Status of PRs towards a release
(and updates on CMS/DY, timers/profiling, sampling...)

Andrea Valassi (CERN)

Madgraph on GPU development meeting, 27th August 2024

https://indico.cern.ch/event/1355159

(include also slides from private discussion with Olivier on August 20 – THANKS OLIVIER!)
(include also slides from the meeting with CMS on August 13 – THANKS JIN!)
(previous update at a proper Madgraph on GPU development meeting was on July 30)

Overview – timeline

- Previous update from me on 30 July
  - Work on master – main pending issue: pp_tt012j xsec mismatch (~Fortran helicity filtering)
  - Work on master_june24 – fix/reimplementation of channelid, pending review and merge
- Interlude: presentation to CMS on 13 August (slides attached at the back)
  - Look at various issues mentioned by CMS on 26 July (speed, xsec mismatch, bugs...)
  - Detailed profiling of where the time is spent e.g. in DY+3 jets
    - Thanks to a lot of infrastructure work: profiling of fortran and python, multi-backend gridpacks etc
- Last week: discussion with Olivier on 20 August (slides also attached at the back)
  - Almost 3-hour discussion, agree on priority order for merging PRs towards a release
- Today: update on the work last week after meeting Olivier – and on the work ahead
  - First: work needed before the release
  - Also: other work (profiling etc) that I want to prepare results for CHEP
    - And other work for later on...

(1) Towards the release

Some ‘easy’ bits last week

As agreed with Olivier last week, do these two before anything else:

- Merged #966 – bug fix for CMS (nvcc installation without nvtx or curand)
  - Let cudacpp.mk find out and act according, no env variables required
- Merged #960 – performance bug fix during helicity filtering in cudacpp
  - Compute MEs only for ~16 events instead of ~16k events, if only 16 are needed!
  - Bug identified during the analysis of CMS DY+3jet performance speed
    - Thanks to the enhancements in gridpack profiling developed to understand CMS DY+3jet

In addition (not discussed with Olivier – I self approved to allow the CI to function):

- Merged #974 – upgrade Mac CI from gfortran-11 to gfortran-14
  - Otherwise all Mac tests were failing on the CI (due to change by github in node config)

Channelid (master_june24)

Merging master and master_june24

- Olivier asked me to look into the merge of master and master_june24
- I created my branch 'june24' in WIP PR #882
  - starting at master_june24
  - with the idea of progressively merging master into it
- To start with, I looked at master_june24 as-is, or with minimal modifications
  - I regenerated all processes with master_june24 codegen (so that the old CI can test them)
    - Processes had not been regenerated with the latest codegen, unlike what we had agreed long ago
  - I included the new CI tmad tests (so that any issues there immediately show up)
    - *NB: tmad tests (cross section and LHE comparisons) were ALREADY available via manual scripts*
- Many issues showed up, including *trivial build errors*, and crashes (see next slide)
  - **My opinion: channelid PR #830 was not sufficiently tested upfront**
    - Some issues may also arise because 'warp' modifications in mg5amcnlo for #765 are incomplete
  - We should agree on a procedure to avoid this happening again in the future...
    - (at the very least: regenerate the code and ensure all CI tests pass...)
- In the meantime, I am working on fixing these issues, so that we can move on...
  - I will come back to Stefan or Olivier when/if I have questions (I already asked some...)

AV – progress in master, early tests of master_june24 8 July 2024 12

Master_june24: following up on three weeks ago

- *I confirm my opinion: PR #830 (Sep 2023 – Jun 2024) was insufficiently tested*
  - *There are issues that could have been spotted with existing tests*
  - *There are new features for which new specific tests should have been added*
  - *There are usage assumptions for which new sanity checks should have been added*
  - *Especially, the SIMD implementation in #830 was almost completely wrong*
  - *Some parts of the code were modified and there was no need for that*
- ***Therefore: I essentially reimplemented channelid from scratch in 2 weeks***

AV – fixes in master, channelid reimplementation in master_june24 30 July 2024 8

- Olivier last week: first big priority (after the easy issues in the last slide) is merging channelid
- PR #882 by AV accepted by OM – changes requested by OM, implemented by AV
  - Fixed tests failing in the new CI, resynced with latest master – Status AV: ready to merge
- My proposed way forward on this – *Olivier is this OK? (I am waiting for a go-ahead)*
  - *1. OM review/accept mg5amcnlo#121 into gpucpp* (NB: forget about "gpucpp_june24"...)
  - 2. AV merge mg5amcnlo#121 into gpucpp *(without squashing! can we disable this?...)*
  - 3. AV merge #882 (branch valassi/june24) into master_june24
  - 4. AV close #830 (same branch valassi/june24) into master
  - 5. AV create/merge PR master_june24 into master (ask OM for review, even if not needed)

Next: Fortran helicity filtering and pp_tt012j

pp_tt012j xsec mismatch – mirror processes

- (#872) Fortran and cudacpp cross sections differ for (gu_ttxgu within) pp_tt012j
- Analysis by AV (with contributions from OM and SR)
  - This only happens for processes with 'mirror processes'
    - It happens for gu_ttxgu within pp_tt012j (mirror is ug_ttxgu – swap g/u from left/right beam protons)
      - It does not happen for gu_ttxgu standalone (g from left beam, u from right beam)
    - It also happens for uux_ttx within pp_tt (now added as a much simpler test)
  - Code signatures: MIRRORPROCS=true in Fortran, nprocesses=2 in cudacpp
    - These must be kept, else the cross section is a factor two off (OM patch #754 August 2023)
    - Note: nprocesses=2 is only used for static asserts in cudacpp, there is no array(2) for this...
  - *Cross sections are not bit-by-bit the same because different numbers of events are processed*
    - Fortran computes helicities twice (once per mirror), cudacpp computes helicities once (overall)
    - Specifically: Fortran helicity recomputation leads to one more RESET_CUMULATIVE_VARIABLE call
- *Fix by AV (under review by OM) in PR #935*
  - *Add one extra RESET_CUMULATIVE_VARIABLE call during cudacpp helicity computation*
    - *IMO, **huge** benefit (fortran and cudacpp xsecs agree bit-by-bit) for no cost (process few events more)*
    - IMO, these bit-by-bit tests are the main reason we have a reasonably solid code now
  - To do (address OM comment): add sanity check that the two fortran helicity lists are identical

