

Accelerating Monte Carlo event generator analysis using efficient scalable data handling

Christian Gütschow

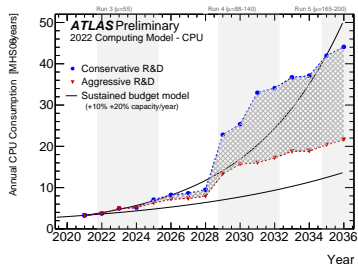
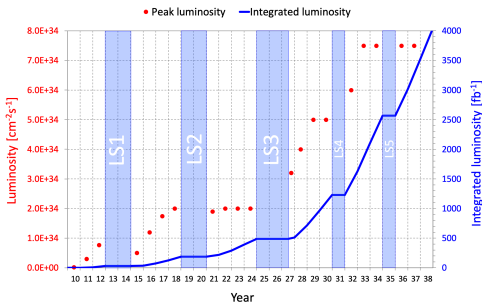
WLCG/HSF workshop, Hamburg

16 May 2024

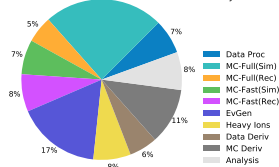


Expected computing requirements

- projected evolution of computing resources sees cost of event generation on par with detector simulation
- LHC measurements in danger of being limited by Monte Carlo statistics



ATLAS Preliminary
2022 Computing Model - CPU: 2031, Conservative R&D
Tot: 33.8 MHS06'y



[CERN-LHCC-2022-005]

Targeted optimisation of CPU-based event generation

- Most event generation CPU spent on multi-leg NLO calculations [[JHEP 08 \(2022\) 089](#)]
 - used for main Standard Model processes: extremely large event sample sizes
 - relevant to measurements and searches alike

Targeted optimisation of CPU-based event generation

- Most event generation CPU spent on multi-leg NLO calculations [[JHEP 08 \(2022\) 089](#)]
 - used for main Standard Model processes: extremely large event sample sizes
 - relevant to measurements and searches alike
- Study CPU performance of Sherpa MEPS@NLO calculations for $e^+e^- + 0, 1, 2j@NLO+3, 4, 5j@LO$ and $t\bar{t} + 0, 1j@NLO+2, 3, 4j@LO$
 - introduction of pilot run in Sherpa brings a factor 5 improvement
 - using analytic QCD loop amplitudes in the unweighting brings another factor 1.5
 - detailed write-up presented in [[EPJC 82 \(2022\) 12](#)]

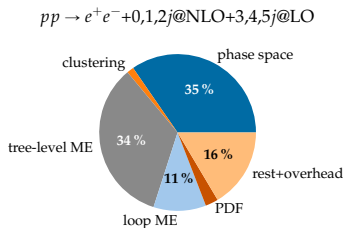
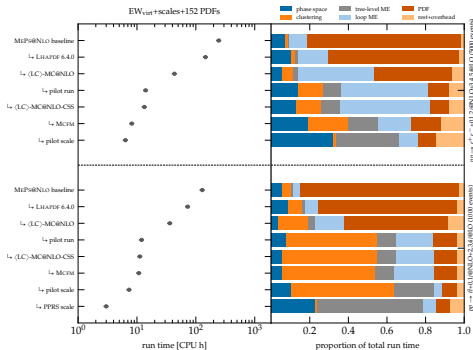
Targeted optimisation of CPU-based event generation

- Most event generation CPU spent on multi-leg NLO calculations [JHEP 08 (2022) 089]
 - used for main Standard Model processes: extremely large event sample sizes
 - relevant to measurements and searches alike
- Study CPU performance of Sherpa MEPS@NLO calculations for $e^+e^- + 0, 1, 2j@NLO+3, 4, 5j@LO$ and $t\bar{t} + 0, 1j@NLO+2, 3, 4j@LO$
 - introduction of pilot run in Sherpa brings a factor 5 improvement
 - using analytic QCD loop amplitudes in the unweighting brings another factor 1.5
 - detailed write-up presented in [EPJC 82 (2022) 12]

cumulative speed-ups for:

