An online GPU hit finder for the STS detector in CBM

Felix Weiglhofer

October 23, 2024

Prof. Dr. Volker Lindenstruth
Frankfurt Institute for Advanced Studies
Goethe University Frankfurt

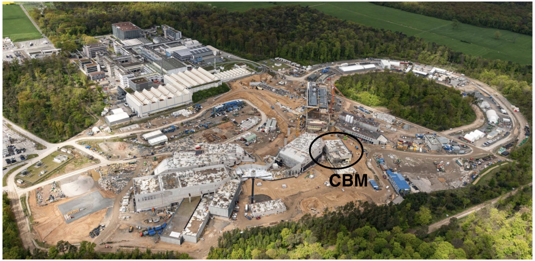
GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

CBM

SPONSORED BY THE
Federal Ministry
of Education
and Research

CHEP 2024, Krakow

The Compressed Baryonic Matter (CBM) experiment at FAIR



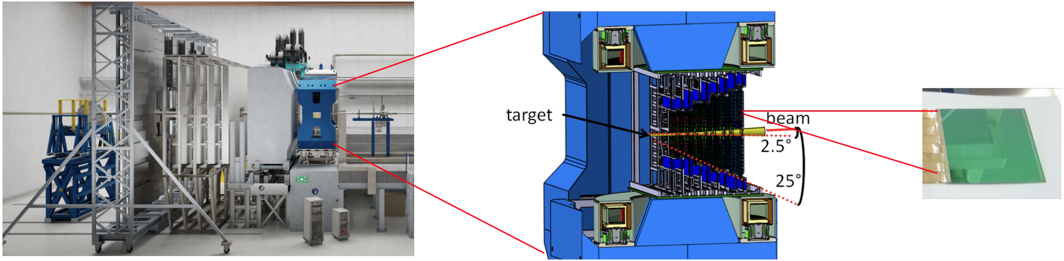
FAIR construction site, Apr. 2024

Render of CBM (© GSI/FAIR, Zeitrausch)

- Fixed target heavy ion experiment
- Under construction at the FAIR facility
- High reaction rates up to 10 MHz

- Exp. data rate: > 500 GB/s average
- Efficient online reconstruction + event selection required

The Silicon Tracking System (STS) detector

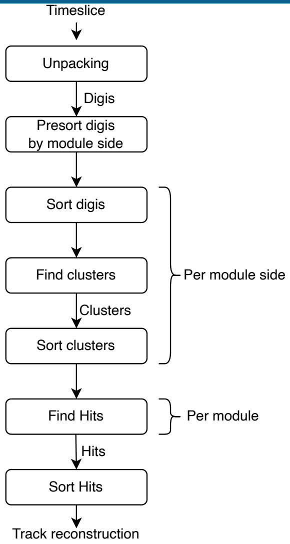


- Closest detector to the target
- High spatial (25 µm) and temporal (5 ns) resolution
- Key detector for track reconstruction
- 8 stations with ∼ 900 modules total

- Module: double sided sensor with 1024 channels on each side
- ∼ 1.8 million read-out channels
- Hit finder: match data from front and back side of same particle

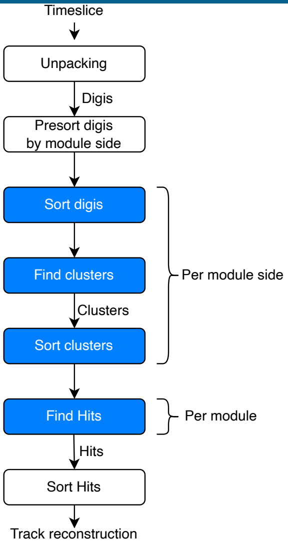
(online) STS reconstruction

- Raw detector data accumulated into timeslices
- Timeslices unpacked into digis
    - Digi: tuple of module, timestamp, channel and charge
- Combine neighboring digis into clusters
    - Cluster: 2D object with channel and time
- Combine clusters from front and back side into hits
    - Hit: 4D object with global coord and time
- Hits used for track reconstruction
- Hits divided into streams, streams sorted again by time before passed to tracking



(online) STS reconstruction

- Focus of this talk:
  - Digi / cluster sorting
  - Cluster finding
  - Hit finding
- Implementation part of CBMRoot online code
- Written in xpu[1]
  - Compiles GPU code to CUDA, HIP, SYCL or regular C++ with OpenMP



[1] https://github.com/fweig/xpu

Sorting on GPU

- Sort contents of module side (digis or clusters)
- Custom sorting algorithm:
  - One GPU block per module side
  - Parallel merge sort within block
  - Parallel merge step per GPU thread via Merge Path[1]
  - Preserves coalesced memory access within blocks

