CPU- and GPU-based Acceleration of Event-Building for Hybrid Pixel Detectors

FACULTY OF MATHEMATICS AND PHYSICS Charles University

Tomáš Čelko[1,2], František Mráz[1], Petr Mánek[2,3], Benedikt Bergmann[2]

[1]Faculty of Mathematics and Physics, Charles University in Prague

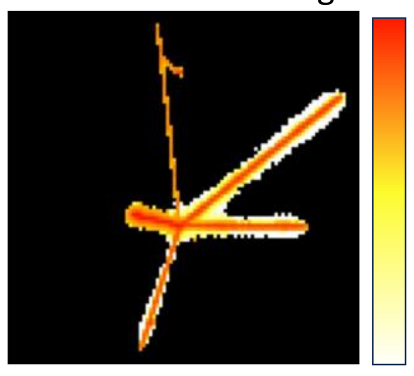
[2]Institute of Experimental and Applied Physics, Czech Technical University in Prague

[3]Department of the Physics and Astronomy, University College London

Introduction

- Hybrid pixel detectors like **Timepix3 and Timepix4** detect individual pixels hit by particles. For further analysis, individual hits from such sensors need to be grouped into spatially and temporally coinciding groups called **clusters.**

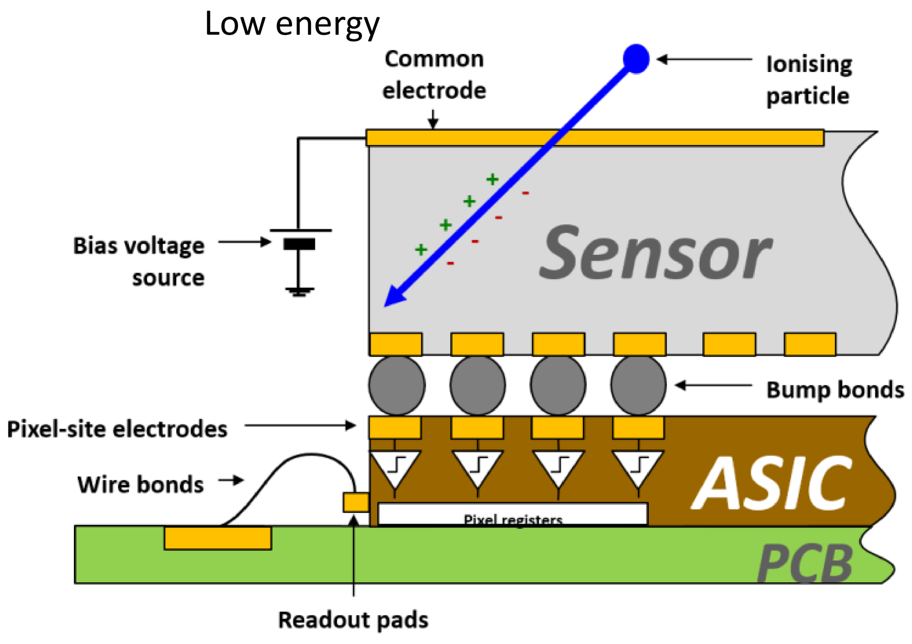
High energy



Low energy

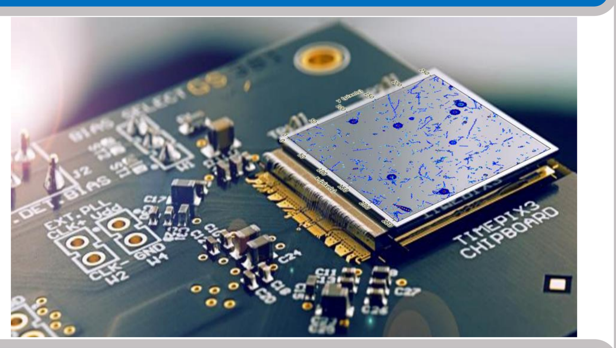
Timepix3 properties	
Pixel matrix	256×256
Pixel size	55 μm × 55 μm
Time resolution	1.56 ns
Bits per hit	48

