A generic reweighting framework

The Rex and teaRex libraries, and their use in the tREX extension to CUDACPP

Zenny Wettersten (CERN, TU Wien)

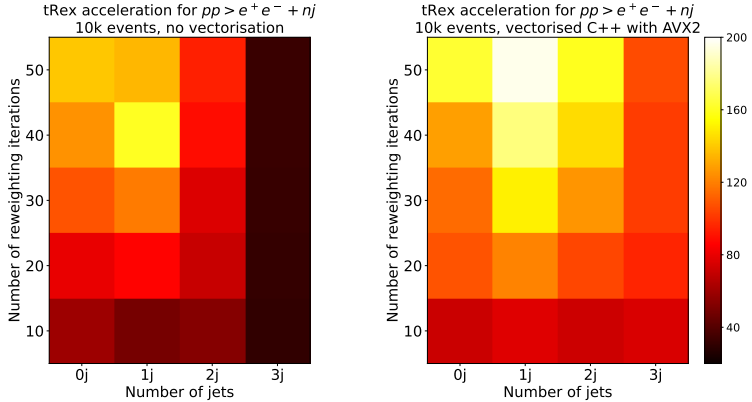
Talk contents

Main point of this talk: `Rex` and `teaRex`

- `Rex`: **R**apid **e**vent e**x**traction
  - C++ library for LHE handling; I/O, internal access, etc.
- `teaRex`: **t**ensorial **e**vent **a**daption with `Rex`
  - Extension to `Rex` for parton-level event reweighting
  - Generic: Function pointers as arguments to reweight over
- tREX: Extension to `CUDACPP` for ME reweighting
  - `CUDACPP` standalone output basis for reweighting
  - Automated SLHA parameter iteration a la MG reweight_cards

Before diving into the algorithms, though...

Reweighting acceleration sneak peek



Measured speedup (t_{MG}/t_{tRex}). No hardware acceleration (left) and AVX2 (right).
For vectorised section, AVX2 provides factor 4 speedup w.r.t. no vectorisation.

Rex and teaRex

Rapid event extraction

- Rex is a a C++ library for working with LHE format files
- Fast I/O and (intended) easy access to LHE data
- At its core: XML parser with specialised LHE functionality
- Transposes between object-oriented (OOP-) LHE and structure of arrays (SoA-)format with user-provided event classifiers
  - E.g. "combine events with same initial and final state particles"

Example: Extracting momenta and weights into SoA

```
REX::lheNode file(lheString); //initialising LHE
REX::lheSoA transpose(file); //transposed structure
//want to extract momenta:
transpose.setRelEvData(REX::relEvArgs().setMomUp(true));
//default event comparison: same external legs
//extracting momenta of all events,
//with events sorted by external legs
transpose.extractEvents();
```

- Goal: Make it this easy to load *and create* LHE files.
  - Going back and forth between OOP- and SoA-formats should be trivial

Rex status and future

- Current status:
  - `Rex` does not have a distinct repository
  - Primarily reads and modifies existing LHE files
  - Written in C++17 with no external dependencies
- In development (pre-1.0 release):
  - Direct transposition between OOP- and SoA-formats
  - Code cleanup and safety mechanisms
- Future possibilities:
  - Separation of data structures and storage formats
    - Future-proofing for new file formats
  - Separation of generic functionality and LHE-specific parts

Interlude: An intro to reweighting

- For Monte Carlo event generation, event weights are given by

$$w = \underbrace{f(p_1, p_2)}_{\text{pdf}} \overbrace{\left|\mathcal{M}\right|^2}^{\text{amplitude}} \, d\Omega.$$

- Factorise: Can e.g. do *matrix element reweighting*:

$$w' = f(p_1, p_2) \left|\mathcal{M}'\right|^2 d\Omega = \frac{w'}{w} w = \frac{\left|\mathcal{M}'\right|^2}{\left|\mathcal{M}\right|^2} w.$$

- (Assumption: New phase space subset of original)

tensorial **e**vent **a**daption with `Rex`

- `teaRex` is an extension to `Rex` for reweighting
- Generic framework for reweighting w.r.t. any LHE data
  - Includes specific implementation for MG-style ME reweighting
- Uses SoA LHE class to sort and map events to "evaluators"
  - Assumes each event belongs to exactly one subprocess
- Takes as input "state" functions and subprocess "evaluators"
  - `iterator`: "State" function, runs before any subprocess evaluation
  - `weightor`: "Evaluation" function, e.g. the amplitude of an event
- Re-evaluates cross sections and runs error propagation

