Efficiency improvements in MC event generation

Stefan Höche

Fermi National Accelerator Laboratory

JLab Computing Round Table

Newport News, 05/05/2020

\[1\text{based on work with Christina Gao, Joshua Isaacson, Claudius Krause, Stefan Prestel and Holger Schulz}\]
What do we need to compute?

Standard Model Production Cross Section Measurements

ATLAS Preliminary
Run 1,2 \( \sqrt{s} = 5,7,8,13 \) TeV

[ATLAS] https://twiki.cern.ch/twiki/bin/view/AtlasPublic/StandardModelPublicResults

[CMS] https://twiki.cern.ch/twiki/bin/view/CMSPublic/PhysicsResultsCombined
What is our pain threshold?
Why improve generation efficiency?

- Event generation will consume significant fraction of resources at LHC soon
- Need to scrutinize both generator usage and underlying algorithms
- Dedicated effort in HEP Software Foundation (HSF)

[ATLAS] Preliminary. 2028 CPU resource needs
MC fast calo sim + standard reco

[ATLAS] Preliminary
CPU resource needs
2017 Computing model
2018 estimates:
- MC fast calo sim + standard reco
- Generators speed up x2
- Flat budget model (+20%/year)

[ATLAS] Preliminary
Preliminary
2018 estimates:
- MC fast calo sim + standard reco
- Generators speed up x2
- Flat budget model (+20%/year)

ATLAS Preliminary. 2028 CPU resource needs
MC fast calo sim + standard reco
MC-Full (Rec)
MC-Fast (Sim)
MC-Fast (Rec)
EvGen
Data Proc
Analysis
HI

Annual CPU Consumption [MHS06]

Year
2018 2020 2022 2024 2026 2028 2030 2032

[ATLAS] https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults
How does the calculation work?

- **Hard interaction**
  - LO, NLO QCD/EW $^2$, NNLO QCD $^3$
  - Various ME generators ...

- **Radiative corrections**
  - Parton / Dipole showers,
  - YFS resummation / QED showers

- **Multiple interactions**
  - Sjöstrand-Zijl / eikonal models

- **Hadronization**
  - String / Cluster model

- **Hadron Decays**
  - Phase space or EFTs,
  - YFS QED corrections

---

$^2$ via interfaces to loop generators

$^3 pp \rightarrow Z/W^\pm/h/W^\pm W^\mp$
Timing distribution: LO merging example

**Weighted** $W + 0..\leq 4j@LO$

**Unweighted** $W + \leq 0..4j@LO$
Timing distribution: LO merging example

**Weighted** $W^{+0..\leq4j@LO}$

**Unweighted** $W^{+\leq0..4j@LO}$
Timing distribution: LO merging example

**Weighted** $W + 0..\leq 4j@LO$

**Unweighted** $W + \leq 0..4j@LO$

Hadron decays + QED radiation
Timing distribution: LO merging example

**Weighted** $W + 0..\leq 4j@LO$

**Unweighted** $W + \leq 0..4j@LO$

Underlying event simulation
Timing distribution: LO merging example

**Weighted**  $W + 0..\leq 4j@LO$

**Unweighted**  $W + \leq 0..4j@LO$

Parton Shower evolution
Timing distribution: LO merging example

**Weighted** $W+0..\leq4j@LO$

**Unweighted** $W+\leq0..4j@LO$

Matrix Element generation
Timing distribution: Details on scaling with multiplicity

Observation

- Hard scattering simulation much more demanding than parton shower & hadronization in multi-jet merged simulations
- Complexity of merging ME&PS grows quickly due to inherent N! scaling of algorithms, but still negligible compared to hard scattering calculation

Where does this hierarchy come from?

- Best case ME computation naively scales as $\approx O(3^N)$
- Monte-Carlo unweighting efficiency degrades quickly due to high dimensionality of phase-space integral (3N-4)
- Overall scaling $\approx O(4^N)$ if not taking cuts into account
- Cut inefficiencies worsen the picture significantly

\[
\begin{align*}
\text{CPUh/Mevt} & = F(N_{\text{jet}}) \\
\text{Sherpa / Pythia + DIY @ NERSC} & \\
W^{+}/jets, \text{LHC@14TeV} & \\
p_{T,j} > 20\text{GeV}, |\eta_j| < 6 & \\
WTA (> 6j) & \\
\end{align*}
\]
The problem at matrix-element level

