Monte-Carlo event generation
–
Tools and challenges

Marek Schönherr

IPPP, Durham University

Overview

1. Monte-Carlo event generators for the LHC
2. Physics challenges in the immediate future
3. Coding Challenges
4. Conclusions

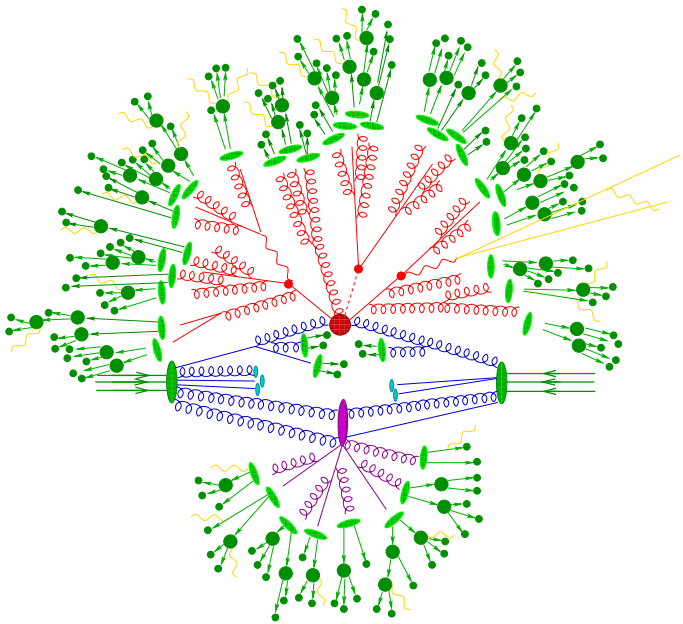
Monte-Carlo event generators for the LHC

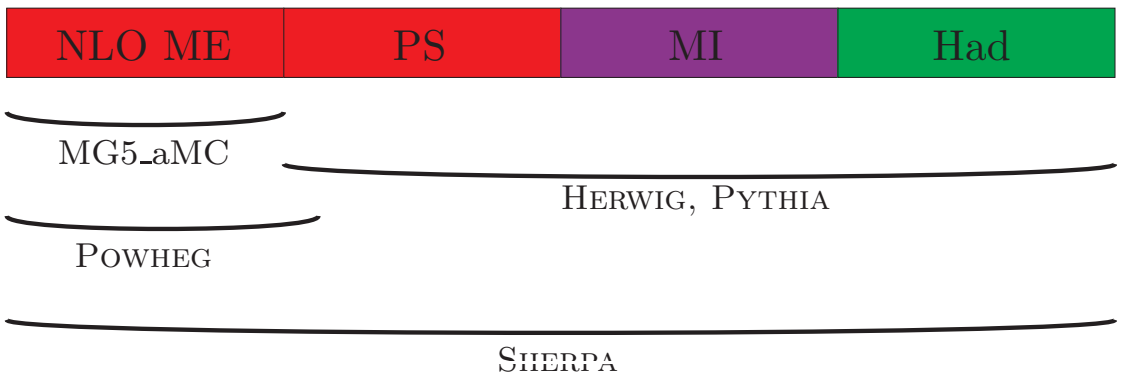
1. Monte-Carlo event generators for the LHC
2. Physics challenges in the immediate future
3. Coding Challenges
4. Conclusions

Monte-Carlo event generators for the LHC

Modern event generation comprises:

- NLO QCD ME
  (sometimes EW corr.)
- matched to PS
- multiple interactions
- hadronisation, hadron decays





+ dedicated codes for loop MEs, specific hadron decays, etc.

