Porting the CMS Trackster to a GPU-Friendly Data Layout with Alpaka

Muhand Emad Alsaiari
malsaiari0043@stu.kau.edu.sa
Computer Science, King Abdulaziz University
KAUST Academy Summer Programme 2026

Project 15 — CMS Experiment, CERN
Mentor: Felice Pantaleo • Supervision: Simone Balducci
TICL group, CMS

Abstract

Reconstruction software for the CMS experiment at CERN must keep pace with the much higher data rates expected from the High Luminosity LHC, and is therefore being moved onto GPUs. Running efficiently on a GPU requires the data to be laid out as a Structure of Arrays rather than an Array of Structs. This project converts the Trackster, a core data structure of the TICL reconstruction framework, to a Structure of Arrays layout using the CMSSW SoA templates and the Alpaka performance-portability layer, together with host and device collections and a back-pointer design for its variable-length members. The layout is verified by execution: data written on the host, transferred to an NVIDIA GPU and read back is unchanged in every field. The layout is then applied to the pattern recognition step of TICL, whose interface was changed to return the collection rather than fill a growable vector, because a Structure of Arrays must know its sizes before it is allocated. That conversion builds; its runtime verification is with CMS supervision. No performance measurement was made.

August 2026

Contents

1 Introduction **2**

2 Related Work **2**
2.1 Performance portability and Alpaka 2
2.2 Clustering in HGCAL 2
2.3 Array of Structs and Structure of Arrays 2

3 Methodology **3**
3.1 Declaring the SoA layout 3
3.2 Variable-length members 4
3.3 Host and device collections 4
3.4 Applying the layout to pattern recognition 4

4 Results **5**

5 Discussion **6**

6 Limitations and Future Work **6**
6.1 Limitations 6
6.2 Future work 6

7 Conclusion **7**

Acknowledgements **7**

References **7**

1 Introduction

The Large Hadron Collider crosses beams of protons forty million times a second, and every crossing has to be turned back into a list of particles by software. This step is called reconstruction, and it runs on a large farm of CPUs and GPUs.

From 2030 the accelerator will run as the High Luminosity LHC, with up to two hundred proton–proton collisions overlapping in a single crossing — roughly three times more particles in every snapshot of the detector. The detectors themselves are being rebuilt for these conditions [1], and the reconstruction software has to keep pace. Adding CPUs alone will not close the gap, so more of the reconstruction chain must run on GPUs.

The upgraded CMS endcap calorimeter, the High Granularity Calorimeter (HGCAL), has about six million readout channels, and a single snapshot leaves on the order of half a million energy deposits in it. On their own these are just points in space carrying an energy value. TICL, The Iterative CLustering, is the framework that makes sense of them: it works out which deposits were left by the same particle and groups them into objects called *tracksters*. A trackster is therefore one particle's shower — its constituent clusters, together with derived quantities such as energy, barycenter and the principal axes of its shape.

TICL is being migrated onto the GPU one step at a time. The first requirement of that migration is not a faster algorithm; it is a data layout the GPU can read efficiently. This report describes the conversion of the Trackster to such a layout, its verification on real GPU hardware, and its application to the first step of TICL that produces tracksters.



Figure 1: The TICL reconstruction chain. This project targets the pattern recognition step, which builds tracksters.

2 Related Work

2.1 Performance portability and Alpaka

Writing one implementation of an algorithm for the CPU and a second for the GPU duplicates code and doubles the maintenance cost. Performance portability libraries avoid this. CMS has adopted Alpaka [2], a header-only C++ library that abstracts the levels of parallelism exposed by different hardware while still allowing hardware-specific details where they matter. Code written once against Alpaka compiles and runs on CPUs, on NVIDIA GPUs and on AMD GPUs. The CMS offline software framework, CMSSW [3], provides Alpaka-backed portable collections and a set of macros for declaring Structure of Arrays layouts, and these were the starting point for this work.

2.2 Clustering in HGCAL

The clustering that feeds TICL is performed by CLUE [4], a parallel density-based algorithm designed for high granularity calorimeters. CLUE and its successors already run on the GPU; the tracksters built from their output did not, because the data structure holding them was not in a layout a GPU kernel could read efficiently. Closing that gap is the subject of this report.

2.3 Array of Structs and Structure of Arrays

Data is normally stored one object at a time — an Array of Structs (AoS) — because that mirrors the way the objects are described in the problem domain. The alternative is a Structure of Arrays

(SoA), in which each property becomes its own contiguous array and one object is a row index shared across all of them.

Which layout is better is a property of the *access pattern*, not of the hardware. When a small number of objects are read in full, AoS is cheap: one fetch brings in a whole object and all of it is used. When many objects are read but only a few of their fields are touched, SoA is cheap: the values read next to each other in the program also sit next to each other in memory.

