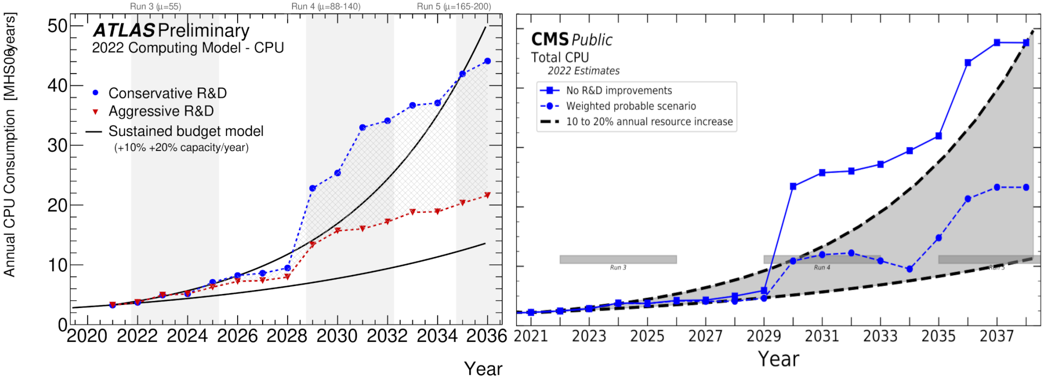
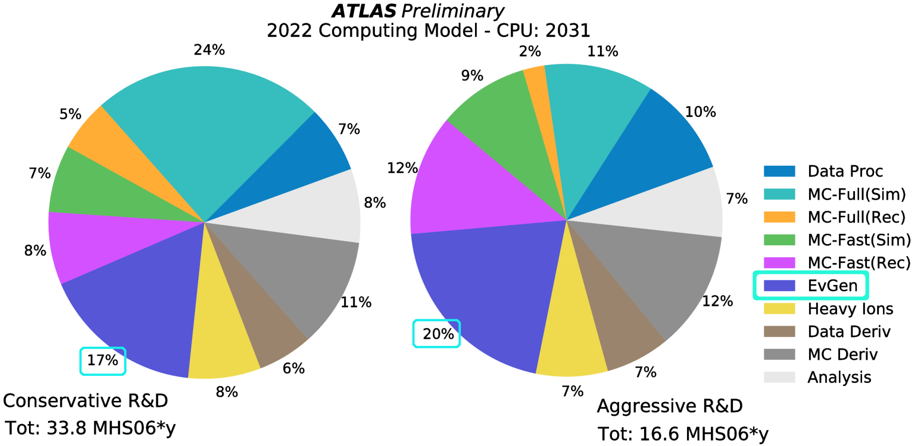
CERN



Projected CPU usage from present to 2036 (ATLAS) and 2038 (CMS).
(CERN-LHCC-2022-005, CMS-NOTE-2022-008)



ATLAS projected CPU usage 2031. (CERN-LHCC-2022-005)

Process	Madevent 262 144 events		
	Total	Momenta+unweight	Matrix elm
$e^+e^- \to \mu^+\mu^-$	17.9 s	10.2 s	7.8 s
+CUDA Tesla A100	10.0 s	10.0 s	0.02s
	1.8 x	1.0 x	390 x
$gg \to t\bar{t}gg$	209.3 s	7.8 s	201.5 s
+CUDA Tesla A100	8.4 s	7.8 s	0.6 s
	24.9 x	1.0 x	336 x
$gg \to t\bar{t}ggg$	2507.6 s	12.2 s	2495.3 s
+CUDA Tesla A100	30.6 s	14.1 s	16.5 s
	82.0 x	0.9 x	151 x

Event generation time comparison, offloading scattering amplitudes on GPU. Measurements are run single-threaded on an AMD EPYC 7313 CPU.

