Implementation of a CUDA clusterization algorithm for traccc

Stephen Nicholas Swatman, Attila Krasznahorkay

Status

- Work over the last two weeks or so has been on investigating and implementing CUDA clusterization in traccc
- Functional, fairly performant work-in-progress implementation is in place
- Issues like build system and EDM still in flux, of course

EDM

- Existing EDM is vector-of-structs-containing-vectors-of-structs (colloquially known as VoScVoS, of course)
- This is very GPU-hostile
- Implement the change proposed by Attila: struct-of-vectors-(of-vectors), more easily processed on GPUs with dedicated jagged vector code (such as what we have in vecmem!)
- Practically: all vectors moved around to be direct neighbours in the type constructor hierarchy, allowing us to traverse them more easily

SparseCCL

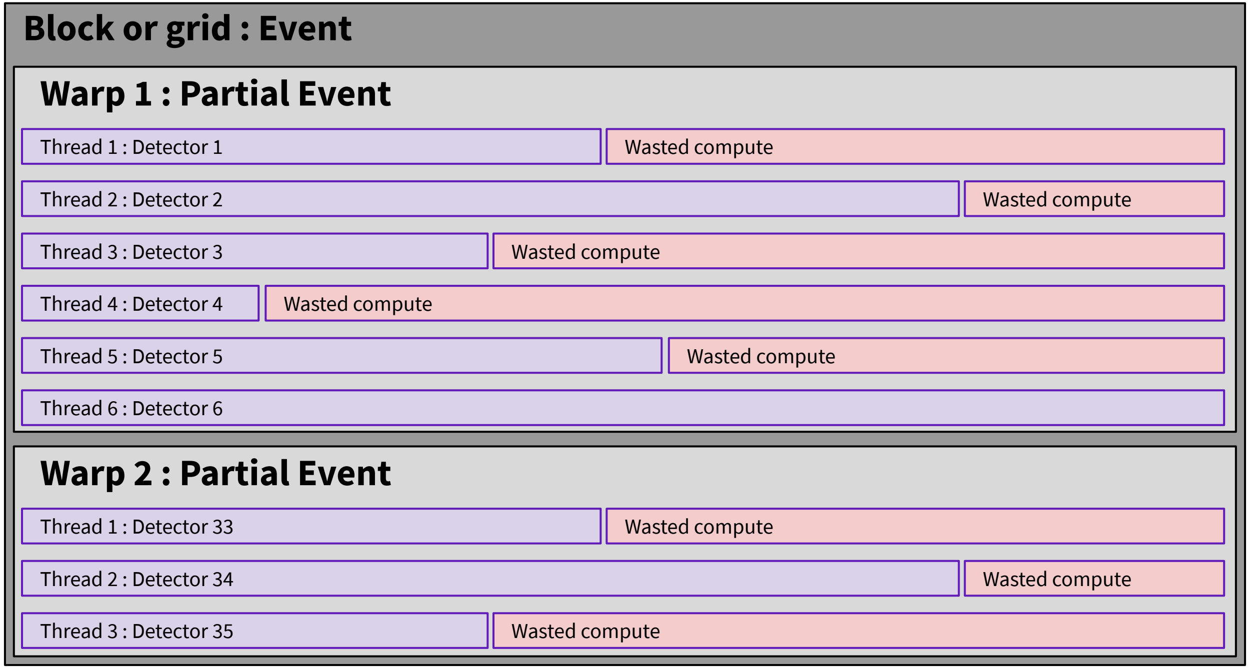
- SparseCCL is a nice algorithm for CPU, but is it optimal for GPUs?
- Inherent parallel nature limits the parallelism possible
- I’ve chosen to implement FastSV (Y. Zhang et al., 2020)
- Implicitly load-balanced implementation is possible here, with a little synchronisation
- Other alternatives: spectral methods or linear-algebraic methods: reduce the problem to other well-studied problems. Or Playne equivalence.