- We want to compute the integral

\[
\langle O \rangle = \sum_{a,b \in \{q,g\}} \int dx_1 \int dx_2 f_a(x_1, Q^2) f_b(x_2, Q^2) \hat{\sigma}_{ab}(Q^2) O
\]

where \( O \) is our observable, and \( \hat{\sigma} \) is the partonic cross section

\[
\hat{\sigma} = \frac{1}{SF} \int d\Phi_{2\rightarrow n} |M_{2\rightarrow n}|^2
\]

- \( d\Phi_{2\rightarrow n} \) - differential phase-space element: \textbf{3n-4 dimensional}
- \( |M_{2\rightarrow n}|^2 \) - matrix element squared: \textbf{exhibits poles and strong peaks}
- Cuts to remove singularities create \textbf{discontinuous integrand}
- Need adaptive MC methods to perform computation
- During adaptation stage, find maximum \( \langle O \rangle \)
  During event generation stage, hit-or-miss against max
Timing of integrand

\[ d\bar{u} \rightarrow e^- \nu_e + Ng \]

\[ gg \rightarrow t\bar{t} + Ng \]

  Worst case scaling factorial with particle multiplicity

  Worst case scaling exponential with particle multiplicity

- **MadGraph** [arXiv:1405.0301](https://arxiv.org/abs/1405.0301) → Feynman diagrams
  Worst case scaling factorial with particle multiplicity

  Worst case scaling \( \sim \) factorial with particle multiplicity
Weight distributions and number of trials

- Unweighted events obtained by performing hit-or-miss against maximum of weight distribution
- If maximum artificially low (or current weight exceeds known max) events acquire relative weight

Higher multiplicity events have broader weight distribution

Leads to much reduced unweighting efficiency

Clearly, if we improve this, we gain more than by just accelerating the integrand
Scaling with MPI and on different architectures

Traditionally, integration is performed with Vegas or similar algorithms

Grid warmup step scales (strong & weak) up to \( \sim 2048 \) cores

Performance limited by number of events being processed per rank

Acceptable (though limited) performance on KNL w/o modifications
Scaling with MPI and on different architectures

- Scaling of event generation step (strong & weak) up to ~ 2048 cores
- Performance limited by number of events being processed per rank (Average timing can only be expected if statistics is large enough)
- Performance of particle-level simulation limited by I/O speed
  Good results with Cori burst buffer, but room for improvement
Improving event generation efficiency with Neural Networks

- Neural Networks can be used in different ways to improve event generation

**Surrogate model technique**

- Generate events with GANs
- Several orders of magnitude faster
- Generates unweighted events
- Needs existing sample to train
- Biased results if not trained right

**Variable transformation technique**

- Learn integrand to improve importance sampling
- Insufficient training yields high uncertainties, but no bias
- Events generated from scratch no pre-existing sample required
- Resulting events still need to be unweighted
Event generation with Normalizing Flows

- Straightforward MC integral estimator

\[ I = \int_{\Omega} f(x) \, dx = \frac{\Omega}{N} \sum_{i=1}^{N} f(x_i) = \Omega \langle f \rangle_x , \quad \sigma_I = \Omega \sqrt{\frac{\langle f^2 \rangle_x - \langle f \rangle_x^2}{N - 1}} \]

- After variable transformation \( dx \rightarrow dx \, g(x) = dG(x) \)

\[ I = \int_{\Omega} \frac{f(x)}{g(x)} \, dG(x) = \Omega \langle f/g \rangle_G , \quad \sigma_I = \Omega \sqrt{\frac{\langle (f/g)^2 \rangle_G - \langle f/g \rangle_G^2}{N - 1}} \]

- Multi-dimensional integrals: \( d\vec{x} \rightarrow d\vec{x}' \mid d\vec{x}'(\vec{x}'')/d\vec{x}'' \mid \rightarrow \) Jacobian changes from \( 1/g(x) \) to \( |d\vec{x}''/d\vec{x}|^{-1} \)

- For bijective map \( g \) of random variable \( \vec{x} \) drawn from base distribution \( q_0 \), the variable \( \vec{x}' = g(\vec{x}) \) follows distribution inferred by chain rule:

\[ q_1(\vec{x}') = q_0(g^{-1}(\vec{x}')) \left| \left| \frac{\partial g^{-1}(\vec{x}')}{\partial \vec{x}''} \right| \right| = q_0(\vec{x}) \left| \left| \frac{\partial g(\vec{x})}{\partial \vec{x}} \right| \right|^{-1} \]