Parallelism for hard event generation in MadGraph5_aMC@NLO

Argonne: Taylor Childers, Walter Hopkins, Nathon Nichols
CERN: Laurence Field, Stephan Hageboeck, Filip Optolowicz, Stefan Roiser, David Smith, Jorgen Teig, Andrea Valassi, Zenny Wettersten
UCLouvain: Olivier Mattelaer
UW-Madison: Carl Vuosalo

What this talk is **not**

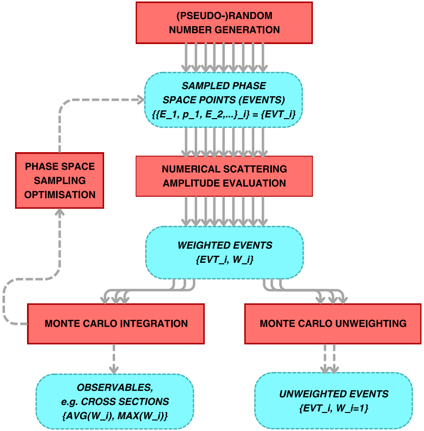
- Official release of the accelerated plugin
- Nitty gritty report on software development
- Detailed analysis on levels of acceleration
- In-depth review of different implementations

What this talk **is**

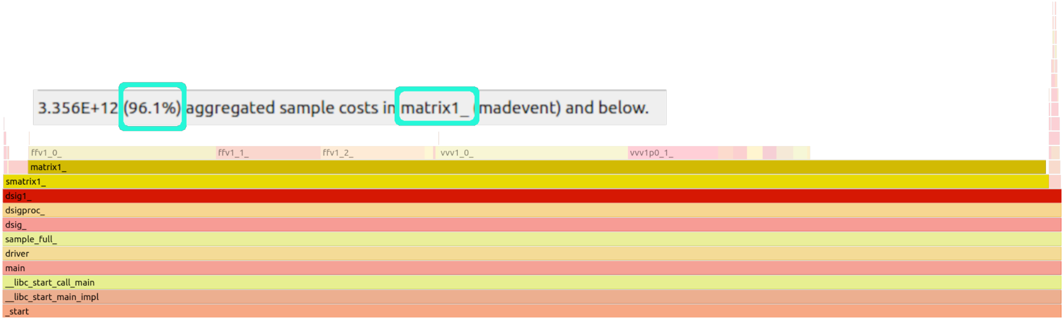
- Brief overview of the philosophy for parallelism
- Status report on current stage of development
  - Focus on CERN team efforts, particularly GPU offloading
- Sidenotes on parallel projects relating to this work
- Project plan for future development
- ...and a call to arms!

  If you would like to test use-cases,
  please contact us!
  And if you're interested in R&D,
  join the event generator acceleration workshop in November!

MadGraph5_aMC@NLO (arXiv:1405.0301)


(PSEUDO-)RANDOM NUMBER GENERATION
SAMPLED PHASE SPACE POINTS (EVENTS) {{E_1, p_1, E_2,...}_i} = {EVT_i}
PHASE SPACE SAMPLING OPTIMISATION
NUMERICAL SCATTERING AMPLITUDE EVALUATION
WEIGHTED EVENTS {EVT_i, W_i}
MONTE CARLO INTEGRATION
MONTE CARLO UNWEIGHTING
OBSERVABLES, e.g. CROSS SECTIONS {AVG(W_i), MAX(W_i)}
UNWEIGHTED EVENTS {EVT_i, W_i=1}

- Hard event generator (MadEvent)
  - Exact kinematics
  - Full scattering amplitude
    - No showering
- Returns unweighted events
  - Properly sampled phase space
  - Events equally important for integration
- Completely analytic
  - Exact formulae for given theory
  - Numeric integration (Monte Carlo)



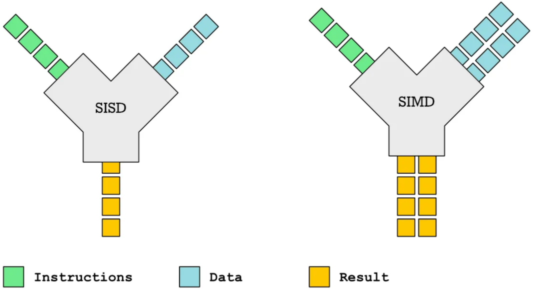
3.356E+12 (96.1%) aggregated sample costs in matrix1_ (madevent) and below.

