An Online GPU Hit Finder for the STS Detector in the CBM Experiment

Felix Weiglhofer[1,2,*] and *Volker* Lindenstruth[1,2,3]

[1]Frankfurt Institute for Advanced Studies, Frankfurt am Main, Germany
[2]Johann Wolfgang Goethe-Universität Frankfurt, Frankfurt am Main, Germany
[3]GSI Helmholtz Centre, Darmstadt, Germany

Abstract. The Compressed Baryonic Matter (CBM) experiment at FAIR will operate at interaction rates up to 10 MHz, generating data streams averaging 500 GB/s. This necessitates efficient online reconstruction capabilities, particularly for the Silicon Tracking System (STS), which is the key detector for track reconstruction and contributes a large fraction of the expected data volume. We present a GPU-accelerated hit reconstruction chain for the STS that achieves a 128× speedup over the sequential CPU implementation. The implementation features optimized data structures reducing memory footprint, parallel algorithms for sorting, cluster finding, and hit reconstruction, and portability across GPU architectures. Our custom merge sort outperforms library implementations by 10 % while using 33 % less memory. Cluster finding employs a two-phase approach with atomic operations for thread-safe connections between signal clusters. Even before GPU acceleration, algorithmic improvements provide a 3× speedup in single-threaded execution. Both NVIDIA and AMD GPUs achieve comparable performance of approximately 0.12 s on a timeframe containing 1000 Au+Au events. The reconstruction chain was successfully deployed during the May 2024 mCBM beamtime, processing data rates up to 2.4 GB/s in real-time, demonstrating its viability for CBM's triggerless data acquisition approach.

1 Introduction

Modern high-energy physics experiments face unprecedented computational challenges due to increasing collision rates and detector complexity. The Compressed Baryonic Matter (CBM) experiment, currently under construction at the Facility for Antiproton and Ion Research (FAIR), exemplifies these challenges. CBM aims to study dense nuclear matter through heavy-ion collisions at interaction rates up to 10 MHz, producing data streams exceeding 500 GB/s [1][2].

Traditional triggered readout systems, which selectively record events based on hardware-level decisions, cannot effectively handle such extreme rates. Instead, CBM employs a free-streaming data acquisition approach where all detector data is recorded continuously and event selection occurs entirely in software. This paradigm shift necessitates efficient real-time processing capabilities to reconstruct particle trajectories from raw detector data.

*e-mail: weiglhofer@fias.uni-frankfurt.de

Figure 1. Rendering of the CBM experiment, with the large blue dipole magnet housing the Silicon Tracking System (STS) at its core, surrounded by additional detector systems. (© GSI/FAIR, Zeitrausch)

Graphics Processing Units (GPUs) have proven effective for reconstruction tasks in high-energy physics. The ALICE experiment pioneered large-scale GPU usage for real-time track reconstruction during LHC Run 2, expanding this approach in Run 3 where the complete TPC reconstruction runs on GPUs handling up to 50 kHz Pb-Pb data at rates exceeding 1 TB/s [3][4]. Similarly, LHCb deployed the Allen GPU trigger system processing up to 40 Tbit/s of detector data for Run 3 [5].

This paper describes the implementation of a parallel algorithm for hit reconstruction on GPUs for the Silicon Tracking System, the central tracking detector in CBM. We discuss the optimization of data structures to minimize memory overhead, efficient sorting strategies for time-ordered data, and parallel approaches to cluster formation and hit reconstruction. Performance measurements demonstrate speedups exceeding 100x compared to single-threaded CPU execution, with detailed analysis of scaling behavior across different GPU architectures.

2 Background

2.1 The STS Detector

The Silicon Tracking System (STS) serves as the primary tracking detector in the CBM experiment [6]. Positioned inside the dipole magnet, see Figure 1, it enables track reconstruction and momentum determination of charged particles. The detector consists of eight stations with double-sided silicon strip sensors arranged at a stereo angle of 7.5°. The STS utilizes 896 sensors with 1.8×10^6 readout channels and achieves spatial and temporal resolutions of 25 μm and 5 ns.

2.2 The STS Reconstruction Chain

The reconstruction chain transforms raw measurements into four-dimensional spacetime points. Starting with digis (time-stamped charge measurements), the pipeline distributes them

to corresponding module sides and sorts them by timestamp. The cluster finding algorithm identifies groups of adjacent strips with charge deposits within a time window [7]. Clusters are then sorted by time before hit finding, which combines time-matched clusters from both sensor sides to reconstruct hit positions [8]. The reconstruction processes data independently for each sensor module, enabling parallel processing on GPUs.

