Madgraph4gpu progress and WIP: madevent + cudacpp integration

(i.e. additional plumbing and soldering...)

Andrea Valassi (CERN)

Madgraph on GPU development meeting, 16th May 2022

https://indico.cern.ch/event/1154531

Overview – progress in last three weeks

- Follow up on my April 25 talk - we also had meetings with POWHEG and ATLAS on May 13
- Alphas, basics within cudacpp (gg_tt* and ee_mumu) – PR #434
  - Extend Stefan's cudacpp gg_tt patch, fix tests, fix memory access to couplings (reported on Apr 25)
  - New: backport to code generation (generalized from gg_tt gc10/gc11 to any process-specific couplings)
  - Added event-by-event Gs to fbridge.inc interface (but not yet connected to Fortran MadEvent)
- Alphas, generalize to other processes (uu_dd and EFT gg_h) – PRs #440, #446, #449, #450
  - Consistent (but improvable) framework to handle mix of dependent/independent couplings/parameters
    - Not tested yet: QCD+EW process with both dependent (evt-by-evt) and independent (fixed) couplings, issue #438
  - Tested also clang and icx (fixed memory access to couplings with different SIMD vector extensions)
  - Warning: the handling of model parameters needs several improvements, see issue #448
    - e.g. EFT only ok '#ifdef MGONGPU_HARDCODE_PARAM' (computing couplings from Gs requires other parameters)
- Alphas, connect cudacpp to ALL_G common in fortran MadEvent – PRs #452, #453
  - Event-by-event comparison of fortran ME and cudacpp ME with varying renormalization scale now ~ 1!
  - *WIP (PR #454) on detailed comparison of physics results and of computing performance – see later*
    - WIP also on doing the ME calculation ONLY in cudacpp and not in fortran – will need multichannel API
    - WIP also on improving Makefiles to build both fortran and cudacpp in .mad (eg 'make avxall –j'), issue #400
- Multichannel
  - TODO: integrate Olivier's ME single-diagram factors GPU patch (from standalone gpu)
  - TODO: port code generation to Olivier's latest 340 branch?
- Unweighting
  - TODO: random color, random helicity – should be easy?

WIP *(preliminary!)* – physics comparison

- Started analysing event-by-event ratios of cudacpp to fortran ME – *should be == 1!*
  - Multichannel is disabled for now (compare "ME", not "ME * channel enhancement factor")
  - New: enabled variable renormalization scale (evt-by-evt Gs and alpha QCD), *ratio now ~ 1*

- Deviations of cudacpp/fortran ratio from 1 are non negligible – and asymmetric!
  - eemumu (commit f6cfe83) → (ratio – 1) is in [–9E-16, +7E-16] OK!
  - ggtt (commit f84150e) → (ratio – 1) is in [–7E-05, +5E-06] NOT OK?
  - ggttggg (commit a949146) → (ratio – 1) is in [–4E-05, +6E-04] NOT OK?
  - NB: results above are for double precision cudacpp, deviations are higher for floats
  - NB: cuda and cpp are in good agreement with each other – they both deviate from fortran

- To be understood:
  - Why do large deviations only happen for QCD processes? (bug in my code?...)
    - Related to running of alphas? (Try the same tests without alphas running?)
    - Related to color algebra?
    - Related to different way of computing jamp2 in cudacpp and fortran (+=, +=... vs = + +...)?
  - Is it possible to remove them? Or otherwise are they acceptable by physicists?
  - (Have not tested yet: comparison of cross sections, or of average ME ratio...)

WIP *(preliminary!)* – performance comparison

- A few simple tests of Madevent, with MEs computed both in fortran and in cudacpp
  - Three timing components
    - Fortran: common overhead (random numbers, sampling, event I/O...) – hopefully small!
    - Fortran: ME calculation
    - Cuda or Cpp: ME calculation (TODO: increase Cuda grids, limited to 32 threads per grid now!)

- Preliminary timing measurements (double precision)
  - ggttggg (commit a949146), 1056 MEs
    - 1.5-2.0s overhead + 39.5s fortran ME + 3.7s cpp/512y ME (or 9.2s cuda ME)
    - cpp/512y 2.86E2 MEs/s vs fortran 2.66E1 MEs/s: ***cpp/512y throughput ~10x higher than fortran***
      - Consistent with cpp/512y standalone throughput 2.90E2 in 8769cb7 (should check "bridge" SA throughput)
  - eemumu (commit f6cfe83), 524320 MEs
    - 4.60s overhead + 3.70s fortran ME + 0.10s cpp/512y : cpp/512y 37x fortran, very high overhead
  - ggtt (commit f84150e), 524320 MEs
    - 22.3s overhead + 9.0s fortran ME + 1.3s cpp/512y : cpp/512y 7x fortran, very high overhead

- To be understood:
  - Where is the factor 10x coming from in ggttggg? (forget the simpler 2 to 2 processes)
    - Probably 4x from "512y" vectorization?
    - Probably 2x from fast math?
    - Maybe non-vectorized non-fast-math cpp is also slightly better than fortran now?
    - NB fortran production version can be a factor ~2 faster through helicity recycling (not included here)
  - Can the overhead be reduced further? (1.5-2.0s not negligible with respect to 3.7s)