- In CS literature, this transformation is called a “Normalizing Flow”
Event generation with Normalizing Flows

- If we use a Neural Network to learn $g$, we need to compute its gradient during inference. This is veery slow. No, really! It’s even slower!
- “Coupling layers” are special bijectors to avoid these gradients:
  - Input variables $\vec{x} = \{x_1, .., x_D\}$ partitioned into two subsets, $\vec{x}_A$ and $\vec{x}_B$
    
    $\begin{align*}
    x'_A &= x_A, \\
    x'_B &= C(x_B; m(\vec{x}_A))
    \end{align*}$
  - $m$ is output of a Neural Network taking $x_A$ as inputs and returning parameters of “Coupling Transform” $C$ that will be applied to $x_B$
  - Inverse map is simple, leading to simple Jacobian (no $\partial m / \partial \vec{x}_A$!)

$$
\begin{aligned}
\left| \frac{\partial g(\vec{x})}{\partial \vec{x}} \right|^{-1} &= \left| \begin{pmatrix}
1 & 0 \\
\frac{\partial C}{\partial m} \frac{\partial m}{\partial \vec{x}_A} & \frac{\partial C}{\partial \vec{x}_B}
\end{pmatrix} \right|^{-1} = \left| \frac{\partial C(\vec{x}_B; m(\vec{x}_A))}{\partial \vec{x}_B} \right|^{-1}.
\end{aligned}
$$

![Diagram of event generation with Normalizing Flows](image-url)
Toy example - 4-dimensional camel function

loss = 3.197938e+00

Toy example - 4-dimensional camel function

loss = 3.140863e+00

Toy example - 4-dimensional camel function

loss = 3.162317e+00

Toy example - 4-dimensional camel function

\[ x_0 = 0.52^{+0.32}_{-0.36} \]

\[ x_1 = 0.51^{+0.33}_{-0.36} \]

\[ x_2 = 0.50^{+0.33}_{-0.33} \]

\[ x_3 = 0.51^{+0.33}_{-0.33} \]

\[ \text{loss} = 2.945214e+00 \]

Toy example - 4-dimensional camel function

\[ x_0 = 0.55^{+0.29}_{-0.39} \]

\[ x_1 = 0.52^{+0.29}_{-0.39} \]

\[ x_2 = 0.50^{+0.37}_{-0.33} \]

\[ x_3 = 0.51^{+0.33}_{-0.33} \]

loss = 2.639655e+00

Toy example - 4-dimensional camel function

loss = 1.771176e+00

Toy example - 4-dimensional camel function

\[ x_0 = 0.63^{+0.12}_{-0.09} \]

\[ x_1 = 0.57^{+0.17}_{-0.45} \]

\[ x_2 = 0.43^{+0.48}_{-0.48} \]

\[ x_3 = 0.51^{+0.56}_{-0.48} \]

loss = 1.184918e+00

Toy example - 4-dimensional camel function

\[ x_0 = 0.65 \pm 0.05 \]

\[ x_1 = 0.55 \pm 0.10 \]

\[ x_2 = 0.41 \pm 0.05 \]

\[ x_3 = 0.51 \pm 0.10 \]

loss = 6.472999e-01

Toy example - 4-dimensional camel function

\[ x_0 = 0.66^{+0.16}_{-0.10} \]

\[ x_1 = 0.59^{+0.15}_{-0.48} \]

\[ x_2 = 0.39^{+0.53}_{-0.03} \]

\[ x_3 = 0.51^{+0.38}_{-0.18} \]

loss = $7.906117e-02$

Toy example - 4-dimensional camel function

\[ x_0 = 0.66^{+0.10}_{-0.10} \]

\[ x_1 = 0.58^{+0.15}_{-0.48} \]

\[ x_2 = 0.39^{+0.93}_{-0.13} \]

\[ x_3 = 0.52^{+0.18}_{-0.18} \]

\[ \text{loss} = 2.640339e-02 \]

Real-life example: $e^+ e^- \rightarrow qqg$


← $g$ color
← $q$ color
← $g$ color spectator
← $\cos \vartheta$ of decaying fermion with beam
← $\varphi$ of decaying fermion with beam
← $\cos \vartheta$ of decay
← $\varphi$ of decay
← propagator of decaying fermion
← multichannel ...