- The Timepix3 detectors can generate more than **40 Mhit/s** (up to 640 Mhit/s with Timepix4) which is far **beyond the current capabilities** of the real-time clustering algorithms, processing at roughly **3 MHit/s.**
- Additionally, the hits from the detector are **not** guaranteed to be fully **temporally ordered.**



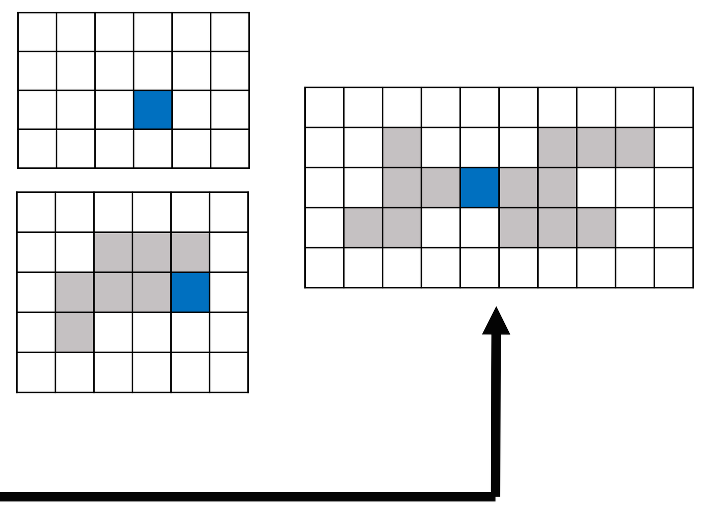
Goals

- Evaluate the capability of **speeding up** the clustering process through parallelization.
- Focus on **real-time** clustering application.
- Measure the clustering performance for clusters of varying sizes.



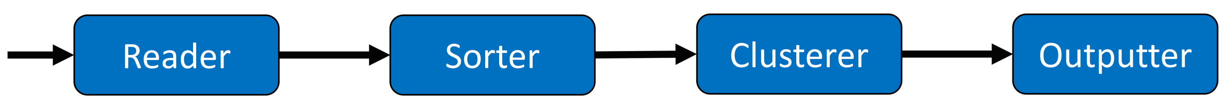
Methods

```
ProcessHit(hit):
  N = findNeighboringClusters(hit) //FIND
  if (|N| == 0)
    createNewCluster(hit)
  else if(|N| == 1)
    addHitToCluster(hit, N.first)
  else
    newCluster = mergeClusters(N) //UNION
    addHitToCluster(hit, newCluster)
  outputOldClusters()
```

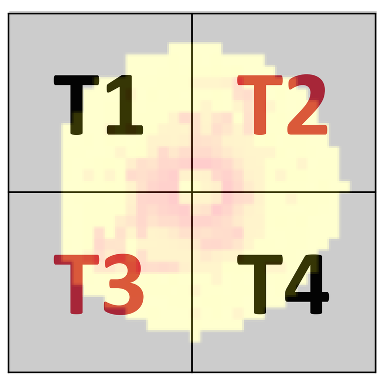


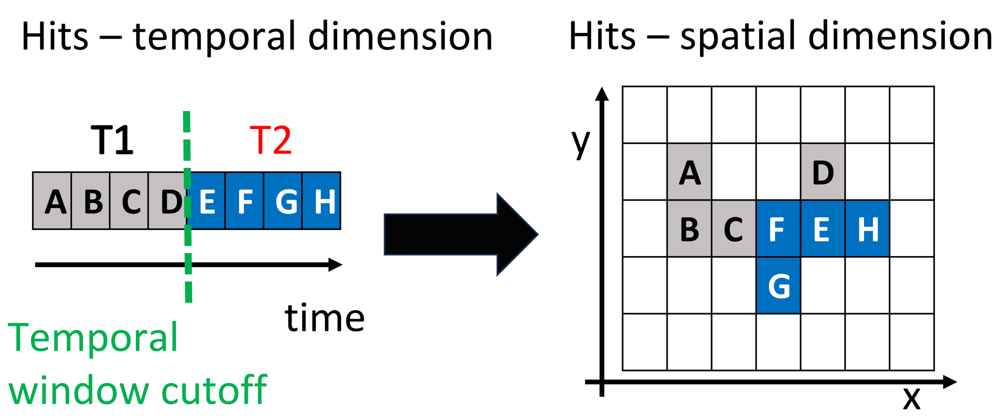
Parallel clustering performs the distributed computation of the clusters

