Computing challenges
for
Monte-Carlo event generators

Marek Schönherr

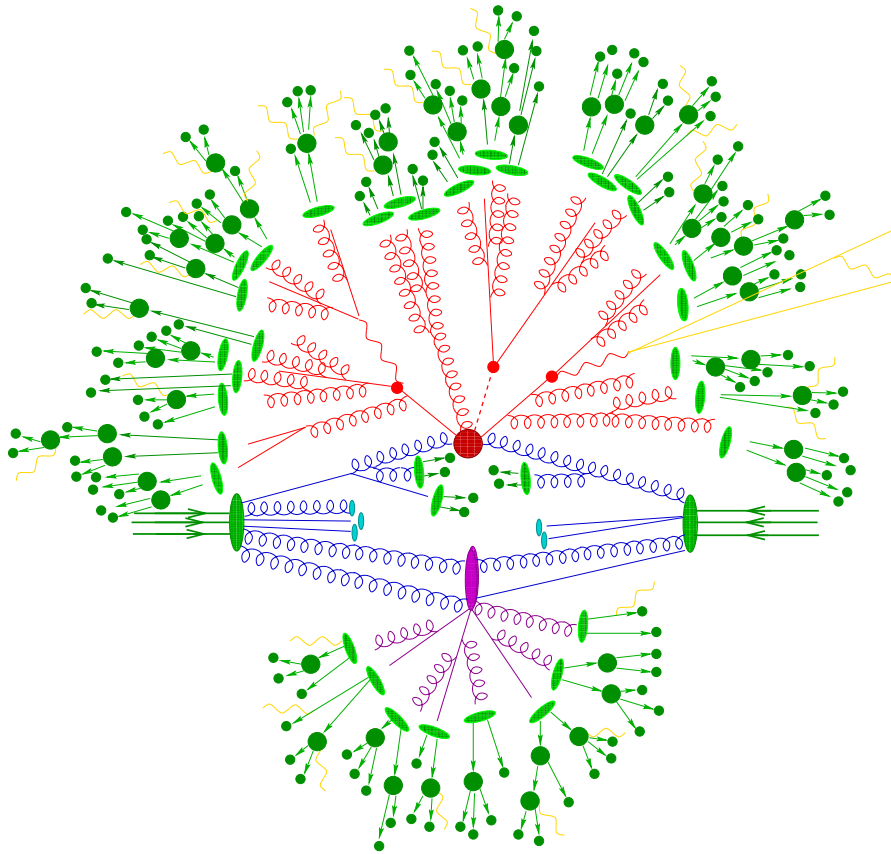
IPPP, Durham University

Content

General event generator frameworks

General event generator frameworks

- hard scattering
  - (mostly) constant program flow
  - evaluation of very large expressions
  - large memory requirements
    few GB for e.g. $V + 2j$ @ NLO
  - challenge: generate good weight distribution in finite time



- parton showers
  multiple interactions
  hadronisation
  hadron decays
  - mostly Markov Chain techniques
  - highly variable program flow
  - generally small expressions
  - low memory requirements,
    largest are typically the hadron decay tables

Matrix elements

- ideal weight distribution: uniform weights
- when all events contribute the same to any given observable, the fewest events are needed to reach a given statistical prowess of a sample
  $\Rightarrow$ **unweighted events**

Parton showers and non-perturbative event phases

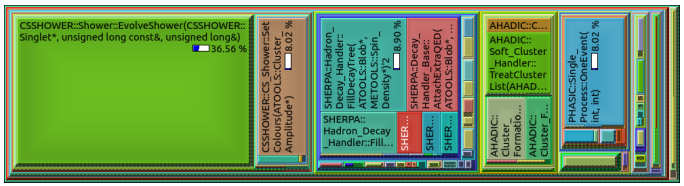
- generally set up to generate unweighted distributions through probabilistic implementations

Timings

- Example: V + jets @ LO and NLO in SHERPA

Weighted W+0..≤4j@LO

0j

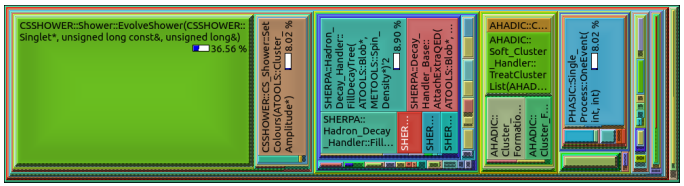


Unweighted W+0..≤4j@LO

Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j

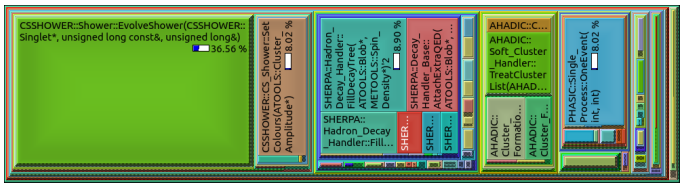


- matrix elements

Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



parton shower (incl. MI)

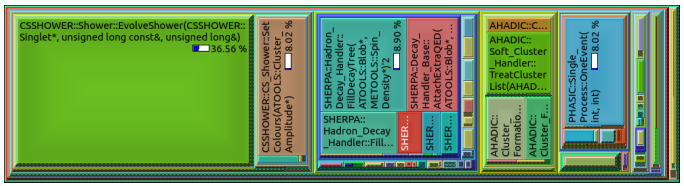
-

matrix elements

Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



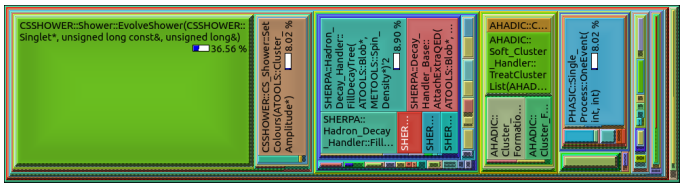
parton shower (incl. MI)

matrix elements mult. interactions

Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



parton shower (incl. MI)

hadronisation

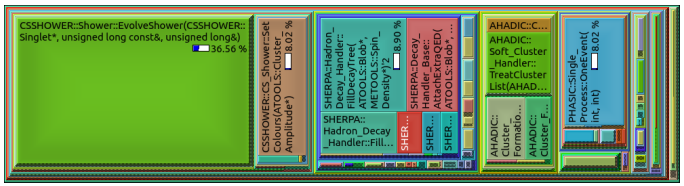
mult. interactions

matrix elements

Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



parton shower (incl. MI)

hadron decays and QED corrs.

hadronisation

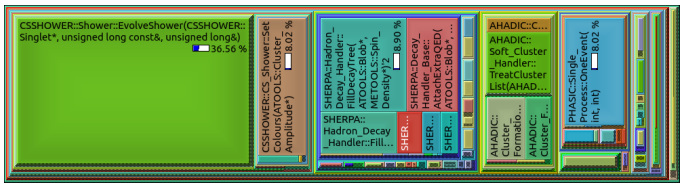
mult. interactions

matrix elements

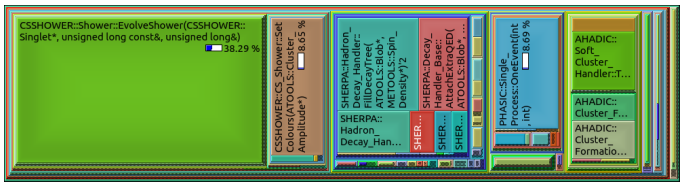
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