AV – fixes in master, channelid reimplementation in master_june24 30 July 2024 5

- AV initial proposal in PR #935 (30 July): add one RESET_CUMULATIVE_VARIABLE
- OM counterproposal in PR #955: remove the second helicity filtering in Fortran!
  - Requires merging gpucpp_goodhel into gpucpp and then fixing cudacpp accordingly
  - En passant, OM also made LIMHEL a runcard parameter – cudacpp integration needed
- Olivier last week: second big priority (after channelid and june24)
- Status AV: agree on the direction, will look at it this week (did not have time yet)

Other issues towards the release
(incomplete list, random order)

Before the release:

- Packaging of cudacpp as a git submodule will be one of the priorities
- Understand and fix FPEs in DY+jets reported by CMS #942
- Check that results are the same with and without vector interfaces #678 (OM)
  - Understand xsec variation with vector_size (32 vs 16384) in DY+3jets #959
- (Check that parameter cards are handled correctly #660)
- ...

Are the following needed before the release?

- Understand xsec mismatch (Fortran vs cudacpp) in DY+4jets reported by CMS #944
- Additional "3rd" CI by OM – PR #865 (still under review by AV, sorry for the delay)
- Sort out various multi-GPU issues from today's meeting with CMS (will open tickets)

(2) For CHEP results – profiling

(follow-up of work done for / with CMS)

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!)

Andrea Valassi – progress on DY+jets for CMS 13 August 2024 8

WIP: 948b

WIP: 948a

Merged: 960

WIP: 962b

WIP: 962a

962a. Low-overhead (rdtsc-based) timers

with subtraction of the estimated overhead

- Changes in timer.h (essentially a new file, but keep the name for simplicity):
  - Rename old timer as ChronoTimer, new API based on ticks, new granularity
  - Add a new rdtsc-based timer, based on reading TSC ticks (faster than chrono) #972
- Changes in timermap.h (check.exe profiling): adapt to new timer.h, default is rdtsc
- Changes in counters.cc (madevent profiling): adapt to new timer.h, default is rdtcs
  - New function names (remove reference to smatrix, use this anywhere) and API
  - Added a way to estimate and subtract the start/stop timer overhead, will become default
- (#962) Status: WIP PR ("prof") exists but also mixes other things, will split it in two:
  - One PR only for the new rdtcs timers/counters/timermap and their usage in other classes
  - Another PR (next slide) for more detailed profiling of madevent components

962a. Low-overhead (rdtsc-based) timers

with subtraction of the estimated overhead

https://github.com/madgraph5/madgraph4gpu/pull/962#issuecomment-2307332171

(DY+3j subprocess with 16k events – profiles include a test timer for sample_get_x which is called 14M times)

```
CUDACPP_RUNTIME_USECHRONOTIMERS=1 \
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 [COUNTERS] *** USING STD::CHRONO TIMERS (do not remove timer overhead) ***
 [COUNTERS] PROGRAM TOTAL                          :    5.3144s
```

Original chrono timers
Program total: 5.3s

```
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 [COUNTERS] *** USING RDTSC-BASED TIMERS (do not remove timer overhead) ***
 [COUNTERS] PROGRAM TOTAL                          :    4.4766s
```

New rdtsc timers
Program total: 4.5s

```
CUDACPP_RUNTIME_REMOVECOUNTEROVERHEAD=1 \
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 INFO: COUNTERS overhead :    0.0338s for 1M start/stop cycles
 [COUNTERS] PROGRAM TOTAL+COUNTEROVERHEAD          :    4.8244s
 [COUNTERS] PROGRAM COUNTEROVERHEAD                :    0.8905s
 ----------------------------------------------------------------
 [COUNTERS] *** USING RDTSC-BASED TIMERS (remove timer overhead) ***
 [COUNTERS] PROGRAM TOTAL                          :    3.9339s
```

New rdtsc timers
Subtract the estimated overhead
Program total: 3.9s

```
CUDACPP_RUNTIME_REMOVECOUNTEROVERHEAD=1 CUDACPP_RUNTIME_DISABLECALLTIMERS=1 \
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 INFO: COUNTERS overhead :    0.0333s for 1M start/stop cycles
 [COUNTERS] PROGRAM TOTAL+COUNTEROVERHEAD          :    4.1897s
 [COUNTERS] PROGRAM COUNTEROVERHEAD                :    0.3330s
 ----------------------------------------------------------------
 [COUNTERS] *** USING RDTSC-BASED TIMERS (remove timer overhead) ***
 [COUNTERS] PROGRAM TOTAL                          :    3.8567s
```

New rdtsc timers
Subtract the estimated overhead
Disable the 14M+ test timer calls
Program total: 3.9s
(i.e. overhead subtraction is good enough)

962b. Fine-grained madevent Fortran profiling

```
CUDACPP_RUNTIME_REMOVECOUNTEROVERHEAD=1 \
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 INFO: COUNTERS overhead :     0.0338s for 1M start/stop cycles
 [COUNTERS] PROGRAM TOTAL+COUNTEROVERHEAD            :     4.8244s
 [COUNTERS] PROGRAM COUNTEROVERHEAD                  :     0.8905s
 --------------------------------------------------------------
 [COUNTERS] *** USING RDTSC-BASED TIMERS (remove timer overhead) ***
 [COUNTERS] PROGRAM TOTAL                            :     3.9339s
 [COUNTERS] Fortran Other                  (  0 ) :     0.2954s
 [COUNTERS] Fortran Initialise(I/O)        (  1 ) :     0.0674s
 [COUNTERS] Fortran PhaseSpaceSampling     (  3 ) :     2.7332s for  1087437 events
 [COUNTERS] Fortran PDFs                   (  4 ) :     0.1003s for    32768 events
 [COUNTERS] Fortran UpdateScaleCouplings   (  5 ) :     0.1688s for    16384 events
 [COUNTERS] Fortran Reweight               (  6 ) :     0.0507s for    16384 events
 [COUNTERS] Fortran Unweight(LHE-I/O)      (  7 ) :     0.0695s for    16384 events
 [COUNTERS] Fortran SamplePutPoint         (  8 ) :     0.0924s for  1087437 events
 [COUNTERS] CudaCpp Initialise             ( 11 ) :     0.4692s
 [COUNTERS] CudaCpp Finalise               ( 12 ) :     0.0263s
 [COUNTERS] CudaCpp MEs                    ( 19 ) :     0.0357s for    16384 events
 [COUNTERS] TEST    SampleGetX             ( 21 ) :     1.8723s for 14136681 events
 [COUNTERS] OVERALL NON-MEs                ( 31 ) :     3.8982s
 [COUNTERS] OVERALL MEs                    ( 32 ) :     0.0357s for    16384 events
```

