SHERPA

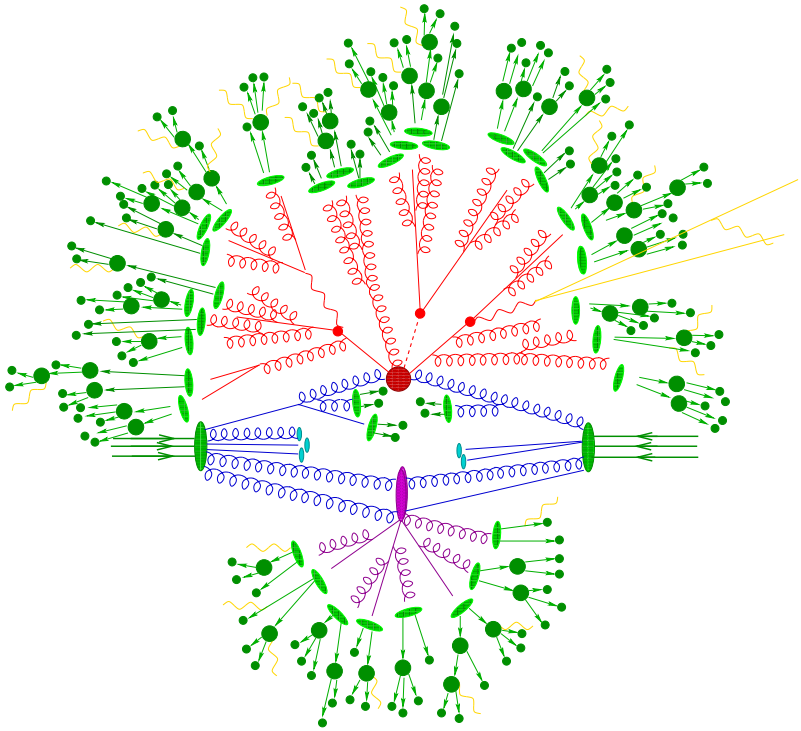
–

Event generator for

high-energy collisions

–

findings and improvements



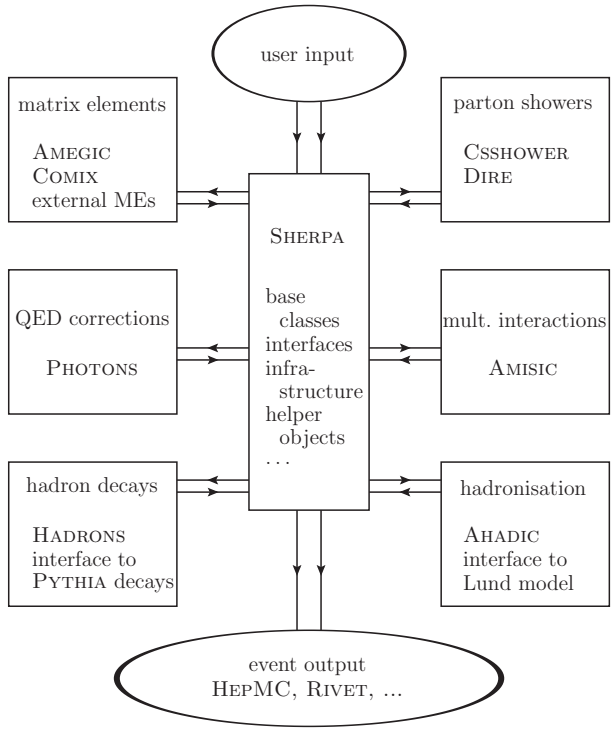
in the workshop

- Enrico Bothmann (Göttingen)
- Lois Flower (Durham)
- Chris Gütschow (UCL)
- Stefan Höche (Fermilab)
- Marek Schönherr (Durham)

http://sherpa.hepforge.org

SHERPA – usage

- SHERPA one of the main tools for theory predictions used by the LHC experiments
- typically used for high-precision processes with large cross sections / event rates
- written in C++, modular
- in active development for about 20 years



- currently about 300k lines of code in 100 libraries,
  (+ scattering process dependent auto-generated code/libs,
  user supplied custom code/libs)

Computational challenge

We are trying to compute the following **integral** (scattering cross section)

$$\sigma = \int \mathrm{d}\Phi_n \int_0^1 \mathrm{d}x_a \int_0^1 \mathrm{d}x_b \; f_a(x_a)\, f_b(x_b)\, \hat{\sigma}_{ab\to n}\, O(\Phi_n)$$