Higher precision – better physics description

Physics advances:

- general NNLOPS matching and multijet merging
  specialised solutions exist for singlet production
  - NNLOPS based on MINLO
  - NNLOPS based on UNLOPS
  - GENEVA

  though, strictly speaking, PS radiation pattern not accurate enough,
  methods fix it enough not to spoil inclusive fixed-order accuracy
- increased parton shower accuracy
  - re-assessment of standard LO shower accuracy
  - inclusion of NLO splitting functions
  - inclusion of beyond-leading-colour effects
- inclusion of precision EW effects

not discussed here, many specialised talks in many sessions

Higher precision – better physics description

Physics advances:

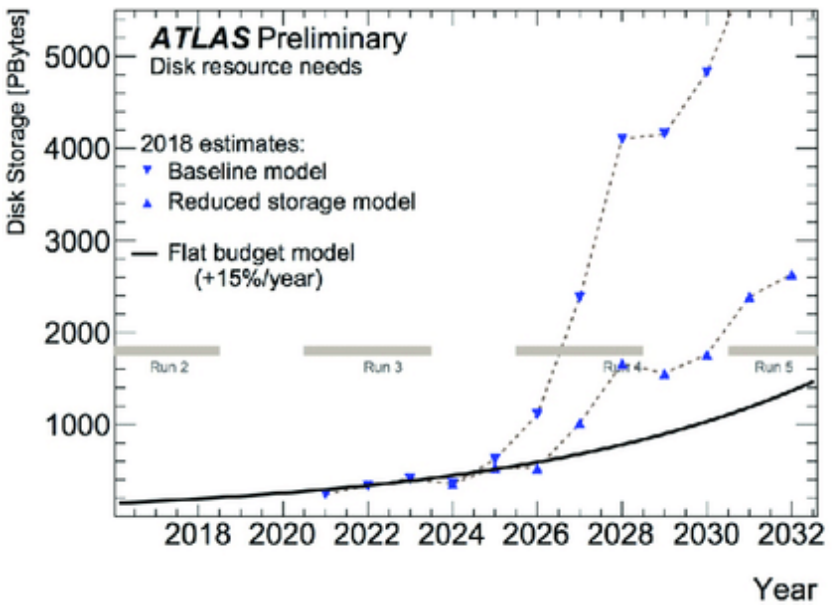
- general NNLOPS matching and multijet merging
  specialised solutions exist for singlet production
  - NNLOPS based on MINLO
  - NNLOPS based on UNLOPS
  - GENEVA

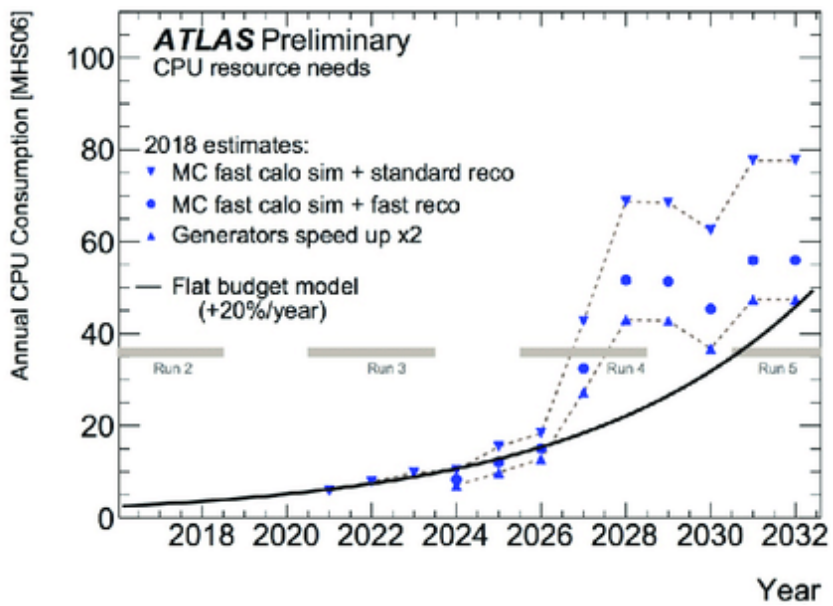
  though, strictly speaking, PS radiation pattern not accurate enough, methods fix it enough not to spoil inclusive fixed-order accuracy
- increased parton shower accuracy
  - re-assessment of standard LO shower accuracy
  - inclusion of NLO splitting functions
  - inclusion of beyond-leading-colour effects
- inclusion of precision EW effects

  not discussed here, many specialised talks in many sessions

Monte-Carlo event generators for the LHC

ATLAS projection





Both CPU and disk resources will be a problem for the LHC collabs.