3 GPU-Accelerated STS Reconstruction

The GPU implementation of the STS reconstruction chain requires significant adaptations to the existing algorithms and data structures. While the basic reconstruction steps remain similar to the offline version, the implementation focuses on maximizing parallel execution and minimizing memory overhead. This section describes the key optimizations and algorithmic changes that enable efficient GPU processing. We first discuss common optimizations applied throughout the reconstruction chain, followed by detailed descriptions of the GPU implementations for sorting, cluster finding, and hit reconstruction.

3.1 Common Optimizations

The implementation reduces memory overhead by redesigning data structures. Cluster objects are reduced from 112 byte to 24 byte and hit objects from 136 byte to 48 byte using single-precision floating point values, 32-bit integer timestamps, and separating simulation information. Specialized container classes partition data by hardware address, eliminating redundant storage. Additional optimizations include direct computation of cluster properties and parallelization across module sides rather than complete modules.

3.2 Sorting on GPU

The STS reconstruction chain requires sorting operations at multiple stages: first to order digis by channel and time for cluster finding, and later to sort clusters by time to facilitate hit reconstruction. The sorting implementation must efficiently handle data from 1792 module sides in parallel while maintaining high GPU occupancy.

Our implementation divides the sorting work along natural partition boundaries, with each GPU thread block processing data from a single module side independently. Within each block, a two-phase sorting approach is employed. First, the data is divided into fixed-size chunks that are sorted using a block-level radix sort. These sorted sequences are then merged in parallel using the Merge Path algorithm [9]. The parallel merge occurs entirely within the thread block, using shared memory to optimize memory access patterns.

3.3 Parallel Cluster Finding

The cluster finding algorithm identifies groups of adjacent strips that registered charge deposits within a configurable time window. While the offline implementation processes digis sequentially, examining each measurement in order, this approach is not suitable for parallel execution on GPUs. Instead, we implement a two-phase strategy, illustrated in Figure 2, that enables efficient parallel processing without requiring complex synchronization between threads.

The first phase establishes connections between digis that belong to the same cluster using a lightweight linked list structure. Each digi maintains a 32-bit connector value that stores both the index to the next digi in the cluster and a flag indicating whether the digi has a

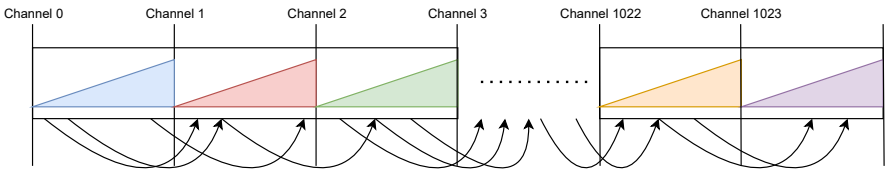

Figure 2. Illustration of the linked-list structure used by the parallel cluster finder, showing connections between digis in adjacent channels. Colored regions represent charge deposits in each channel with increasing time stamps, while arrows indicate connections established between digis that form clusters.

predecessor. The highest bit serves as the predecessor flag, while the remaining 31 bits store the index of the next digi or zero to indicate the end of a cluster. This compact representation allows atomic updates to the entire structure using compare-and-swap operations, ensuring thread safety when multiple GPU threads modify connections simultaneously.

During the connection phase, each GPU thread processes a single digi, examining neighboring channels for potential cluster members. Two digis are considered part of the same cluster if they are in adjacent channels and their time difference falls within a configurable window. The algorithm exploits the fact that digis are already sorted by channel and time, using binary search to efficiently locate the first digi within the time window in the neighboring channel. When a match is found, the thread atomically updates the connector to establish a forward edge to the next digi and sets the predecessor flag on the target digi.

The second phase creates the actual cluster objects. Each GPU thread that processes a digi without a predecessor (indicating the start of a cluster) is responsible for creating the corresponding cluster. The thread follows the chain of connections established in the first phase, computing cluster properties like total charge, position, and timing information directly during traversal. This approach eliminates the need to store intermediate arrays of digi indices, reducing memory usage compared to the offline implementation to an additional 4 byte per digi.

3.4 Parallel Hit Finding

The hit finding algorithm transforms two-dimensional cluster measurements from both sensor sides into four-dimensional spacetime points. While the offline version processes data at the module level with one CPU thread per module, the GPU implementation instead assigns one thread per front-side cluster, enabling full parallelization across all potential hits.