1j

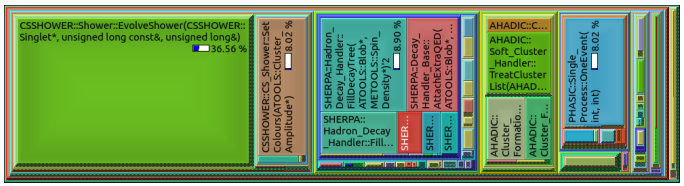


- matrix elements

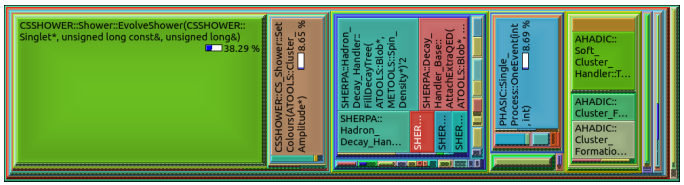
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



1j

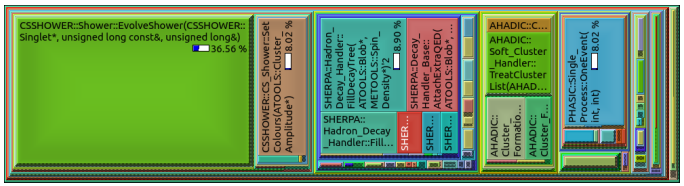


parton shower (incl. MI) - matrix elements

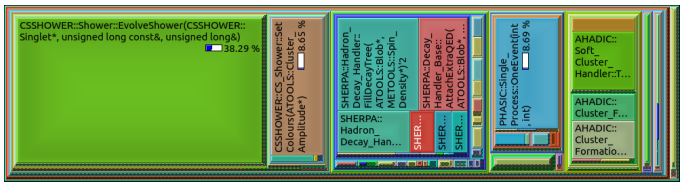
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



1j



parton shower (incl. MI)

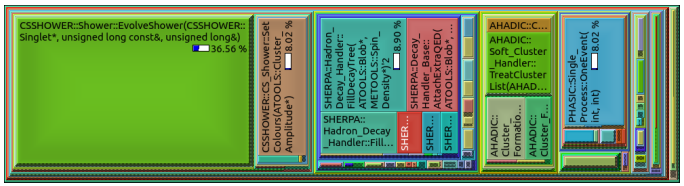
mult. interactions

matrix elements

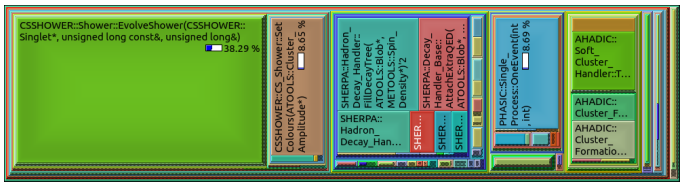
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



1j



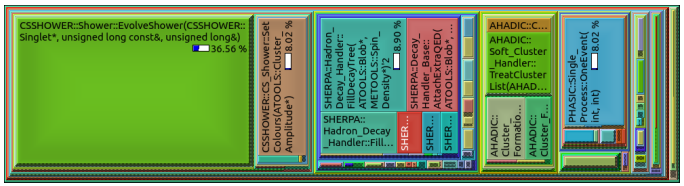
parton shower (incl. MI)

matrix elements
hadronisation
mult. interactions

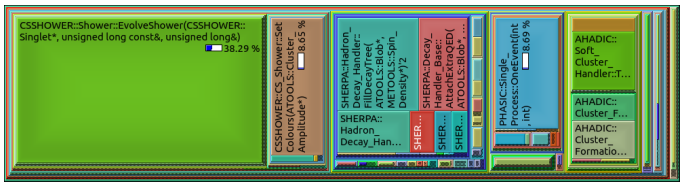
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

0j



1j



parton shower (incl. MI)

hadron decays and QED corrs.

mult. interactions

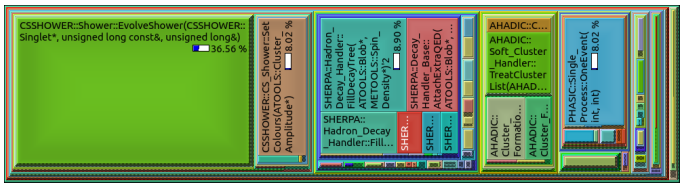
hadronisation

matrix elements

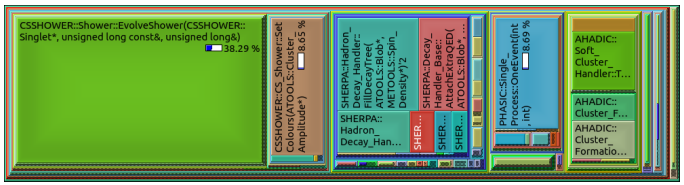
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

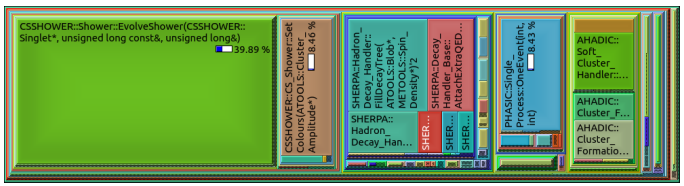
0j



1j



2j



parton shower (incl. MI)

hadron decays and QED corrs.

