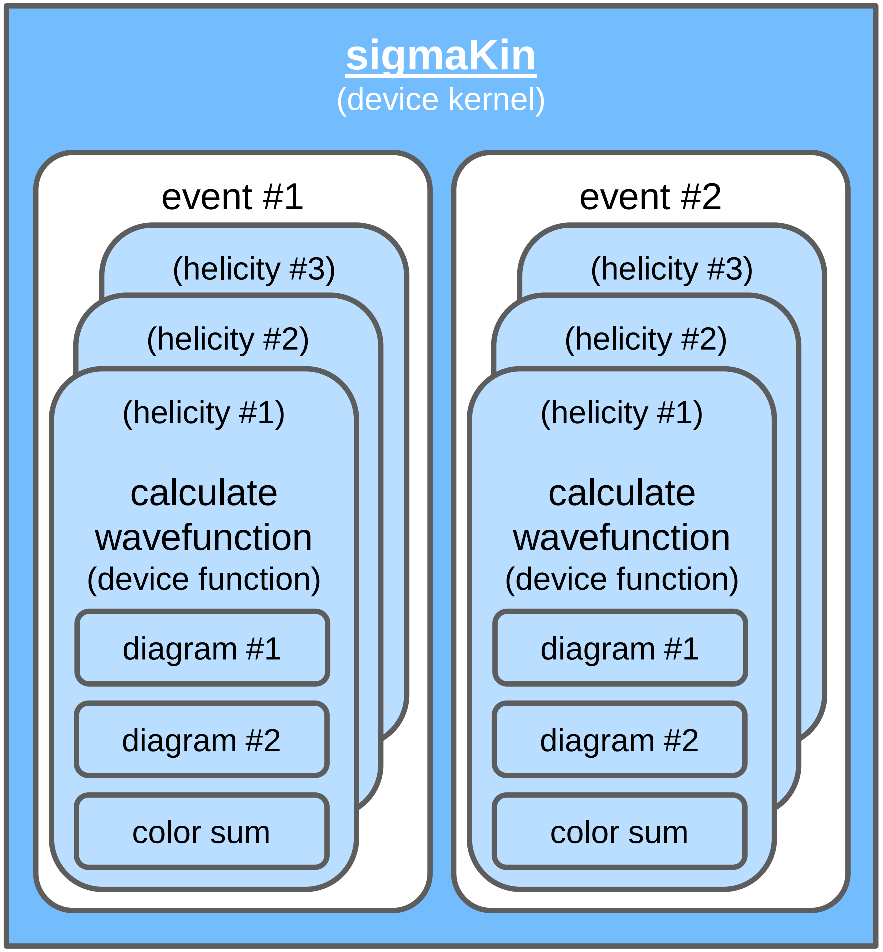
Kernel splitting, Streams, cuBLAS

Andrea Valassi (CERN)