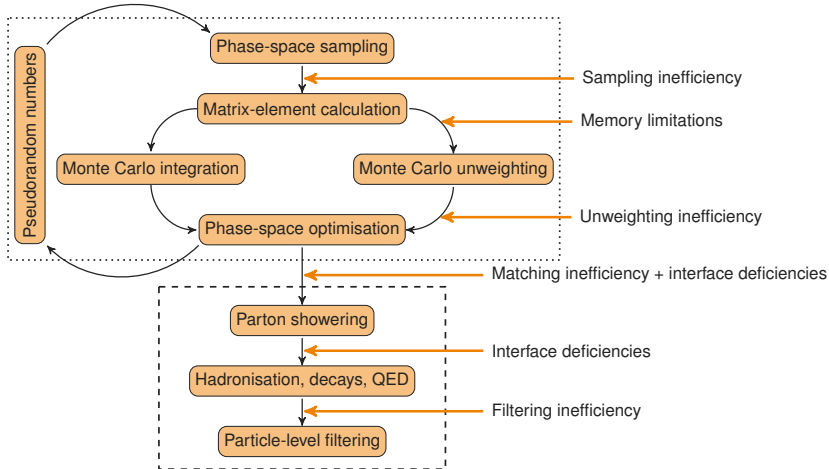
setup variant	$pp \rightarrow e^+e^- + \text{jets}$			$pp \rightarrow t\bar{t} + \text{jets}$		
	runtime [CPU h/5k events] old	runtime [CPU h/5k events] new	speed-up	runtime [CPU h/5k events] old	runtime [CPU h/5k events] new	speed-up
no variations	20 h	5 h	4×	15 h	8 h	2×
EW _{virt}	35 h	5 h	6×	20 h	8 h	2×
EW _{virt} +scales	45 h	5 h	7×	25 h	8 h	4×
EW _{virt} +scales+100 PDFs	90 h	5 h	15×	55 h	8 h	7×
EW _{virt} +scales+1000 PDFs	725 h	8 h	78×	440 h	9 h	51×

Case study: ATLAS baseline configuration



→ CPU consumption **overall improved by factors of $\times 39$ and $\times 43$** for $V+jets$ and $t\bar{t}+jets$

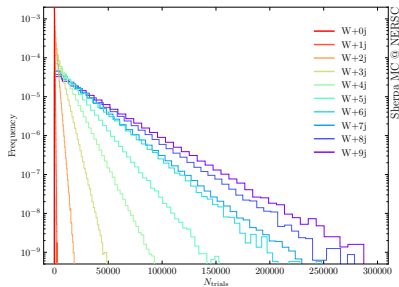
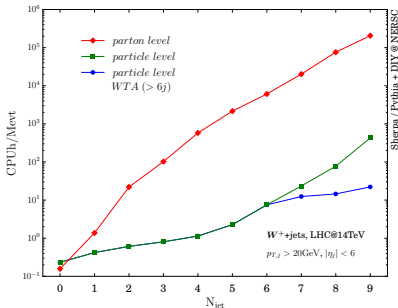
Other bottlenecks



➔ Lack of active development on infrastructure tools (LHE, HepMC, ...) set to become major bottleneck going forward (➔ upcoming [IPPP workshop] on MC support tools)

Parton vs particle level

- Scaling of parton- and particle level analysed in [PRD 100 (2019) 1]
- cost of showering matrix elements with extra emissions **dominated by parton level**
 - number of diagrams grows factorially with every additional emission (at best exponentially when exploiting recursions a la COMIX)
- low-multiplicity matrix elements cheaper to regenerate entirely than to store on disk

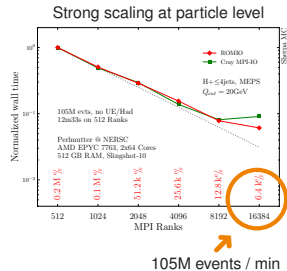
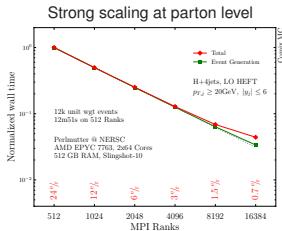
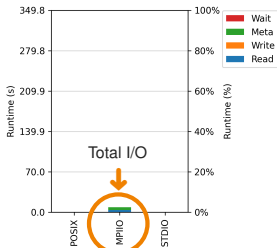