Data-parallel kernels almost always fall into the second case, and this is why SoA dominates on GPUs. A GPU fetches memory in wide contiguous transactions. Under an AoS layout each thread pulls in an entire object in order to use one number from it, and most of the fetched bandwidth is discarded. Under an SoA layout the values needed by neighbouring threads are adjacent, and a single transaction serves all of them. CPUs benefit from the same effect through their cache lines; the difference is that on a GPU the penalty for getting it wrong is much larger.

Figure 2 shows both layouts for the same four fields of the same tracksters. Reading the energy of consecutive tracksters, one fetch of eight values yields two useful values under AoS and eight under SoA. TICL reads thousands of tracksters and touches only a few fields of each, so its access pattern is the second case.

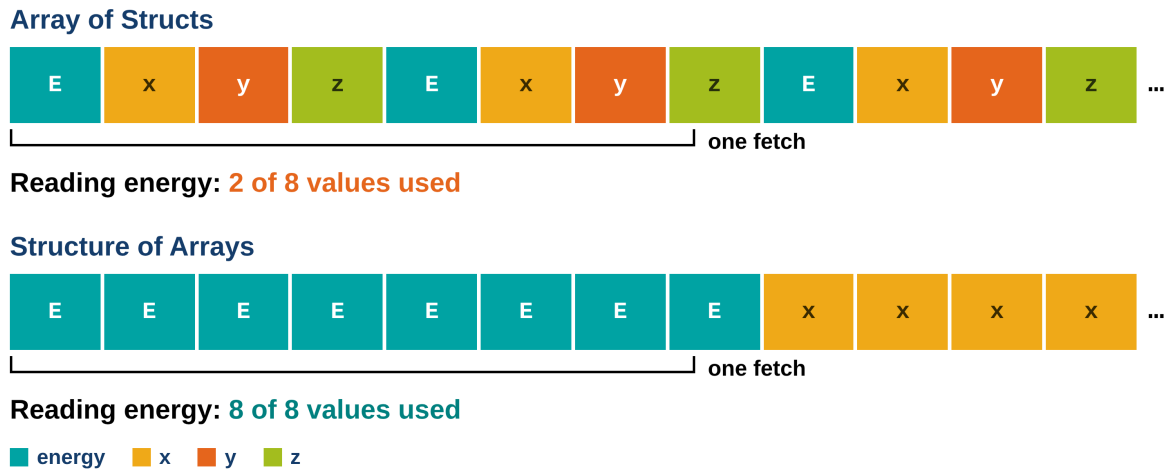


Figure 2: The same four fields of the same tracksters, rearranged. Simplified to four fields; a trackster carries more.

3 Methodology

3.1 Declaring the SoA layout

CMSSW provides a set of macros for declaring SoA layouts. A layout is declared once — which properties it holds and of what type — and the framework then generates the code to allocate it in host memory or in device memory, to move it between the two, and to access it through a lightweight *view*. The Trackster layout was written using these macros and contains:

- scalar-per-trackster columns for the derived quantities: raw energy, raw electromagnetic energy, transverse momentum, the barycenter components, the eigenvalues of the principal component analysis, the identification probabilities and the iteration index;
- Eigen columns for the three principal axes, declared with `SOA_EIGEN_COLUMN` so that each row is read as a vector while the underlying storage stays column-wise;
- variable-length blocks for the members whose length differs from one trackster to the next.

3.2 Variable-length members

A trackster does not have a fixed number of constituent clusters. In the AoS version these were `std::vector` members, which cannot survive the conversion: an SoA has a fixed extent decided when it is allocated.

The solution adopted, agreed with the day-to-day supervisor, is a *back-pointer* design. The constituent clusters of every trackster in the event are stored in one flat block, and each row of that block records the index of the trackster it belongs to (Figure 3). This replaces per-object variable-length storage with a single flat array plus an ownership column, which is the form a GPU kernel can iterate over directly. The same pattern carries the graph edges, the associated track indices and the associated GSF track indices. The blocks are combined into a single collection, so a Trackster collection is allocated in one operation with one size per block.

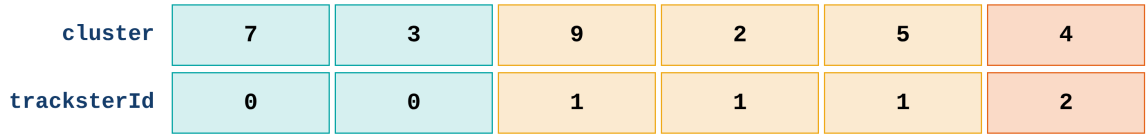


Figure 3: A flat array plus an ownership column — the form a GPU kernel can walk directly. Colour indicates the owning trackster.