Madgraph on GPU development meeting, 12th November 2024

https://indico.cern.ch/event/1355165

Step 0 – the current upstream/master

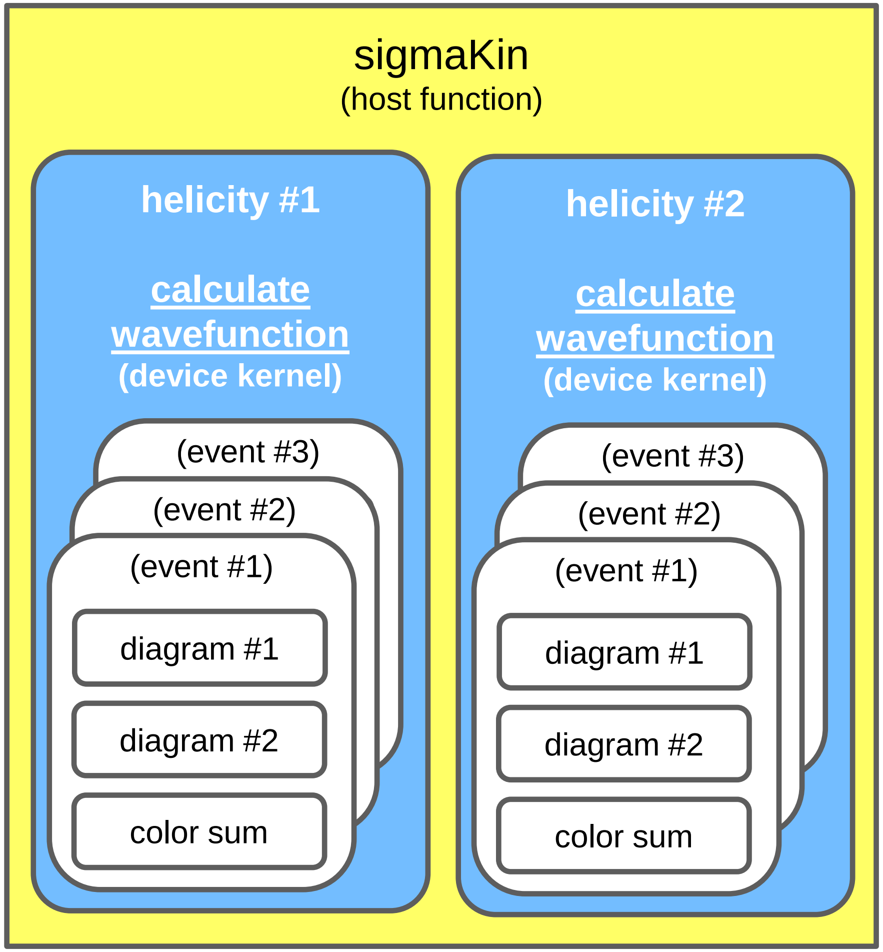


- Event-level parallelism
  - One single kernel launch
  - One event per GPU thread
  - *External loop is over events*
    - Loop over helicities within each event
- Performance for standalone gg_ttgg:
  - throughput peak 4.2 E5
  - throughput plateau at ~128k events

Results for gg_ttgg
check.exe –p thr blk 1

8.194081e+03	1	256
1.646537e+04	2	256
3.275429e+04	4	256
6.264182e+04	8	256
1.203012e+05	16	256
2.155087e+05	32	256
3.329072e+05	64	256
3.508559e+05	128	256
3.688962e+05	256	256
4.006369e+05	512	256
4.149977e+05	1024	256

Step 1a – invert event and helicity loops

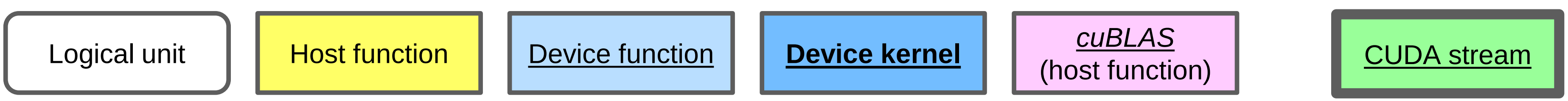


- Event-level and helicity-level parallelism
  - One kernel launch per helicity
  - One event per GPU thread
  - *External loop is over helicities*
    - Loop over events within each helicity
- Performance for standalone gg_ttgg:
  - throughput peak 4.0 E5 (was 4.2 E5)
  - throughput plateau at ~8k events (was 128k)