Introducing LHEH5

- established LHEF format is based on XML
 - flexible enough to add any desired feature
 - poses a challenge for I/O operations at scale
- new efficient LHE-like data format based on HDF5+HighFive proposed in [\[PRD 109 \(2024\) 1\]](#)

Name	Data type	Contents
VERSION	3 × int	Version ID
INIT	10 × double	beamA, beamB, energyA, energyB, PDFgroupA, PDFgroupB, PDFsetA, PDFsetB, weightingStrategy, numProcesses
PROCINFO	6 × double	procl, npLO, npNLO, xSection, error, unitWeight
EVENTS	9 × double	pid, nparticles, start, trials, scale, fscale, rscale, aqed, aqcd
PARTICLES	13 × double	id, status, mother1, mother2, color1, color2, px, py, pz, e, m, lifetime, spin
CTEVENTS	9 × double	ijt, kt, i, j, k, z1, z2, bbpsw, tpsw
CTPARTICLES	4 × double	px, py, pz, e

I/O performance



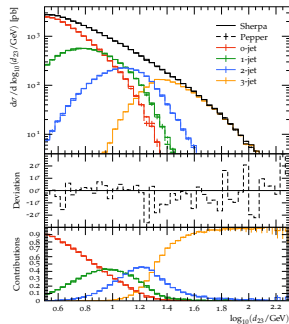
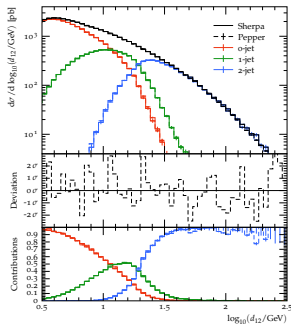
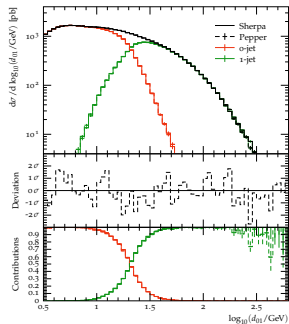
→ overall I/O time reduced to below 1s per rank

→ time spent in I/O operations less than 5% when reading 128.85 GiB

→ ideal for accessing back-fill queues at large computing centres

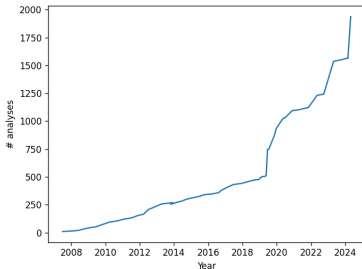
Comparison of parton-level event generators

- validated for standard candle processes (Z +jets shown) at various multiplicities
- can mix and match generators to reduced computing time to the absolute minimum required for event simulation



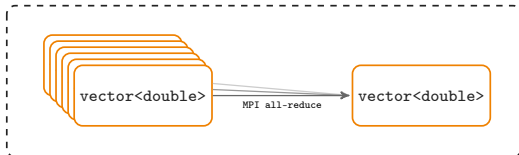
Truth-level analysis updates

- New major versions Rivet4 and YODA2 released!
[rivet.hepforge.org] [yoda.hepforge.org]
- written in C++ and programmatically usable from C++ and Python, complemented by a set of command-line tools for dataset inspection, manipulation and combination
- emerged from the sub-field of MC event generator analysis and tuning in HEP, but library deliberately agnostic of any particular application
- YODA: lightweight and general purpose library for binned statistical data analysis
 - first released in 2013, now undergone a ground-up re-design
- see also detailed write-ups for YODA [[arxiv:2312.15070](https://arxiv.org/abs/2312.15070)] and Rivet [[arxiv:2404.15984](https://arxiv.org/abs/2404.15984)] respectively
- statistical analysis objects generalised to arbitrary dimensions and edge types along different axes – with the help of modern C++ design patterns
 - Rivet now requires C++17, drops support for HepMC2 and Python2