Design space

- A very wise lady once told me to always look at the *dimensions on which parallelism can be applied* to a problem
- What are our options in terms of parallel granularity?
- For SparseCCL, the minimum granularity is detector-level, leading to a lot of possible load imbalance: need to map detectors to threads

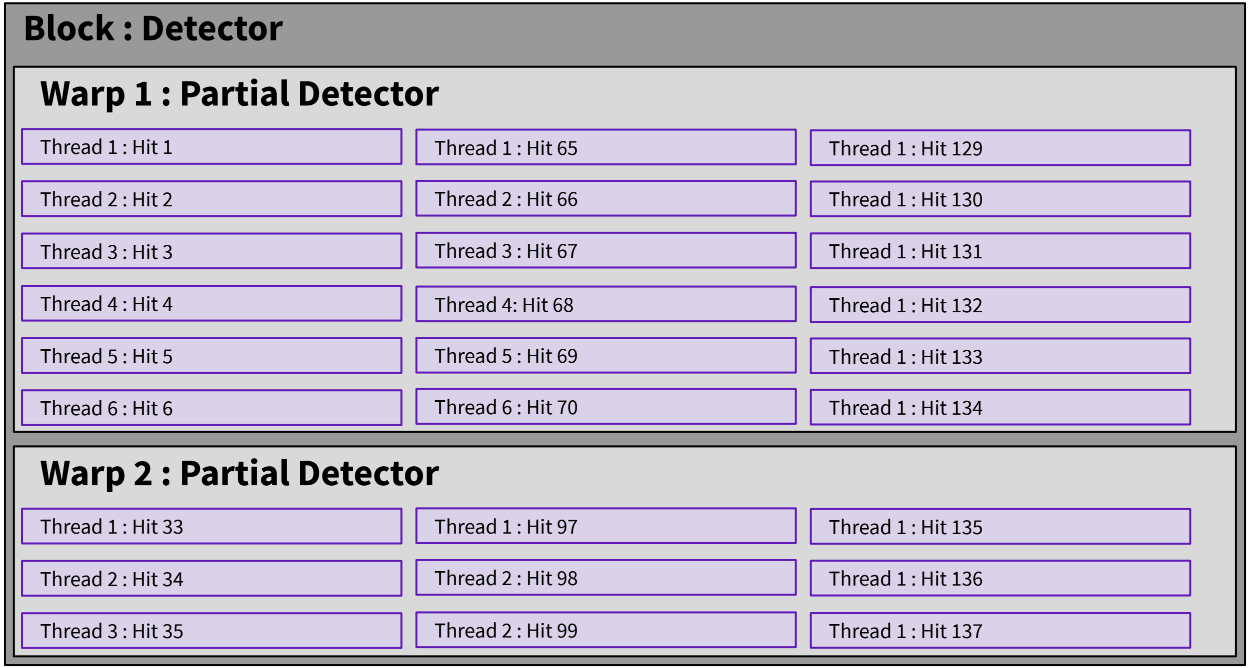
Design space

- Example of what processing one event might look like when using SparseCCL on a GPU: lots of wasted compute due to lock-stepping!



Design space

- Applying parallelism at the hit level makes this potentially easier



Design space

- Hit level parallelism can be easily scaled to the event-scale by adding more blocks following this correspondence:
  - Thread ↔ Hit
  - Warp ↔ Partial detector
  - Block ↔ Detector
  - Grid ↔ Event
- This allows us to leave the load-balancing up to the GPU, which seems like a good idea
- Also gives us the ability to share data between threads using the fast shared memory!

Design space

- Risk of this design is that the overhead of kernel launches might dominate, depends on the number of detectors per event, and number of hits per detector
- Will need to run more complex tests on more representative data to get a better feeling for how the overhead relates to the compute time
- Currently, performance is okay, but largely dominated by overhead due to per-detector kernel launch design - switch to per-event kernel should alleviate this! Requires some rethinking of the way the algorithm classes are designed though, and will no longer correspond 1:1 with the CPU code

Merging clusterization, measurements, SP

- Clusterization is relatively cheap, and leaves the EDM in a rather awkward state (cluster list)
- Not a problem on CPU, but this architecture is not ideal for GPUs (requires either work on the CPU or some very unorthodox processing on the kernel side)
- Proposed solution would be to merge clusterization, measurement creation (maybe spacepoint formation) into a single kernel
- Sacrifice composability for performance

Additional findings

- I strongly suspect that the current implementation of clusterization in traccc is broken
- Existing CPU algorithm consistently overestimates the number of clusters in generated testing data
- Other algorithms and implementations get the exact answer every time

Conclusions

- Code is available at:
  https://github.com/stephenswat/traccc/tree/enhancement/cuda_clusterization
- Or in merge request #19. Still ***very*** WIP!
- Seems to work well in my preliminary testing
- Design space is large, only part of it explored today