En passant, note:

- phase space sampling is 75% of total with CUDA MEs *(DY+3j subproc gux_taptamggux)*

- See details in CMS slides two weeks ago (tuned until "Fortran Other" is small)
- (#962) Status: WIP PR ("prof") exists but will strip this off to a separate PR
- To do as discussed with Olivier:
  - add the profiling sections in upstream mg5amcnlo Fortran and protect them with #ifdef's
  - i.e. disable fine-grained profiling unless *users choose to enable profiling in the runcards*

948a. Aggregated madevent python profiling

```
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
```

En passant, note:

- phase space sampling is >50% of total with CUDA MEs *(DY+3j overall)*

- time spent in python+bash is negligible with respect to Fortran

- See details in CMS slides two weeks ago (keep, parse, aggregate madevent logs)
- (#948) Status: WIP PR ("grid") exists but mixes other things, might split it in two:
  - (948a) aggregated madevent profiling; (948b, next slide) multi-backend gridpacks
- To do as discussed with Olivier:
  - add the profiling commands in upstream mg5amcnlo python and make them optional
  - i.e. disable fine-grained profiling unless *users choose to enable profiling in the runcards*
    - (use the same setting as for fine-grained madevent profiling 962b)

948b. Multi-backend gridpacks

```
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
```

En passant, note:
phase space sampling is
>50% of total with CUDA MEs

(DY+3j overall)

but could also do this in CUDA, see the next slide...

- See CMS slides two weeks ago (pre-build all backends, optimize Vegas in Fortran)
- (#948) Status: WIP PR ("grid") exists but mixes other things, might split it in two:
  - (948a, previous slide) aggregated madevent profiling; (948b) multi-backend gridpacks
- To do as discussed with Olivier:
  - (keep current default: multi-backend disabled unless users enable it in the runcards)
  - clarify how symlinks are re-created for a new backend after untarring the gridpack
    - or maybe add one backend parameter to the run.sh script?
    - or maybe add a madevent script that switches between backends, instead of a symlinks? #693
    - (bottom line: choosing a backend to run a gridpack is a run-time choice, not a build-time choice...)

948b. Multi-backend gridpacks... PS!

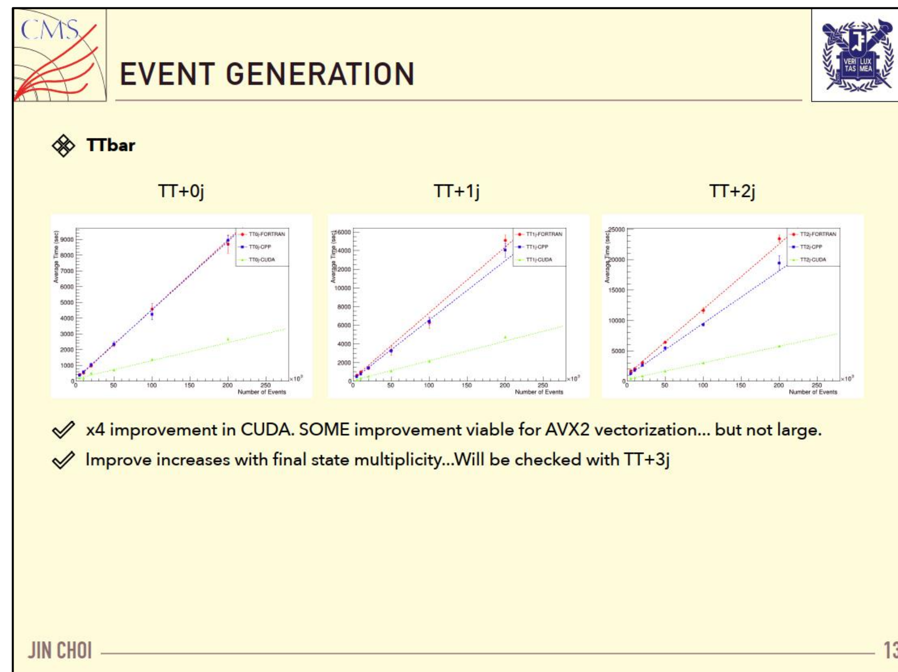
GRIDPACK PRODUCTION

TTbar

	nb_core = 16 FORTRAN	nb_core = 16 CPP	nb_core = 16 CUDA	nb_core = 12 CUDA - H100
TT+0j	5m 47s	7m 15s	4m 41s	-
TT+1j	11m 8s	10m 43s	7m 7s	-
TT+2j	74m 52s	38m 25s	21m 47s	-
TT+3j	> 119h...(19%)	> 19h (6%)	nb_core = 6 8h 11m	4h 53m
TT+0123j	> 118h...(20%)	31h 6m	8h 24m	4h 52s

- Improvements observed throughout the whole processes - x2 for CPP / x3.5 for CUDA for TT+2j
- Expecting huge improvements in TT+3j/0123j!
- Only 6 madevents possible to be submitted for TT+3j/0123j - gg →ttxggg takes ~ 6GB GPU memory
- Additional test with 12 madevents using H100 (~ 96 GB)
- Super-fast gridpack production viable if multi-gpu supports available! (H100 x 4 ~ 384 GB)
- Madevents in TT+3j possesses 2~6 GB for GPU memory → could it be allocated dynamically? e.g. Check the remaining memory in GPU and submit the madevent...