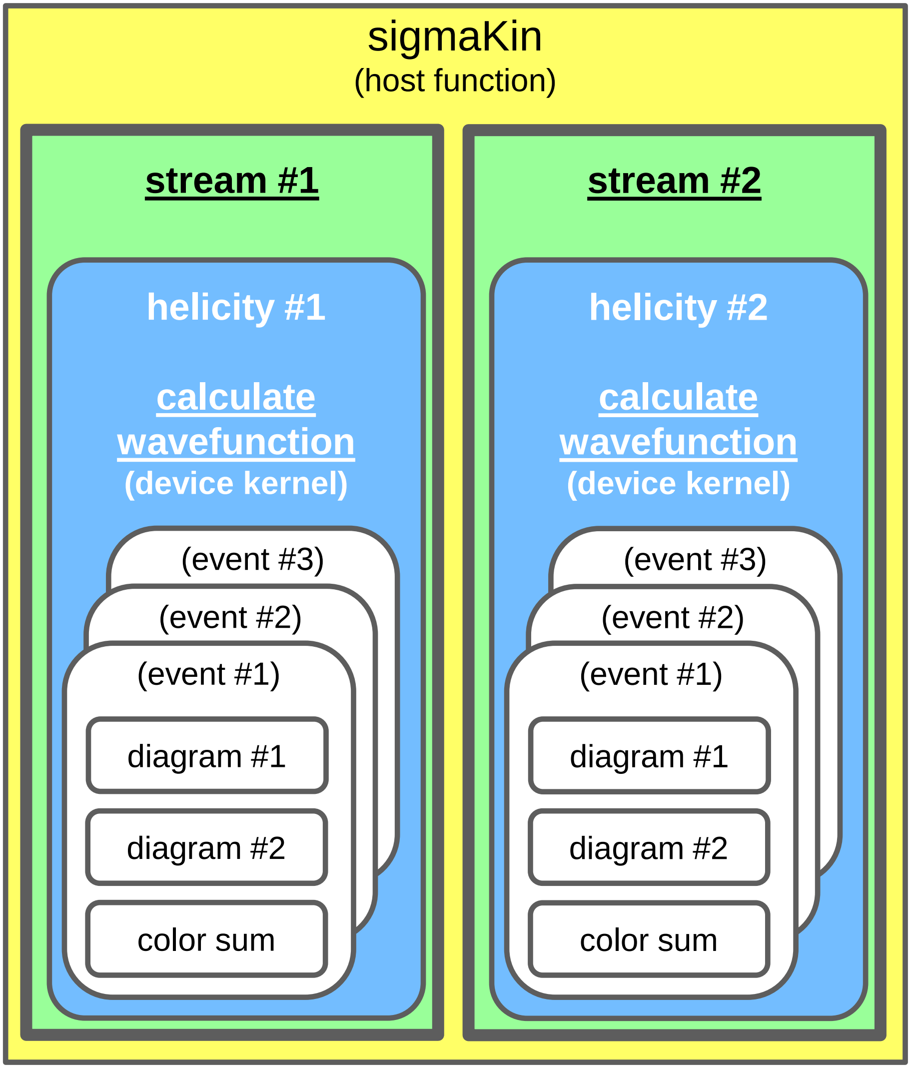
Results for gg_ttgg
check.exe –p thr blk 1

1.425079e+04	1	256
2.851574e+04	2	256
5.644591e+04	4	256
7.653334e+04	8	256
1.422753e+05	16	256
2.455957e+05	32	256
3.679183e+05	64	256
3.663693e+05	128	256
3.634467e+05	256	256
3.970560e+05	512	256
4.048658e+05	1024	256

Work started at Lugano hackathon with Olivier!