For each front-side cluster, the assigned GPU thread searches for matching clusters on the back side of the sensor. To efficiently find potential matches, the algorithm exploits the fact that clusters are already sorted by time. The thread performs a binary search to locate the first back-side cluster within the configurable time window relative to its front-side cluster. This optimization significantly reduces the number of cluster pairs that need to be evaluated for geometric intersection, as clusters outside the time window can be quickly discarded.

The algorithm implements early exit conditions based on time differences between clusters. As both cluster arrays are sorted by time, the thread can stop searching once it encounters a back-side cluster whose time difference exceeds the maximum window. This approach, combined with the binary search for the initial matching cluster, means that each thread typically only needs to evaluate a small subset of back-side clusters.

For time-matched clusters, the algorithm calculates their geometric intersection point in the sensor's local coordinate system. Due to the sensor's design, multiple intersections may

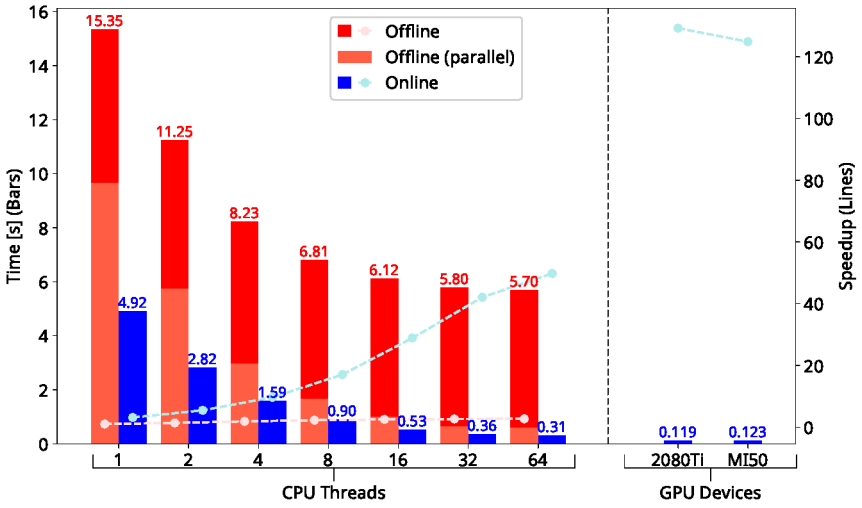

Figure 3. Performance comparison between offline and online STS reconstruction implementations across different hardware configurations. The plot shows the execution time in seconds (bars) and the speedup (lines) depending on the number of CPU threads (1–64) and tested GPU devices (RTX 2080Ti and MI50).

occur for a single cluster pair. The implementation handles these ambiguities, known as ghost hits, by creating hits for all valid intersection points, leaving their resolution to the subsequent track reconstruction stage.

The memory management for output hits employs a bucket-based approach with pre-allocated space for each module, sized according to the number of input clusters. Each thread uses atomic operations to obtain an index in the appropriate module's bucket when writing a new hit. This approach ensures thread-safe parallel output while maintaining spatial locality of hits from the same module.

4 Performance Evaluation

4.1 Benchmark Setup

Performance evaluations used a simulated dataset of 1000 central Au+Au collisions producing 17.2×10^6 STS digis, 5.9×10^6 clusters, and 4.8×10^6 hits. Tests ran on an AMD MI50 GPU, NVIDIA RTX 2080 Ti GPU, and Intel Xeon Gold 6130 CPU (16 cores per socket, dual-socket configuration with 64 hardware threads total). The software environment used Ubuntu 22.04, GCC 11, CUDA 12.6, and ROCm 6.0. Each configuration executed for 11 iterations, with results reporting the median of 10 runs after discarding the warmup iteration.

4.2 Overall Performance

Figure 3 presents the end-to-end runtime comparison between the *offline* and *online* implementations. The single-threaded offline implementation requires 15.35 s to process the benchmark dataset, with 9.5 s spent in the parallel section and 5.85 s in sequential overhead. The

Table 1. Performance comparison of individual processing steps in the STS reconstruction chain. GPU measurements use the best performance between NVIDIA RTX 2080 Ti and AMD MI50. Pre/Post Processing includes data distribution, collection, and host device DMA transfers for GPU.