Better support for massively parallel applications

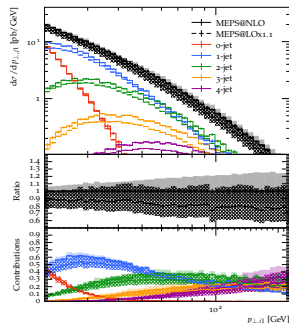
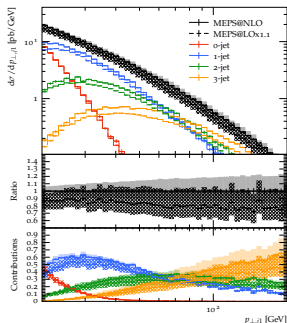
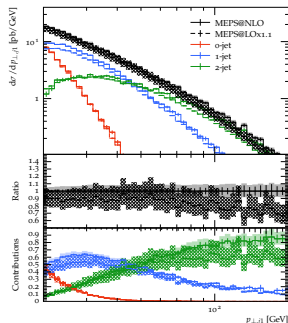
- YODA2 inheritance structure makes it straightforward to `serialize` object data
 - numerical content of `AnalysisHandler` can be translated into `std::vector<double>`
 - contiguous arrays of primitive types lend themselves much better to MPI communication
- memory block of data can be loaded back into an `AnalysisHandler` for `deserialize-ing` and `finalize-ing`



- also: reduced I/O load from parsing info files in the initialisation phase
- alternative YODA-HDF5 output format also in the works

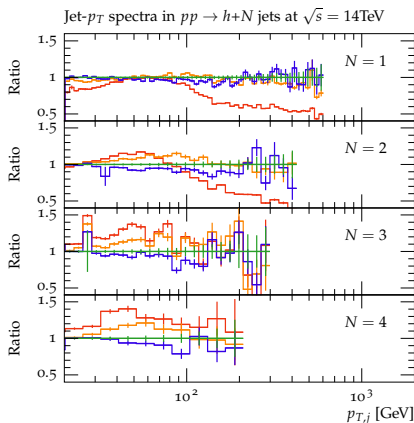
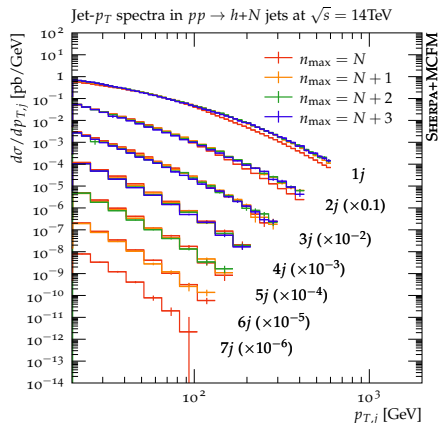
Improved modelling through high-multiplicity final states

- simulation of additional radiation at tree level clearly necessary for proper physics modelling of high-multiplicity final states
- hatched bands indicate the scale uncertainties from 7-point scale variations at LO, solid bands represent the corresponding band at NLO
- uncertainties inevitably increase with additional jet multiplicities as more of the phase space is systematically varied



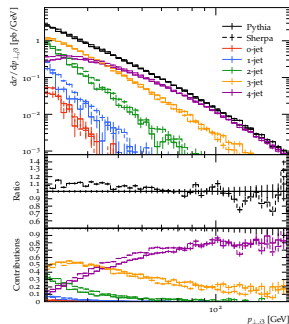
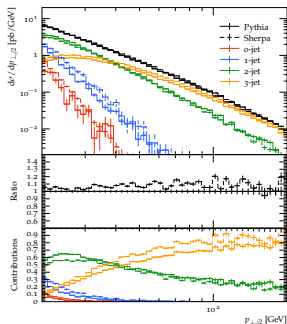
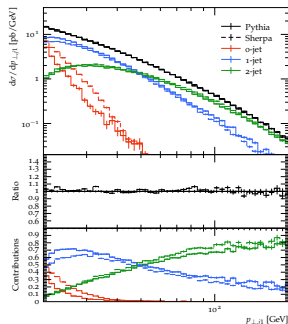
Case study: Higgs plus many jets at NLO