Step 1b – one stream per helicity

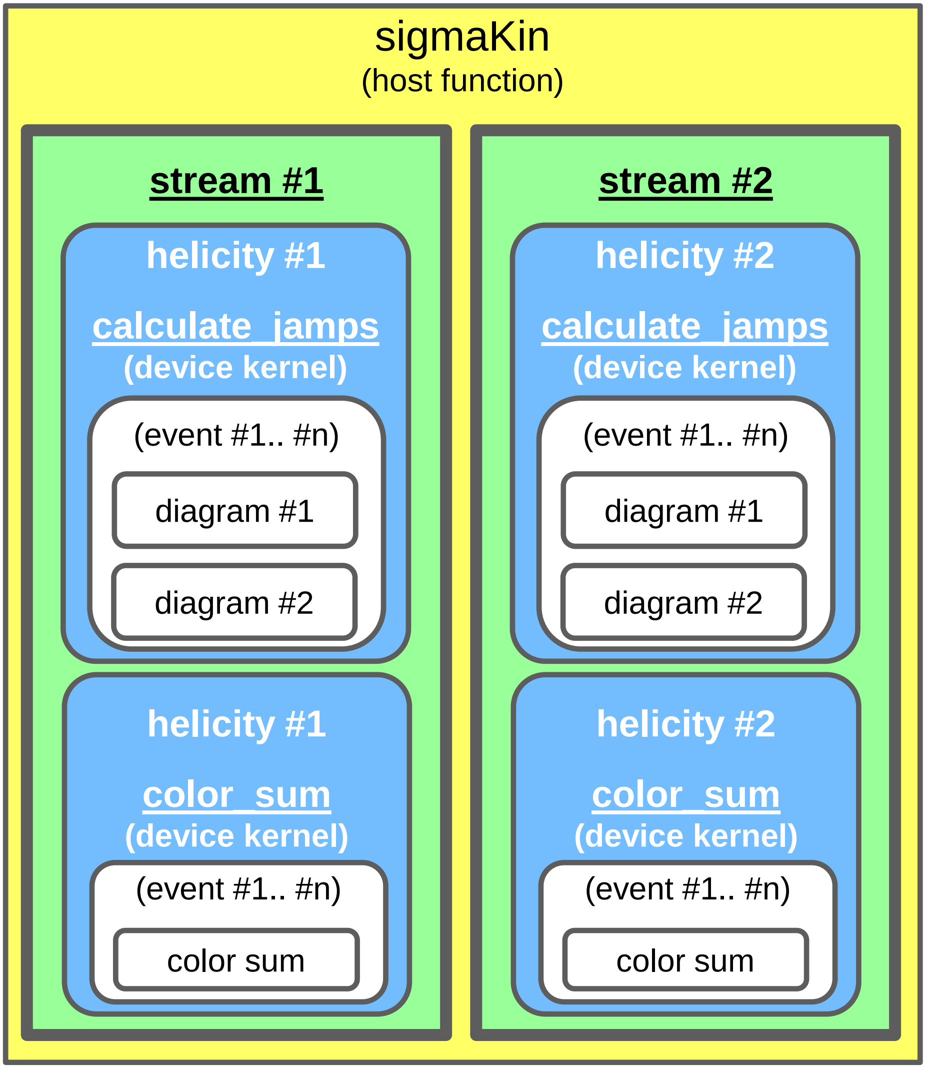


- Event-level and helicity-level parallelism
  - **One stream per helicity**
  - One kernel launch per helicity
  - One event per GPU thread
    - Loop over events within each helicity
- Performance for standalone gg_ttgg:
  - throughput peak 4.2 E5 (was 4.2 E5)
  - throughput plateau at ~2k events (was 128k)
    - only 30% lower than the peak with 256 events

Results for gg_ttgg
check.exe –p thr blk 1

2.871507e+05	1	256
3.592277e+05	2	256
3.540120e+05	4	256
3.866724e+05	8	256
3.995685e+05	16	256
3.978852e+05	32	256
4.000465e+05	64	256
4.050206e+05	128	256
4.084850e+05	256	256
4.171658e+05	512	256
4.130526e+05	1024	256

Step 2 – split diagrams from color sum



- Event-level and helicity-level parallelism
  - One stream per helicity
  - Two kernels launched per helicity
  - One event per GPU thread
    - Loop over events within each helicity
- Performance for standalone gg_ttgg:
  - throughput peak 4.6 E5 (was 4.2 E5)
  - throughput plateau at ~2k events (was 128k)
    - only 30% lower than the peak with 256 events

