Progress on DY+jets for CMS

Andrea Valassi
(CERN IT-GOV-ENG)

With many thanks especially to Jin Choi, Olivier Mattelaer, Daniele Massaro!

Madgraph on GPU meeting with CMS, 13th August 2024

https://indico.cern.ch/event/1373474

Overview: follow-up on Jin's reports in July

- Jin reported several issues during the last meetings in July
  - https://indico.cern.ch/event/1373473/ (July 30)
  - https://indico.cern.ch/event/1441554/ (July 26, CMS gen meeting)
  - https://indico.cern.ch/event/1373472/ (July 16)

- Here I describe some followup on those issues (which I linked to github tickets)
  - Also profiting from work and results by Olivier and Daniele (thanks!)

- (1) CMS sees some Floating Point Exceptions in various DY processes
  - Details on https://github.com/madgraph5/madgraph4gpu/issues/942

- (2) CMS sees a discrepancy in DY+4 jets cross section for Fortran vs Cuda/C++
  - Details on https://github.com/madgraph5/madgraph4gpu/issues/944

- (3) CMS sees a speedup for DY+4 jets, but not for DY+3 jets
  - Details on https://github.com/madgraph5/madgraph4gpu/issues/943

(1) Floating Point Exceptions in DY

https://github.com/madgraph5/madgraph4gpu/issues/942

Followup of FPEs in DY

- I initially thought this might be related to SIMD (we saw many FPEs in SIMD code)
  - I asked Jin to do various tests with –O3 and –O flags (thanks Jin!)
  - But it soon was clear that this is not the source of the problem
- Later on I generated and tested some DY processes and I also saw the issue
  - Details: reproducible; at events 11 and 12; also without –O3; *comes from pdf=0 (!?)*
  - Many suggestions by Olivier (thanks!), e.g. check if this comes from a reset after 10 events
  - *Status: reproducible bug, need to follow up* (e.g. I will check this reset after 10 events)
- Work around: must disable FPE crashes to be able to do anything with DY
  - Essentially, comment out or remove “feenableexcept” calls
  - I understand that this is what Jin has done (modifying all code manually?)
  - *For convenience: I added an env variable CUDACPP_RUNTIME_DISABLEFPE*
    - This is in a WIP PR, not yet merged (but Jin ask me if you are interested...)

(2) Cross-section mismatch in DY+4jets

https://github.com/madgraph5/madgraph4gpu/issues/944

XSECS

Least validation

Compatible

	FORTRAN [pb]	CPP [pb]	CUDA [pb]
DY+0j	5704 \pm 10.11	5711 \pm 1.053	5710 \pm 1.484
DY+1j	3539 \pm 8.096	3535 \pm 1.263	3536 \pm 1.442
DY+2j	2228 \pm 3.143	2236 \pm 0.503	2237 \pm 0.4618
DY+3j	1375 \pm 1.265	1387 \pm 0.3515	1385 \pm 0.3288
DY+4j	883.4 \pm 0.3813	845.8 \pm 0.21	843.8 \pm 0.2022

A bit large errors / different xsecs for FORTRAN?

FORTRAN: Original MG
CPP: Vectorized CPU
CUDA: GPU

JIN CHOI 10

Followup of cross-section mismatch in DY+4j

- My doubt is whether the *statistical* (MC) errors quoted are reliable or underestimated
  - We know there is a large *systematic* bias, but this should be the same for all results?
    - Zenny (thanks!) suggests that this is not necessarily the case (each event has a different scale)