Performance profile, MadEvent ($gg \to t\bar{t}gg$)

For an introduction to flamegraphs, see e.g. Brendan Gregg.

How do we speed this up?

Data parallelism for CPUs (SIMD)



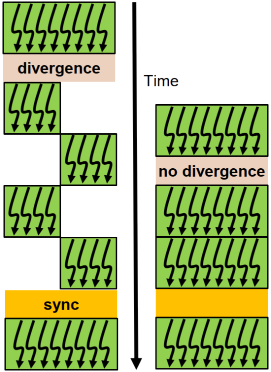
(E. Engheim, ITNEXT)

- Single instruction, single data
  - Run operations sequentially
  - One instruction on one datum
  - → Repetitive tasks take long
- Single instruction, multiple data
  - Vector of data in register
  - Run instruction on entire register
  - → Accelerate repetitive tasks
  - *(No performance cost)*
  - E.g. AVX architectures on Intel, AMD HPC-CPUs

Data parallelism for GPUs (SIMT)

- Simultaneous instruction, multiple threads
  - Many processing units in parallel (threads)
  - Operations on distinct data sets (registers)
  - Simultaneous operations $\Longrightarrow$ **lockstep**
- Lockstep processing
  - Each thread runs same operation at same time
  - Threads "pause" for branching code

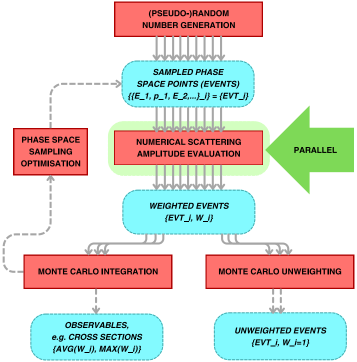
  ⇒ Need to minimise thread divergence when programming SIMT hardware, e.g. GPUs



(NVidia Volta whitepaper, here modified)

Parallel helicity amplitudes

"[E]vent-level parallelism [seems] an appropriate approach" (arXiv:2004.13687)



(PSEUDO-)RANDOM NUMBER GENERATION

SAMPLED PHASE SPACE POINTS (EVENTS) {{E_1, p_1, E_2,...}_i} = {EVT_i}

PHASE SPACE SAMPLING OPTIMISATION

NUMERICAL SCATTERING AMPLITUDE EVALUATION

PARALLEL

WEIGHTED EVENTS {EVT_i, W_i}

MONTE CARLO INTEGRATION

MONTE CARLO UNWEIGHTING

OBSERVABLES, e.g. CROSS SECTIONS {AVG(W_i), MAX(W_i)}

UNWEIGHTED EVENTS {EVT_i, W_i=1}

- Parallelism in event generation
  - Same amplitudes, many phase-space points
  - Same calculations for each one
  - → No divergence between events
- Scattering amplitudes in MadEvent
  - MadEvent generates phase space points
  - Sends events to amplitude evaluation
  - → Evaluate amplitudes in parallel

Parallelised implementations

Two separate codebases

- `CUDACPP` *(subject of this talk)*
  - Vectorised `C++` routines with `CUDA` integration