JIN CHOI 9

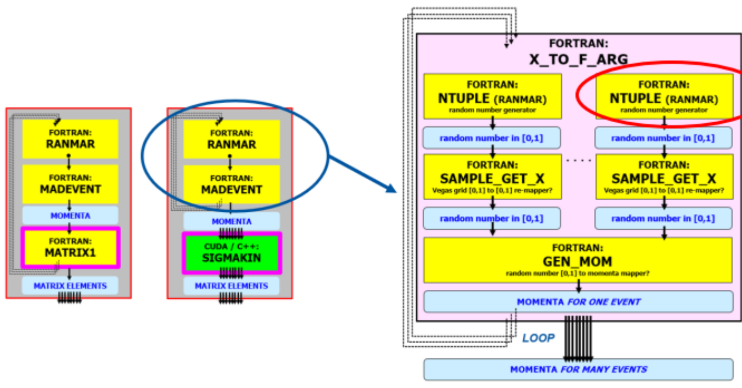


- See *Jin's slides at the CMS meeting earlier today* (https://indico.cern.ch/event/1373475)
  - Event generation in cpp/fortran not tested yet because gridpack creation is not done yet...
- Possible solution? (for CMS tests before CHEP, not for physics production yet...)
  - Create multi-backend gridpacks using CUDA (i.e. Vegas optimization using CUDA MEs)
  - Most of the O(100-1000) hours in fortran/cpp gridpack creation are Vegas optimization
    - The software builds are also slow but not the bottleneck: expect to create gridpacks in O(10) hours
  - *Question for OM: increase priority and merge this PR soon so that CMS can test this?*

(3) For CHEP or beyond – sampling improvements

(follow-up of work done for CMS)

Improving phase space sampling?



NB the name ntuple originally referred to a quasi-MC function in htuple.f, but this is no longer used!

(ntuple is now just a ranmar call!)

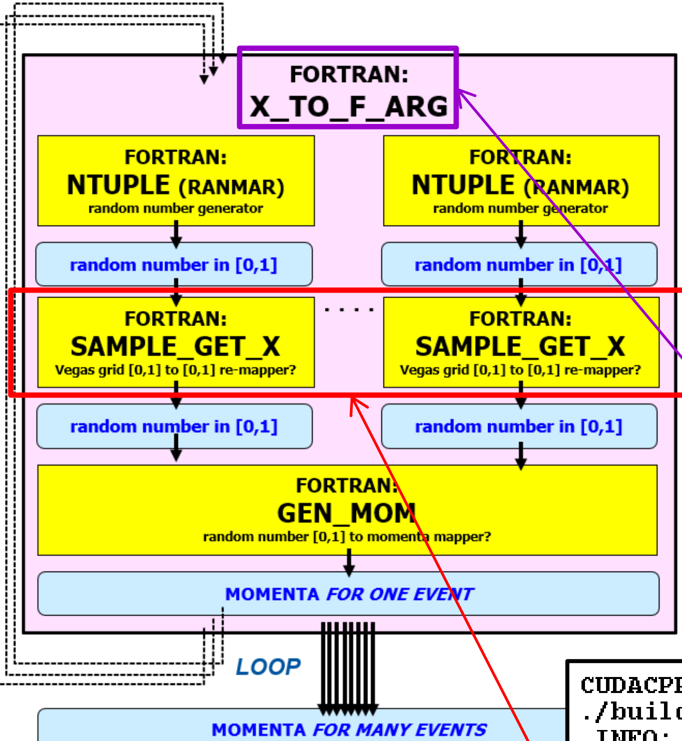
- What goes inside phase space sampling (x_to_f_arg) is more or less the above...
  - one x_to_f_arg (calling one gen_mom internally) *for each event*
  - which internally calls one sample_get_x (calling ranmar-based ntuple) *for each particle*
    - *sample_get_x is the bottleneck (around or more than 50% of phase space sampling for DY+3j?)*

- Largely speaking, two (or three) strategies forward
  - low hanging fruits: trivial improvements are possible in sample_get_x
  - vectorization and GPU port of the whole phase space sampling chain
  - (but does Madnis completely replace the sample_get_x internal code?)

Andrea Valassi – CMS DY+jets, timers/profiling, first sampling improvements   20 August 2024   4

- Next three slides:
  - sample_get_x profiling numbers
  - picking up some low hanging fruits
  - some thoughts on vectorization

Profiling sampling: sample_get_x



One DY+3j subprocess *gux_taptamggux*

- Sampling (x_to_f_arg) ~ 75% of total
  - Within that: sample_get_x ~ 50% of total
    - Within that: xbin takes a large fraction

```
CUDACPP_RUNTIME_REMOVECOUNTEROVERHEAD=1 \
./build.cuda_d_inl0_hrd0/madevent_cuda < /tmp/avalassi/input_ggtt_x1_cudacpp
 INFO: COUNTERS overhead :     0.0338s for 1M start/stop cycles
 [COUNTERS] PROGRAM TOTAL+COUNTEROVERHEAD            :     4.8244s
 [COUNTERS] PROGRAM COUNTEROVERHEAD                  :     0.8905s
 ----------------------------------------------------------------------
 [COUNTERS] *** USING RDTSC-BASED TIMERS (remove timer overhead) ***
 [COUNTERS] PROGRAM TOTAL                            :     3.9339s
 [COUNTERS] Fortran Other                  (  0 ) :     0.2954s
 [COUNTERS] Fortran Initialise(I/O)        (  1 ) :     0.0674s
 [COUNTERS] Fortran PhaseSpaceSampling     (  3 ) :     2.7332s for  1087437 events
 [COUNTERS] Fortran PDFs                   (  4 ) :     0.1003s for    32768 events
 [COUNTERS] Fortran UpdateScaleCouplings   (  5 ) :     0.1688s for    16384 events
 [COUNTERS] Fortran Reweight               (  6 ) :     0.0507s for    16384 events
 [COUNTERS] Fortran Unweight(LHE-I/O)      (  7 ) :     0.0695s for    16384 events
 [COUNTERS] Fortran SamplePutPoint         (  8 ) :     0.0924s for  1087437 events
 [COUNTERS] CudaCpp Initialise             ( 11 ) :     0.4692s
 [COUNTERS] CudaCpp Finalise               ( 12 ) :     0.0263s
 [COUNTERS] CudaCpp MEs                    ( 19 ) :     0.0357s for    16384 events
 [COUNTERS] TEST    SampleGetX             ( 21 ) :     1.8723s for 14136681 events
 [COUNTERS] OVERALL NON-MEs                ( 31 ) :     3.8982s
 [COUNTERS] OVERALL MEs                    ( 32 ) :     0.0357s for    16384 events
```

Time estimates from rdtcs, after subtracting the estimated timer overhead (should be reliable enough?...)