- **Step based (pipeline)** – perform individual steps of the algorithm in the parallel



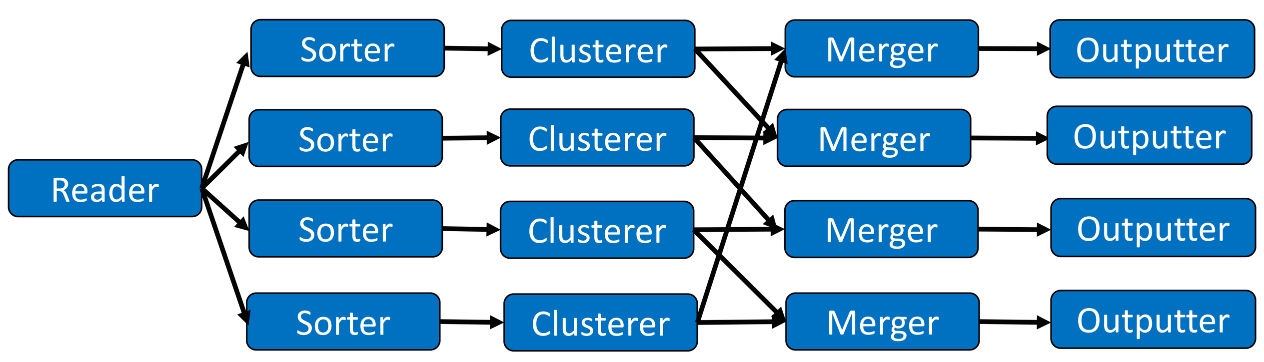
- **Data based** – split the data between workers, which can produce incomplete clusters.
  - **Spatial** – divides the area of the sensor into sectors.
  - **Temporal** – divides the hits into time windows.





Merging incomplete clusters split by the parallelization

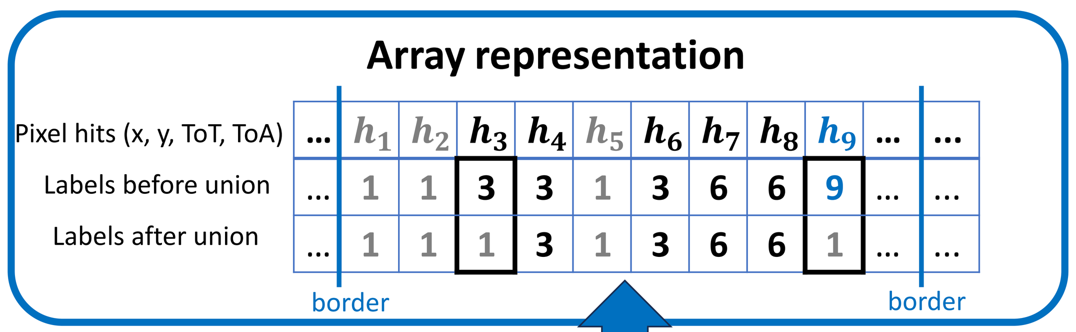
- Merging must be performed quickly**.** A **cascade approach** is used to quickly detect complete clusters. Moreover, the **merging is parallelized** – clusters from each clustering node are split among a pair of merging nodes. This way, we obtain multiple streams of complete clusters, which may or may not be concatenated.