Processing step	**Execution time**				
	Off. CPU (1t)	Off. CPU (64t)	On. CPU (1t)	On. CPU (64t)	GPU
Sorting	2.22 s	0.15 s	1.13 s	0.06 s	0.009 s
Clustering	5.45 s	0.31 s	2.16 s	0.07 s	0.008 s
Hit Finding	2.01 s	0.15 s	1.15 s	0.06 s	0.010 s
Pre/Post Processing	5.67 s	5.09 s	0.48 s	0.12 s	0.092 s
Total	15.35 s	5.70 s	4.92 s	0.31 s	0.119 s

online implementation achieves 4.92 s in single-threaded execution, a 3.1× speedup before parallelization due to optimized data structures and algorithms.

Both implementations scale with increasing thread count but exhibit different characteristics. The offline implementation's performance becomes dominated by the sequential overhead at higher thread counts, limiting parallelization to a 3× speedup. The online implementation scales up to 32 physical cores, with performance flattening when hyperthreading is enabled at 64 hardware threads. At maximum CPU utilization, the online implementation achieves 0.31 s runtime, representing a 15.9× improvement over its sequential performance.

GPU execution provides substantial additional speedup. Both tested devices achieve similar performance: 0.119 s on the NVIDIA RTX 2080 Ti and 0.123 s on the AMD MI50. This represents a 128× speedup compared to sequential offline execution and is 2.6× faster than the best CPU performance of the online version.

Table 1 provides a breakdown of execution times for individual processing stages. The cluster finding stage shows the most dramatic improvement with a 681× speedup, followed by sorting (247×) and hit finding (201×). Even pre/post processing overhead achieves a 62× speedup through optimized data structures and parallelization of data movement operations.

These results demonstrate two key points. First, significant performance improvements come from algorithmic changes and data structure optimizations before considering hardware acceleration. Second, the STS reconstruction algorithms effectively leverage GPU parallelism, achieving substantial speedups while maintaining identical functionality across different GPU architectures.

4.3 Sorting Performance on GPU

To evaluate the performance of our custom merge sort implementation, we compared it against CUB's DeviceSegmentedSort [10] using both key-value pair sorting and key-only variants. For the full STS configuration with 1792 segments, our merge sort implementation showed superior performance, completing sort operations in 93.8 ms on the RTX 2080 Ti compared to 189.1 ms for key-value pair sorting and 104.6 ms for the key-only variant. Performance degrades with fewer active segments or unbalanced load distribution. In such scenarios, library implementations may provide better runtime performance.

Memory requirements also differ between implementations. For n digis, our merge sort operates with $2n$ memory usage, requiring only input and output buffers. In contrast, CUB's key-value pair sorting demands $6n$ memory for input, output, and temporary storage buffers,

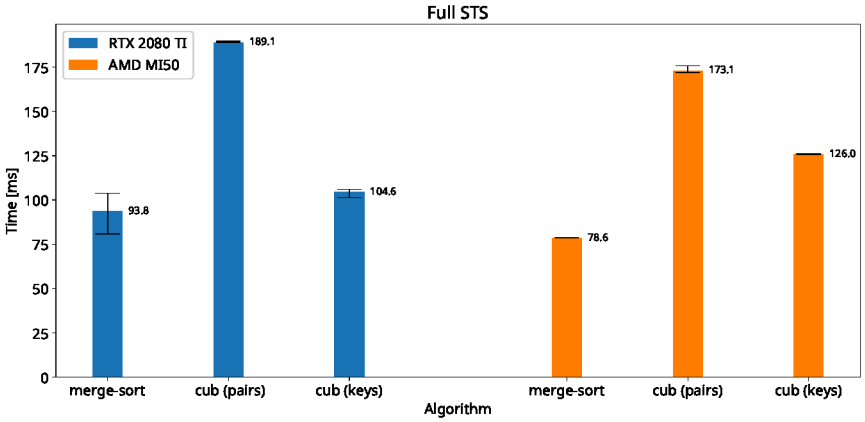

Figure 4. Comparison of sorting algorithm performance on RTX 2080 Ti and AMD MI50 GPUs for full STS configuration.

while its key-only variant reduces this to $3n$. This memory efficiency makes our implementation particularly suitable for processing large data volumes from the full STS detector.

5 Production Deployment

The mini-CBM (mCBM) experiment is a FAIR Phase-0 experiment that serves as a proof of concept for the CBM hardware [11]. Operating at the SIS18 synchrotron since 2018, mCBM incorporates small-scale prototypes of all major CBM subsystems, including an mSTS detector with three tracking stations comprising 12 modules. This setup provides an opportunity to validate both detector hardware and online software under real experimental conditions.

