Intro to “fast event generation” (and a bit on HepMC tools and LHAPDF 6!)

Andy Buckley
University of Glasgow

Fast Monte Carlo in HEP #2, DESY Zeuthen, 14 Jan 2014

Introduction to the evgen session

"Traditionally" event generation has been forgotten in attempts to optimise simulation CPU usage.

"1 event per second" throughput target assumes that the evgen step is already well-within that limit. But far from true these days!

Majority of LHC process simulation (at least for SM and Higgs processes) is well beyond Born+LL level: NLO or multileg hard process calculations (or both at once).

Complex processes and huge integration phase spaces mean that evgen can approach 1 event/hour in bad cases! (Also easily possible with very aggressive filtering such as used to create b-physics samples from $\sim$ min bias MC.)

Generator / theory trend has been towards higher accuracy, usually meaning slower code: what are the best "operating points" for experimental needs?

Introduction to the evgen session (2)

Several efforts underway in experiments to (re)organise evgen production for LHC Run 2: mainly based on factorisation of hard process and "dressing" steps (via LHEF data exchange), and use of dedicated HPC resources for evgen.

For very high-stats processes is there a better option than "fall back to Born"? Balance of processing time between pre-integration / grid-building and event sampling? Parallelisability of each step?

Focus of session: updates from generator authors on code developments and prospects for speed increases in various generator components (hard process, showers, ...) and with various technologies.

Also experimental talks on HPC and GPU work so far (applications to evgen, detector sim and analysis).

But first a brief diversion on some relevant evgen/MC analysis tool developments...

HepMC and speed issues

I was asked to speak about HepMC developments and how they might impact on speed.

HepMC and speed issues

I was asked to speak about HepMC developments and how they might impact on speed. Lowest order answer:

THEY DON’T!

HepMC and speed issues

I was asked to speak about HepMC developments and how they might impact on speed. Lowest order answer:

THEY DON'T!

HepMC itself not *usually* a speed bottleneck, but several developments:

- Reading from ASCII format (and writing?) can be much faster by using C functions (no widechar support, etc.)
- Event records full of *debug info*. Strip down to what's *needed*.
- Experiment analysis formats already do MC stripping, but break graph structure: MCUtils (**https://bitbucket.org/andybuckley/mcutils**) provides functions for more intelligent filtering.
- Built-in multi-weight / systematics treatments starting to gain momentum. One approach to fast evgen is to generate less, reweight more. . . . . . if done correctly. Impact on production & analysis codes.
- HepMC development restart planned ⇒ version 3. Interface fixes and improvements: speed-up possible in refactoring/removing current iterators? (Unknown due to *code hell*!)
- Tool development (MCUtils, HepMC, . . .) will continue: please use and contribute to shared tool packages.

LHAPDF speed (and other) improvements

And now something less speculative: real speed (and other) improvements in the C++ rewrite of LHAPDF.

The main problems with LHAPDF 5:

- **Fortran code with static, unshared memory** $\Rightarrow$ *huge* memory requirements $\mathcal{O}(2-10\text{ GB})$
  $\Rightarrow$ Grid/batch issues
- **Multiset, low-memory, etc. modes are *hacks*** on original code. Speed, VMEM, flexibility, and correct operation all suffer depending on build-time configuration.
  - NMXSET determined at build-time and too restrictive for e.g. error set reweighting
  - Re-init very slow. Esp. later PDF members at end of file.
  - Some functions don't (can't) respect multi-set indexing: **`alphaS`**, **`xMin`**, etc.
- **Unmaintainable!**

LHAPDF 6

- **Ground-up rewrite (in C++, with Python wrapper) attempting to learn from and solve all these problems**

  AB, Martin Ruefenacht, Karl Nordstrom, James Ferrando, Steve Lloyd, others
- **Key feature: dynamic allocation:** allocate only what you use — and no concurrency limitation
- Uniform data format with arbitrary flavours and powerful metadata
- **Speed:**
  - Single-flavour, single member loading and interpolation are always faster than LHAPDF 5: less data to read, only interpolate the required flavour. Can be a factor of ~ 3.
  - All-flavour interpolation can be slower: depends on set. Possibility of caching, vectorization, ...
  - Very significant speed-ups for event generation if code is migrated to support single-flavour mode: Sherpa has already done so
  - Very significant speed-up for reweighting, since only one flavour needed per incoming parton.