Need to be addressed to ensure detailed theory predictions exist to interpret data (meaning)fully.

Problem: Not a traditional theoretical physics problem.

Physics Challenges

1. Monte-Carlo event generators for the LHC
2. Physics challenges in the immediate future
3. Coding Challenges
4. Conclusions

Unweighted events

Unweighted events represent the *optimal statistical distribution* (all weights are equal, variance is minimal), giving the sample with the largest statistical power with the smallest sample size.

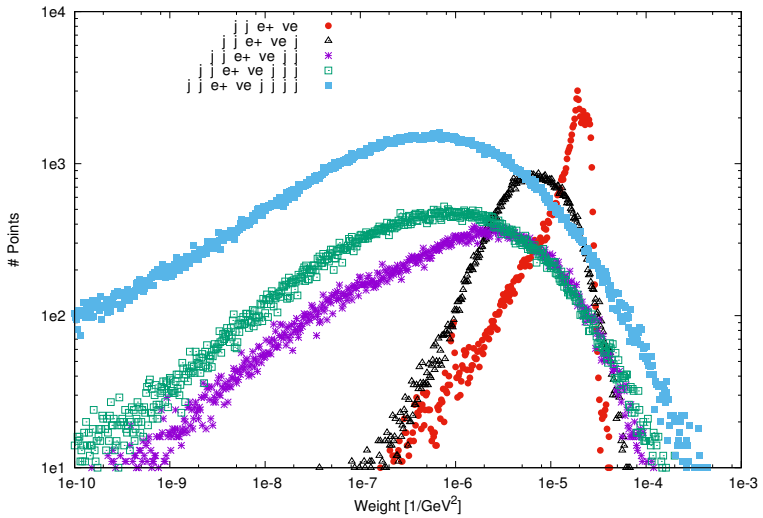
Higher-order calculations invariably introduce **negative weights** due to the organisation of the perturbative series.

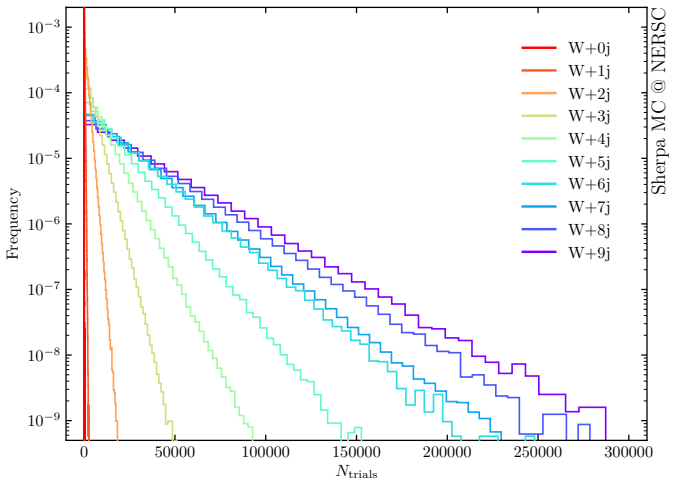
Limited computing resources demand improvements in the efficiency with which unweighted events are produced and the fraction of negative weights in a sample.

Unweighting efficiency

- hit-and-miss of weight of ME config vs. maximum
- the larger distance of mean to the maximum the smaller unw. efficiency

⚠ small variance $\neq$ large unw. efficiency





- width of distribution governed by how well we can sample the integrand
- typically: the more complex the process, the worse the unweighting efficiency **and** the more computationally intensive each trial point

Unweighting efficiency

Looks cut out for Machine Learning

- tradionally VEGAS (integrand is facorised as $1 \times 1 \times \ldots \times 1$), problems if structures of integrand do not align with int. variables
- train *N*-dim. NN to give importance sampling map
- promising on toy examples and simple cases

Bendavid arXiv:1707.00028
Klimek, Perelstein arXiv:1810.11509
Otten et.al arXiv:1901.00875
...

