Status of PRs towards a release
(and a few other things)

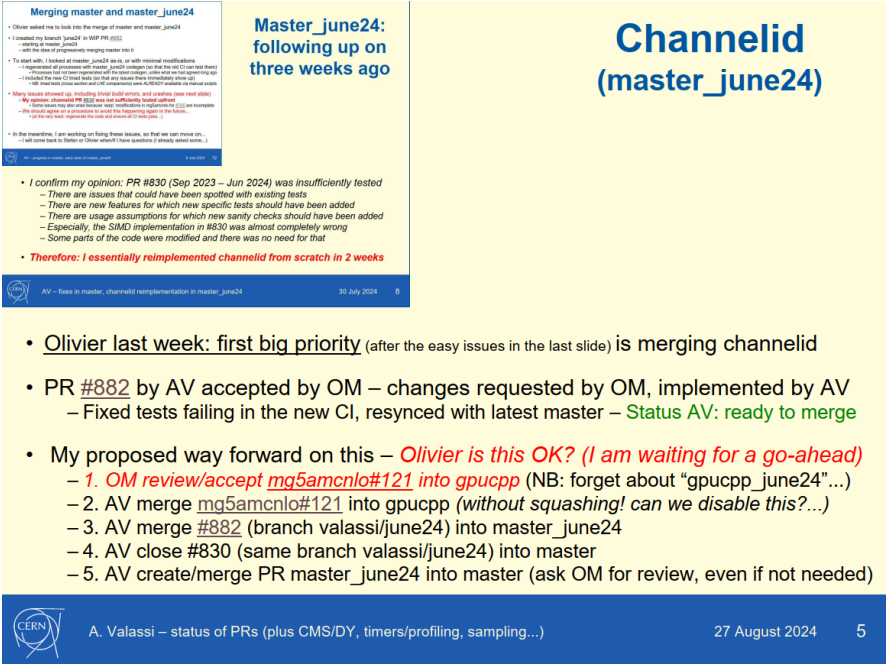
Andrea Valassi (CERN)

(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

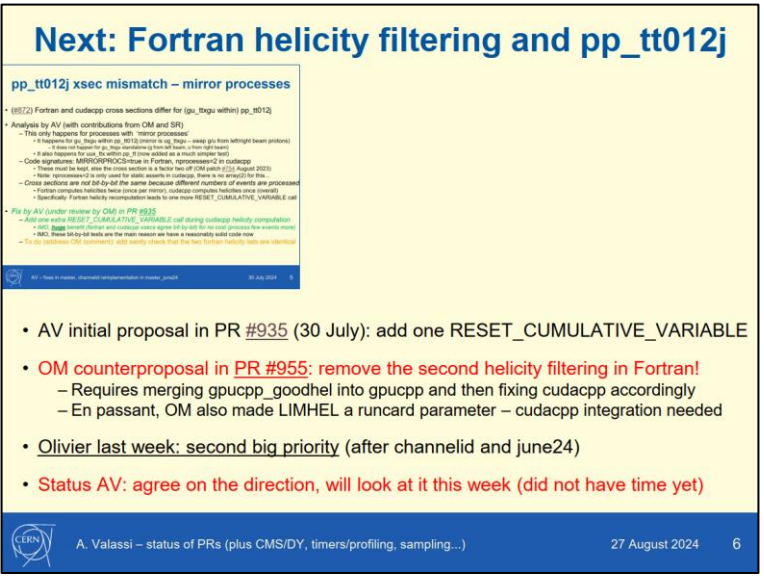


Channelid (master_june24)

MERGED!

- 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



- 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 gpuccp_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 `-flto` 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 equiavelent of link time optimizations (while compiling CPPProcess.o). The runtime speed can get a signifcant 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 ineresting, 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 ttggg (2 to 6) become more manageable?
Preliminary answer: NO unfortunately