Examples: usage from C++

Single member:

```
#include "LHAPDF/LHAPDF.h"
...
LHAPDF::PDF* pdf = LHAPDF::mkPDF("CT10nlo", 0);
size_t num_mems = pdf->numMembers();
// One value:
double xf_g = pdf->xfxQ(21, 1e-3, 126.0);
// Quark and gluon values:
vector<double> xfs;
pdf->xfxQ(1e-3, 126.0, xfs);
// All values (partons, photon, gluino, ...):
map<int, double> xfs = pdf->xfxQ(1e-3, 126.0);
delete pdf;
```

PDF set:

```
vector<unique_ptr<LHAPDF::PDF>> pdfs;
LHAPDF::mkPDFs("CT10nlo", pdfs);
for (const auto& p : pdfs)
  double xf_g = p->xfxQ(21, 1e-3, 126.0);
```

Examples: usage from Python

Single member:

```
>>> import lhapdf
>>> pdf0 = lhapdf.mkPDF("CT10nlo", 0)
>>> pdf0.xfxQ(21, 1e-3, 126)
31.199466144272378
```

PDF set:

```
>>> pdfs = lhapdf.mkPDFs("CT10nlo")
>>> len(pdfs)
52
>>> [pdf.xfxQ(21, 1e-3, 126) for pdf in pdfs]
[31.199466144272378, 31.10261967456719, ...
...
```

Memory

Numbers from 6.0.0beta1 but still relevant

LHAPDF 5

```
$ size -B -d ~/heplocal/lib/libLHAPDF.so
text     data      bss       dec
1509082  142048 2039405376 2041056506
```

⇒ 1.5 MB functions, 140 kB data, **2 GB** uninitialised data!

⇓

LHAPDF 6

```
$ size -B -d ~/heplocal/lib/libLHAPDF.so
text     data      bss       dec
265310    8504     1552   275366
```

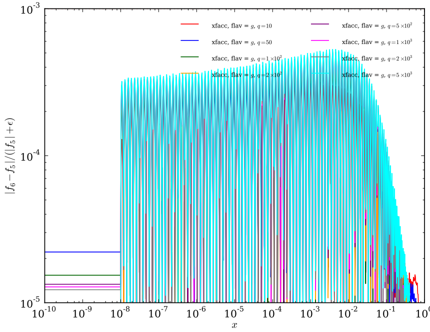
⇒ 2.6 kB functions, 8 kB data, **280 kB** uninitialised data!

WIN!

Set migration and validation

- We set a nominal LHA5 $\rightarrow$ 6 reproduction accuracy target of per-mille (1/1000)
- Regularised deviation measure $\Delta = |f_6 - f_5|/(|f_5| + \epsilon)$
- Get sets from **http://www.hepforge.org/archive/lhapdf/pdfsets/6.0/**

xf* vs. *x

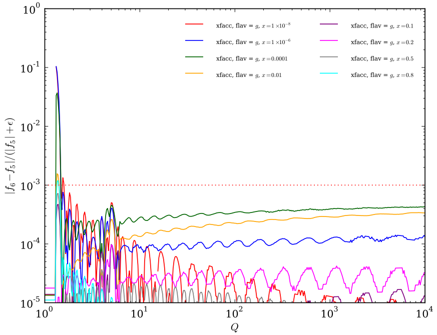


Most set migrations now final – CTEQ, HERA, ATLAS, ABM sign-off awaited. NNPDF & MRST/MSTW sets approved.

Set migration and validation

- We set a nominal LHA5 $\rightarrow$ 6 reproduction accuracy target of per-mille (1/1000)
- Regularised deviation measure $\Delta = |f_6 - f_5|/(|f_5| + \epsilon)$
- Get sets from **http://www.hepforge.org/archive/lhapdf/pdfsets/6.0/**

xf* vs. *Q

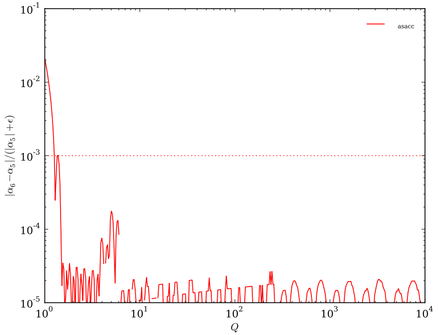


Most set migrations now final – CTEQ, HERA, ATLAS, ABM sign-off awaited. NNPDF & MRST/MSTW sets approved.

Set migration and validation

- We set a nominal LHA5 $\rightarrow$ 6 reproduction accuracy target of per-mille (1/1000)
- Regularised deviation measure $\Delta = |f_6 - f_5|/(|f_5| + \epsilon)$
- Get sets from **http://www.hepforge.org/archive/lhapdf/pdfsets/6.0/**

α_s vs. Q



Most set migrations now final – CTEQ, HERA, ATLAS, ABM sign-off awaited. NNPDF & MRST/MSTW sets approved.

Summary of everything

Event generation is no longer a trivial consumer of HEP experiment CPU resources. Will get worse as need for precision MC increases: higher orders, higher stats and more systematics variations.

Fast detector simulation needs to be combined with emphappropriate organisation of event generation.

We also hope that there are prospects for “built-in” speedups (algorithmic and technical) from the generator authors’ side: let’s see from the talks!

Common event generation infrastructure like HepMC and LHEF aren’t usually a performance bottleneck. But the way they are used can be key to efficient generation strategy.

Developments are starting up again in these areas, to make the toolkits more user-friendly and powerful: please participate so that we all benefit and spend less time swearing at our monitors.

LHAPDF 6 is ready to use “for real”: latest version is easy to install and faster than ever. Speed gains w.r.t. LHAPDF 5 should be helpful for evgen, and very important for reweighting.