Algorithm sketch

1. Load and sort LHE file as a `reweightor` object
2. Set `iterators` and `weightors`
   - Each subprocess can have multiple `weightors`
3. Run:
   - 3.1 Determine original weight factor (e.g. $w/|\mathcal{M}|^2$)
   - 3.2 Call `iterator`
   - 3.3 For each subprocess, call `weightor`(s)
   - 3.4 Multiply return values element-wise by weight factors
   - 3.5 Repeat 2-4 for each iterator
4. Append weights to LHE file and export

- (Note that `teaRex` sorts events upfront, unlike MG)

teaRex status and future

- Current status:
  - Like Rex, teaRex is not yet readily available
  - Written in C++17 with no external dependencies
  - Generic reweighting functionality exists; not (yet) easily accessible
- In development (pre-1.0 release):
  - Ease of use; more immediate access to functionality
- Future possibilities:
  - Multithreading (over weightors, coalescing for iterator calls)

The tREX plugin

tREX reweighting module

- tREX is an extension to the CUDACPP plugin
  - Pull request open; not yet integrated into main project
- Uses CUDACPP standalone output as base
- Wrappers for comparing and running different subprocesses
  - Sorter; determines whether an event belongs and sorts its particles
  - Bridge; calls and runs scattering amplitude on input events
- Compiles each subprocess as a library
- Writes and compiles executable, linking to these libraries
- Runs the resulting reweight driver

Code sample: `rwgt_driver.cc`, minimal example

```
164  // Get event sorting operators
165  static std::vector<std::function<bool(REX::event&)>> comps = {P1_Sigma_sm_gg_ttx::getComp()};
166  static auto runner = REX::tea::reweightor(lheFile, comps); // Initialise rwgt runner
167  if(!runner.extractEvents()) throw std::runtime_error("Failed to extract events from LHE file.");
168  // Initialise amplitude evaluation bridges
169  static std::vector<rwgt::fBridge> fBridgeVec = {P1_Sigma_sm_gg_ttx::bridgeConstr()};
170  // weightor is an alias for std::function with input procSoA, output std::vector<double>
171  static std::vector<REX::tea::weightor> amplitudes;
172  for( size_t k = 0 ; k < fBridgeVec.size() ; ++k )
173  { // Internal setup for fBridge class, append weightors
174      fBridgeVec[k].init(runner.sortedEvents[k], 32);
175      amplitudes.push_back(fBridgeVec[k].getAmp());
176  }
177  runner.setAmps( amplitudes ); // Set weightors (assuming same order as comps)
178  runner.setOriginator(std::function<bool()>([](){return true;})); // Use existing param_card.dat
179  auto writeSlha = std::function<bool()>([slhaCard, slhaPath](){
180      return REX::filePusher(slhaPath, *slhaCard); // Re-write original param_card.dat
181  });
182  // Set iterators to write new param cards and end by rewriting old one
183  runner.setIterators(rwgtCard.getIterators(slhaPath)).setTerminator(writeSlha);
184  runner.run(); // Run reweighting iterations
185  runner.appendWgts(lheFile, rwgtCard.getProcs(), rwgtCard.getNames()); // Add wgts to LHE file
186  if(!REX::filePusher(outputPath, *lheFile.nodeWriter()))
187          throw std::runtime_error( "Failed to write LHE file." );
```

Quickstart guide

Available in my fork of the CUDACPP repo

- `git clone --recurse-submodules -b tRex_dev git@github.com:zeniheisser/madgraph4gpu.git`
- `cd madgraph4gpu/MG5_aMC/mg5amcnlo`
- Install the plugin (ie link `../../epochX/cudacpp/CODEGEN/PLUGIN/` and `PLUGIN/`)
- `./bin/mg5_aMC`
- `generate PROC`
- `output`, `output madevent_simd`, or `output madevent_gpu`
- `launch`
- `reweight=trex`
- Set your run card and reweight card and run

With this, tREX will automatically generate, output, and run the reweighting executable after event generation. The makefile determines whether to compile for a GPU or vector CPU at compile time.