Further studies: compare with `cub::DeviceSegmentedSort`[2]

→ not available during first implementation

[1] O. Green et al., Merge Path - A Visually Intuitive Approach to Parallel Merging, 2014

[2] `https://nvidia.github.io/cccl/cub/api/structcub_1_1DeviceSegmentedSort.html`

Cluster finder (sequential)

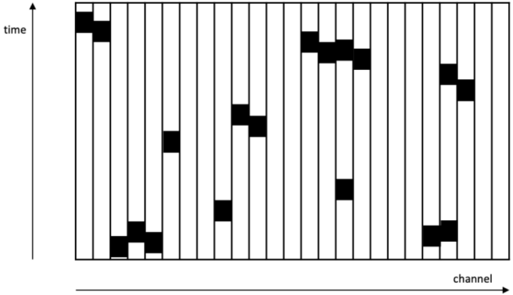
Sort digis by time

For all digis:

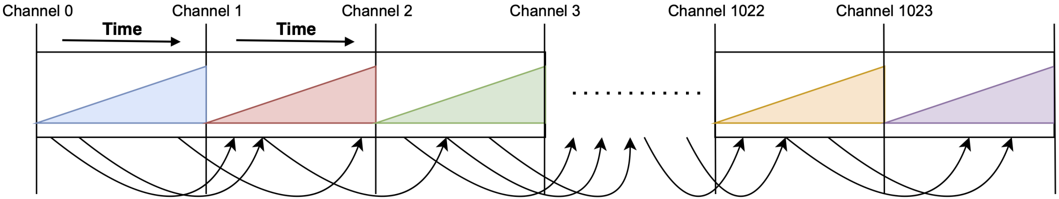
1. Try to mark channel of digi as active
2. If channel already active:
    - 2.1 Create cluster around digi (neighboring digis must fall within Δ time)
    - 2.2 Mark channels of cluster as inactive

Assumes sequential processing of digis, can't parallelize across digis

→ not suited for GPU

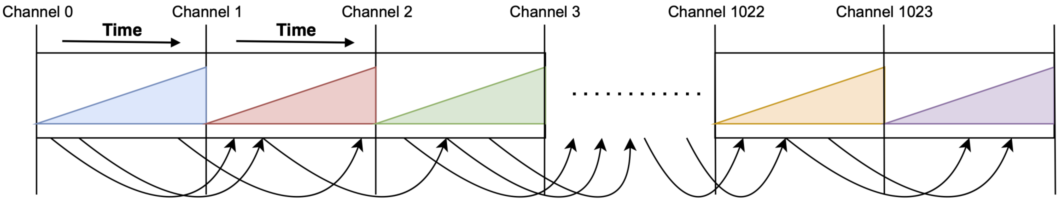


Parallel cluster finder



- Sort digis by channel and time instead
- Connect digis in same cluster via linked list
- → Store additional 32 bit connector object per digi

Parallel cluster finder



1. Find offset of each channel
2. Create connections:
   - Look for candidate *C* in next channel
   - If *C* found: Set index to *C* in connector, set prev bit for *C*
3. Create clusters:
   - Thread of first digi (prev bit not set) creates clusters
   - Iterate connectors to combine digis

Step 2 + 3: Parallel across all digis via atomic operations

Hit finding

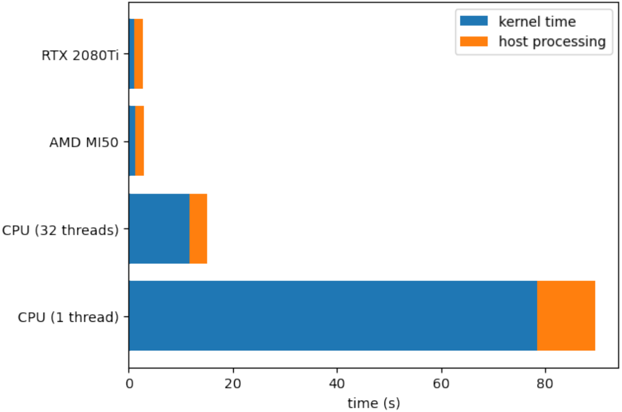
- Attempt to combine clusters from front and back side into hits if they overlap in time
- Time sorting required to reduce combinatorics
- Original algorithm[1] was straightforward to move to GPU
- Parallel across front side clusters

[1] H. Malygina, Hit reconstruction for the Silicon Tracking System of the CBM experiment, 2018