Low hanging fruits?

Low hanging fruits in sample_get_x

- Identified and fixed a couple of simple possible changes
  - the xbin() function called by sample_get_x is one of the bottlenecks: avoid it!
  - non-controversial(?) changes
    - (1) xbin is very often (not always) called with the same arguments, e.g. 0 or 1: cache it!
    - (2) xbin is sometimes called in dead or repeated code, avoid those calls
  - more controversial(?) changes
    - (3) expensive xbin calls take place in some internal checks to issue warnings: are these needed?
      - I have the impression this code is not completely functional anyway... (e.g. warning counters look strange)
  - some nice gains from (1) and especially (3)... will give details another time
- Are other improvements possible in the xbin function?
  - I had no time to look at this in more detail than caching it or avoiding it...
    - internals look reasonable, there is a binary tree search... but maybe can be improved?

Andrea Valassi – CMS DY+jets, timers/profiling, first sampling improvements 20 August 2024 5

#946 Status: WIP PR exists, to be rediscussed with Olivier

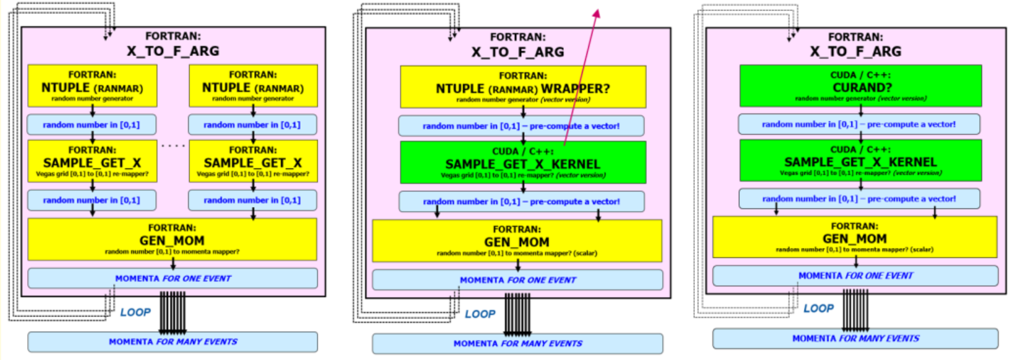
```
For the cuda backend is now, skipping xbin checks #968
Phase space sampling in dy+3j has decreased from 78s to 53s (down by 30%)
> [GridPackCmd.launch] GRIDPCK TOTAL                 135.1144
> [madevent COUNTERS]  PROGRAM TOTAL                 130.8140s
> [madevent COUNTERS]  Fortran PhaseSpaceSampling     53.0338s for   44652395 events
> ...
> [madevent COUNTERS]  CudaCpp MEs                    35.4908s for    1769472 events
> [madevent COUNTERS]  OVERALL NON-MEs                95.3232s
> [madevent COUNTERS]  OVERALL MEs                    35.4908s for    1769472 events

For the cuda backend was, including xbin checks but including trivial improvements #969
Phase space sampling in dy+3j has decreased from 93s to 78s (down by 15%)
< [GridPackCmd.launch] GRIDPCK TOTAL                 160.1718
< [madevent COUNTERS]  PROGRAM TOTAL                 155.8605s
< [madevent COUNTERS]  Fortran PhaseSpaceSampling     78.1023s for   44652395 events
< ...
< [madevent COUNTERS]  CudaCpp MEs                    35.4320s for    1769472 events
< [madevent COUNTERS]  OVERALL NON-MEs               120.4290s
< [madevent COUNTERS]  OVERALL MEs                    35.4320s for    1769472 events

For the cuda backend was in 2e59eca00, without trivial improvements
< [GridPackCmd.launch] GRIDPCK TOTAL                 176.8891
< [madevent COUNTERS]  PROGRAM TOTAL                 172.6370s
< [madevent COUNTERS]  Fortran Random2Momenta         93.2907s for   44651014 events
< ...
< [madevent COUNTERS]  CudaCpp MEs                    35.4557s for    1769472 events
< [madevent COUNTERS]  OVERALL NON-MEs               137.1806s
< [madevent COUNTERS]  OVERALL MEs                    35.4557s for    1769472 events
```

946b. (issue 968)
"More controversial" changes?
Save an additional 30%

946a. (issue 969)
"Non-controversial" changes?
Save 15%

https://github.com/valassi/madgraph4gpu/commit/348664c66d90f47d1d9e6fd72d7dd7f4b0fa7cff

Vectorizing phase space sampling?



- I had a first quick look at possibly vectorizing sample_get_x
  - these are relatively short functions with simple operations, it is not rocket science
    - API: could start by preparing baskets and then looping internally as Olivier did for MEs
  - *the main problem I see is that there are many COMMON's making this stateful*
    - can the hidden inputs/outputs requiring these COMMON's be avoided?
    - or can these hidden inputs/outputs be moved outside the event/particle loop?