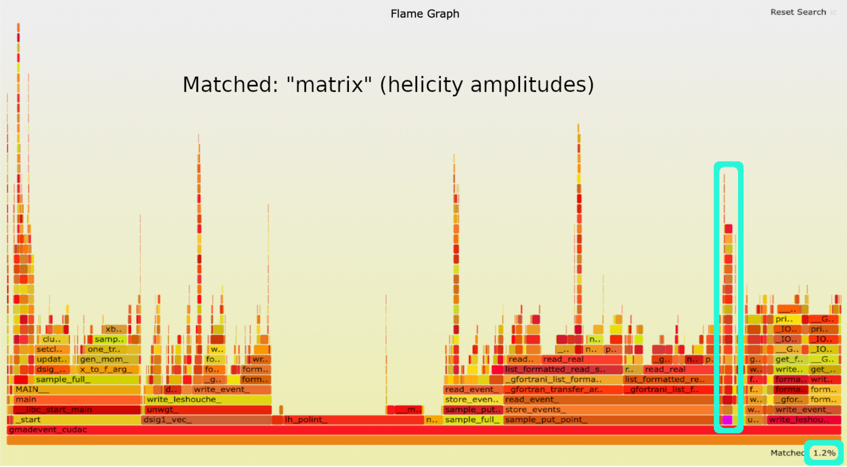
    ⇒ vector CPU output

    ⇒ NVidia GPU output

    ⇒ *(NEW!)* HIP backend (compile time toggle), AMD GPU output
- `SYCL`
  - Tracks `CUDACPP` development, ports it to SYCL portability framework

    ⇒ Intel hardware output (CPU, GPU)

    ⇒ NVidia, AMD GPU output
  - Testing on Intel Aurora cluster



Matched: "matrix" (helicity amplitudes)

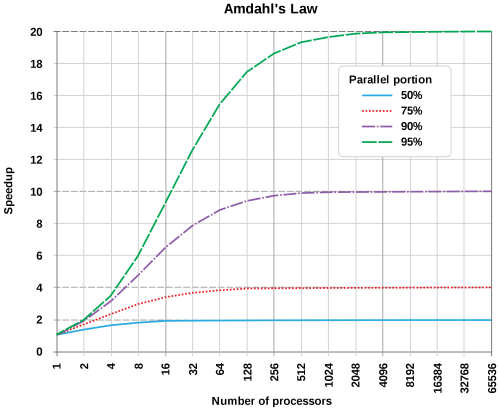
Performance profile, CUDACPP ($gg \to t\bar{t}gg$)

For an introduction to flamegraphs, see e.g. Brendan Gregg.

Amdahl's law

- Acceleration limited by time spent in part of code *not* being parallelised
- Code which originally takes fraction p of total runtime can improve runtime by

$$\text{Acceleration} \leq \frac{1}{1-p}.$$



(Wikimedia)

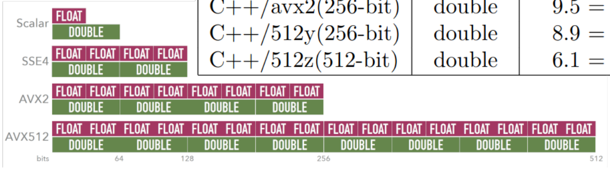
CUDACPP performance, NVidia GPU

Process	Madevent 262 144 events		
	Total	Momenta+unweight	Matrix elm
$e^+e^- \to \mu^+\mu^-$	17.9 s	10.2 s	7.8 s
+CUDA Tesla A100	10.0 s	10.0 s	0.02s
	1.8 x	1.0 x	390 x
$gg \to t\bar{t}gg$	209.3 s	7.8 s	201.5 s
+CUDA Tesla A100	8.4 s	7.8 s	0.6 s
	24.9 x	1.0 x	336 x
$gg \to t\bar{t}ggg$	2507.6 s	12.2 s	2495.3 s
+CUDA Tesla A100	30.6 s	14.1 s	16.5 s
	82.0 x	0.9 x	151 x

Event generation time comparison, offloading scattering amplitudes on GPU.
Measurements are run single-threaded on an AMD EPYC 7313 CPU.
Comparisons are between single-threaded CPU with and without GPU offloading.