In May 2024, the online STS reconstruction was deployed during a four-day mCBM beamtime, marking its first production use. The system operated continuously alongside raw data recording, processing the free-streaming data in real-time. Processing performance remained robust throughout the deployment period. Average data rates reached 800 MB/s, with peaks up to 2.4 GB/s. The STS detector constituted the largest data contributor at 310 MB/s average and 900 MB/s peak rates.

The system consistently processed between 24–32 million hits per second, derived from 120–160 million clusters. These reconstructed hits provided input for track finding, which maintained consistent yields of approximately 500,000 tracks per second throughout data taking.

This successful deployment validated the online reconstruction chain under real experimental conditions, confirming that the algorithmic approaches and optimizations scale effectively to real-world data. The experience gained provides valuable insights for future CBM operations with higher data rates and more complex detector configurations.

6 Conclusion

We have presented a GPU-accelerated hit reconstruction for the Silicon Tracking System of the CBM experiment. The implementation achieves a 128× speedup compared to the offline single-threaded implementation, enabling real-time processing of the high data volumes expected during CBM operation.

The implementation uses lightweight data structures, custom sorting algorithms, and parallel processing techniques that demonstrate portability across GPU architectures. Both NVIDIA RTX 2080 Ti and AMD MI50 achieved comparable performance of approximately 0.12 s per timeframe, while even without GPU acceleration, the optimized algorithms showed a 3× speedup in single-threaded CPU execution.

The hit finder has been integrated into the CBM online framework and successfully deployed during the May 2024 mCBM beamtime. During this four-day period, the system processed data rates up to 2.4 GB/s in real-time, reconstructing approximately 24–32 million hits per second. The successful deployment demonstrates the viability of GPU acceleration for meeting the real-time processing requirements in CBM.

Future work includes integration with other GPU-accelerated components such as decoding the raw detector data and the track finder, enabling an end-to-end GPU processing pipeline. Additionally, optimization for the full CBM detector configuration will be needed to ensure scalability to the 10 MHz interaction rates expected during production data taking.

This work is supported by BMBF (05P21RFFC1).

References

[1] P. Senger, Probing dense QCD matter in the laboratory—The CBM experiment at FAIR, Physica Scripta **95**, 074003 (2020). 10.1088/1402-4896/ab8c14

[2] J. Cuveland, D. Emschermann, P. Gasik, D. Hutter, W. Müller, C. Sturm, A. Bercuci, V. Friese, I. Frohlich, J. Fruhauf et al., Technical Design Report for the CBM Online Systems – Part I DAQ and FLES Entry Stage (2022)

[3] ALICE Collaboration, Real-time data processing in the ALICE High Level Trigger at the LHC (2019).

[4] G. Eulisse, D. Rohr, The O2 software framework and GPU usage in ALICE online and offline reconstruction in Run 3, EPJ Web of Conf. **295**, 05022 (2024). 10.1051/epjconf/202429505022

[5] R. Aaij, J. Albrecht, M. Belous, P. Billoir, T. Boettcher, A. Brea Rodríguez, D. vom Bruch, D.H. Cámpora Pérez, A. Casais Vidal, D.C. Craik et al., Allen: A High-Level Trigger on GPUs for LHCb, Computing and Software for Big Science **4** (2020). 10.1007/s41781-020-00039-7

[6] J. Heuser, W. Müller, V. Pugatch, P. Senger, C.J. Schmidt, C. Sturm, U. Frankenfeld, eds., [GSI Report 2013-4] Technical Design Report for the CBM Silicon Tracking System (STS) (GSI, Darmstadt, 2013), https://repository.gsi.de/record/54798

[7] V. Friese, A cluster-finding algorithm for free-streaming data, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 01008

[8] H. Malygina, Hit reconstruction for the Silicon Tracking System of the CBM experiment, Ph.D. thesis, Goethe University (2018)

[9] O. Green, S. Odeh, Y. Birk, Merge Path - A visually intuitive approach to parallel merging (2014), 1406.2628

[10] Nvida, Nvidia CUB, https://nvidia.github.io/cccl/cub/, [Accessed 06-02-2025]

[11] CBM Collaboration, mCBM@SIS18 - A CBM full system test-setup for high-rate nucleus-nucleus collisions at GSI / FAIR, CBM (GSI, Darmstadt, 2017), https://repository.gsi.de/record/220072