Algorithm	# of Func. Evals	$\sigma_w / < w >$	σ_I / I (2e6 add. evts)
VEGAS	1,500,000	19	$\pm 1.3 \times 10^{-2}$
Generative BDT	3,200,000	0.63	$\pm 4.5 \times 10^{-4}$
Generative BDT (staged)	3,200,000	0.31	$\pm 2.2 \times 10^{-4}$
Generative DNN	294,912	0.15	$\pm 1.1 \times 10^{-4}$
Generative DNN (staged)	294,912	0.081	$\pm 5.7 \times 10^{-5}$

performance on sampling a 9D camel function

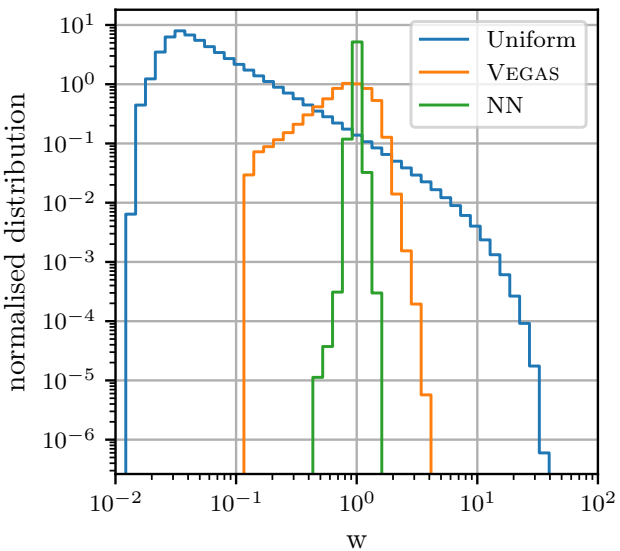
⚠ finite training sample, boundaries in activation functions, etc, may introduce phase space boundaries or unphysical extrapolation beyond traning region

Unweighting efficiency

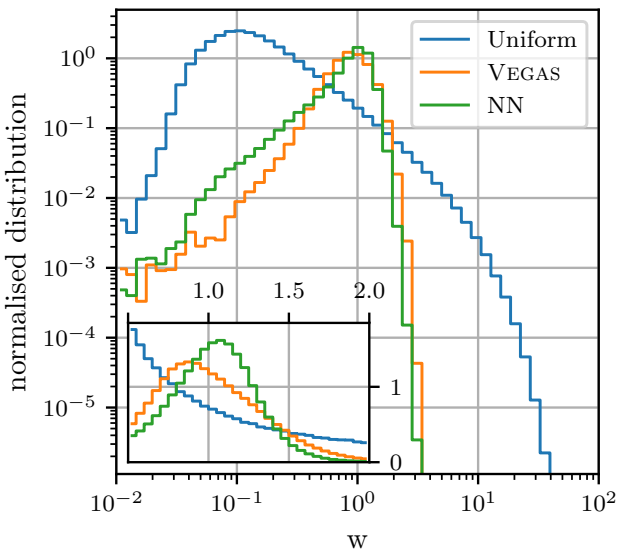
Realistic use case:

- realistic implementation in working event generator

Bothmann et.al arXiv:2001.05478
Gao et.al arXiv:2001.10028



$gg \to 3g$



$gg \to 4g$

improvements only for cases that need no improvements,
worse than VEGAS for bottleneck cases not smart enough yet?

Negative weights

Frederix et.al. arXiv:2002.12716

In aMC@NLO negative weights are encountered in

- $\mathbb{S}$-events
  when MC estimate of integral over shower emission is accidentally negative
  $\Rightarrow$ folding, evaluate MC integral with more points
- $\mathbb{H}$-events
  when shower overestimates exact emission probility
  $\Rightarrow$ MC@NLO-Δ,
  additional Sudakov suppr.