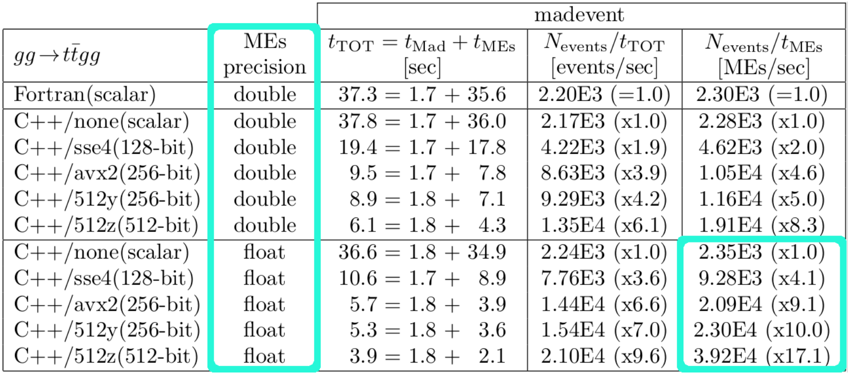
CUDACPP performance, vector CPU

		madevent		
$gg \to t\bar{t}gg$	MEs precision	$t_{\text{TOT}} = t_{\text{Mad}} + t_{\text{MEs}}$ [sec]	$N_{\text{events}}/t_{\text{TOT}}$ [events/sec]	$N_{\text{events}}/t_{\text{MEs}}$ [MEs/sec]
Fortran(scalar)	double	37.3 = 1.7 + 35.6	2.20E3 (=1.0)	2.30E3 (=1.0)
C++/none(scalar)	double	37.8 = 1.7 + 36.0	2.17E3 (x1.0)	2.28E3 (x1.0)
C++/sse4(128-bit)	double	19.4 = 1.7 + 17.8	4.22E3 (x1.9)	4.62E3 (x2.0)
C++/avx2(256-bit)	double	9.5 = 1.7 + 7.8	8.63E3 (x3.9)	1.05E4 (x4.6)
C++/512y(256-bit)	double	8.9 = 1.8 + 7.1	9.29E3 (x4.2)	1.16E4 (x5.0)
C++/512z(512-bit)	double	6.1 = 1.8 + 4.3	1.35E4 (x6.1)	1.91E4 (x8.3)



Event generation time comparison, levels of vectorisation.
Measurements taken on an Intel Gold 6148 CPU.
Comparisons are of compilation with different levels of vectorisation, no further modifications.