- $H + 0, 1, 2j@NLO+3, 4, 5, 6, 7j@LO$, ratios normalised to $n_{\max} = 2$
- maximum jet multiplicity (n_{\max}) set to the number of measured jets, N (red), $N + 1$ (green), $N + 2$ (blue) and $N + 3$ (purple)



More robust uncertainty estimates

- LHEH5 enables efficient substitution of various parts in the event generation chain
 - already supported by Pepper, Sherpa and Pythia!
- 10% uncertainty seen in Z+jets due to different algorithmic choices in the parton showers



Future event generation workflows

- Approach 1: produce parton-level samples centrally with input from the MC developers, provide them in a shared space for all experiments
 - experiments run their preferred shower setup (✓)
 - allows for affordable plug & play between different models (✓)
 - lowers cost threshold for reproducing larger setups after some time if need be (✓)
 - requires more storage for parton-level events (✗)
 - new infrastructure needs to be set up and maintained (✗)

Future event generation workflows

- Approach 1: produce parton-level samples centrally with input from the MC developers, provide them in a shared space for all experiments
 - experiments run their preferred shower setup (✓)
 - allows for affordable plug & play between different models (✓)
 - lowers cost threshold for reproducing larger setups after some time if need be (✓)
 - requires more storage for parton-level events (✗)
 - new infrastructure needs to be set up and maintained (✗)
- Approach 2: run everything in one go, harnessing heterogeneous resources, possibly with in-memory transfer of GPU-accelerated calculation components
 - no intermediate storage for parton level events needed (✓)
 - minimal infrastructure changes required (✓)
 - parton-level events continue to cost twice as strictly necessary (✗)
 - regenerating larger setups from scratch will become painful (✗)

Summary

- computing cost of traditional CPU-based multi-leg event generation significantly reduced by factor 40–80 following dedicated profiling
 - first production-ready portable LO event generator (cf. Max's talk) allows to incorporate GPU resources into high-precision simulations
- new LHEH5 format allows for efficient parton-level event generation
 - excellent I/O performance for massive MPI applications
 - additional factor 3–6 speed-up for traditional grid resources
- new major releases of Rivet and YODA to allow for more efficient validation of massively parallel applications
- new tools chain facilitates more robust uncertainty estimates of parton-shower effects
- allows better exploitation/recycling of existing resources in subsequent large-scale production campaigns
- seeing latest performance improvements reflected in up-to-date projections from the experiments paramount for defining appropriate objectives going forward

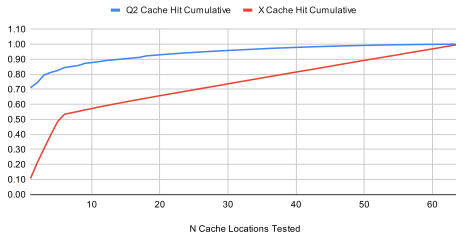
Backup

Improving LHAPDF

→ first PDF-grid cache introduced in v6.3.0

→ rendered ineffective by PDF-call strategy used in Sherpa

→ nevertheless useful as case study



→ follow-up release v6.4.0 with improved interpolation logic

→ revised cache implementation with improved memory layout (but well-matched call strategy in the generator still crucial)

→ pre-computation of shared coefficients of the interpolation polynomial along (x, Q^2) grid lines

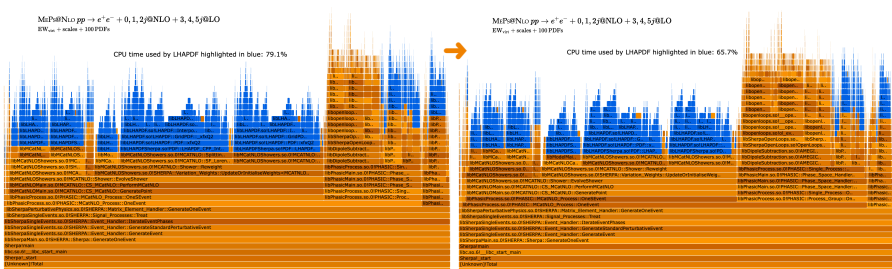
→ results in factor 3 speed-up for single flavour computations