MG5_aMC	MC@NLO		MC@NLO-Δ	
	111	411	111	411
$pp \to e^+e^-$	6.9%	3.2%	5.7%	2.0%
$pp \to Hb\bar{b}$	40.3%	38.0%	36.6%	31.3%
$pp \to t\bar{t}$	23.0%	19.6%	13.6%	7.7%

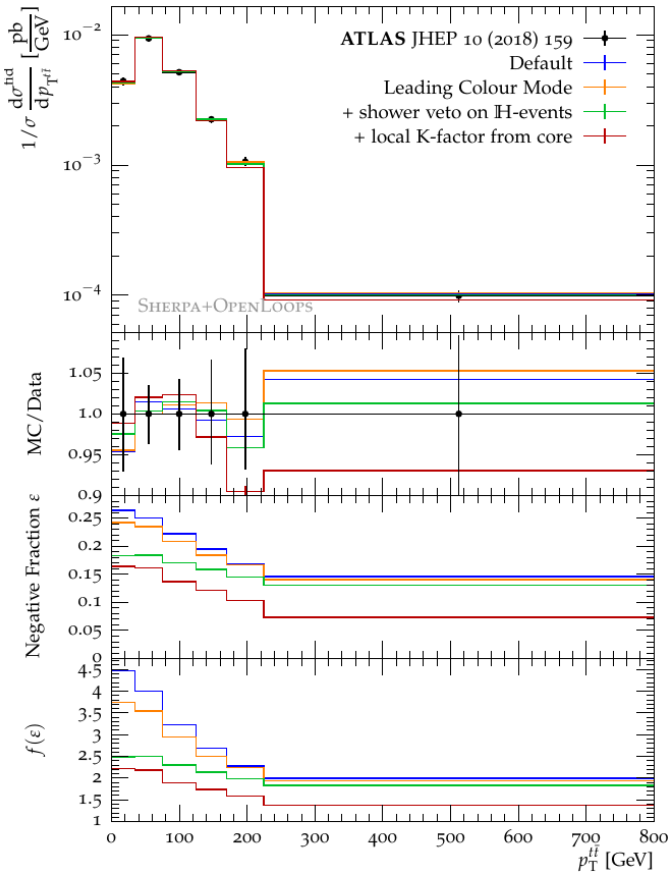
Negative weights

Danziger CERN-THESIS-2020-024

Neg. weights in SHERPA MEPS@NLO

- origins same as in aMC@NLO, plus $N_c = 3$ matched PS emission and local K-factor for higher LO ME
- use standard $N_c \to \infty$ PS
- introduce Sudakov suppr. for $\mathbb{H}$ events as in Hoeche et.al. arXiv:1207.5030
- calculate local K-factor for $t\bar{t}jj(j)$ on $t\bar{t}$ instead of $t\bar{t}j$

$pp \to t\bar{t} + 0, 1j$@NLO $+ 2, 3j$@LO	Negative Weight Fraction
Default	24.8%
Leading Colour Mode	18.7%
+ shower veto on $\mathbb{H}$-events	14.5%
+ local K-factor from core	12.6%