CUDACPP performance, float precision

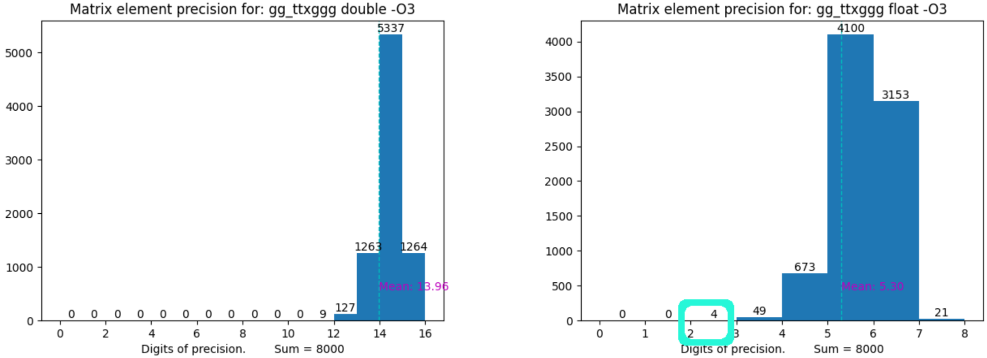


		madevent		
$gg \to t\bar{t}gg$	MEs precision	$t_{\mathrm{TOT}} = t_{\mathrm{Mad}} + t_{\mathrm{MEs}}$ [sec]	$N_{\mathrm{events}}/t_{\mathrm{TOT}}$ [events/sec]	$N_{\mathrm{events}}/t_{\mathrm{MEs}}$ [MEs/sec]
Fortran(scalar)	double	37.3 = 1.7 + 35.6	2.20E3 (=1.0)	2.30E3 (=1.0)
C++/none(scalar)	double	37.8 = 1.7 + 36.0	2.17E3 (x1.0)	2.28E3 (x1.0)
C++/sse4(128-bit)	double	19.4 = 1.7 + 17.8	4.22E3 (x1.9)	4.62E3 (x2.0)
C++/avx2(256-bit)	double	9.5 = 1.7 + 7.8	8.63E3 (x3.9)	1.05E4 (x4.6)
C++/512y(256-bit)	double	8.9 = 1.8 + 7.1	9.29E3 (x4.2)	1.16E4 (x5.0)
C++/512z(512-bit)	double	6.1 = 1.8 + 4.3	1.35E4 (x6.1)	1.91E4 (x8.3)
C++/none(scalar)	float	36.6 = 1.8 + 34.9	2.24E3 (x1.0)	2.35E3 (x1.0)
C++/sse4(128-bit)	float	10.6 = 1.7 + 8.9	7.76E3 (x3.6)	9.28E3 (x4.1)
C++/avx2(256-bit)	float	5.7 = 1.8 + 3.9	1.44E4 (x6.6)	2.09E4 (x9.1)
C++/512y(256-bit)	float	5.3 = 1.8 + 3.6	1.54E4 (x7.0)	2.30E4 (x10.0)
C++/512z(512-bit)	float	3.9 = 1.8 + 2.1	2.10E4 (x9.6)	3.92E4 (x17.1)

Event generation time comparison, levels of vectorisation and double vs float precision.
Measurements taken on an Intel Gold 6148 CPU.
Comparisons are of compilation with different levels of vectorisation, no further modifications.

What about numerical stability?

CADNA measurements for MadEvent

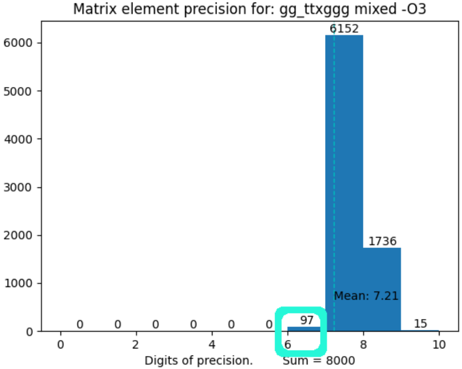


Measurements taken with CADNA over 8 000 phase space points for process $gg \to t\bar{t}\,ggg$.

Mixed precision

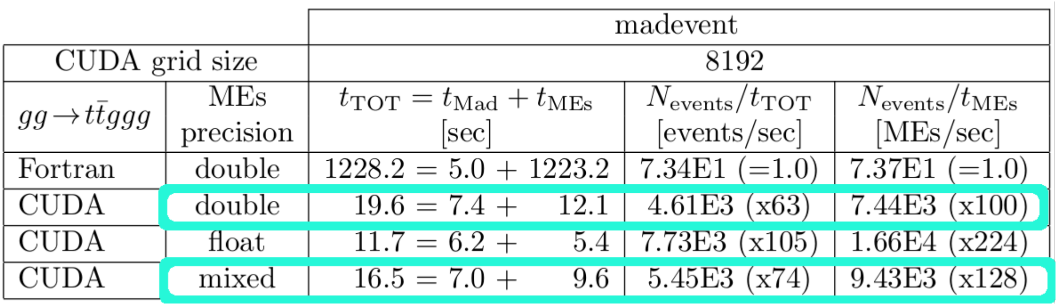
- Helicity amplitudes vary orders of magnitude
- E.g. colour algebra is very consistent in magnitude

⇒ Check stability with only colour algebra in float (*mixed precision*)



Measurements taken with CADNA over 8 000 phase space points for $gg \to t\bar{t}ggg$, mixed precision.

Acceleration from mixed precision



		madevent		
CUDA grid size		8192		
$gg \to t\bar{t}ggg$	MEs precision	$t_{\mathrm{TOT}} = t_{\mathrm{Mad}} + t_{\mathrm{MEs}}$ [sec]	$N_{\mathrm{events}}/t_{\mathrm{TOT}}$ [events/sec]	$N_{\mathrm{events}}/t_{\mathrm{MEs}}$ [MEs/sec]
Fortran	double	1228.2 = 5.0 + 1223.2	7.34E1 (=1.0)	7.37E1 (=1.0)
CUDA	double	19.6 = 7.4 + 12.1	4.61E3 (x63)	7.44E3 (x100)
CUDA	float	11.7 = 6.2 + 5.4	7.73E3 (x105)	1.66E4 (x224)
CUDA	mixed	16.5 = 7.0 + 9.6	5.45E3 (x74)	9.43E3 (x128)

Event generation time comparison, offloading scattering amplitudes on GPU, levels of precision.
Measurements taken single-threaded on an Intel Silver 4216 CPU and an NVidia V100 GPU.
Comparisons are between single-threaded CPU with and without GPU offloading.