Andrea Valassi – CMS DY+jets, timers/profiling, first sampling improvements 20 August 2024 6

- IMO: should clarify two things (OM) before doing more work in this direction
  - 1. is there any hope of removing the need for Fortran commons?
    - especially those that seem to analyse each individual event and keep a state somewhere
  - 2. would sample_get_x continue to exist when Madnis is introduced?
    - otherwise, gen_mom ([0,1] to momenta) may be more relevant than sample_get_x ([0,1] to [0,1])

CMS DY+jets, timers/profiling, phase space sampling improvements

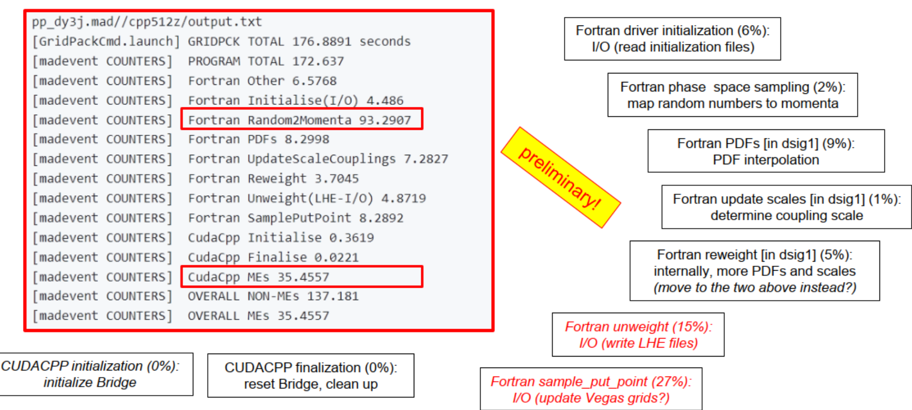
Andrea Valassi (CERN)

Madgraph on GPU development (discussion with Olivier), 20th August 2024

(Quick update on the work last week since the CMS meeting – CMS slides are included at the back)

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)*



Andrea Valassi – progress on DY+jets for CMS 13 August 2024 11

Overview:
follow-up on the meeting with CMS last week

- Phase space sampling is the bottleneck in DY+3j for CUDA – see next slides
  - Had a first quick look at the internals (some trivial improvements, possible strategies...)
  - This required more profiling, which has an overhead: developed lower overhead timers

- Other ‘minor’ new issues
  - New issue #965 in CMS user support, nvcc installed without nvtx: *PR #966 ready/approved*
  - New issue #971 in Mac CI, Fortran not installed, *pending* (github changed its node config?)

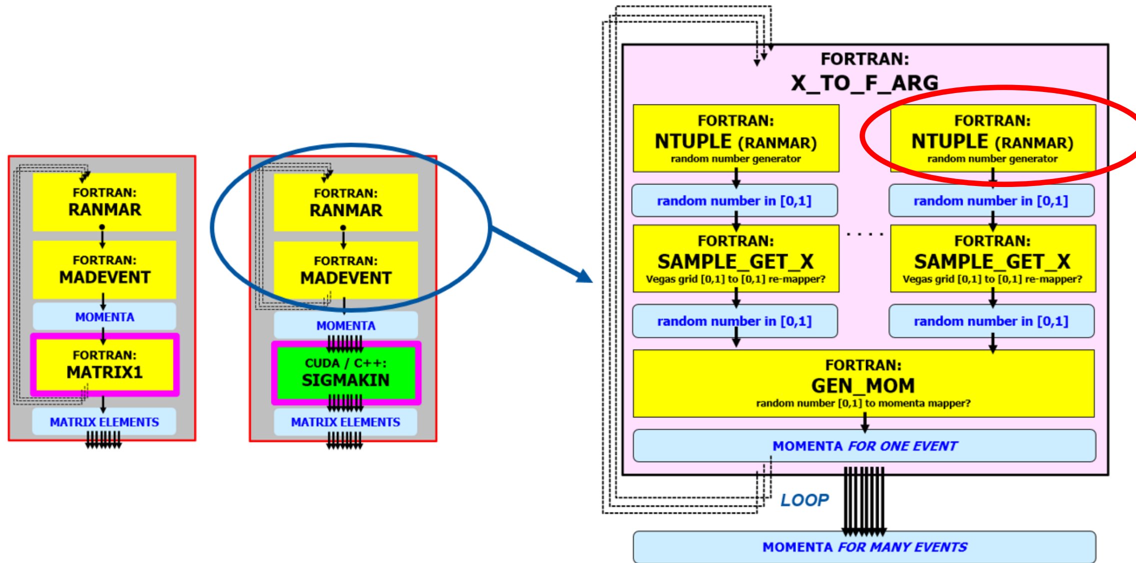
Low-overhead (rdtsc-based) timers

- Timers based on std::chrono have some overhead
  - (Even if now much less than we had in the past due to O/S issues #116)
  - This overhead is especially obvious with very heavy profiling (e.g. for sample_get_x)
- I developed new timers based on rdtsc #972 (ready in PR #962)
  - Individual timers in code instrumentation just read TSC ticks, which is very fast
  - The calibration from TSC ticks to time is only done once at the end
    - All relevant updates completed in timermap.h (for check_sa.cc) and counters.cc (for fortran code)
  - Chrono based timers also have a new API and internal implementation (nanosec ticks)
  - Main file is still called timer.h but is essentially a brand new timing machinery
- See this commit in PR #970 for CMS DY+3jet phase space sampling studies

```
[COUNTERS] *** USING STD::CHRONO TIMERS ***
[COUNTERS] PROGRAM TOTAL                         :    4.7930s
[COUNTERS] Fortran Other                  (  0 ) :    0.1701s
[COUNTERS] Fortran Initialise(I/O)        (  1 ) :    0.0672s
[COUNTERS] Fortran Random2Momenta         (  3 ) :    3.5324s for  1170103 events
[COUNTERS] Fortran PDFs                   (  4 ) :    0.1024s for    49152 events
[COUNTERS] Fortran UpdateScaleCouplings   (  5 ) :    0.1323s for    16384 events
[COUNTERS] Fortran Reweight               (  6 ) :    0.0525s for    16384 events
[COUNTERS] Fortran Unweight(LHE-I/O)      (  7 ) :    0.0647s for    16384 events
[COUNTERS] Fortran SamplePutPoint         (  8 ) :    0.1415s for  1170103 events
[COUNTERS] CudaCpp Initialise             ( 11 ) :    0.4695s
[COUNTERS] CudaCpp Finalise               ( 12 ) :    0.0258s
[COUNTERS] CudaCpp MEs                    ( 19 ) :    0.0346s for    16384 events
[COUNTERS] TEST    SampleGetX             ( 21 ) :    2.0375s for 15211307 events
[COUNTERS] OVERALL NON-MEs                ( 31 ) :    4.7584s
[COUNTERS] OVERALL MEs                    ( 32 ) :    0.0346s for    16384 events
```

```
[COUNTERS] *** USING RDTSC-BASED TIMERS ***
[COUNTERS] PROGRAM TOTAL                         :    3.9808s
[COUNTERS] Fortran Other                  (  0 ) :    0.1248s
[COUNTERS] Fortran Initialise(I/O)        (  1 ) :    0.0676s
[COUNTERS] Fortran Random2Momenta         (  3 ) :    2.7899s for  1170103 events
[COUNTERS] Fortran PDFs                   (  4 ) :    0.1042s for    49152 events
[COUNTERS] Fortran UpdateScaleCouplings   (  5 ) :    0.1327s for    16384 events
[COUNTERS] Fortran Reweight               (  6 ) :    0.0504s for    16384 events
[COUNTERS] Fortran Unweight(LHE-I/O)      (  7 ) :    0.0652s for    16384 events
[COUNTERS] Fortran SamplePutPoint         (  8 ) :    0.1165s for  1170103 events
[COUNTERS] CudaCpp Initialise             ( 11 ) :    0.4685s
[COUNTERS] CudaCpp Finalise               ( 12 ) :    0.0261s
[COUNTERS] CudaCpp MEs                    ( 19 ) :    0.0349s for    16384 events
[COUNTERS] TEST    SampleGetX             ( 21 ) :    1.6663s for 15211307 events
[COUNTERS] OVERALL NON-MEs                ( 31 ) :    3.9459s
[COUNTERS] OVERALL MEs                    ( 32 ) :    0.0349s for    16384 events
```