Caveats

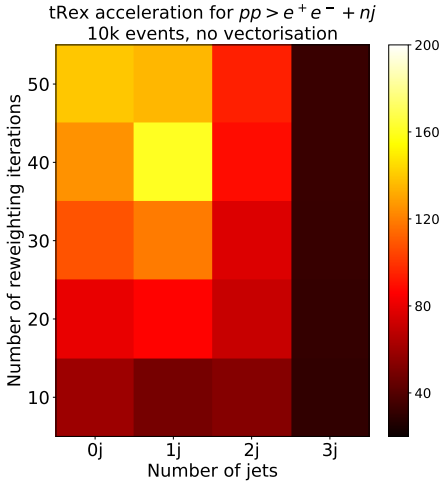
- Only supports single-model reweighting
  - Not a fundamental issue; code generation considers one model at a time and the executable needs to link to all subprocesses
- Only supports helicity-summed/helicity ignorant reweighting
  - `CUDACPP` does not have distinct helicity kernels yet
- Does not amend kinematics for mass reweighting
- (Still has some bugs and lacks safety features)

TREX benchmarks

Measurements and functionality

- TREX can run any process supported by CUDACPP
  - At the moment, all UFO models we have tested (at LO) work
- Although ME reweighting not particularly useful in SM, can be used as first benchmark
- Several things to test:
  - Speedup from running a compiled executable
  - Speedup from CPU vector instructions
  - Speedup from GPU offloading

TREX speedup without hardware acceleration

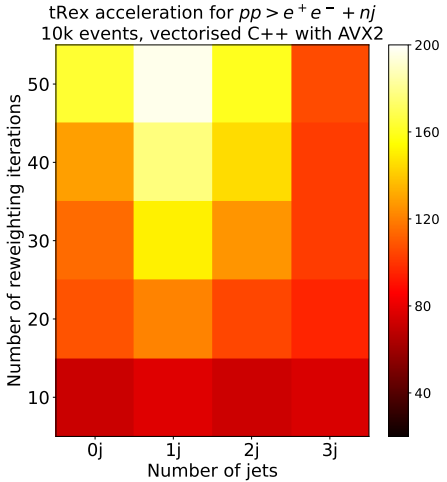


- Reweight Drell-Yan in α_{EW}, SM
- Without vector instructions, TREX 1-2 orders faster
- Two differences to MGaMC
  1. Sorting events before evaluation
  2. Compiled executable
- In computational (complexity) limit, TREX seems to be $\sim 30 - 40\times$ faster than MGaMC reweighting

Runtime ratio $t_{\text{MG}}/t_{\text{TREX}}$, $\sim 30 - 150$.

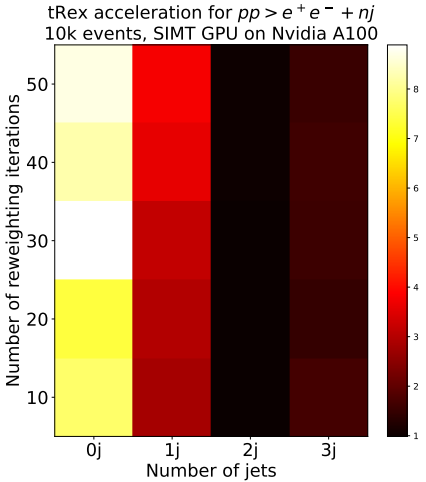
TREX speedup with vector instrections

- AVX2: 256-bit register, up to 4× faster
- For computational bound, AVX2 should be *exactly* 4× faster than scalar instructions
- At $3j$: $2.5 - 3.3\times$ faster than TREX without vectorisation



Runtime ratio $t_{\mathrm{MG}}/t_{\mathrm{TREX}}$, $\sim 80 - 200$.