How about an official release?

Requirements for a plugin release

- Process $pp \to t\bar{t} + nj, n \in \{0, 1, 2\}$ fully validated
  - Multiprocess, i.e. consists of many distinct particle configurations
  - Each amplitude should match MadGraph to numerical precision
  - Total cross section should match MadGraph to numerical precision
  - Set events (LHEF) should match MadGraph to numerical precision
- Native plugin integration
  - Plugin installed in local MadGraph copy (non-release branch)
  - Use regular MadGraph CLI and syntax (`generate`, `output`, `launch`)
  - Output with plugin amplitude evaluation exporter

Requirements for a plugin release

- Process $pp \rightarrow t\bar{t} + nj, n \in \{0, 1, 2\}$ fully validated
  - Multiprocess, i.e. consists of many distinct particle configurations
  - Each amplitude should match MadGraph to numerical precision
  - Total cross section should match MadGraph to numerical precision
  - Set events (LHEF) should match MadGraph to numerical precision
- Native plugin integration ✓
  - Plugin installed in local MadGraph copy (non-release branch) ✓
  - Use regular MadGraph CLI and syntax ✓
    (`generate`, `output`, `launch`)
  - Output with plugin amplitude evaluation exporter ✓

Status report

- Generic *pp* generation is not quite functional
- ...but most[1] subprocesses do work
- And complex subprocesses show considerable acceleration
- ... but running multiprocess does not trivially exploit it fully

Aside from event-level parallelism, in the process have also:

- Improved unweighting routine (being merged in MadGraph)
- Implemented parallel event reweighting (not fully integrated)

[1] Recent checks indicate full functionality, but validation tests are not yet finished.

...but what about NLO?

NLO event generation in MadGraph

- MadGraph supports generic NLO event generation
- Uses FKS subtraction (arXiv:hep-ph/9512328, arXiv:hep-ph/9706545)
  - Splits amplitudes into Born, real, and loop contributions
  - Born and real contributions are tree-level
  - Loop contributions necessitate integration over undefined momenta
- Output exclusively in `Fortran`

⇒ Can reuse `CUDACPP` tree-level amplitudes

Initial project plan for NLO

Entering planning stages, looking into performance right now

- Treating Born and real amplitudes *(simple but non-trivial)*
  - Call tree-level `C++/CUDA` helicity amplitudes for NLO event generation
  - Run heterogeneously: loops on host; Born and real amplitudes on device
- Treating loops *(longer-term project)*
  - Study fraction of proper loop evaluation vs approximators (tree-level)
  - Study fraction of proper loops in double vs quadruple precision
    - Quadruple precision **not** supported on GPGPUs/TPUs
    - If mostly double precision: Port **one** loop library to GPUs
    - Prioritise GPU loop library, evaluate on CPU when unstable

Summary

Summary

- Official plugin beta release (timeplan: coming months)
  - Guarantee $pp \to t\bar{t} + 0 - 2j$ working and validated
  - Guarantee plugin integration with native MadGraph CLI
- Other gains along the way
  - Optimised unweighting routines
  - Reduce numerical precision in parts without instabilities
  - Helicity amplitudes reused for parallel event reweighting
- Future developments
  - Extend plugin to generic BSM models
  - Parallelise additional parts of MadEvent
  - Starting a project plan for NLO development

A call to arms

- High-performance computing is becoming necessary in HEP
- If you want to test our code on your own hardware
  - Get into contact with us!
- If you want to work with this yourself
  - Sign up for the event generator and N(n)LO code acceleration workshop!

- And if nothing else, start thinking in parallel