→ can achieve factor 10 speed-up when combining with multi-flavour caching

Impact of new LHAPDF

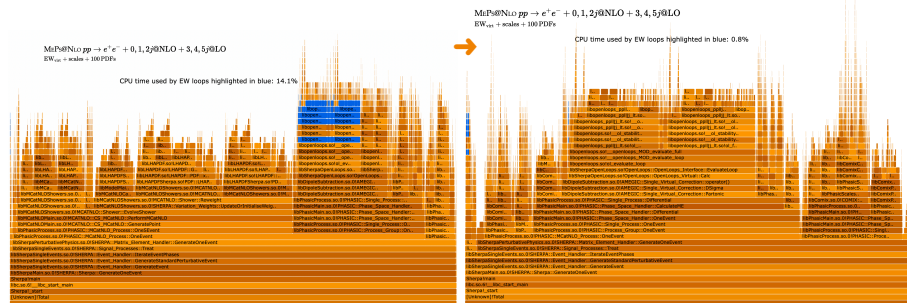
→ ATLAS $V+$ jets setup **overall 30% faster** using new LHAPDF release

→ switching from old ATLAS production default v6.2.3 to new v6.4.0 release



Internal restructuring in Sherpa 2.2.12: the pilot run

- CPU spent on calculating EW one-loop amplitudes going from 19% down to 0.8% when using the pilot run with the ATLAS V+jets setup
- nevertheless, ~40% of the CPU still spent on calculating QCD loops

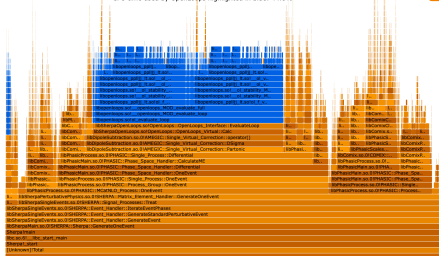


Analytic vs numerical QCD loop amplitudes

- employ analytic one-loop amplitudes (if available) in the pilot run using Sherpa-MCFM interface [EPJC 81 (2021) 12]
- yields **additional ~35% speed improvement** for the V +jets setup

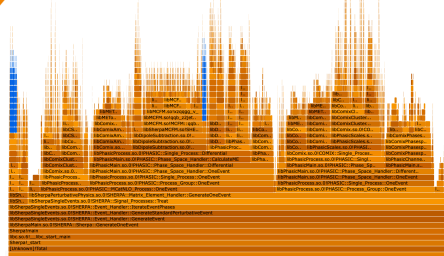
MePs@NLO $pp \rightarrow e^+e^- + 0, 1, 2j$ @NLO + 3, 4, 5j@LO
EW_{cor.} + scales + 100 PDFs

CPU time used by OpenLoops highlighted in blue: 44.8%

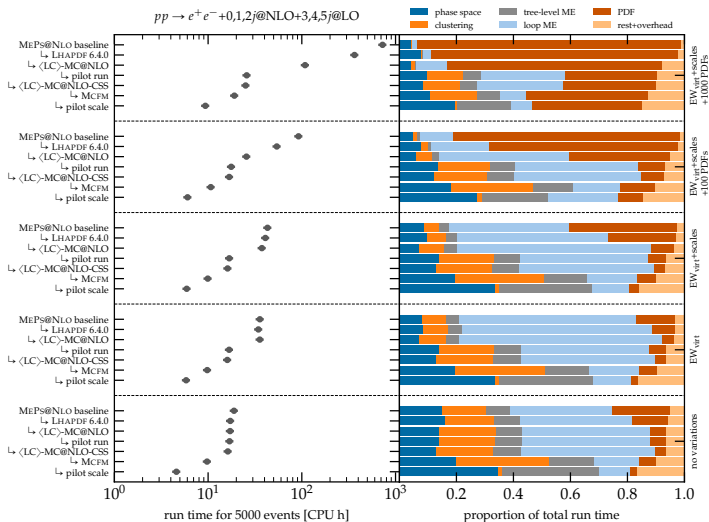


MePs@NLO $pp \rightarrow e^+e^- + 0, 1, 2j$ @NLO + 3, 4, 5j@LO
EW_{cor.} + scales + 100 PDFs