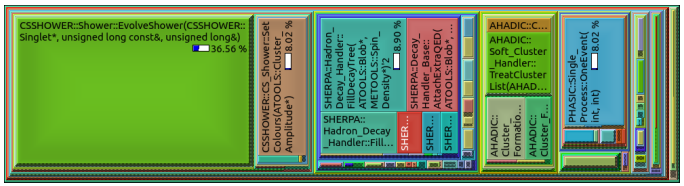
mult. interactions

hadronisation

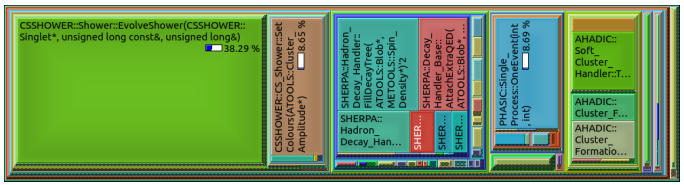
matrix elements

Weighted W+0..≤4j@LO

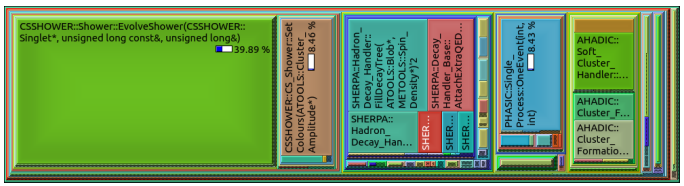
0j



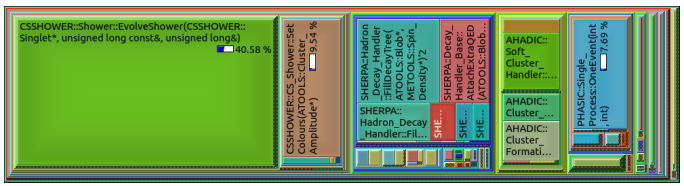
1j



2j



3j

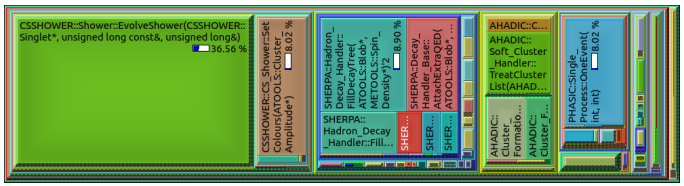


Unweighted W+0..≤4j@LO

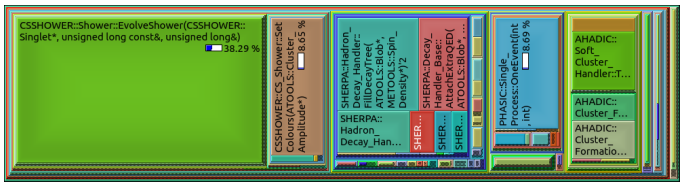
Weighted W+0..≤4j@LO

Unweighted W+0..≤4j@LO

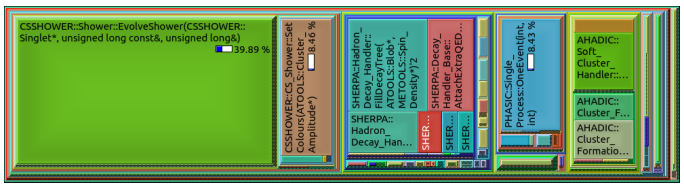
0j



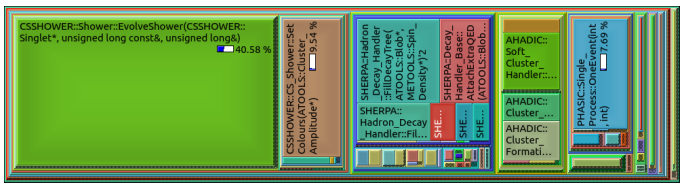
1j



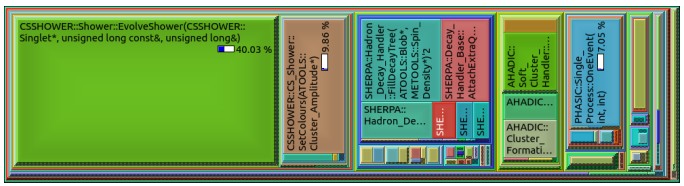
2j



3j



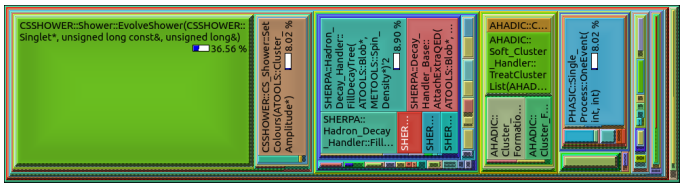
4j



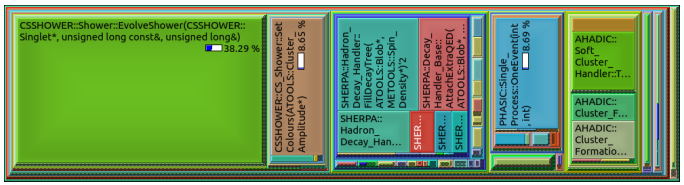
– matrix elements

Weighted W+0..≤4j@LO

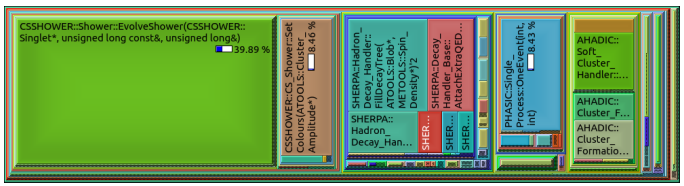
0j



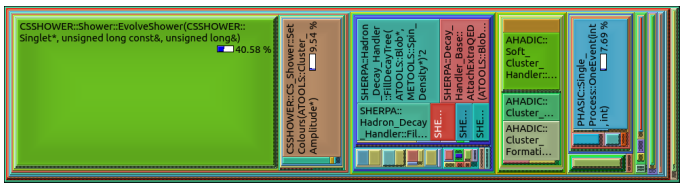
1j



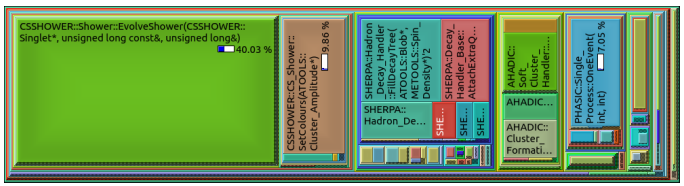
2j



3j

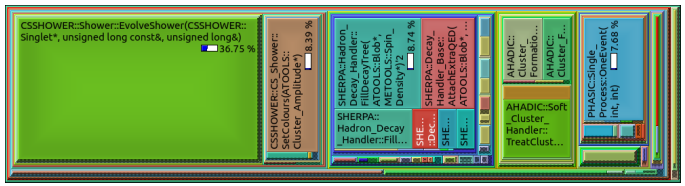


4j



Unweighted W+0..≤4j@LO

0j



parton shower (incl. MI)

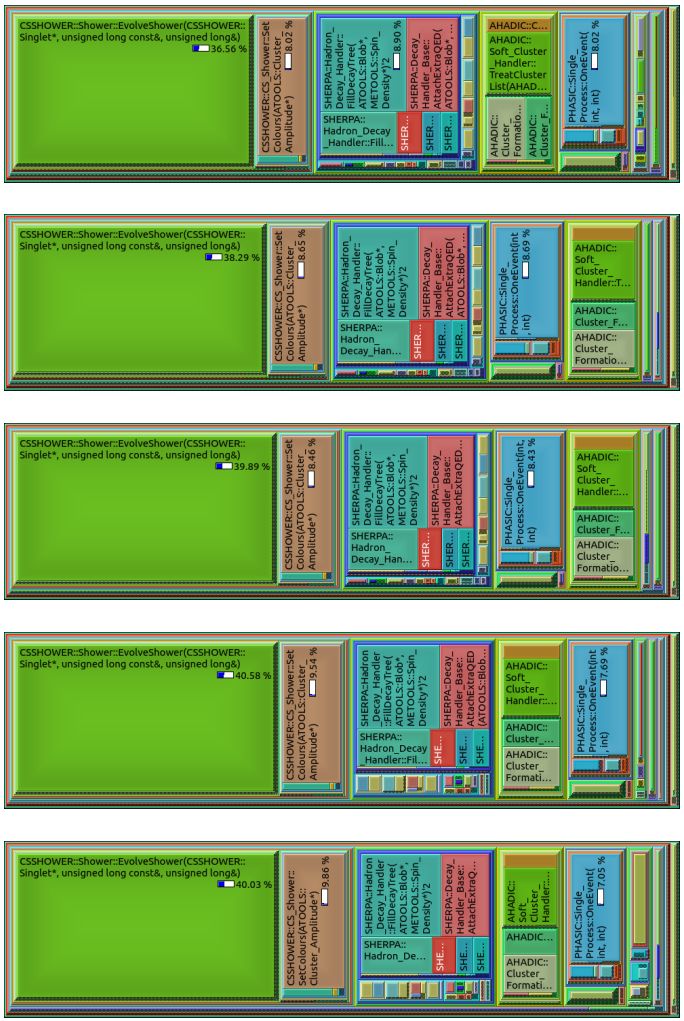
hadron decays and QED corrs.