Negative weights

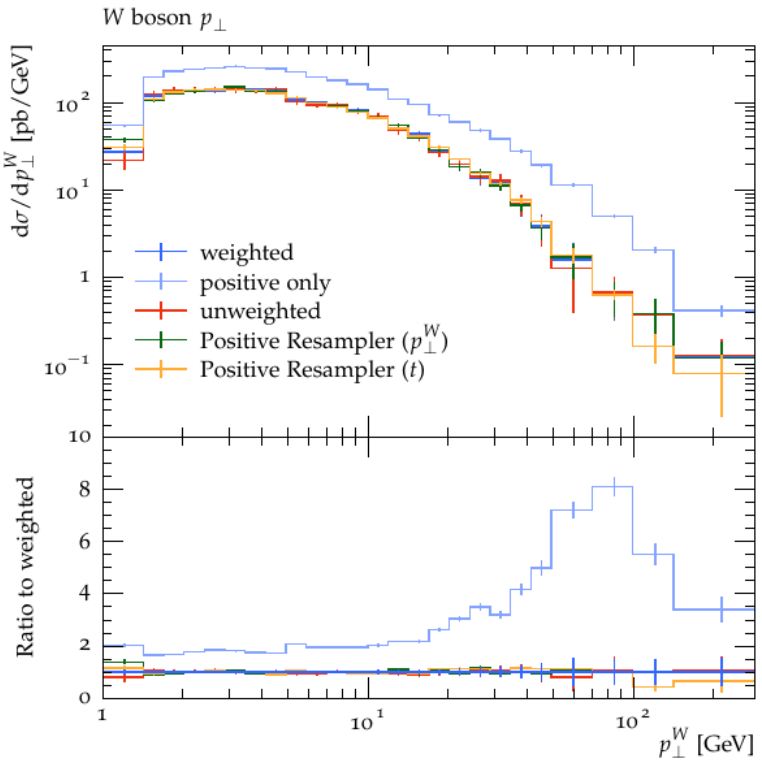
Andersen et.al. arXiv:2005.09375

A posteriori resampling

- replace weight w_i with $P|w_i|$ where $P = \sum w_i / \sum |w_i|$ in a given observable (bin)

$$\sigma = \sum_i w_i = \sum_i P|w_i|$$

- transfers to other observables or subdivided bins only if P is phase space independent



Biased phase space generation and hadronisation

Unweighted events represent the *optimal statistical distribution* (all weights are equal, variance is minimal), giving the sample with the largest statistical power with the smallest sample size.

True for inclusive observables, but not for exclusive ones where only part of the sample is used.

Can introduce biases to overpopulate regions of (statically) overproportional interest and correct weight appropriately.

Resulting **weight distribution** *degrades overall statistical power* of inclusive sample, but *enhances that of the region* in question.

Need-cases for one-size-fits-all inclusive sample needs to be estimated before-hand and impact of biases needs to be carefully assessed.

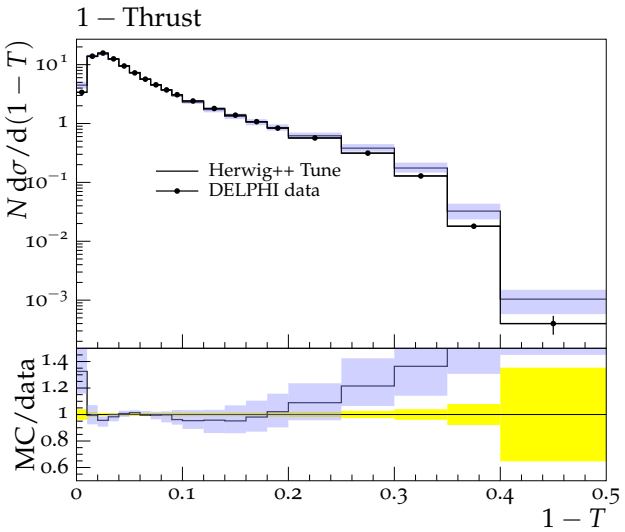
Uncertainty of phenomenological models

Intrinsic uncertainty:

- needs to be assessed and factored into global uncertainty budget
- in principle known how to do, just not how to do efficiently, same problem as for non-perturbative part of PDFs, only larger and more varied parameter space

Eigentunes:

- construct χ^2 of tune, vary along eigenvectors
- available from PROFESSOR
- could extract a lot of additional information about model and observable
- necessitates **separate run for every eigentune**, impractical



Richardson, Winn arXiv:1207.0380

Coding Challenges

1 Monte-Carlo event generators for the LHC

2 Physics challenges in the immediate future

3 Coding Challenges

4 Conclusions

Design phases & prevalent design paradigms

- 3000 BC – 2000 one monolithic fortran file with a handful of common blocks
  $\rightarrow$ limited capabilities, no maintainability, no user customisability
- 2000-2020 object orientated code design, typically in C++
  $\rightarrow$ vastly extended capabilities, multi-author maintenance, highly customisable
  - one code that can calculate all processes using the same methods, code-generating code
  - dynamic library loading (on demand)
  - plenty of casting operations, virtual table look-ups
  - dynamic mem. allocation: object sizes not clear at compile time
  - euphemistic: we understand the problem, but are too stupid to explain it in simple terms (even a computer easily understands)
  - designed and programmed by physicists
- 2020+ back to monolithic structure for comp. intensive parts of the code? predictable program flow? process specific programs?

