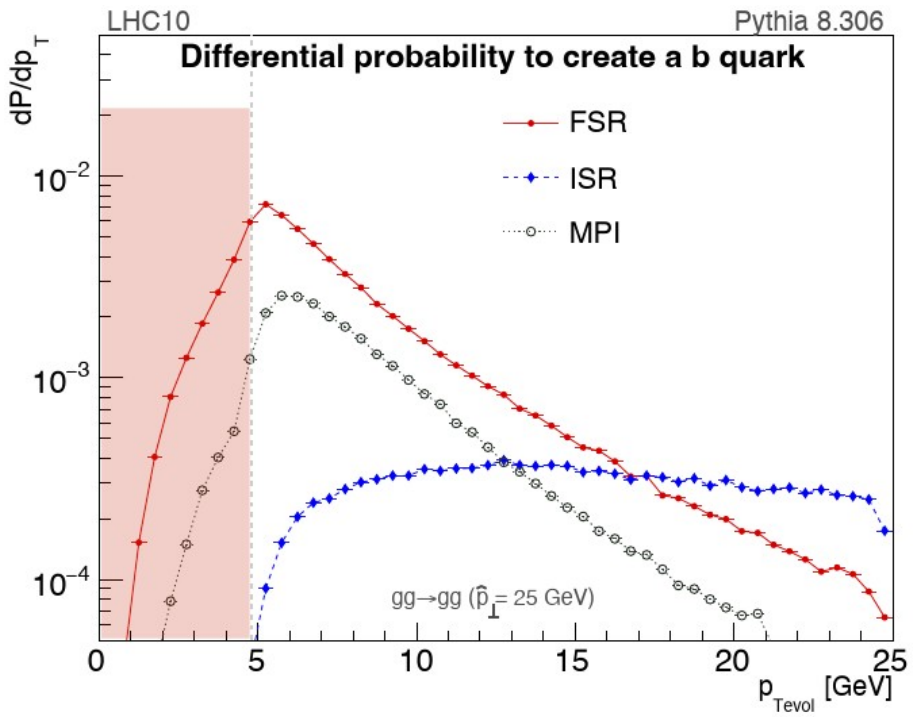
Using `UserHooks` you can tell PYTHIA8 to veto an event, e.g. when the evolution scale has dropped below some scale, $\hat{p}_T$.





Problem: aggressive speedup misses part of the b cross section

[Ulrik Egede et al. arXiv:2205.15681]



- ISR has the expected behaviour, FSR and MPI not.
- WIP: reformulate shower to have a more physical evolution scale so that all *b*s are produced at scales above m_b.

Beyond the default PYTHIA8

While the default PYTHIA8 is fast, we sometimes need speed-up.

In addition PYTHIA8 includes important optional modules which are quite CPU demanding, e.g.

- Hadronic rescattering has complexity N_h^2 (CPU$\times$2)
- QCD-based Colour reconnections, N_d^2 or even N_d^3 (CPU$\times$15)
- CKKW-L type merging: N_l!

ME+PS merging

For CKKW-L style merging (UMEPS, NL3, UNLOPS, . . .) you need to take all contributing shower histories into account for a given ME state.

In conventional PS each possible phase-space point receives contributions from many possible branching “histories” (or “clusterings”). Approximately sums over (singular) diagrams to give the full singularity structure.

# branchings	1	2	3	4	5	6	7
# histories	2	8	48	384	3840	46080	645120

histories $\sim$ # feynman diagrams grow faster than factorial

Sector Showers

The default VINCIA shower in PYTHIA8 is unique in being a "Sector Shower"

- Divide N-gluon Phase Space into N "sectors", with step functions.
- Each PS sector corresponds to one specific gluon being the "softest" in the event — the one you would cluster if you were running a jet algorithm (specifically one called ARCLUS)
- Inside each sector, only a single kernel is allowed to contribute (the most singular one)!

