



Status of PRs towards a release

(and a few other things)

Andrea Valassi (CERN)

(THANKS to Olivier for the team work on all these PRs!)

Madgraph on GPU development meeting, 3rd September 2024

<https://indico.cern.ch/event/1355160>

(previous update was last week on August 27 – only mentioning changes since then)

(1) Towards the release

Merging master and master_june24

Master_june24:
following up on three weeks ago

- Olivier asked me to look into the merge of master and master_june24
- I looked my branch june24 in WP PR #832
- with the aim of progressively merging master in it
- To start with, I looked at master_june24 with its existing modifications
- Implementation of new features and of new sanity checks, so that the CI can catch them
- I looked for new CI tests that are not in master but in master_june24
- All these tests were either not implemented or already available as regular tests
- My opinion: channelid PR #832 was not sufficiently tested against
- These issues are implemented with dependencies implemented in #832 and in master
- For the CI, I will merge the tests and the code of CI tests in #832
- In the meantime, I am working on fixing these issues, so that we can move on.
- I will come back to Olivier in a few weeks to see whether a strategy emerges.

AV - Fees in master, channelid reimplement in master_june24 30 July 2024 8

- 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

Channelid (master_june24)

Channelid (master_june24)

MERGED!

- 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)*
 - OM review/accept [mg5amcnlo#121](#) into gpucpp (NB: forget about “gpucpp_june24”...)
 - AV merge [mg5amcnlo#121](#) into gpucpp (*without squashing! can we disable this?...*)
 - AV merge #882 (branch valassi/june24) into master_june24
 - AV close #830 (same branch valassi/june24) into master
 - AV create/merge PR master_june24 into master (ask OM for review, even if not needed)

A. Valassi – status of PRs (plus CMS/DY, timers/profiling, sampling...) 27 August 2024 5

- Olivier two weeks ago: first big priority is merging channelid
- Status: FINALLY MERGED!** – In detail
 - 1. Merged [mg5amcnlo#121](#) (branch valassi_gpucpp_june24) into gpucpp
 - 2. Merged [#882](#) (branch valassi/june24) into master_june24
 - 2. Merged [#985](#) (branch master_june24) into master
- And now... must merge/rebase everything else with the gpucpp+master baseline*
 - This is where some conflicts and issues start to appear...*

Next: Fortran helicity filtering and pp_tt012j

Next: Fortran helicity filtering and pp_tt012j

pp_tt012j xsec mismatch – mirror processes

(8)(2) Fortran and cudaccpp cross sections differ for (pp_tt012j within) pp_tt012j

- Analysis by AV (with contributions from OM and DY)
 - This only happens for processes with "mirror processes"
 - It happens for pp_tt012j (mirror to gg_tt012j) – needs gas from whitlight beam problem
 - It happens for pp_tt012j (mirror to gg_tt012j) – needs gas from whitlight beam problem
 - It happens for pp_tt012j (mirror to gg_tt012j) – needs gas from whitlight beam problem
- Code signature: `MIRRORPROC=true` in Fortran, `mirrorproc=2` in cudaccpp
 - These must be kept, and the cross section is a factor of 2 off OM value (22, August 2023)
 - Note: `mirrorproc=2` only used for data points in cudaccpp, there is no `mirrorproc=1` for this
- Cross sections are not the same because different numbers of events are processed
 - Fortran computes helicity twice (once per mirror), cudaccpp computes helicity once (overall)
 - Specify Fortran helicity reweighting factor in one more `RESET_CUMULATIVE_VARIABLE` call
- Fix by AV (under review by OM) in PR #935
 - Add a new `RESET_CUMULATIVE_VARIABLE` call during cudaccpp helicity computation
 - Call `pp_tt012j` in Fortran and cudaccpp twice (once for helicity, once for the real process) to avoid mismatch
 - Note: this is not the real reason we have a mismatch with our code
 - To do: check if `CF` is needed and add sanity checks that the two helicity helicity helicity are identical

AV - Status of PRs (plus CMS/DY, timers/profiling, sampling...)