hadronisation

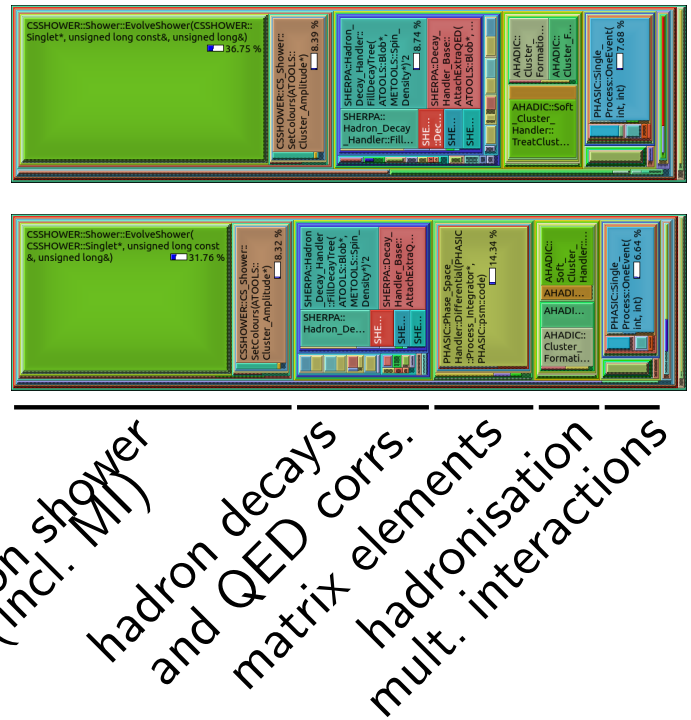
mult. interactions

matrix elements

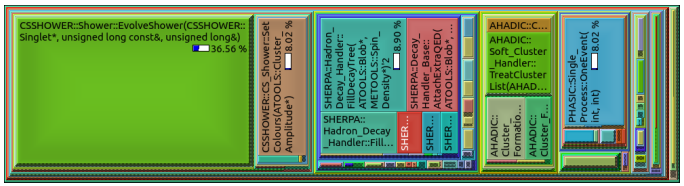
Weighted W+0..≤4j@LO

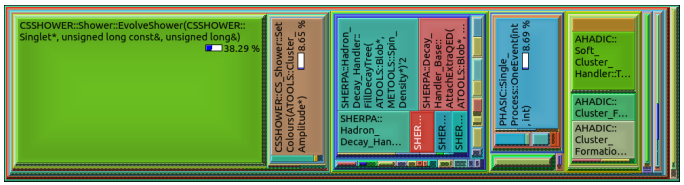


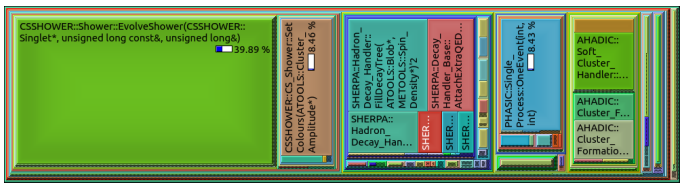
Unweighted W+0..≤4j@LO

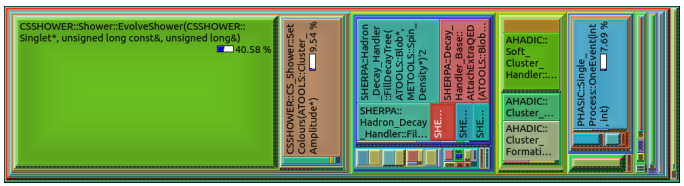


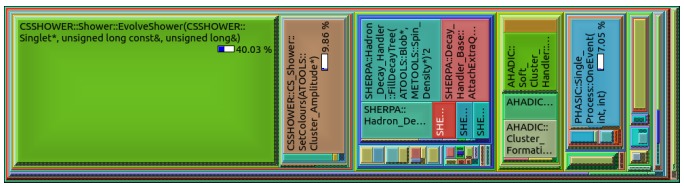
Weighted W+0..≤4j@LO

0j


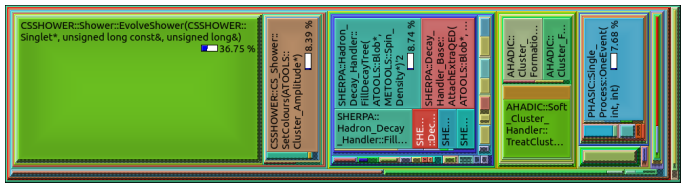
1j


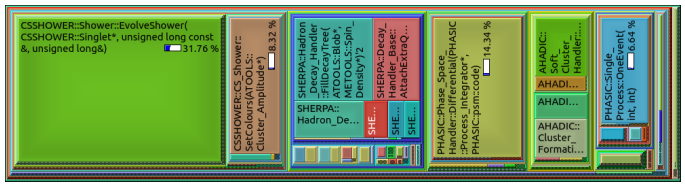
2j


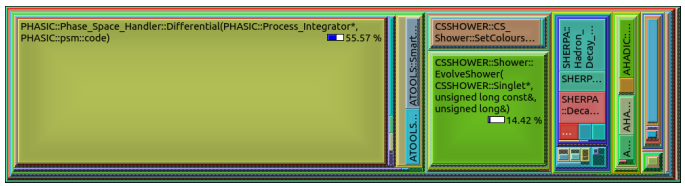
3j


4j


Unweighted W+0..≤4j@LO

0j


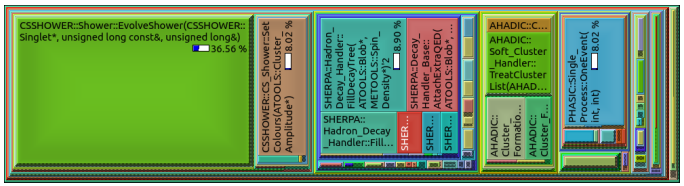
1j


2j


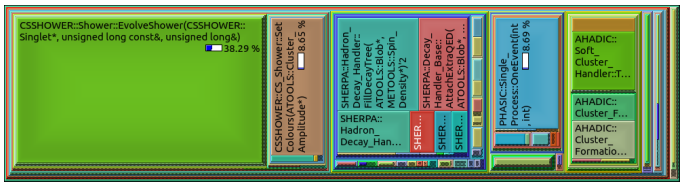
matrix elements
dominated by CKKW clustering