- My approach: *use different random numbers and observe the distribution!*
  - I only had time for a first quick test (DY + 0,1,2 jets), results not really conclusive?
    - But my first impression is that the errors are somewhat underestimated – some big outliers
    - https://github.com/madgraph5/madgraph4gpu/issues/944#issuecomment-2271099576
  - *Status: to be followed up...*
    - I need to repeat this for DY+2 alone or DY+3, and with more than 10 data points...

```
more tlau/logs_ppdy012j.mad_fortran/*txt | egrep '(Current est)'
- Current estimate of cross-section: 22604.882597000003 +- 25.69693417269259
- Current estimate of cross-section: 22736.487131999995 +- 26.02223931415431
- Current estimate of cross-section: 22606.672284000004 +- 25.982101016390413
- Current estimate of cross-section: 22680.418818000002 +- 30.296789851771535
- Current estimate of cross-section: 22598.979159 +- 29.095684586947588
- Current estimate of cross-section: 22661.842675000004 +- 28.504426906822836
- Current estimate of cross-section: 22594.760607 +- 25.30150482309723
- Current estimate of cross-section: 22562.885393999994 +- 27.53350228395446
- Current estimate of cross-section: 22783.444705999995 +- 24.879796947884447
- Current estimate of cross-section: 22699.778944 +- 24.883887513199372
```

  - Aside: #959 *new bug found? DY+3j xsection changes by x10 depending on vector_size?*

- NB: Daniele is also doing tests with a different approach (e.g. try SDE flags etc)...

(3) No speedup from SIMD/GPU in DY+3jets?

https://github.com/madgraph5/madgraph4gpu/issues/943

GENERATION TIME

Results

- Producing 100K events w/ single thread

	FORTRAN	CPP	CUDA
DY+2j	80m 10s	59s 2s	40m 2s
DY+3j	130m 51s	153m 46s	101m 25s
DY+4j	never ends (>4000m)	1366m 49s	426m 54s

- Improvement starts with DY+2j, ~x10 faster for DY+4j

JIN CHOI 22

SIMD/GPU speedups – preliminary work

- To follow up on the CMS DY+3jet speed issue I did a lot of (general) preliminary work
  - Condensed summary below – NB these are all WIP PRs (not yet reviewed or merged...)

- (1) *Multi-backend gridpacks*
  - Create gridpacks that contain Fortran, CUDA and all SIMD builds; the madevent executable symlink is updated when running the gridpack (issue #945, WIP PR #948)

- (2) *Profiling infrastructure for python/bash orchestrator of many madevent processes*
  - Special gridpack creation in private "tlau/gridpacks" scripts; modified python scripts keep, parse and aggregate individual madevent logs (issue #957, WIP PR #948)

- (3) *Performance bug fix: compute MEs for only ~16 events during helicity filtering*
  - Only 16 events were used in SIMD to filter good helicities, but MEs were computed for 16k events; now fixed with "compute good helicities only" flag (issue #958, WIP PR #960)
  - Note1: this improves SIMD runs with vector_size=16384; less relevant if vector_size=32
  - Note2 (to do): maybe a similar bug is lurking for CUDA too, but is probably less relevant?

- (4) *More fine-grained profiling of fortran/cudacpp components in a madevent process*
  - Progressively identified all major scalar bottlenecks and added individual timers/counters for all of them (WIP PR #962, generic; WIP PR #946, CMS DY+jets)
  - Note: this also benefits from earlier profiling flamegraphs by Daniele (thanks!)

Tuning fine-grained madevent profiling

- I progressively added individual timers/counters to new distinct code sections
  - Goal: *reduce generic "Fortran Other" contribution to negligible* (say <2% of total time)...
    - ... while taking care to avoid double counting (which would make "Fortran Other" negative)
  - I used a very simple gg to tt process for this exercise (fast MEs, high non-MEs contribution)
    - https://github.com/madgraph5/madgraph4gpu/pull/962#issuecomment-2284597295
  - *NB: the relative weight of each contribution is highly process-dependent!* (see DY later...)

```
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 [COUNTERS] PROGRAM TOTAL                         :    1.0988s
 [COUNTERS] Fortran Other                  (  0 ) :    0.0117s
 [COUNTERS] Fortran Initialise(I/O)        (  1 ) :    0.0697s
 [COUNTERS] Fortran Random2Momenta         (  3 ) :    0.0167s for    16399 events
 [COUNTERS] Fortran PDFs                   (  4 ) :    0.0910s for    32768 events
 [COUNTERS] Fortran UpdateScaleCouplings   (  5 ) :    0.0098s for    16384 events
 [COUNTERS] Fortran Reweight               (  6 ) :    0.0473s for    16384 events
 [COUNTERS] Fortran Unweight(LHE-I/O)      (  7 ) :    0.1488s for    16384 events
 [COUNTERS] Fortran SamplePutPoint         (  8 ) :    0.2702s for    16399 events
 [COUNTERS] CudaCpp Initialise             ( 11 ) :    0.4077s
 [COUNTERS] CudaCpp Finalise               ( 12 ) :    0.0250s
 [COUNTERS] CudaCpp MEs                    ( 19 ) :    0.0010s for    16384 events
 [COUNTERS] OVERALL NON-MEs                ( 21 ) :    1.0979s
 [COUNTERS] OVERALL MEs                    ( 22 ) :    0.0010s for    16384 events
```