- 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 `cudaccpp` accordingly
 - En passant, OM also made `LIMHEL` a runcard parameter – `cudaccpp` 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)

A. Valassi – status of PRs (plus CMS/DY, timers/profiling, sampling...) 27 August 2024 6

- Olivier two weeks ago: second big priority (after `channelid`/`june24`)
- Status AV: PR #986 (branch `goodhel`) based on Olivier's 955 is ready and tested
 - With its `mg5amcnlo` counterpart #137 (branch `valassi_goodhel`)
 - Note: this includes many bits of the upgrade to 3.6.0 (but Olivier has more... next slide)
- WIP now: update this to the latest `gpucpp/master` including `june24`
 - First issue: merge conflicts (maybe better solved by Olivier's `gpucpp_for360`, next slide)
 - Second issue: with my WIP version, I get a 0.1% cross-section mismatch #991



Next: Olivier's gpucpp_for360

- Olivier this week: third big priority (after channelid/june24 and goodhel)
 - Our cudacpp release is meant to be in v3.6.0...
- This is a series of patches that are needed on top of june24 and goodhel
 - They fix issues resulting from the update to 3.6.0 (in goodhel) and the interplay with the rest
- Status: WIP WIP
 - Done AV/OM: merged gpucpp_goodhel [#138](#) into gpucpp_for360
 - To do: more conflict resolution, on the mg5amcnlo side
 - To do: and then, the integration with the cudacpp side

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)

Other issues towards the release

Daniele's talk

done, three issues

done

(fixci branch, not 865)



(2) Miscellanea

Build times: from templates to linked objects

- Just some quick tests after a discussion at the meeting last week
 - WIP PR [#978](#) – reusing bits and pieces of previous work for splitting kernels

HELINL=0 (default) aka "templates with moderate inlining".

This has templated helas functions FFV. The templates are in the memory access classes, i.e. essentially the template specialization depends on the AOSOA format used for momenta, wavefunctions and couplings. The sigmakin and calculate_wavefunction functions in CPPProcess.cc use these templated FFV functions, which are then implemented (and possibly inlined). The build times can be long, because the same templates are reevaluated all over the place, but the runtime speed is good.

HELINL=1 aka "templates with aggressive inlining".

This is the mode that I had introduced to mimic `-f1to` i.e. link time optimizations. The FFV functions (and others) are inlined with `always_inline`. This significantly increases the build times because in practice it does the equivalent of link time optimizations (while compiling CPPProcess.o). The runtime speed can get a significant boost for simple processes, where data access is important but the speedups tend to decrease for complex processes, where arithmetic operations dominate. In a realistic madevent environment, this is probably not interesting: for simple processes, it can be interesting, but the ME calculation is outnumbered by non-ME fortran parts and so it is not interesting to have faster MEs; in complex processes, the build times become just too large.

HELINL=L aka "linked objects".

This is the new mode I introduced here. The FFV functions are pre-compiled for the appropriate templates into .o object files. A technical detail: the HelAmps.cc file is common in Subprocess, but it must be compiled in each P* subdirectory, because the memory access classes may be different: for instance, a subprocess with 3 final state particles and one with 4 particles have different AOSOA, hence different memory access classes. My tests so far show that **the build times can decrease/improve by a factor two, while the runtime can increase/degrade by around 10%** for complex processes. (More detailed studies should show if it is the cuda or c++ build times that improve, or both). This is work that goes somewhat in the direction of splitting kernels and that I imagined in that context, but it is not exactly the same. It may become interesting for users especially for complex processes, and especially as long as the non-ME part is still important (eg DY+3j where cuda ME becomes 25% and sampling non-ME is over 50%, there having a ME that is 10% slower is acceptable).

To do: test build times separately for cuda and each SIMD mode

*Quick test using HELINL=L mode: does gg to ttgg (2 to 6) become more manageable?
Preliminary answer: NO unfortunately*