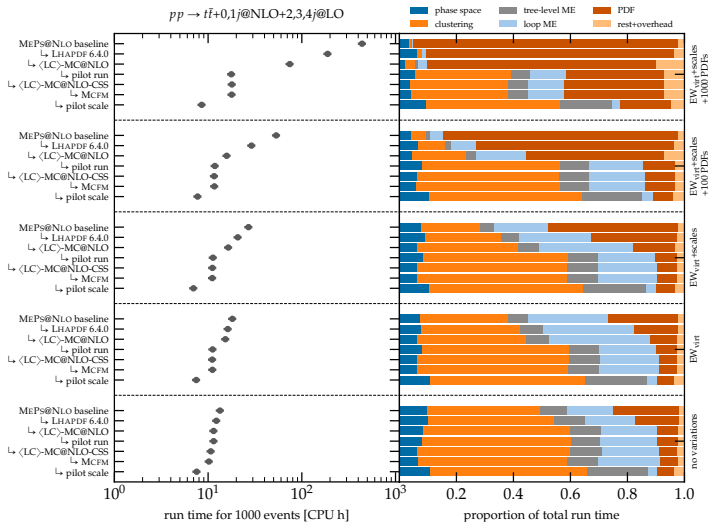
CPU time used by OpenLoops highlighted in blue: 2.5%



Breakdown of CPU budget in $V+jets$

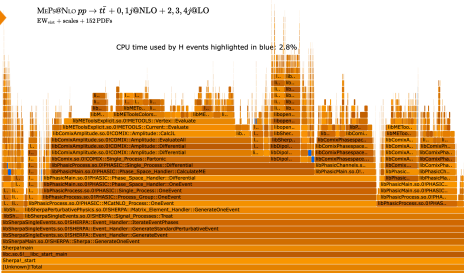
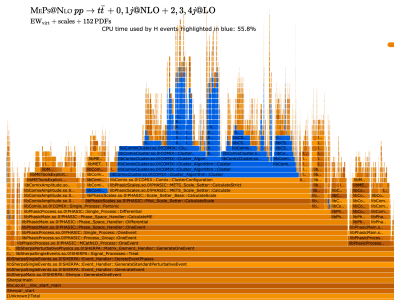


Breakdown of CPU budget in $t\bar{t}+jets$

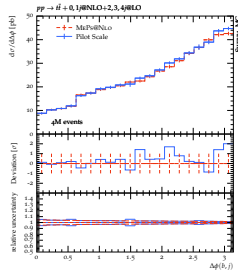
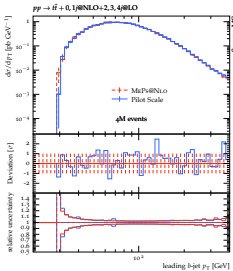
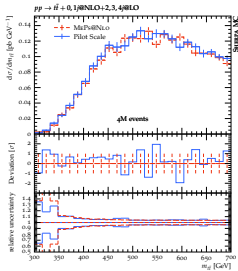
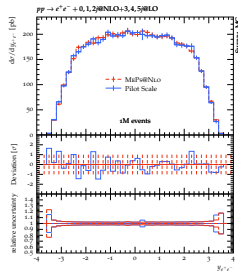
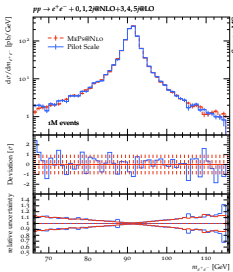
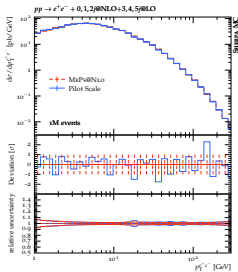


Cluster-independent scale definition

- employ clustering-independent scale definition ($H_T^c/2$) for \mathcal{H} -events in $t\bar{t}$ +jets (already used in V +jets baseline setup)
- yields **additional factor 2 speed-up** of the overall run time