Design phases & prevalent design paradigms

- 3000 BC – 2000 one monolithic fortran file with a handful of common blocks
  $\rightarrow$ limited capabilities, no maintainability, no user customisability
- 2000-2020 object orientated code design, typically in C++
  $\rightarrow$ vastly extended capabilities, multi-author maintenance, highly customisable
  - one code that can calculate all processes using the same methods, code-generating code
  - dynamic library loading (on demand)
  - plenty of casting operations, virtual table look-ups
  - dynamic mem. allocation: object sizes not clear at compile time
  - euphemistic: we understand the problem, but are too stupid to explain it in simple terms (even a computer easily understands)
  - designed and programmed by physicists
- 2020+ back to monolithic structure for comp. intensive parts of the code? predictable program flow? process specific programs?

Design phases & prevalent design paradigms

- 3000 BC – 2000 one monolithic fortran file with a handful of common blocks
  → limited capabilities, no maintainability, no user customisability
- 2000-2020 object orientated code design, typically in C++
  → vastly extended capabilities, multi-author maintenance, highly customisable
  - one code that can calculate all processes using the same methods, code-generating code
  - dynamic library loading (on demand)
  - plenty of casting operations, virtual table look-ups
  - dynamic mem. allocation: object sizes not clear at compile time
  - euphemistic: we understand the problem, but are too stupid to explain it in simple terms (even a computer easily understands)
  - designed and programmed by physicists
- 2020+ back to monolithic structure for comp. intensive parts of the code? predictable program flow? process specific programs?

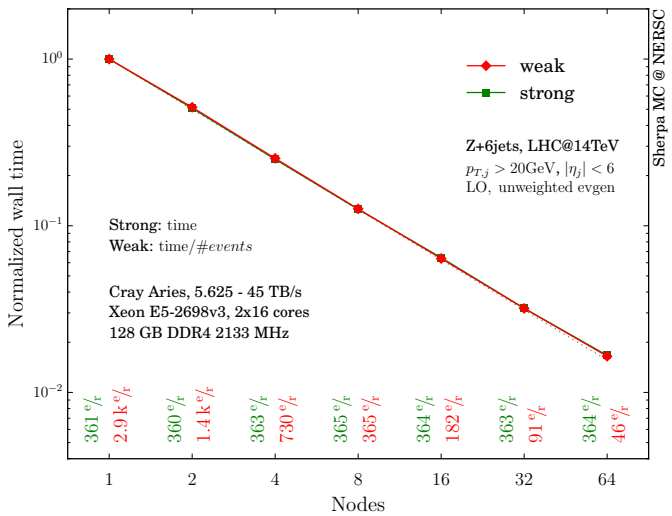
Adapting to modern HPC architectures

Generators built to $> 15a$ ago, to computing paradigms of the time:

- POWHEG, MG5_aMC .. FORTRAN
- HERWIG, PYTHIA, SHERPA .. C++

most applications are memory bound, not suitable for architectures of modern HPCs

- some tools support MPI parallelisation, at least for part of the comp.
- efforts to put MEs on GPUs exist



On the other hand, if memory requirements are met, Monte-Carlo event generation is embarrassingly parallel.

Conclusions

- modern event generators are mature tools ready to be used for precision physics at the LHC and beyond
- vast amounts of data and limited computing budgets puts pressure on theory community to move beyond their comfort zone
  - specific physics knowledge needs to be applied to make the algorithms better, approximations converge faster, etc.
  - specific computing knowledge needs to be applied to make the codes more efficient, adapt to new paradigms, etc.
- vast amounts of data and limited computing budgets puts pressure on experimental community to move beyond their comfort zone
  - compromise accuracy vs. speed will have to be made
  - compromises in storage format (which details can be stored) will have to be made
- drive for higher precision calculations stronger than ever, but computing demands increase exponentially

Thank you!