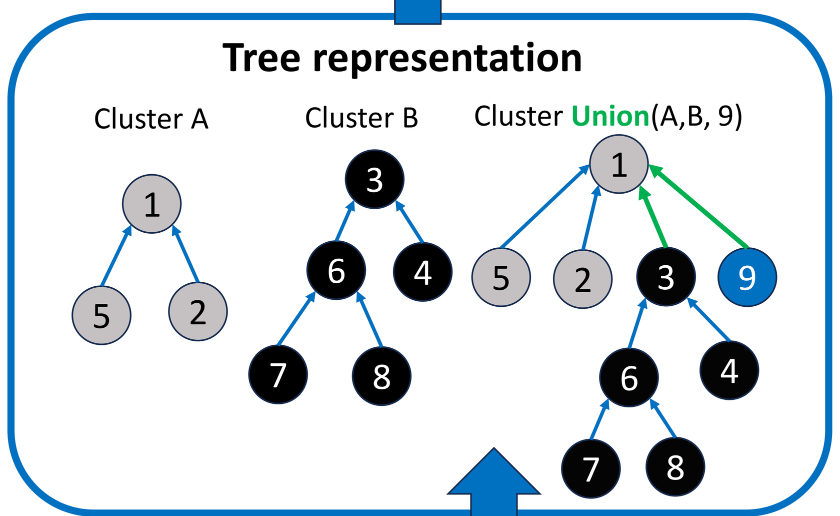


GPU parallel clustering

Disjoint union find

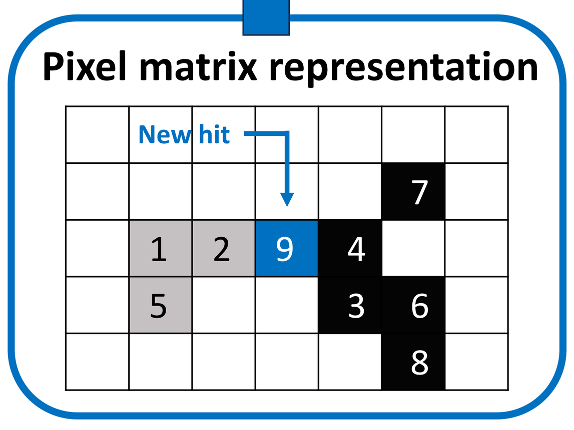
- Common data structure for **connected component labeling**.
- Clusters are represented by the **root** (min ToA hit).
- Determine which cluster is parent: Sorted output -> merge by ToA; Non-sorted output -> merge by size.





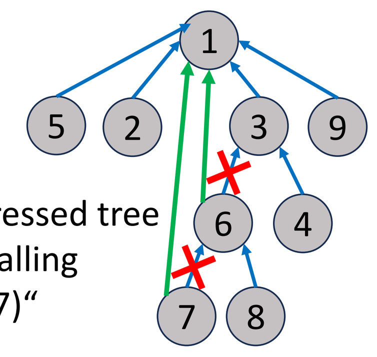
Algorithm

1. **Copy** data buffer from host to GPU (from pinned memory).
2. **Sort** hits **temporally** (parallel radix sort).
3. **Use disjoint-union-find** clustering for each chunk in parallel.
4. **Apply the step 3 again** to hits around the **border** of each chunk (to avoid splitting clusters)
5. **Sort** each hit by "**cluster id**" = root of the disjoint-union-find tree (hit with minimum ToA in cluster)
6. **Copy** data buffer from GPU to host



Path compression:

- Every time we visit a path to the root, set the root as the parent for each visited node.
- This makes the tree shallower and faster for next access.