Weighted W+0..≤4j@LO

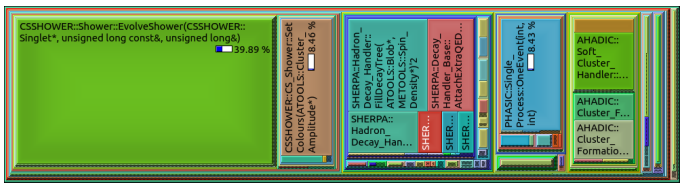
0j



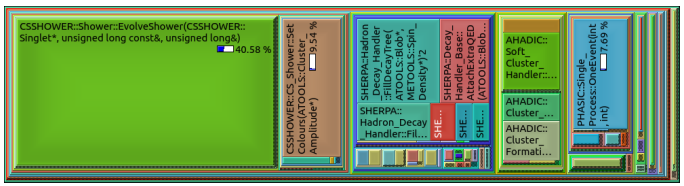
1j



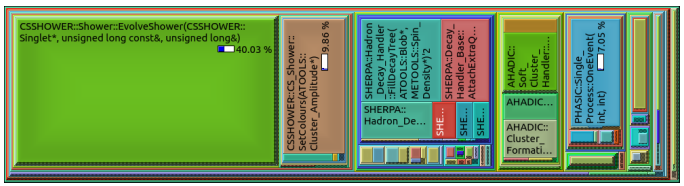
2j



3j

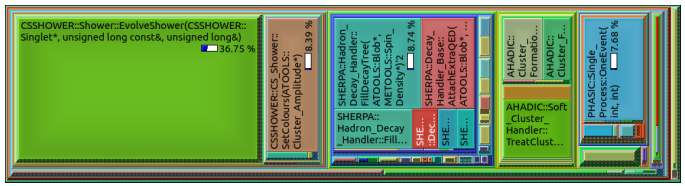


4j

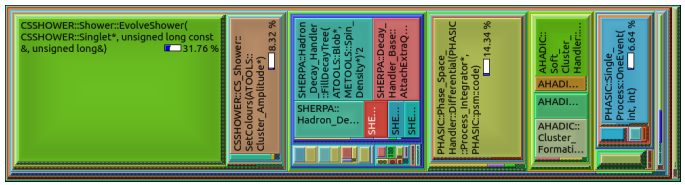


Unweighted W+0..≤4j@LO

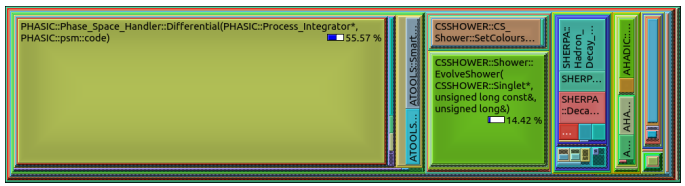
0j



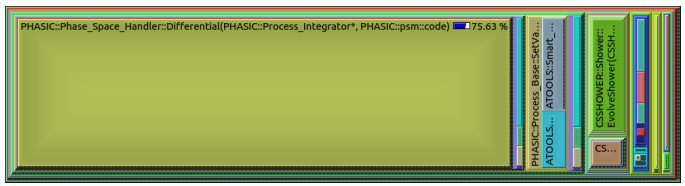
1j



2j

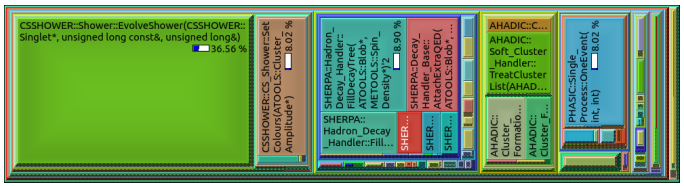


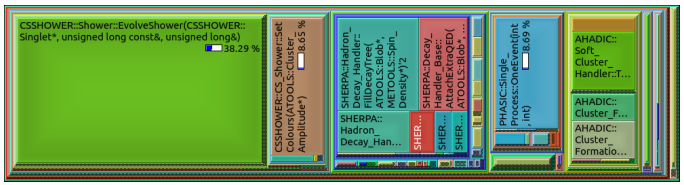
3j

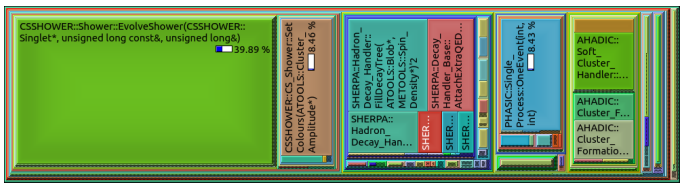


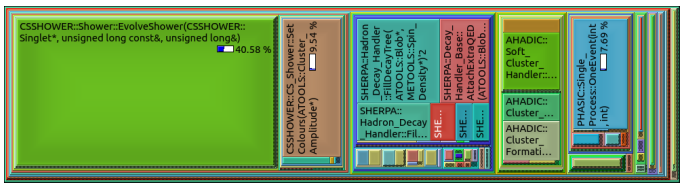
matrix elements
dominated by CKKW clustering

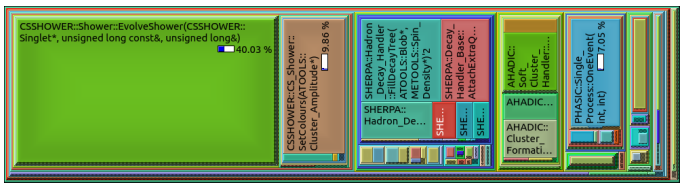
Weighted W+0..≤4j@LO

0j


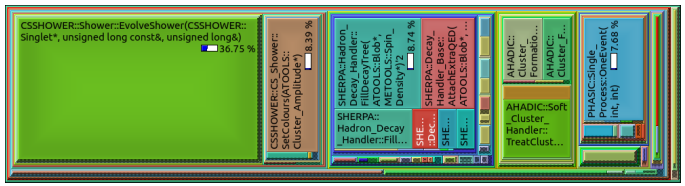
1j


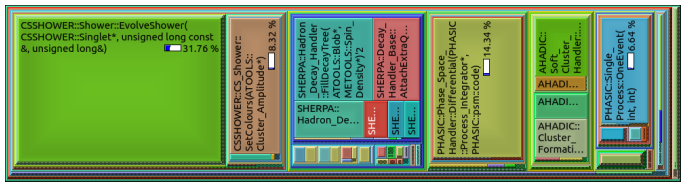
2j


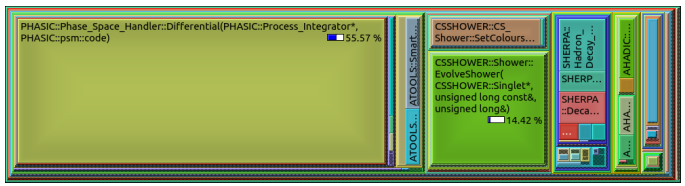
3j


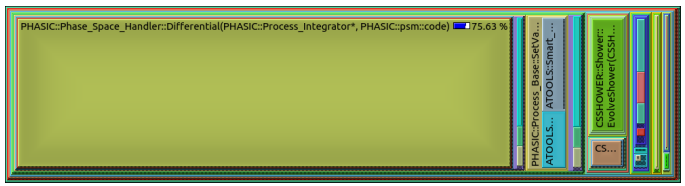
4j


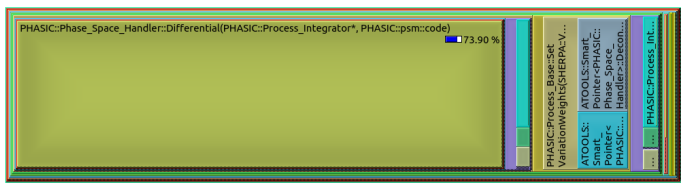
Unweighted W+0..≤4j@LO

0j


1j


2j


3j


4j


ME

Timing distribution: NLO merging using BlackHat

Weighted W+≤0..2j@NLO

parton shower (incl. MI)