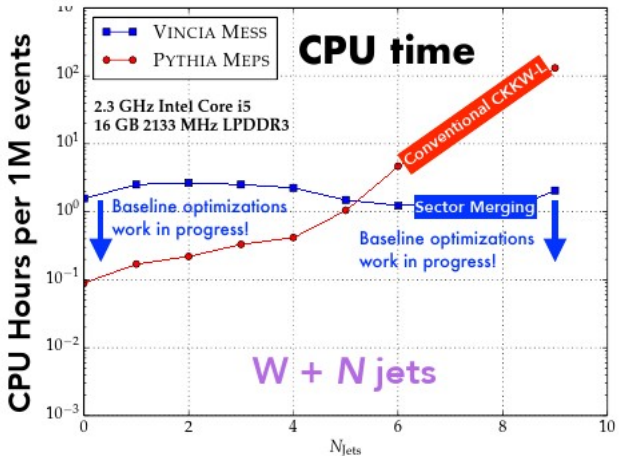
Sector Kernel = the eikonal for the soft gluon and its collinear DGLAP limits for $z > 1/2$.

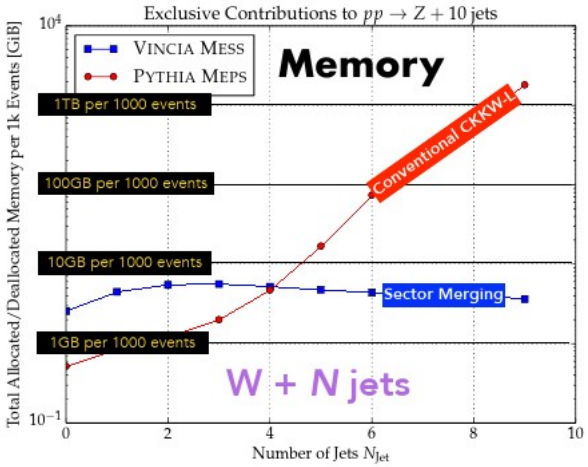
Only a single (product of) kernel(s) contributes to each phase-space point. $N! \to 1!$

[Skands & Villajero arXiv:1109.3608, Brooks, Preuss, Skands arXiv:2003.00702]

Sector Merging with VINCIA Sector Shower

Available since PYTHIA 8.306





Extentsons are coming:

- Sectorized matching at NNLO
  (proof of concept: arXiv:2108.07133, arXiv:2310.18671)
- Sectorized iterated tree-level ME corrections
  (demonstrated in arXiv:1109.3608)
- Sectorized multi-leg merging at NLO

[Brooks & Preuss arXiv:2008.09468]

Preview: VINCIA NNLO+PS for $H \to b\bar{b}$

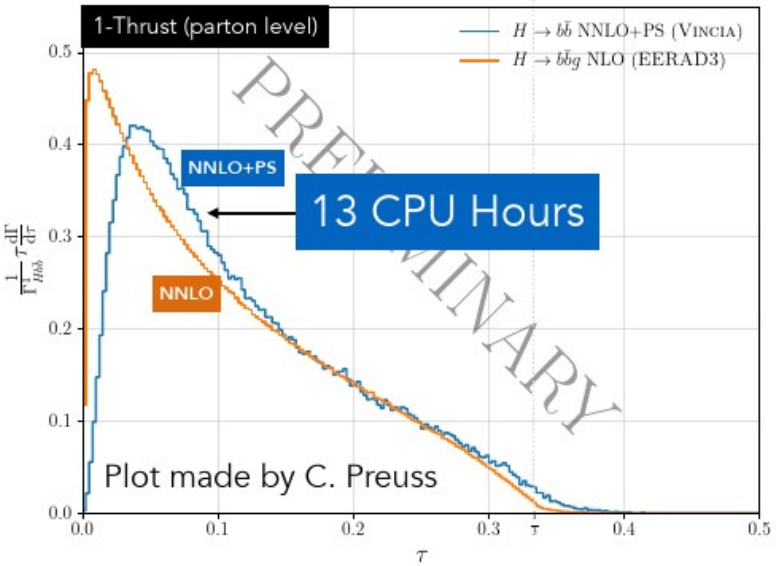
Coloretti, Gehrmann-de Ridder, Preuss, JHEP 06 (2022) 009