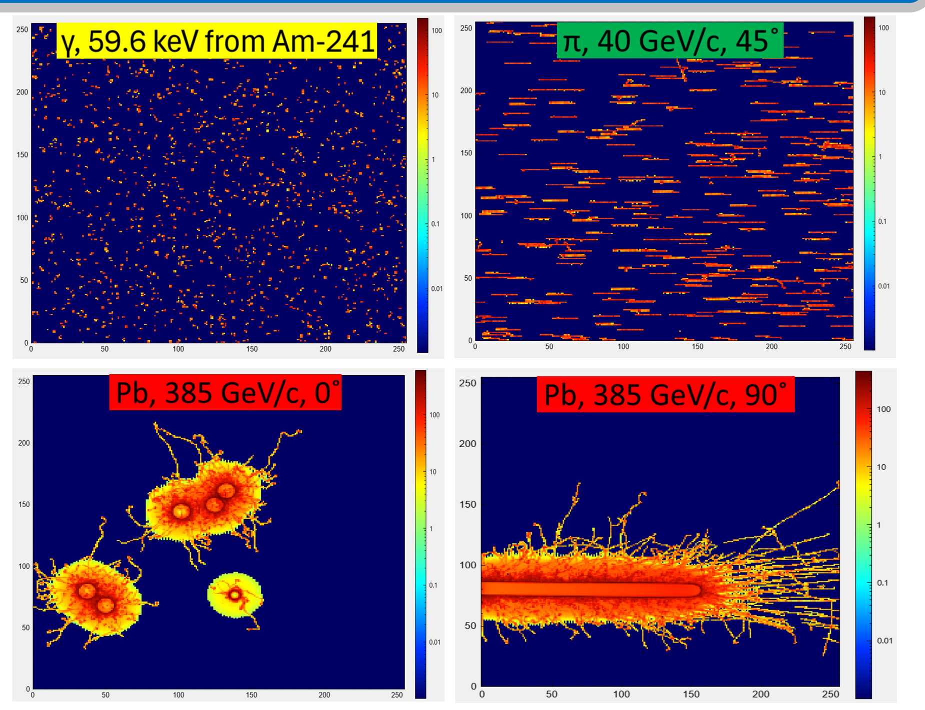


Compressed tree after calling "Find(7)"

Experiments

- **Benchmarking dataset**

Dataset	Mean cluster size	Stadard deviation of cluster size
γ, 59.6 keV from Am-241	1.46	1.65
π, 40 GeV/c, 0˚	3.86	6.66
π, 40 GeV/c, 45˚	20.09	10.58
π, 40 GeV/c, 75˚	56.02	30.26
Pb, 385 GeV/c, 0˚	422.81	860.71
Pb, 385 GeV/c, 50˚	280.27	939.95
Pb, 385 GeV/c, 90˚	210.82	1305.84
Pb, 385 GeV/c, 0˚, subset	2200.96	363.65
Pb, 385 GeV/c, 50˚, subset	3606.30	834.28
Pb, 385 GeV/c, 90˚, subset	7303.24	5081.69

