WP2.1 plans

Christian Gütschow

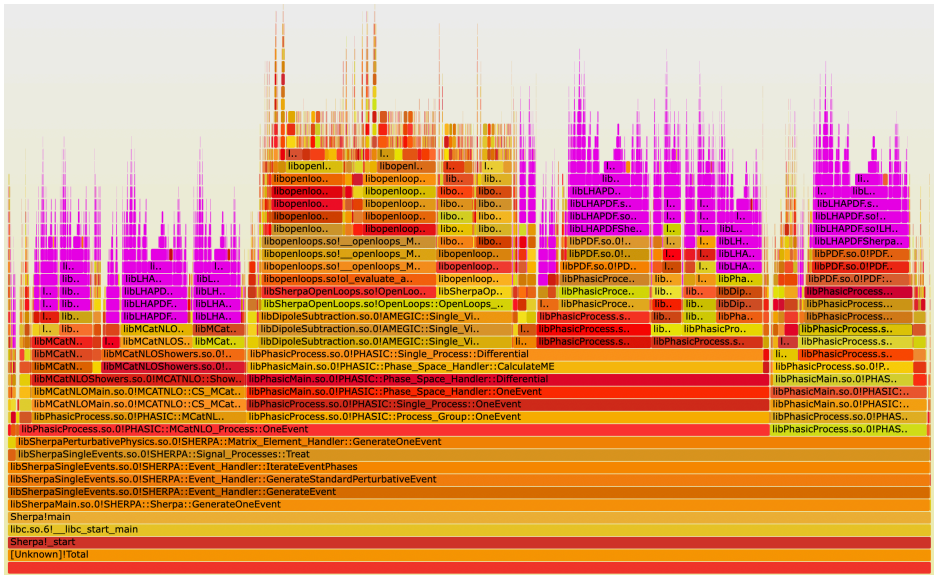
obo SWIFT-HEP Generators team

3 November 2021

Overview

- first ECHEP generator profilings by Tim Martin showed LHAPDF uses a lot of the per-event CPU time (60–70%)
  - maybe not completely surprising: multiweights are a releatively recent feature and code originally not designed with hundreds of variations in mind

- RHS: Sherpa+OpenLoops $\ell\ell$+0,1,2j@NLO+3,4,5j@LO with PDF calls (60%) highlighted in magenta



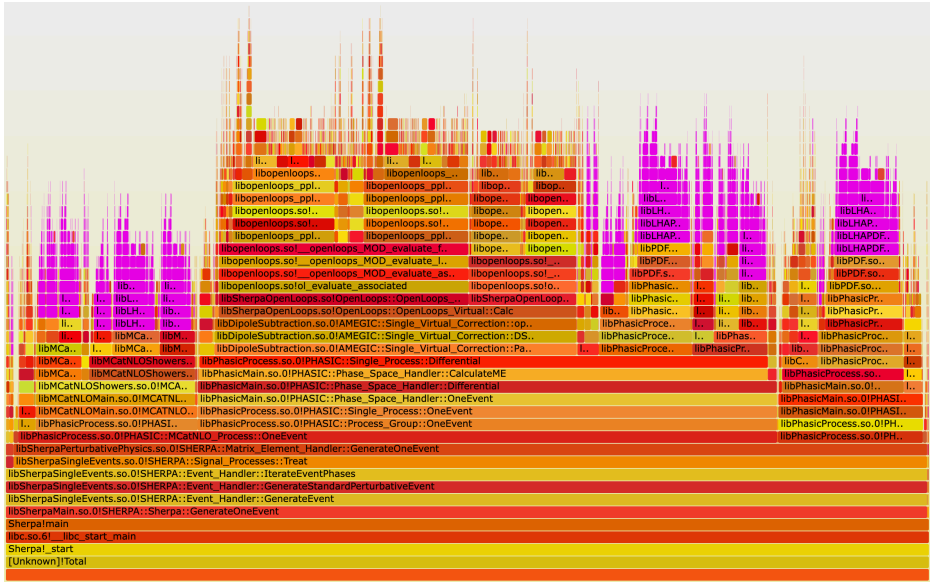
- using LHAPDF 6.2.3

- explore two approaches in parallel:
  - make LHAPDF faster (lots of time spent interpolating and evaluating transcendental functions)
  - rework LHAPDF call strategy