Performance

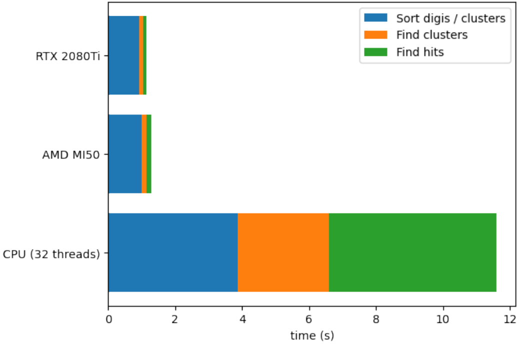
The mCBM setup

- Time accumulated over 20 timeslices (2.5 s of data)
- Real data from mCBM ($\sim 160 \cdot 10^6$ digis)
- CPU: Intel Xeon 6130 (16 cores, 32 threads)



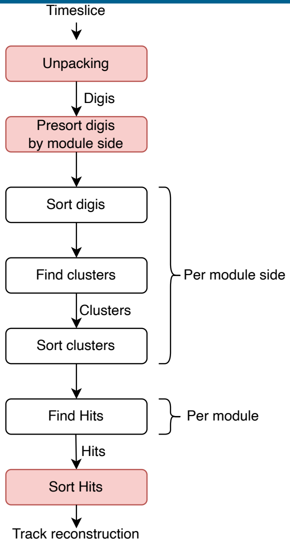
Kernel times

- Sorting predominant on GPU
  - only uses 22 blocks / compute units, occupies $\sim 1/3$ of device!
- Cluster and hit creation can exploit parallelism on GPU . . .
- . . . but the unmodified GPU code doesn't work that well here on CPU
- Hit creation on CPU: static thread distribution → few threads stuck on clusters with high combinatorics



Conclusion and next steps

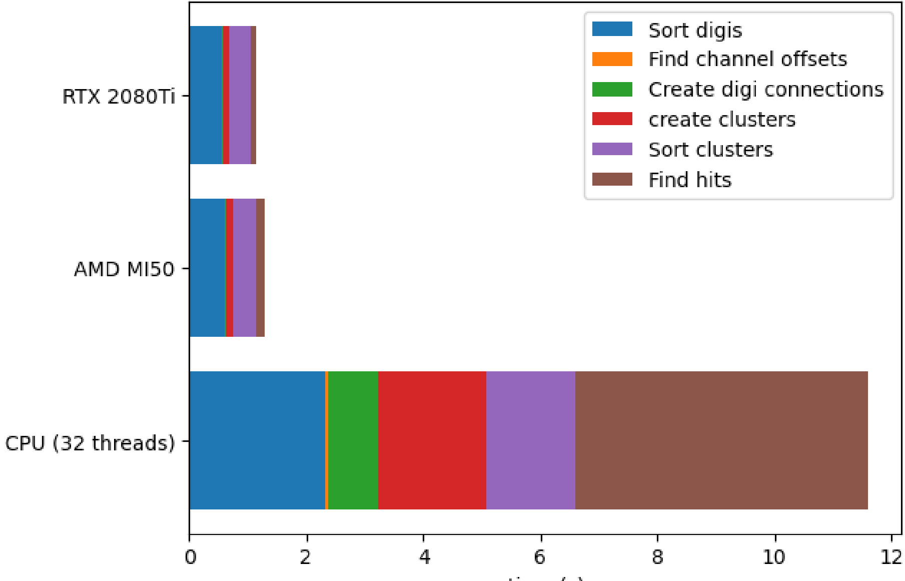
- Key steps of hit finder show promising performance on GPU
- 50 % runtime still spent on host
- Unpacking proof-of-concept on GPU
    - Shows great performance: 40 GB/s per timeslice
    - Currently being integrated into CBMRoot
- Remaining steps should be moved to GPU (digi presorting, hit sorting)



Thank you for your attention!

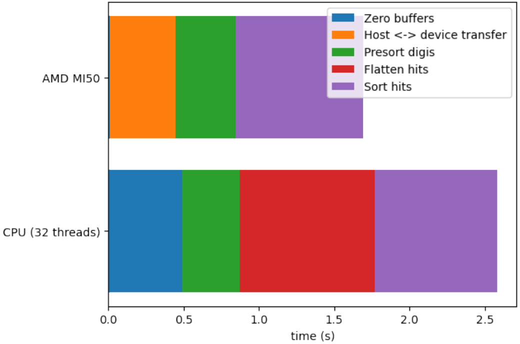
Backup Slides

Kernel times (detailed)



Processing on host

- Digi presorting and hit sorting: should be moved to GPU eventually
- Hits are stored into buckets, need separate step to flatten into single array
  - → Done on GPU during device to host copy.
- Zeroing buffers not yet parallel on host


Zero buffers
Host <-> device transfer
Presort digis
Flatten hits
Sort hits
AMD MI50
CPU (32 threads)
time (s)