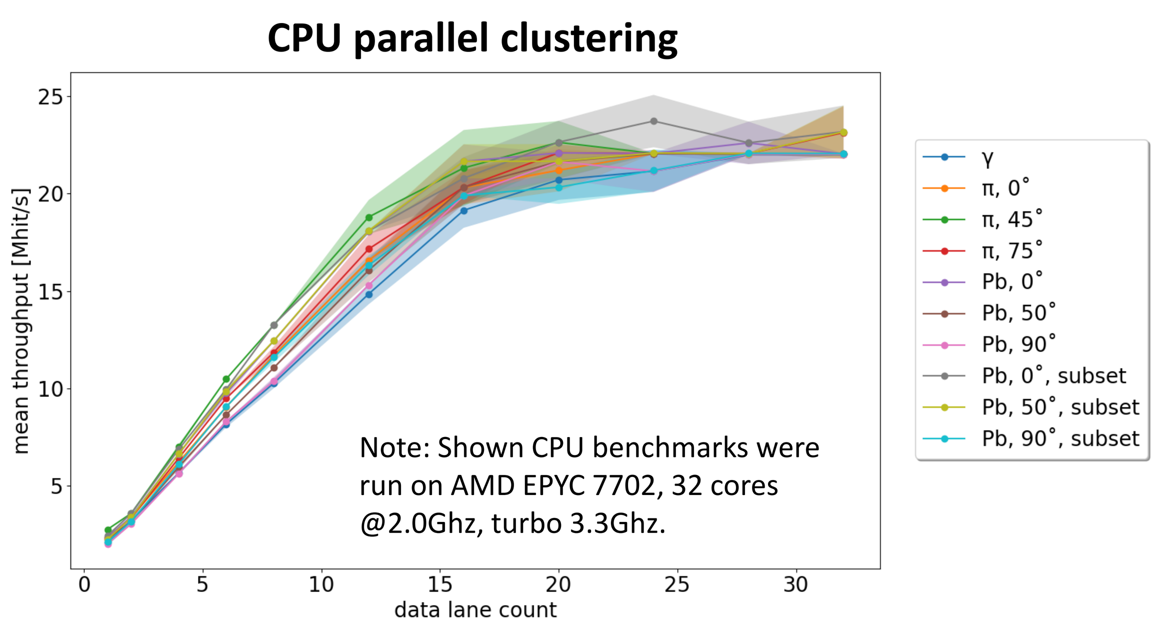


- **Clustering throughput scaling (with respect to the thread count)**

We performed multiple **I/O-independent** benchmarks – the data stream was generated by repeating a fixed-sized buffer of hits.

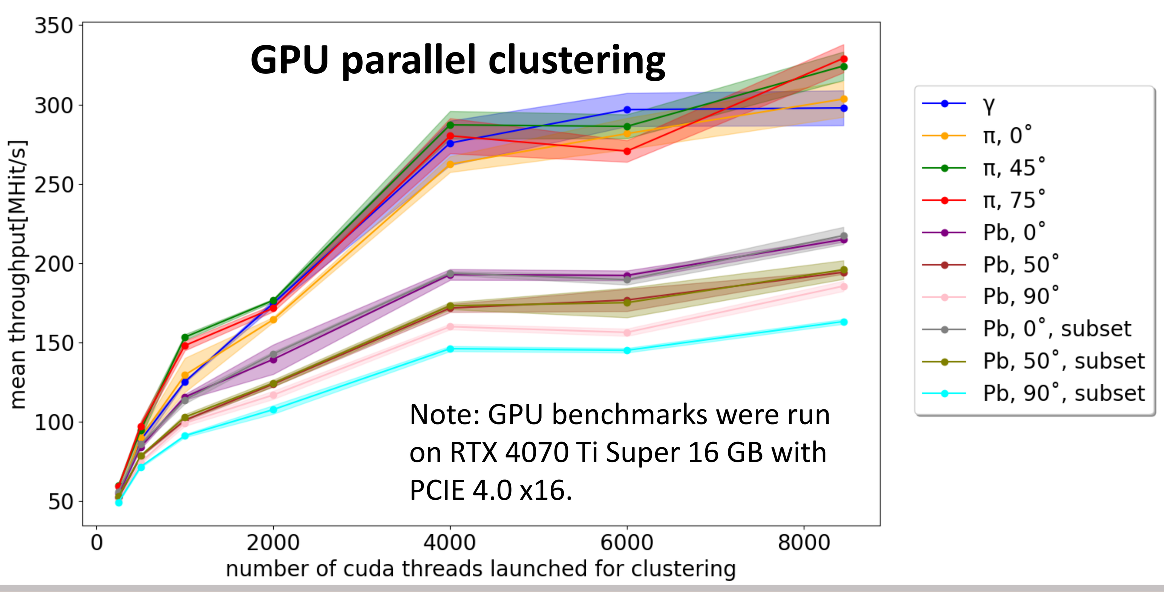
CPU clustering observations:

- The CPU parallel clustering **scales well** with the number of cores.
- Data dependence is insignificant for high thread count.



GPU clustering observations:

- The GPU parallel clustering also **scales well** with number of cores.
- There is a **negative** correlation between **clustering speed** and the mean **cluster size**. This is expected, as large clusters imply more extensive border checking (step 4).



Conclusion

- In both CPU- and GPU-based clustering algorithms we achieve a speed-up scaling with the number of used cores – up to **7× speed-up** for CPU clustering and **100× speed-up** for GPU clustering with respect to the baseline.
- Due to buffered processing, GPU clustering is suitable for processing data from **multiple devices at once** (quad) with little synchronization – only at the start/end of the buffer.
- Further improvements were implemented (copy & compute overlap, shared memory use…) with more coming up.

30th June – 4th July 2024, iWoRiD 2024, Lisbon

Email: celko.tom@gmail.com

Acknowledgements:

This research was funded by Grant Agency of Charles University under grant no. 142424 and the Czech Science Foundation grant number GM23-04869M.