Target distribution with learning color
Real-life example: \( pp \rightarrow V + \text{jets} \)

- Check results in most performance-critical applications
- To make unweighting efficiency independent of weight outliers, define \( \max \) as median of maxima in bootstrap approach [Campbell, Neumann] arXiv:1909.09117

<table>
<thead>
<tr>
<th>unweighting efficiency</th>
<th>( \langle w \rangle/w_{\max} )</th>
<th>( n = 0 )</th>
<th>( n = 1 )</th>
<th>( n = 2 )</th>
<th>( n = 3 )</th>
<th>( n = 4 )</th>
<th>( n = 0 )</th>
<th>( n = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W^+ + n \text{jets} )</td>
<td>Sherpa</td>
<td>2.5 ( \cdot 10^{-1} )</td>
<td>3.4 ( \cdot 10^{-2} )</td>
<td>6.7 ( \cdot 10^{-3} )</td>
<td>1.7 ( \cdot 10^{-3} )</td>
<td>6.6 ( \cdot 10^{-4} )</td>
<td>6.5 ( \cdot 10^{-2} )</td>
<td>2.9 ( \cdot 10^{-3} )</td>
</tr>
<tr>
<td></td>
<td>NN+NF</td>
<td>5.8 ( \cdot 10^{-1} )</td>
<td>1.2 ( \cdot 10^{-1} )</td>
<td>8.8 ( \cdot 10^{-3} )</td>
<td>1.6 ( \cdot 10^{-3} )</td>
<td>8.9 ( \cdot 10^{-4} )</td>
<td>1.7 ( \cdot 10^{-1} )</td>
<td>4.0 ( \cdot 10^{-3} )</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>2.3</td>
<td>3.6</td>
<td>1.3</td>
<td>0.99</td>
<td>1.4</td>
<td>2.7</td>
<td>1.4</td>
</tr>
<tr>
<td>( W^- + n \text{jets} )</td>
<td>Sherpa</td>
<td>2.4 ( \cdot 10^{-1} )</td>
<td>3.9 ( \cdot 10^{-2} )</td>
<td>8.4 ( \cdot 10^{-3} )</td>
<td>1.7 ( \cdot 10^{-3} )</td>
<td>8.8 ( \cdot 10^{-4} )</td>
<td>6.0 ( \cdot 10^{-2} )</td>
<td>3.3 ( \cdot 10^{-3} )</td>
</tr>
<tr>
<td></td>
<td>NN+NF</td>
<td>6.2 ( \cdot 10^{-1} )</td>
<td>1.3 ( \cdot 10^{-1} )</td>
<td>1.2 ( \cdot 10^{-2} )</td>
<td>2.3 ( \cdot 10^{-3} )</td>
<td>9.8 ( \cdot 10^{-4} )</td>
<td>1.6 ( \cdot 10^{-1} )</td>
<td>3.8 ( \cdot 10^{-3} )</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>2.6</td>
<td>3.2</td>
<td>1.5</td>
<td>1.4</td>
<td>1.17</td>
<td>2.8</td>
<td>1.2</td>
</tr>
<tr>
<td>( Z + n \text{jets} )</td>
<td>Sherpa</td>
<td>4.3 ( \cdot 10^{-1} )</td>
<td>4.3 ( \cdot 10^{-2} )</td>
<td>1.3 ( \cdot 10^{-2} )</td>
<td>2.7 ( \cdot 10^{-3} )</td>
<td>1.1 ( \cdot 10^{-3} )</td>
<td>1.1 ( \cdot 10^{-1} )</td>
<td>4.9 ( \cdot 10^{-3} )</td>
</tr>
<tr>
<td></td>
<td>NN+NF</td>
<td>5.1 ( \cdot 10^{-1} )</td>
<td>1.1 ( \cdot 10^{-1} )</td>
<td>1.3 ( \cdot 10^{-2} )</td>
<td>2.6 ( \cdot 10^{-3} )</td>
<td>1.8 ( \cdot 10^{-3} )</td>
<td>1.8 ( \cdot 10^{-3} )</td>
<td>4.9 ( \cdot 10^{-3} )</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>1.2</td>
<td>2.6</td>
<td>1.1</td>
<td>0.97</td>
<td>1.7</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
Summary and Outlook

- New workflow to generate LO-merged event samples on CPU-based HPC resources
- Working towards similar workflow at NLO
- Novel technique to improve event generation with Neural Network based adaptive integration techniques
- All this heavily reliant on detailed understanding of underlying physics models and algorithms

There are no free lunches!