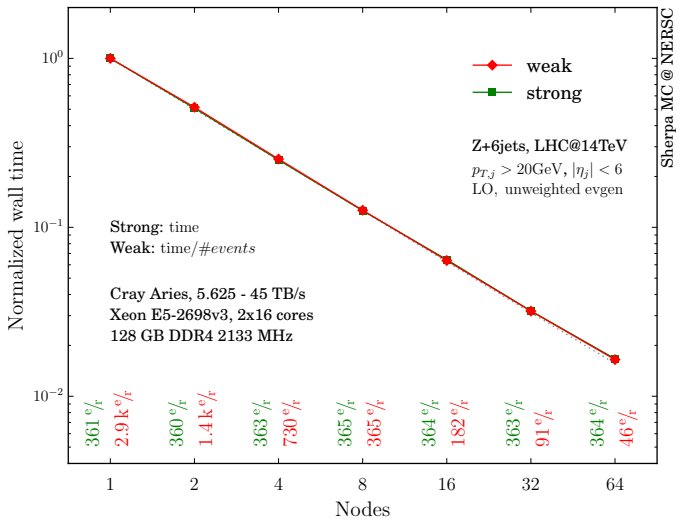
using Monte-Carlo techniques.

Challenges:

- phase space $\mathrm{d}\Phi_n$ is $3n-4$ dimensional
  - strongly peaked integrand, different peak structures in different terms
  - needs better (analytic) understanding of integrand
  - $O(\Phi_n)$ contains a series of Θ-functions, defines detector volume
- scattering amplitudes $\hat{\sigma}_{ab\to n}$ can be somewhat large for $n \gtrsim 4$
  - $> 1\,\mathrm{GB}$ RAM
  - slow to evaluate, need to be evaluated $\mathcal{O}(10^5)$ times to get one statistically representative point

MPI parallelisation

- problem embarrassingly parallel, parallelisation only to collect statistics for Vegas optimisation ⇒ MPI
- minimal data-transfer between ranks, memory is the limiting factor
- performance determined by single-core performance