Improving phase space sampling?



NB the name ntuple originally referred to a quasi-MC function in htuple.f, but this is no longer used!

(ntuple is now just a ranmar call!)

- What goes inside phase space sampling (x_to_f_arg) is more or less the above...
  - one x_to_f_arg (calling one gen_mom internally) *for each event*
  - which internally calls one sample_get_x (calling ranmar-based ntuple) *for each particle*
    - *sample_get_x is the bottleneck (around or more than 50% of phase space sampling for DY+3j?)*

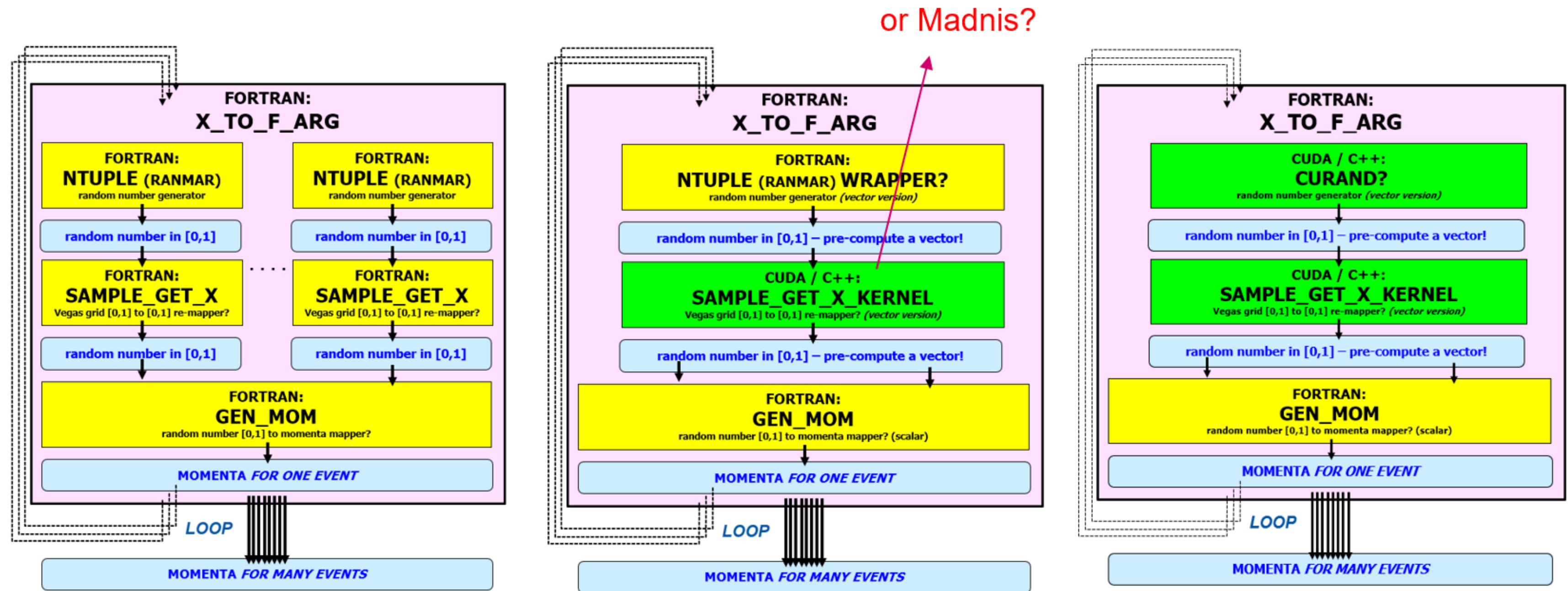
- Largely speaking, two (or three) strategies forward
  - low hanging fruits: trivial improvements are possible in sample_get_x
  - vectorization and GPU port of the whole phase space sampling chain
  - (but does Madnis completely replace the sample_get_x internal code?)

Low hanging fruits in sample_get_x

- Identified and fixed a couple of simple possible changes
  - the xbin() function called by sample_get_x is one of the bottlenecks: avoid it!
  - non-controversial(?) changes
    - (1) xbin is very often (not always) called with the same arguments, e.g. 0 or 1: cache it!
    - (2) xbin is sometimes called in dead or repeated code, avoid those calls
  - more controversial(?) changes
    - (3) expensive xbin calls take place in some internal checks to issue warnings: are these needed?
      - I have the impression this code is not completely functional anyway... (e.g. warning counters look strange)
  - some nice gains from (1) and especially (3)... will give details another time

- Are other improvements possible in the xbin function?
  - I had no time to look at this in more detail than caching it or avoiding it...
    - internals look reasonable, there is a binary tree search... but maybe can be improved?
    - maybe even vectorized, but this clearly includes heavy branching... lockstep seems difficult

Vectorizing phase space sampling?