Fortran driver initialization (6%):
I/O (read initialization files)

Fortran phase space sampling (2%):
map random numbers to momenta

Fortran PDFs [in dsig1] (9%):
PDF interpolation

Fortran update scales [in dsig1] (1%):
determine coupling scale

Fortran reweight [in dsig1] (5%):
internally, more PDFs and scales
(move to the two above instead?)

CUDA initialization (41%):
initialize GPU (one-off)

CUDACPP finalization (3%):
reset GPU, clean up

preliminary!

Fortran unweight (15%):
I/O (write LHE files)

Fortran sample_put_point (27%):
I/O (update Vegas grids?)

Followup of SIMD/GPU speedups in DY+3j (1)

- I prepared a multi-backend gridpack (vegas optimized in fortran)
  - Then I executed the gridpack on all Fortran and SIMD backends (no CUDA on this node)

- Overall results for the different backends
  - https://github.com/madgraph5/madgraph4gpu/issues/943#issuecomment-2284882990
  - Total time of gridpack including python/back orchestrator
  - Total aggregated time of madevent executables only
  - *First observation: python/bash contribution is negligible (gridpack minus madevent)*
  - *Second observation: I do see a speedup by a factor x2.5 from SIMD!?* To cross check...
    - Note: this includes the helicity filtering fix (but irrelevant for Jin who already uses vector_size=32?)
    - Note: maybe this is using a more recent version of the code with fixes which Jin is missing?

```
pp_dy3j.mad//fortran/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 447.7169 seconds
[madevent COUNTERS]  PROGRAM TOTAL 443.48
pp_dy3j.mad//cppnone/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 448.1598 seconds
[madevent COUNTERS]  PROGRAM TOTAL 443.898
pp_dy3j.mad//cppsse4/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 295.7847 seconds
[madevent COUNTERS]  PROGRAM TOTAL 291.523
pp_dy3j.mad//cppavx2/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 204.7001 seconds
[madevent COUNTERS]  PROGRAM TOTAL 200.453
pp_dy3j.mad//cpp512y/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 201.0406 seconds
[madevent COUNTERS]  PROGRAM TOTAL 196.745
pp_dy3j.mad//cpp512z/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 176.8891 seconds
[madevent COUNTERS]  PROGRAM TOTAL 172.637
```

preliminary!

Followup of SIMD/GPU speedups in DY+3j (2)

- Results of fine-grained madevent profiling
  - The profile is VERY different from that of a simpler gg to tt!
  - Observation 1 (not shown here): *the overall non-ME contribution is identical in all backends*
  - Observation 2: *the scalar bottleneck is phase space sampling! (~50% for AVX512)*
  - Observation 3: *PDFs scalar contribution is important but not dominant! (~5% for AVX512)*

```
pp_dy3j.mad//cpp512z/output.txt
[GridPackCmd.launch] GRIDPCK TOTAL 176.8891 seconds
[madevent COUNTERS]  PROGRAM TOTAL 172.637
[madevent COUNTERS]  Fortran Other 6.5768
[madevent COUNTERS]  Fortran Initialise(I/O) 4.486
[madevent COUNTERS]  Fortran Random2Momenta 93.2907
[madevent COUNTERS]  Fortran PDFs 8.2998
[madevent COUNTERS]  Fortran UpdateScaleCouplings 7.2827
[madevent COUNTERS]  Fortran Reweight 3.7045
[madevent COUNTERS]  Fortran Unweight(LHE-I/O) 4.8719
[madevent COUNTERS]  Fortran SamplePutPoint 8.2892
[madevent COUNTERS]  CudaCpp Initialise 0.3619
[madevent COUNTERS]  CudaCpp Finalise 0.0221
[madevent COUNTERS]  CudaCpp MEs 35.4557
[madevent COUNTERS]  OVERALL NON-MEs 137.181
[madevent COUNTERS]  OVERALL MEs 35.4557
```

preliminary!