NLO MEs and matching

hadron decays and QED corrs.

hadronisation mult. interactions

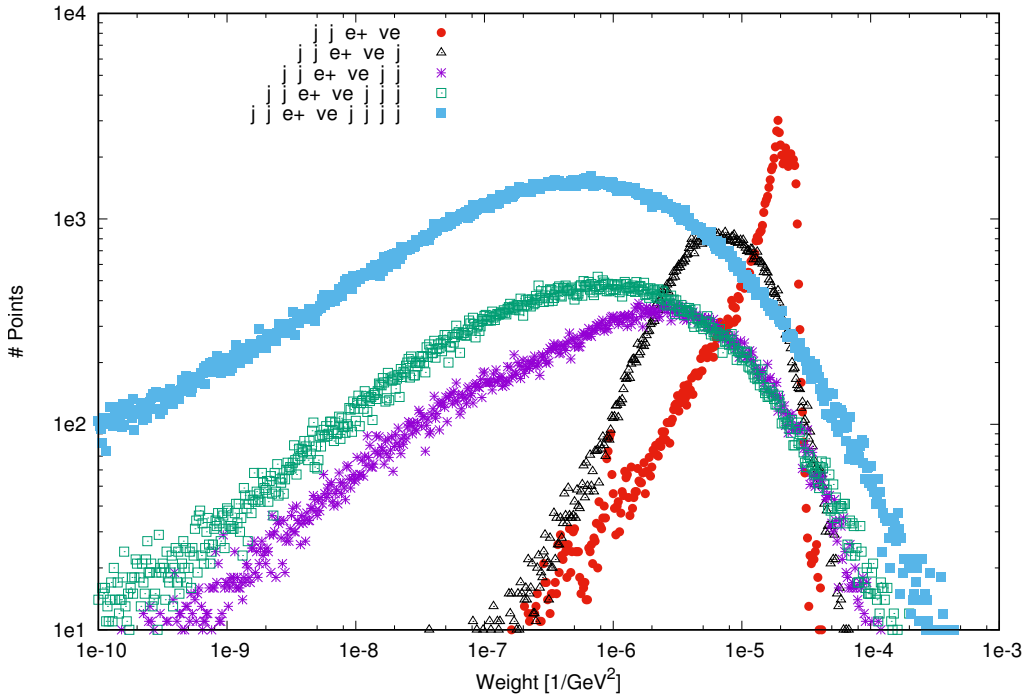
Unweighted W+≤0..2j@NLO

NLO MEs and matching

dominated by CKKW clustering in H events, cf. slide 12

Weight distributions

LO weight distribution



Unweighting

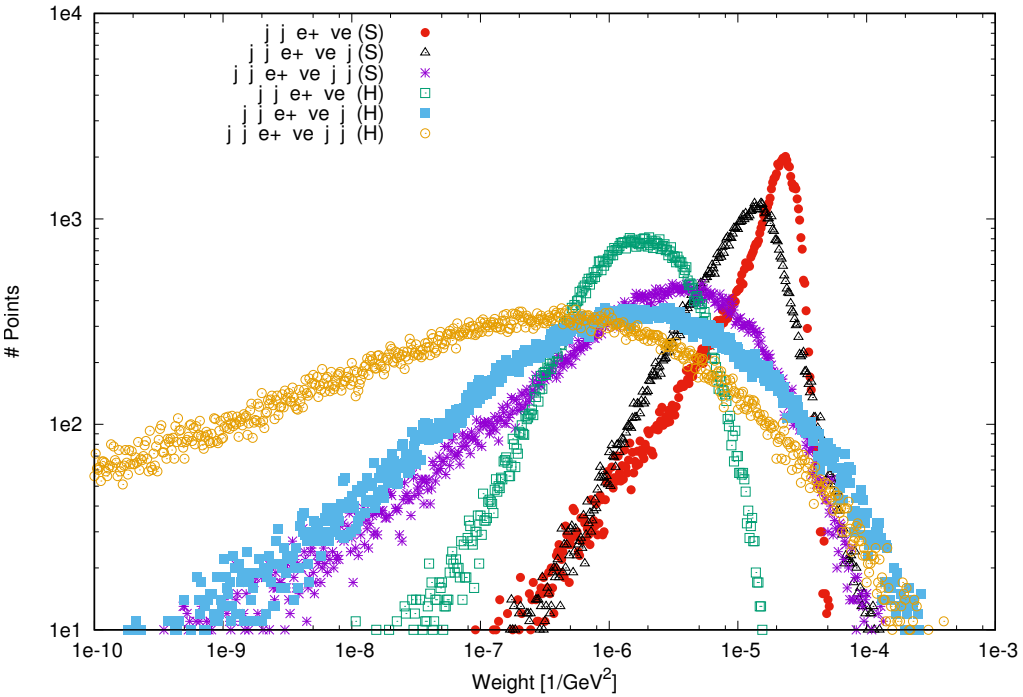
- hit-and-miss against maximum of weight distribution
- if "max" is artificially lowered (or event beyond "max" encountered)
  evts must acquire rel. weight wrt. set "max"

- higher multiplicity → wider weight distribution
  ⇒ worse unweighting efficiency
- bad unweighting efficieny at NLO dominated by H events

⇒ **performance limited by unweighting efficiency**

Weight distributions

NLO weight distribution



Unweighting

- hit-and-miss against maximum of weight distribution
- if "max" is artificially lowered (or event beyond "max" encountered)
  evts must acquire rel. weight wrt. set "max"

- higher multiplicity → wider weight distribution
  ⇒ worse unweighting efficiency
- bad unweighting efficieny at NLO dominated by H events

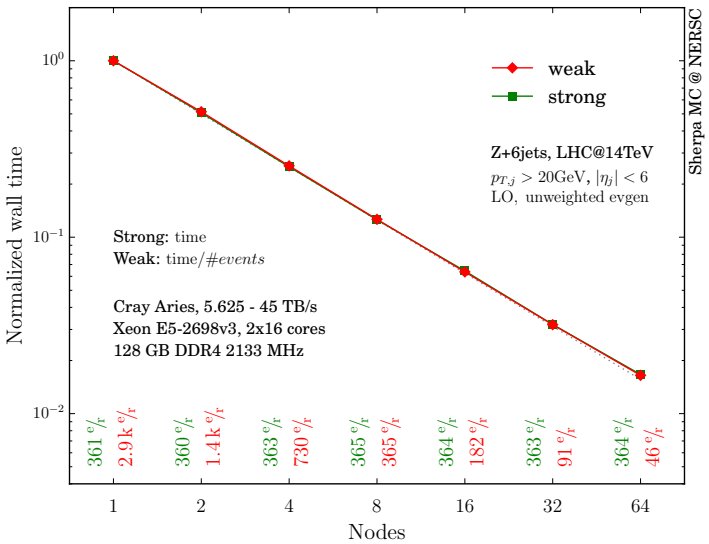
⇒ **performance limited by unweighting efficiency**