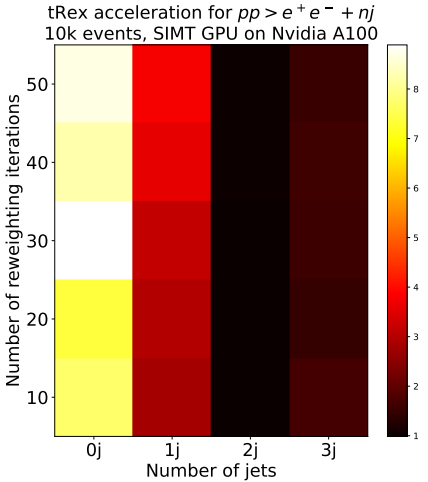
tREX speedup with GPU offloading



- This doesn't look favourable...

Runtime ratio, $t_{\text{MG}}/t_{\text{tREX}}$, $\sim 1 - 10$.

tREX speedup with GPU offloading



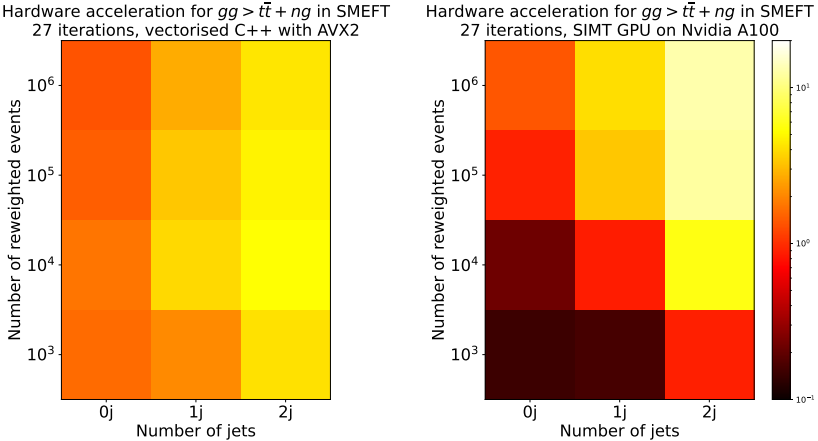
Runtime ratio, $t_{\text{MG}}/t_{\text{TREX}}$, $\sim 1 - 10$.

- This doesn't look favourable...
- ...until you consider the process:
  - 0j: 4 subprocesses
  - 1j: 12 subprocesses
  - 2j: 46 subprocesses
  - 3j: 70 subprocesses
- `teaRex` has no multithreading; one job per subprocess per iteration, running sequentially on the GPU
- 2-3j, each job average $\sim$ 200 events
- Drell-Yan; not computationally heavy
- *Still on par with MGaMC!*

More typical usecase

- Drell-Yan: EW SM process, GPU latency outweighs speedup
- As mentioned: Typically reweight *from* SM *to* BSM
- Consider e.g. SM Effective Field Theory (SMEFT)
  - Parametrisation of generic heavy BSM physics
  - Dozens of free parameters; simulating all combinations impossible
- Consider process $gg \to t\bar{t} + ng$,
  linear and quadratic combinations of related parameters
  - More interesting case: Four-fermion interactions,
    but code size prevents compiling these processes at present
- Running in MGaMC impractical for these measurements;
  compare only TREX with and without hardware acceleration

TREX hardware acceleration for SMEFT



Runtime ratio between TREX without and with hardware acceleration.
At three final-state gluons, compilation fails due to code size.

Conclusions

Summary

- Rex makes for simple handling of LHE files
  - Particularly, Rex allows easy switch between OOP- and SoA-formats
- teaRex provides generic yet accessible reweighting
  - Assuming corresponding functions already exist,
    setting up a teaRex runner takes a couple dozen lines of C++
- tREX combines the CUDACPP plugin with teaRex
  to allow for accelerated matrix element reweighting
  - Compiled execution speeds up reweighting by $\sim 30 - 150\times$
  - Hardware acceleration provides even further speedup
- Currently, code complexity limits tREX usage

Future prospects

- Both `Rex` and `teaRex` have mostly full functionality
  - What remains is code cleanup and additional safety
- `teaRex` could be used as a base for generic reweighting
  - E.g. pdf reweighting would be as simple as passing in libraries
- TREX will benefit from continued `CUDACPP` developments
  - Simplifying compilation will allow for more complex process reweighting
  - Parallelising over helicities enables helicity aware reweighting
- Backporting the TREX program structure to `CUDACPP`?
  - Compiling each parton configuration as a library, not executable
  - Would remove cumbersome communication to subprocesses

CERN