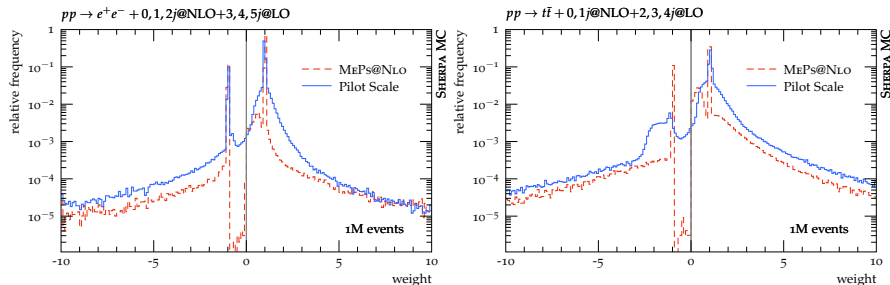


Comparison of MEPS@NLO vs Pilot Scale strategy



Weight distribution for pilot scale

→ weight distributions for partially unweighted events after matching and merging:



→ second unweighting would reduce the efficiency by less than factor 2 for large N_{events}

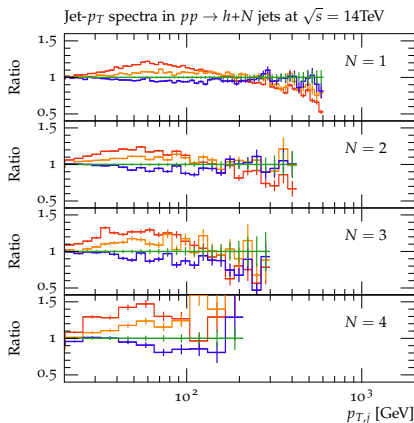
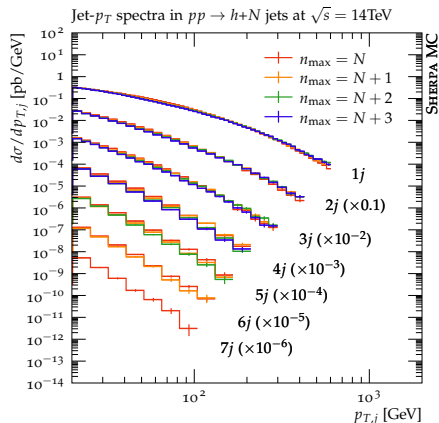
Benchmarking with state-of-the-art event generators

- comparison of Sherpa's COMIX with PEPPER+CHILI, on a single core Intel(R) Core(TM) i3-8300 CPU at 3.70GHz with 8MB L3 cache.
- samples generated with a given target for the total cross section uncertainty ("Tot. unc.")
- "Speed-up" gives the walltime gain factor of PEPPER+CHILI vs. COMIX
- PEPPER+CHILI: $Z + 0, 1j$ generated using helicity summing, while the higher ones use helicity sampling, thereby achieving the best possible performance in each case
- factorial scaling in PEPPER causes COMIX to win at very high multiplicities

Process	Tot. unc. [%]	Sherpa (Comix)			Pepper+Chili			Speed-up
		Walltime [s]	Mem. (USS) [MB]	Eff. [%]	Walltime [s]	Mem. (USS) [MB]	Eff. [%]	
Z+0j	0.089	68	62	22	10	40	43	6.8
Z+1j	0.19	76	66	5.3	31	33	10	2.5
Z+2j	0.99	92	64	0.28	10	35	1.4	9.2
Z+3j	3.8	95	65	0.037	36	43	0.097	2.6
Z+4j	14	122	115	0.0050	71	133	0.016	1.7

Case study: Higgs plus many jets at LO

- $H + 0, 1, 2, 3, 4, 5, 6, 7j@LO$, ratios normalised to $n_{\max} = 2$
- maximum jet multiplicity (n_{\max}) set to the number of measured jets, N (red), $N + 1$ (green), $N + 2$ (blue) and $N + 3$ (purple)



Plotting

- matplotlib-based plotting machinery produces **self-consistent Python scripts** allowing for better customisation of plots (no YODA installation required)