V plus multijet production

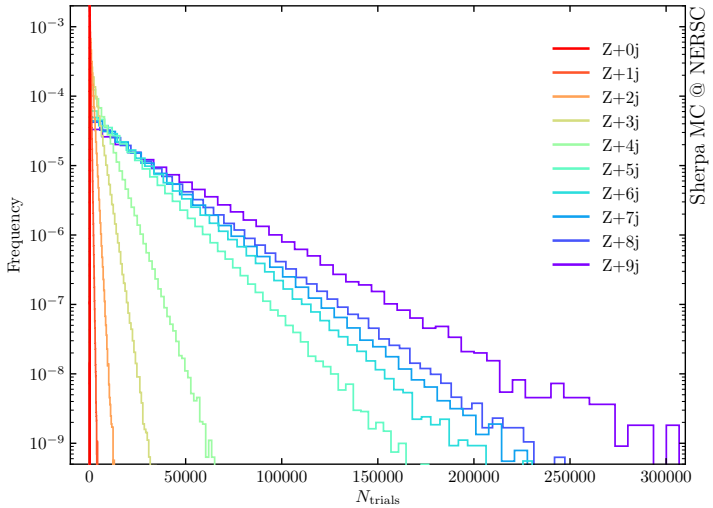
SHERPA+PYTHIA

$$pp \to V + 0, \ldots, 9\,\text{jets}$$

- generate unweighted events on large clusters
- memory is bottleneck



MPI scaling up to 64x32 cores



number of trials in unweighting

- write HDF5 files (MPI comp.)
- process w/ parton shower, large memory requirements for creating CKKW histories

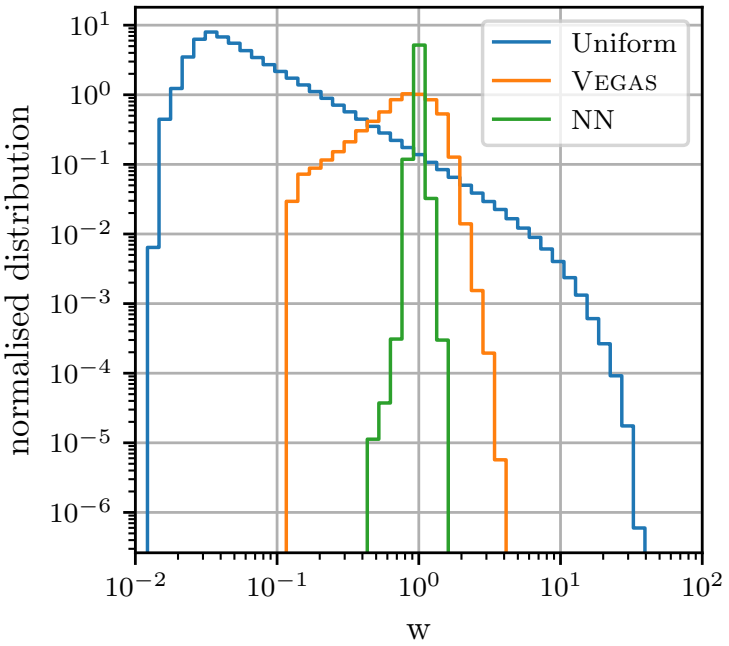
Improvements throught machine learning

Phase space sampling / unweighting

- replace old-school 1D machine learning algoirthms (VEGAS) by multivariate / NN techniques
- many tries, promising on toy examples

  Bendavid arXiv:1707.00028
  Klimek, Perelstein arXiv:1810.11509
  Otten et.al arXiv:1901.00875
  ...
- realistic implementation in working event generator

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



$gg \to 3g$

improvements only for cases that need no improvements, worse than VEGAS for bottleneck cases

not smart enough yet?

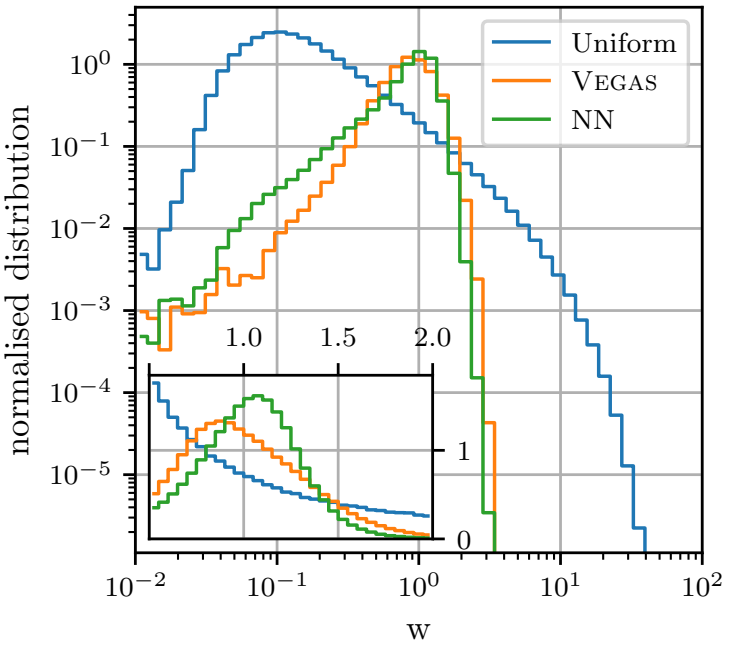
Improvements throught machine learning

Phase space sampling / unweighting

- replace old-school 1D machine learning algoirthms (VEGAS) by multivariate / NN techniques
- many tries, promising on toy examples

  Bendavid arXiv:1707.00028
  Klimek, Perelstein arXiv:1810.11509
  Otten et.al arXiv:1901.00875
  ...
- realistic implementation in working event generator

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



$gg \to 4g$

improvements only for cases that need no improvements, worse than VEGAS for bottleneck cases

not smart enough yet?

Additional sources for degredation of statistical power

Negative weights

- NLO matching methods (to var. degree) introduce neg. weights

$$N_{\text{eff}} = N\,(1-2f)^2$$

- typically arise as correction for previous overestimate

$\Rightarrow$ **neg. weighted events must be kept to minimum**

Parton shower weights

- parton showers operate at LO and leading colour, such that all splitting functions positive definite
- corrections beyond introduce weights
  $\rightarrow$ spoil hard-won uniform weights of unweighted ME configurations
  first ideas for improvements, eg. resampling Olson et.al arXiv:1912.02436

Code structure

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 memory allocation: object sizes not clear at compile time
  - designed and programmed by physicists
- 2020+ back to monolithic structure for computatationally intensive parts of the code? predictable program flow? process specific programs?