- I had a first quick look at possibly vectorizing sample_get_x
  - these are relatively short functions with simple operations, it is not rocket science
    - API: could start by preparing baskets and then looping internally as Olivier did for MEs
  - *the main problem I see is that there are many COMMON's making this stateful*
    - can the hidden inputs/outputs requiring these COMMON's be avoided?
    - or can these hidden inputs/outputs be moved outside the event/particle loop?

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...

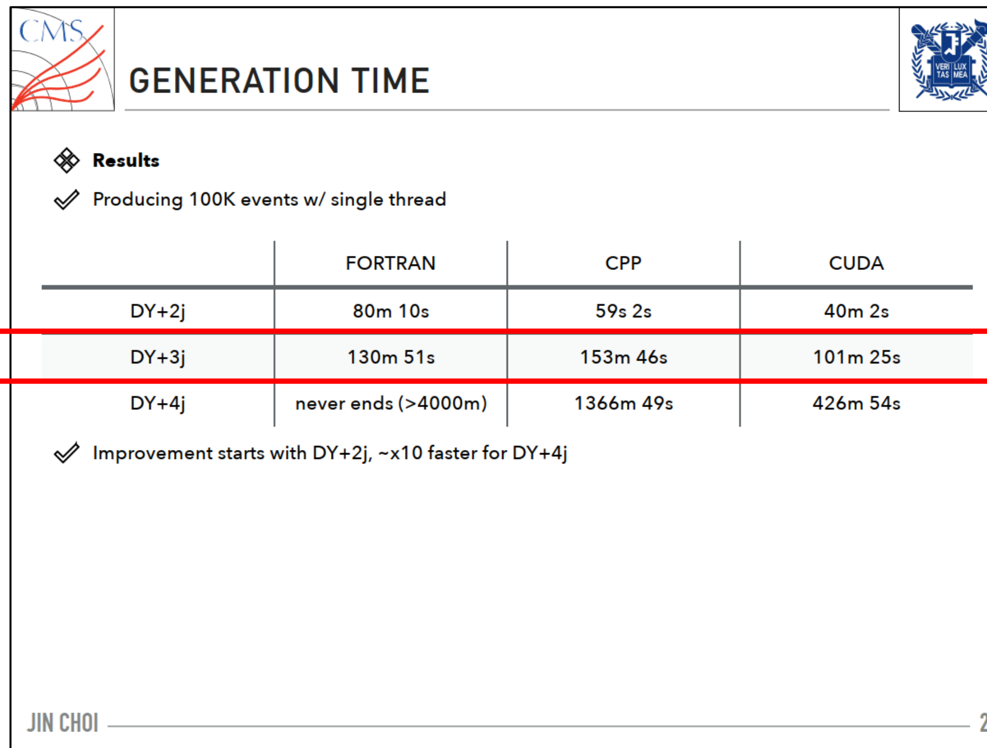
```
more tlau/logs_ppdy012j.mad_fortran/*txt | egrep '(Current est)'
- Current estimate of cross-section: 22604.882597000003 +- 25.69693417269259
- Current estimate of cross-section: 22736.487131999995 +- 26.02223931415431
- Current estimate of cross-section: 22606.672284000004 +- 25.982101016390413
- 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



CMS

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)

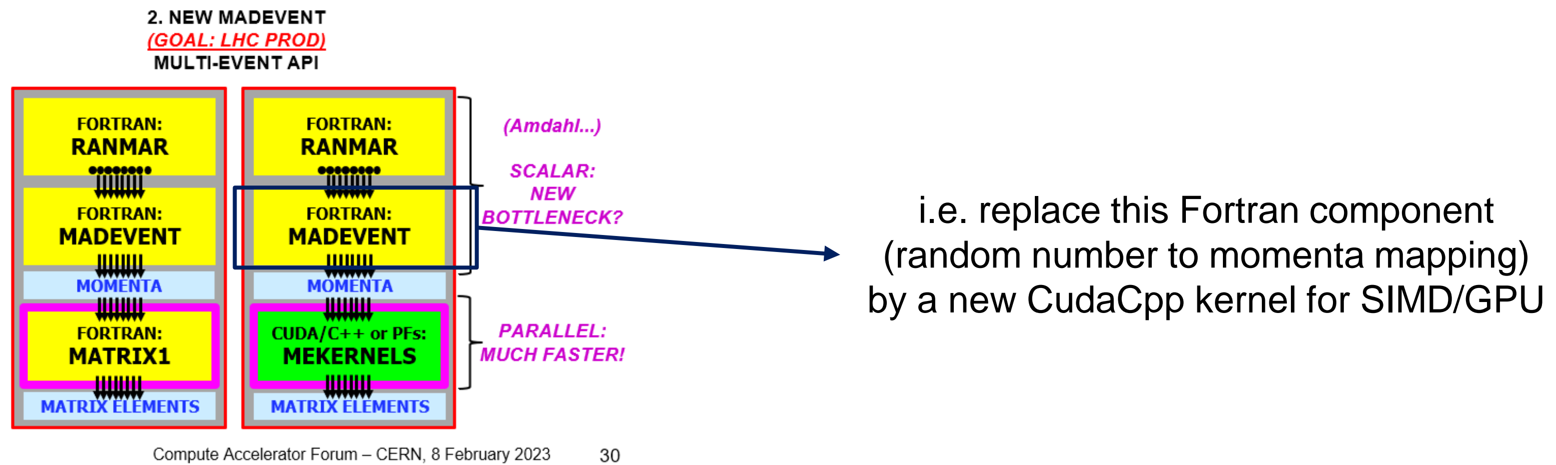
Fortran sample_put_point (27%):
I/O (update Vegas grids?)

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...