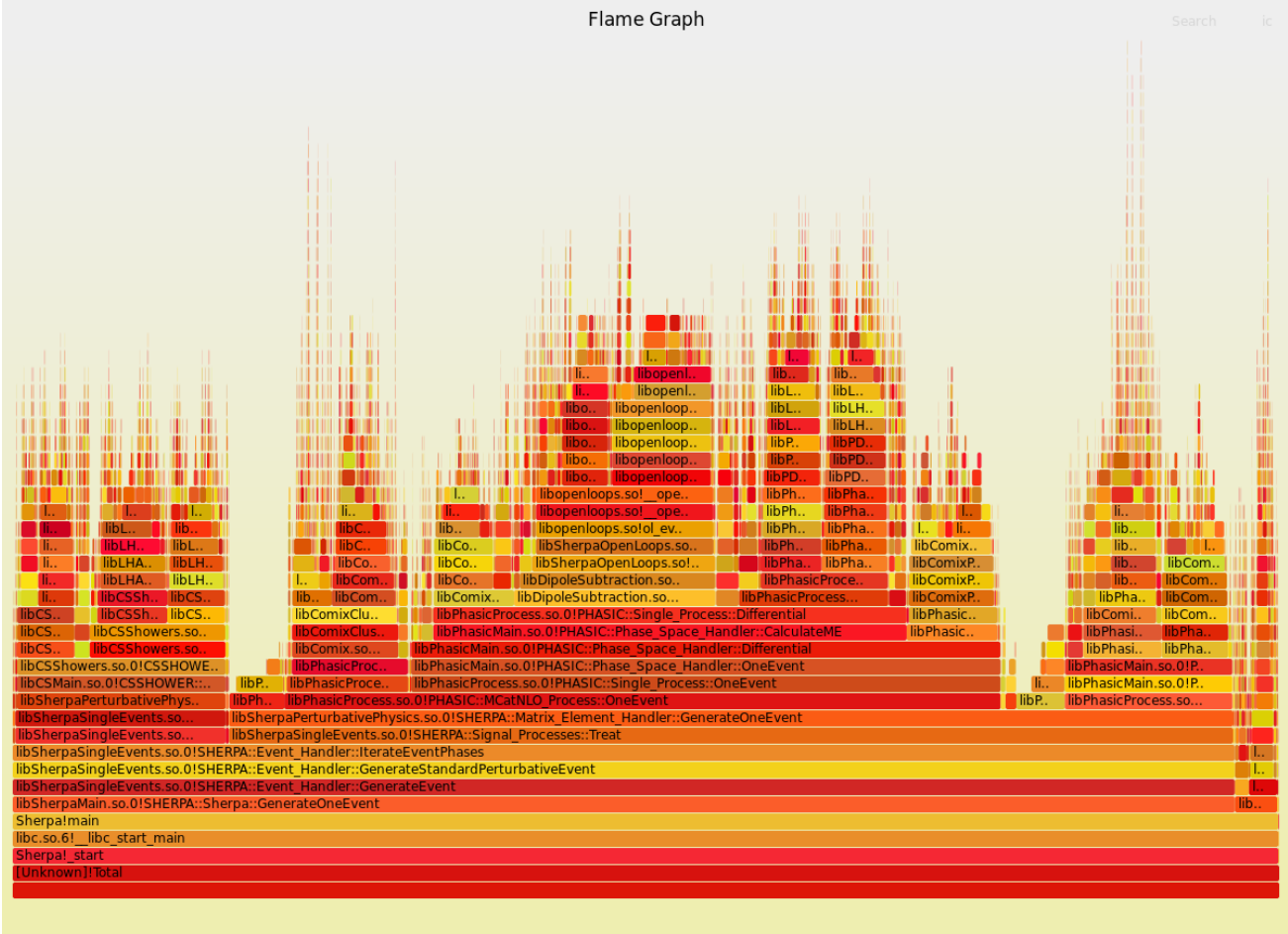


MUST

Rank(s)	Type	Message
	Information	MUST detected no MPI usage errors nor any suspicious behavior during this application run.

Single-core performance

- measure with VTune, visualise with FlameGraph
- identify methods where most time is spent, determine why



Single-core performance – findings & improvements

Sherpa employs a hit-and-miss technique to produce uniformly weighted phase space points, so-called events (unweighting),
simultaneously compute value with variations of input parameters

Findings:

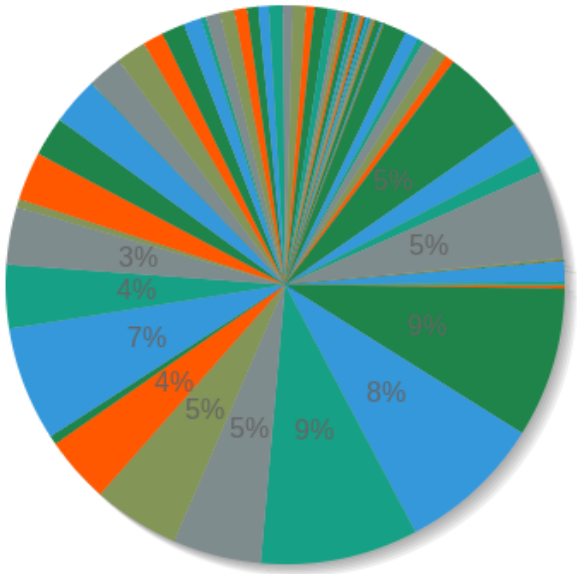
- all parameter variations computed for all trial phase space points, but hit-and-miss only based on the nominal value
  (typical unweighting efficiencies $\mathcal{O}(10^{-4})$)

Improvements:

- use pilot-run strategy: calculate trial phase space points in minimal setup, once accepted recompute using all param. variations
- implemented dedicated `pow(int)` function,
  replaced recurring `exp` with look-up table
- typical setup w/ $\mathcal{O}(100)$ vars: **4.7 s/evt** $\longrightarrow$ **1.0 s/evt**

Single-core performance

MAQAO



LIKWID

Event	Counter	HWThread 0
INSTR_RETIRED_ANY	FIXC0	12793580000000
CPU_CLK_UNHALTED_CORE	FIXC1	13565440000000
CPU_CLK_UNHALTED_REF	FIXC2	11709840000000
...	...	...
FP_COMP_OPS_EXE_SSE_FP_PACKED_DOUBLE	PMC0	238869800000
FP_COMP_OPS_EXE_SSE_FP_SCALAR_DOUBLE	PMC1	669314900000
SIMD_FP_256_PACKED_DOUBLE	PMC2	0
...	...	...

- 5% of all instructions are floating point operations

Name	Module	Coverage (%)
PHASIC::Process_Integrator::SelectionWeight(int) const	libPhasicMain.so.0.0.0	5.92
fourpoint_reduction_ol	libopenloops.so	4.5
PHASIC::Vegas::GenerateWeight(double const*) const	libPhasicChannels.so.0.0.0	3.42
__ieee754_log_avx	libm-2.17.so	3.12
__ieee754_pow_sse2	libm-2.17.so	3.07
_int_malloc	libc-2.17.so	2.65
__exp1	libm-2.17.so	2.09

- lots of places for small improvements