Results for gg_ttgg
check.exe –p thr blk 1

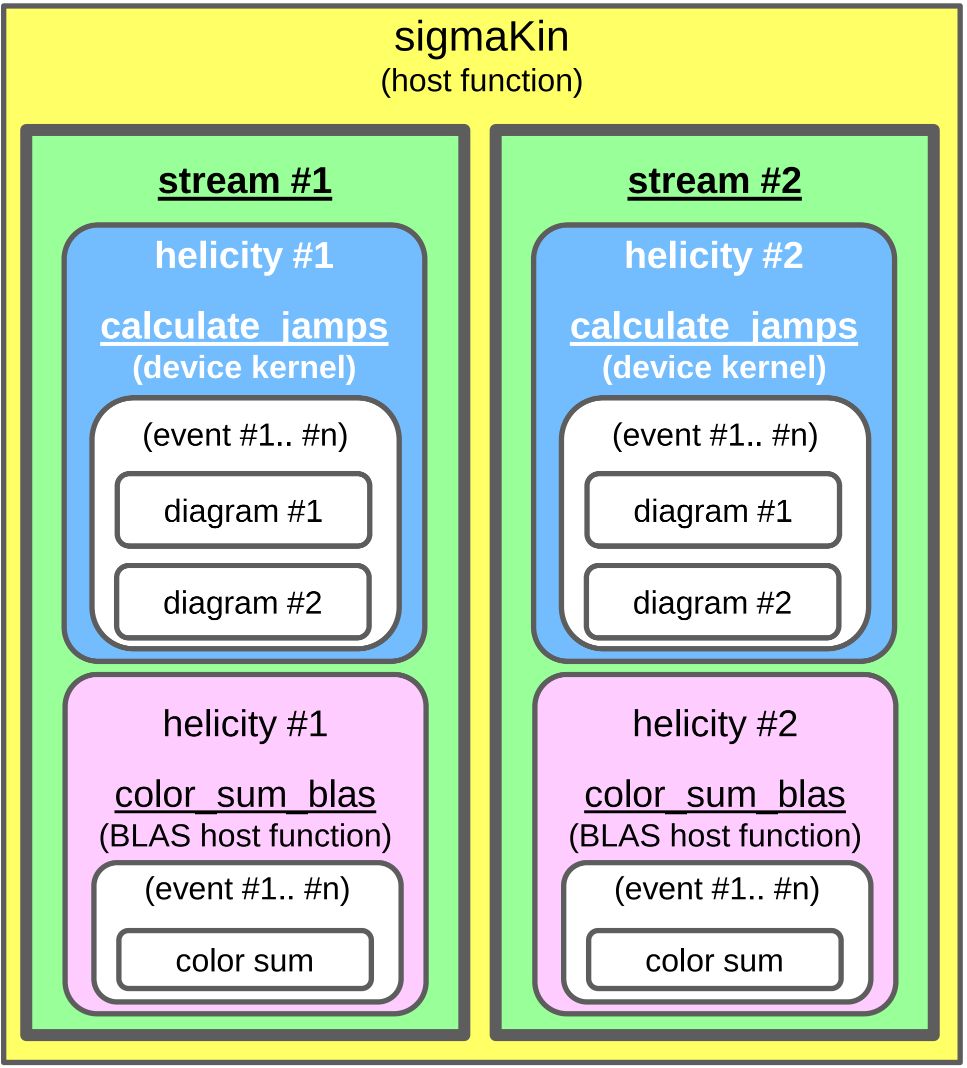
2.933287e+05	1	256
3.768369e+05	2	256
3.876321e+05	4	256
4.261835e+05	8	256
4.434271e+05	16	256
4.454906e+05	32	256
4.461988e+05	64	256
4.471014e+05	128	256
4.520026e+05	256	256
4.616800e+05	512	256
4.577239e+05	1024	256

Comments on steps 1a, 1b, 2 – done, IMO

- IMO: we could merge the three steps 1a, 1b, 2 described in the previous slides
  - 1a moving the helicity loop outside the event loop (one kernel per helicity)
  - 1b using different streams for different helicities (one stream per helicity)
  - 2 splitting the calculate_jamps and color_sum kernels (two kernels per helicity)

- Most obvious advantage: reduce #events in flight (even ~256 is quite good!)
  - much lower Fortran RAM, no need to generate huge numbers of events
  - splitting the color sum kernel also brings in a ~10% speedup (and cleaner code)
  - everything is ready including code generation, I will open a PR

Step 3 – test cuBLAS for color sum



- Event-level and helicity-level parallelism
  - One stream per helicity
  - One calculate_jamps kernel per helicity
  - One cuBLAS handle per helicity stream
    - ***THIS USES TENSOR CORES!***
- Performance for standalone gg_ttgg:
  - worse with BLAS than without BLAS
  - various things could be tested

Results for gg_ttgg
check.exe –p thr blk 1

1.182088e+05	1	256
2.395139e+05	2	256
2.880468e+05	4	256
3.376440e+05	8	256
3.531210e+05	16	256
3.532737e+05	32	256
3.534684e+05	64	256
3.557095e+05	128	256
3.763270e+05	256	256
3.858371e+05	512	256
3.910899e+05	1024	256

Step 4 – split diagrams from one another

- Status: completed the separation of diagrams in different device functions
  - Did this for gg_tt for simplicity, no codegen yet: no performance results for gg_ttggg
- WIP (4a): turn each diagram into a kernel (and calculate_jamps into a host function)
  - Minor issue to be fixed: handling of couplings in constant/device/host memory
- Next step (4b): turn each kernel diagram into a separate file
  - Check build times and check buildability of more complex processes like gg_ttggg
- Should take a few hours/days hopefully – stay tuned

Logical unit | Host function | Device function | **Device kernel** | *cuBLAS* (host function) | CUDA stream