Fixed-Order Reference = EERAD3 NLO $H \to b\bar{b}g$: already **Highly optimised**

Uses analytical MEs, "folds" phase space to cancel azimuthally antipodal points, and uses antenna subtraction (→ smaller # of NLO subtraction terms than Catani-Seymour or FKS).

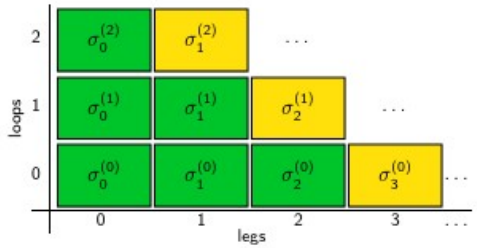
VINCIA NNLO + PS: uses the shower as phase-space generator: **extremely efficient** & everything formulated to be positive definite ⟹ **no negative weights**

➤ **About factor 5 *faster* than EERAD3** (for comparable unweighted stats) + can be hadronised, etc.



Note:

NNLO accuracy in $H \to 2j$ implies **NLO correction in first emission** and **LO correction in second emission.**



So for Thrust, NNLO $H \to b\bar{b}$ is effectively

NLO for $\tau < 1/3$

LO for $\tau > 1/3$

Expectation: VINCIA NLO MEC approach order-of-magnitude faster than anything less optimised than EERAD3

GPU plans

Both hadronic rescattering and QCD-based colour reconnections can be improved algorithmically.

QCD-CR actively being worked on with M. Kreps (Warwick): Purely code-based modifications may result in speed increases of factors 2-3.

But some of the N^2 and N^3 calculations should be possible to parallelise and be delegated to a GPU.

This would in particular be important for heavy ion collisions, where $N \sim \mathcal{O}(1000)$ are common

Conclusions

PYTHIA8 may not be an important bottleneck, but there are improvements than can be made and are being made.

Other Thoughts (from Peter Skands)

Optimisation also crucial to reduce computational footprint / environmental impact

But funders do not (currently) score on this criterion at all

E.g., ERC allow a "Do No Significant Harm" statement — but assessors told **(in boldface)** to ignore it

ARC does not even have such a statement. Not sure about other agencies… ?

Tricky choice if one has to compromise on scientific ambition? Some thoughts on this in *"Computational scientists should consider climate impacts and grant agencies should reward them"*, P Skands, Nature Rev.Phys. 5 (2023) 3, 137-138

All grants I am connected with now include minimisation of footprint as explicit goal

ARC DP22 "Tackling the **computational bottleneck** in precision particle physics" — on **sector-based approaches**

ARC DP23 "Beautiful Strings" — on more **efficient** (and better) models of heavy flavour production, fragmentation, and decays (incl matching), QED showers in hadron decays, collective effects in fragmentation, and Colour Reconnections:

POST DOC AT MONASH NOW OPEN FOR APPLICATIONS

Monash-Warwick Alliance for Particle Physics: including **optimisation and improvements** to **EVTGEN and PYTHIA for HF physics** (incl QED showers and QCD CR)

Royal Society Wolfson Visiting Fellowship "Piercing the **precision barrier** in high-energy particle physics": to develop **efficient techniques for NNLO matching** and beyond + interact with PanScales and with Warwick on **(multithreaded)** multipole **QED showers in hadron decays**

DECRA 23 [L. Scyboz, Monash]**:** "Bridging the accuracy gap: **High-precision parton showers** for colliders" — on **PanScales and VINCIA**.