Madgraph4gpu WIP: compilers, sycl::vec, ggttgggg

Andrea Valassi (CERN)

Madgraph on GPU development meeting, 7th March 2023

https://indico.cern.ch/event/1262942/

Compilers

- As reported two weeks, moved from CentOS7 to Alma9 and upgraded to CUDA12.0
  - Using CUDA12.0 and gcc11.3 (native Alma9) as baseline, was CUDA11.7 and gcc11.2
  - Took the opportunity to also check a few new compilers: gcc12.1, clang14, icpx2023
- gcc has no surprises: gcc11.3 and gcc12.1 are very similar to gcc11.2
  - Note: aggressive inlining is generally worse than the default no inlining
- clang14 is a bit better than gcc without inlining?
  - And it is especially x1.5 to x2 better with aggressive inlining (-flto emulation) for ggttgg
- icpx2023 (based on clang16) is very similar to clang14, or very slightly better?
  - With aggressive inlining it gives the same extra speedups as clang14
  - (Note: dpcpp not supported anymore? “Please use icpx –fsycl instead”...)
- A few comments
  - There is clearly work to do on the C++ implementation in cudacpp
    - inlining gives a speedup sometimes, but not always
    - inlining gives much longer build times (I never attempted ggttggg, only ggttgg at most)
    - profiling/disassembling gcc vs clang may be interesting
  - The extra x1.5/x2 speedups in inlined icpx are compatible to those observed by Nathan?
    - the speed of the sycl implementation comes from both the code and the compiler I imagine

A quick look at sycl::vec

- I understand Nathan's SIMD implementation uses sycl::vec
  - This type looks remarkably close to the compiler vector extensions (CVE) used in cudacpp
- I checked, and sycl::vec is indeed based on clang compiler vector extensions
  - Using CVEs directly should give the same performance
  - And more programming flexibility (e.g. for mixed double/float types in the color matrix)

A quick look at ggttgggg

- Again motivated by some interesting comments by Nathan two weeks ago
  - I added the generation of ggttgggg (2 to 6 process) to cudacpp
- Code generation:
  - standalone ggttgggg for cudacpp was successfully generated (took 10-15 minutes?)
  - madevent ggttgggg could not be generated (out of memory IIRC)
- Code build (for standalone ggttgggg):
  - gcc builds all crash (probably out of memory, but it seems like a bug worth reporting)
  - clang builds without inlining succeed (36 hours?), tests of the code are successful
  - clang builds with inlining crash (within one hour?), probably out of memory
  - cuda builds just take too long (I killed them after 7 days...)
- I assume that code generation and build will be easier when we split kernels...
  - Many small functions (many small files?!) rather than one 32MB source code function...