Fortran driver initialization (6%):
I/O (read initialization files)

Fortran phase space sampling (2%):
map random numbers to momenta

Fortran PDFs [in dsig1] (9%):
PDF interpolation

Fortran update scales [in dsig1] (1%):
determine coupling scale

Fortran reweight [in dsig1] (5%):
internally, more PDFs and scales
(move to the two above instead?)

Fortran unweight (15%):
I/O (write LHE files)

Fortran sample_put_point (27%):
I/O (update Vegas grids?)

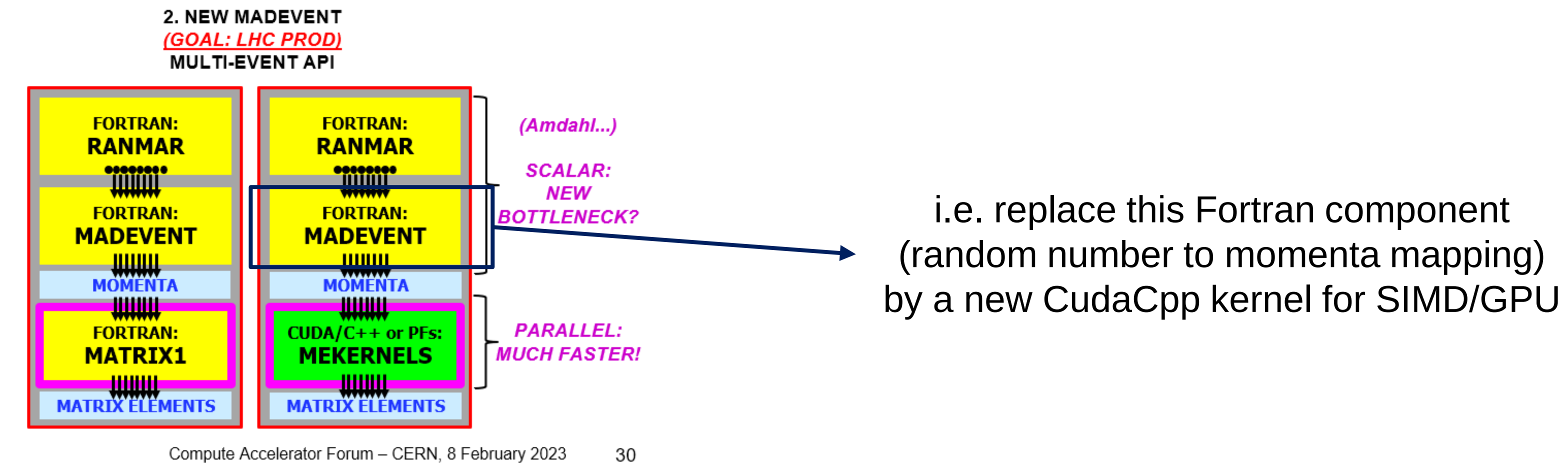
CUDACPP initialization (0%):
initialize Bridge

CUDACPP finalization (0%):
reset Bridge, clean up

Outlook: vectorizing other components

- Further speedup for DY+3 jets would require vectorizing other components
  - (Or speeding them up in much more trivial ways, if low hanging fruits exist...)

- *Phase space sampling (random to momenta mapping) is the first IMO*
  - It represents a very significant fraction (~50% in DY+3 jets with AVX512/zmm)
  - And it should normally be *“easy” to parallelize with lockstep processing?* (few branches)
    - Probably a few months of work, anyway...



- PDFs are certainly another very important component to parallelize
  - Work in this direction already exists and/or is already planned

- Other components
  - Update of coupling scales? Too many branches for lockstep data parallelism?
  - I/O (Vegas grids and LHE files) also need optimization...