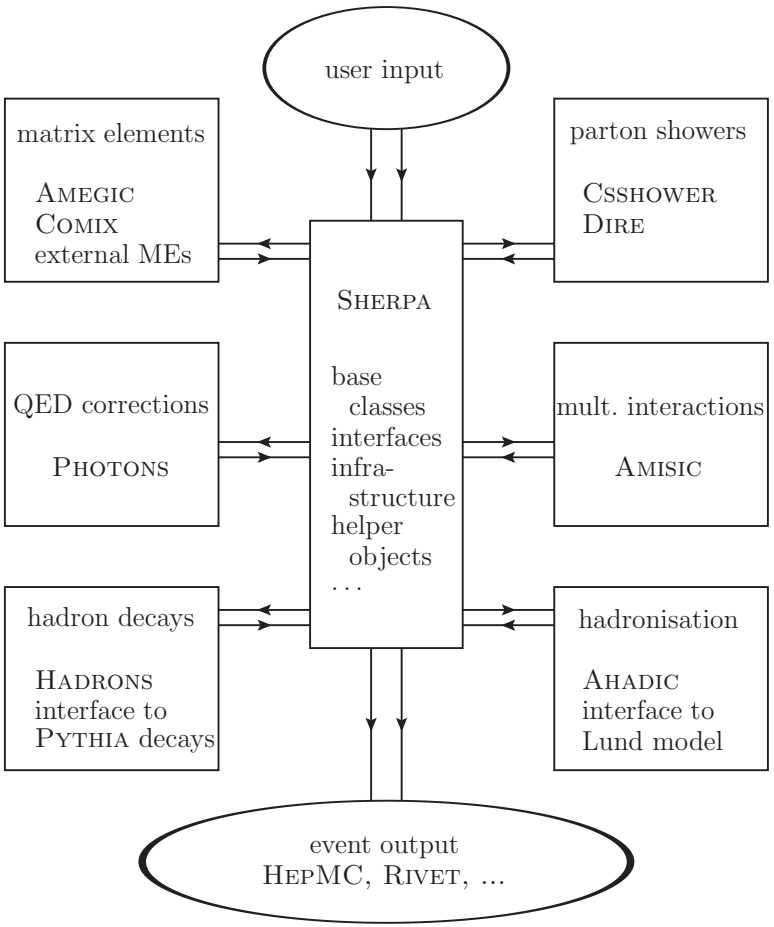
Design phases & prevalent design paradigms

- 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 memory allocation: object sizes not clear at compile time
  - designed and programmed by physicists
- 2020+ back to monolithic structure for computatationally 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 memory allocation: object sizes not clear at compile time
  - designed and programmed by physicists
- 2020+ back to monolithic structure for computatationally intensive parts of the code? predictable program flow? process specific programs?

Code structure: example SHERPA



- most physics modules reside in shared libraries loaded dynamically at run time
  - base classes defining interface functionalities reside in SHERPA
  - ME, PS, PDFs, etc. loaded at run time
- same technology also used for key user definable functions:
  - scale definitions
  - phase space cuts
  - ...

  → customisable by user without needing to touch or recompile SHERPA

Run strategy: example SHERPA

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)
2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database
3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

Run strategy: example SHERPA

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)
2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database
3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

Run strategy: example SHERPA

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)

   single core

2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database

3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

Run strategy: example SHERPA

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)

single core

2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database

MPI parallisable
linear scaling up to few thousand cores

3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

Run strategy: example SHERPA

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)

   → single core

2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     → make weights as uniform as possible
   - store in database

   → MPI parallisable
   linear scaling up to few thousand cores

3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

   → single core
   MPI parallisable but no benefits

Conclusions

- **matrix elements**
  unweighting is the bottleneck
  - high complexity of the integrand (large memory requirements)
  - poor understanding of its structure (unweighting efficiency)
    (machine learning currently does not seem to be the answer)

  $\rightarrow$ nonetheless, compared to other fields we need relatively few evaluations of the integrand, but one evaluation is complex

- **parton showers & non-perturbative modeling**
  currently fine (small part of the overall computational budget)
  - may need attention when improvements lead to large weight spread
- **code structure**
  not compatible with currently favoured computing infrastructure
  - trying to make computers do physics, not exploit their strength
  - too many run-time decisions
  - any theory code is essitially a prototype

Backup

Code performance

LO merging using COMIX

Process W^-+	0j	≤1j	≤2j	≤3j	≤4j
RAM Usage	39 MB	44 MB	49 MB	64 MB	173 MB
Initialization time	<1s	<1s	3s	22s	7m 7s
Startup time	<1s	<1s	<1s	<1s	2s
Integration time	25s	3m 19s	34m 8s	3h 12m	2d 17h
10k weighted evts	3m 24s	3m 51s	4m 2s	4m 4s	4m 21s
10k unweigthed evts	3m 20s	4m 39s	11m 47s	35m 54s	4h 3m

NLO merging using AMEGIC+BLACKHAT/COMIX (S/H-events)

Process W^-+	0j	≤1j	≤2j
RAM Usage	51 MB	112 MB	572 MB
Initialization time	1s	20s	4m 6s
Startup time	<1s	2s	18s
Integration time	20m 48s	4h 45m	5d 23h
10k weighted evts	3m 58s	4m 38s	6m 48s
10k unweighted evts	4m 14s	4h 8m	24h 54m

On dual 8-core Intel® Xeon® E5-2660 @ 2.20GHz using MPI parallelisation, timings quoted cumulative.

Code performance

LO merging using COMIX

Process W^-+	0j	≤1j	≤2j	≤3j	≤4j
RAM Usage	39 MB	44 MB	49 MB	64 MB	173 MB
Initialization time	<1s	<1s	3s	22s	7m 7s
Startup time	<1s	<1s	<1s	<1s	2s
Integration time	25s	3m 19s	34m 8s	3h 12m	2d 17h
10k weighted evts	3m 24s	3m 51s	4m 2s	4m 4s	4m 21s
10k unweigthed evts	3m 20s	4m 39s	11m 47s	35m 54s	4h 3m

NLO merging using AMEGIC+BLACKHAT/COMIX (S/H-events)

Process W^-+	0j	≤1j	≤2j
RAM Usage	51 MB	112 MB	572 MB
Initialization time	1s	20s	4m 6s
Startup time	<1s	2s	18s
Integration time	20m 48s	4h 45m	5d 23h
10k weighted evts	3m 58s	4m 38s	6m 48s
10k unweighted evts	4m 14s	4h 8m	24h 54m

On dual 8-core Intel® Xeon® E5-2660 @ 2.20GHz using MPI parallelisation, timings quoted cumulative.

bottleneck, steps taken to reduce this look promising

factor 10 improvements seem feasible

Changing scales and PDFs/$\alpha_s(m_Z)$

- on-the-fly scale and PDF variations for ME part
  $\rightarrow$ use named weights in HepMC (available since HepMC 2.06)
- two avenues:
  1) output weights for predetermined alternative scale, PDF, $\alpha_s(m_Z)$
     + number of weights grows linearly with requested variations
     – possible variations fixed at generation time
     $\rightarrow$ used currently by LHC experiments
  2) output coefficients
     + arbitrary a posteriori variations possible
     $\pm$ number of scales, parameters & coefficients dependent on type of event and multiplicity, e.g.

$V + 0j$ @ LO	**6**	(type, B, $\mathcal{O}(\alpha_s)$, $\mathcal{O}(\alpha)$, μ_R, μ_F)
$V + 4j$ @ LO	up to **37**	(as above + PDF wgt. from merging)
$V + 2j$ @ NLO (S)	up to **90**	(as above + comp. of S event)
$V + 2j$ @ NLO (H)	up to **158**	(as above + comp. of H event)

     number much lower on avarage, as often many components zero
     – need generator-specific lib to recombine into weights
     $\rightarrow$ used in e.g. MCGrid