LHAPDF 6.4.0

- lots of work went into reviewing the interpolation logic inside LHAPDF over the summer months
- new release out since end of September

Andy Buckley, Max Knobbe



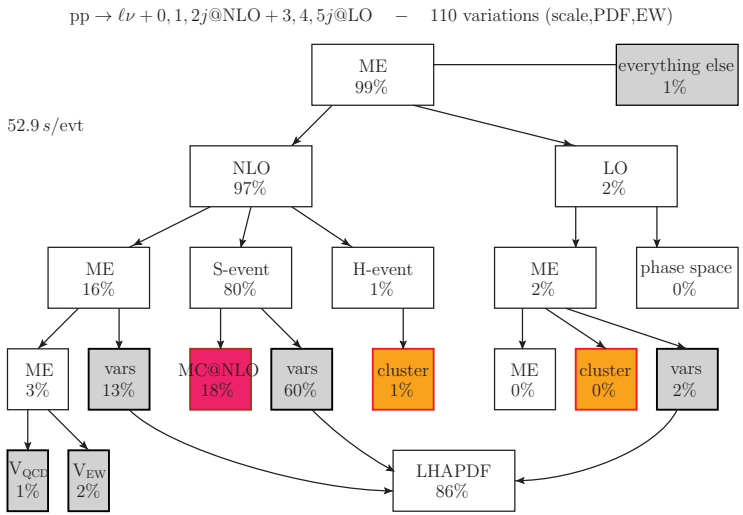
- RHS: Sherpa+OpenLoops $\ell\ell$+0,1,2j@NLO+3,4,5j@LO with PDF calls (40%) highlighted in magenta
- using LHAPDF 6.4.0

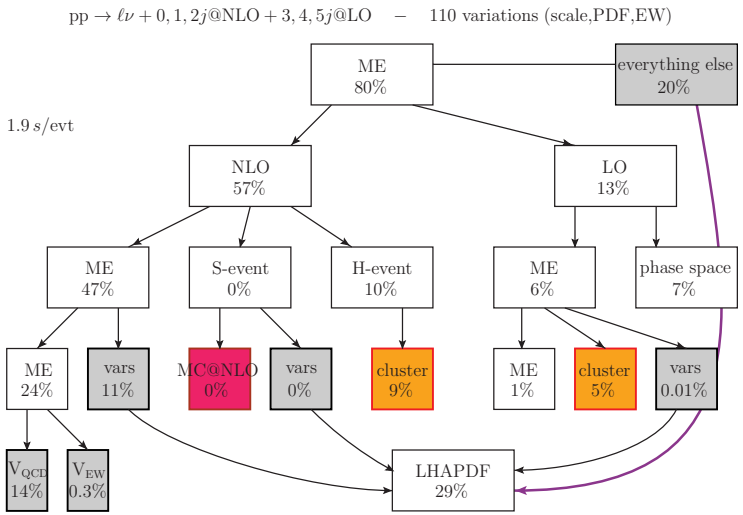
- new release **already brings 30% overall speed-up** for expensive multi-leg NLO setups
  - next biggest blob coming from OpenLoops (virtual corrections) . . .

Review of call strategies

Enrico Bothmann, Stefan Höche, Marek Schönherr

- rearranging program flow to compute expensive variations after the unweighting leads to significant performance improvements
- still to be quantified on a representative machine
- need fresh CPU profiling to re-assess bottlenecks after pulling together all latest developments





Summary

- latest LHAPDF release brings major performance improvements that already have a noticeable impact on overall run time
- parallel effort to revise the evaluation strategy for weight variations in Sherpa
- plan to kick off with fresh CPU profiling accounting for all recent developments
- low hanging fruits in Sherpa have been picked, future developments require dedicated effort
  - likely next target: try replacing expensive numerical QCD loops with analytic results where available (☞ ARXIV:2107.04472)
  - involve RSE soon to look at specific CPU-intensive modules