3.3 Host and device collections

Two collections were built on top of the layout: a host collection, which owns the data in CPU memory, and a device collection, which owns it in the memory of whichever accelerator Alpaka is compiled for. The framework handles the copy between them. Support for multi-block layouts required constructors selected by a `requires` clause on the presence of the generated block-count member, so that a multi-block layout takes one size per block while the single-size constructor is excluded for it.

3.4 Applying the layout to pattern recognition

With the layout in place, it was applied to the first step of TICL that produces tracksters: pattern recognition, which runs the CLUE algorithm [4] over the clustered energy deposits and builds tracksters out of the result.

In the AoS version the entry point filled a `std::vector<Trackster>` passed in by reference as an output parameter. This pattern cannot survive the conversion. A vector can grow as elements are appended; an SoA cannot, because its extents are fixed when it is allocated, and the number of tracksters and the total number of constituent clusters only become known *inside* the function, after the clustering has run. The signature was therefore changed to return the collection by value:

```
- void makeTracksters(...,
        std::vector<Trackster>& out)
+ TracksterHost makeTracksters(...)
```

The sizes are read from the clustering result, the collection is allocated once with those sizes, and the data is written into it. The blocks for edges, track indices and GSF track indices are allocated empty, because pattern recognition does not produce them: edges come from the trackster linking step and track indices from track linking.

The AoS version computed the derived quantities of each trackster in a loop nested inside a loop over tracksters. The SoA version is written as three passes over the data: one to initialise the per-trackster scalars, one to accumulate energy and the energy-weighted barycenter over the flat block of constituent clusters, and one to normalise and compute the transverse momentum.

This is not a mechanical translation of the layout; it is the shape the flat block imposes, and it is also the shape a parallel kernel needs.

Two pieces of dead code were removed in the process: a boolean flag hard-coded to true, so that the branch guarded by it was unreachable, and a vector of per-cluster energies that was built but never read. The producer wrapping the algorithm was updated accordingly: the event put token now carries the host collection type, and the value returned by the algorithm is placed directly into the event.

4 Results

A layout can compile and still be silently wrong. Missing members, misspelled names and headers that are never actually reached all compile without complaint; a clean build proves syntactic validity and nothing more. The layout was therefore tested by execution rather than by inspection. Three tests were written:

1. a host-only test that writes every scalar, every column, every Eigen column and every block, reads them back and checks the values;
2. a device test on the serial backend;
3. a round-trip test that fills the collection on the host, copies it to the device, copies it back and verifies that every field is unchanged (Figure 4).

All three pass. The round-trip test was run on an NVIDIA GPU node using the CUDA asynchronous backend and exits with status 0, which establishes that the layout, its block structure and its Eigen columns survive a transfer to GPU memory and back without corruption. The AMD backend compiles, but no AMD hardware was available to run it on.



Figure 4: The round-trip test. Scalars, columns, Eigen columns and every block return unchanged; the test exits with status 0 on a real GPU node.

The pattern recognition conversion described in the previous section compiles cleanly for the serial backend. It has not been executed. Late in the project the integration build that the working area targeted was retired from the shared software distribution, which removed the ability to rebuild or run locally; the branch was handed to the day-to-day supervisor as a scoped patch so that runtime verification can be carried out in a working area.

One risk identified during the conversion was resolved by inspection of the clustering library rather than deferred to runtime. The number of constituent-cluster rows allocated is taken from the `extents` of the clustering result, and the fill loop that follows writes one row per (trackster, cluster) pair without a bounds check; if the two disagree the result is a write past the end of the block or rows left uninitialised, neither of which necessarily crashes or produces an obviously wrong number. Reading the CLUEstering sources establishes that they agree by construction: the association map is built with its element count set to the number of points carrying a non-negative cluster identifier, that is, with outliers excluded, and the same set of points is what the fill loop traverses. The allocation is therefore exact, and the equality is worth asserting in the code as a regression guard rather than as a diagnostic.

No performance measurement was made. This work produces the data layout that a fast implementation requires; it does not itself claim a speed-up, and no benchmark was run.

5 Discussion

The result that matters is not that the code builds but that the layout survives a real transfer to GPU memory with every field intact. That distinction shaped how the project was carried out, and it is worth stating plainly: a clean build proves syntactic validity and nothing more, so every claim in this report is labelled by the evidence behind it — executed, established from source, or merely compiled.

One diagnostic is worth recording because it cost significant time and is not documented anywhere obvious. `Eigen/Core` must be included *before* the SoA headers, and the check that enforces this is a `static_assert` that fires per translation unit and only when a view is instantiated. A single file that includes the SoA headers first produces dozens of lines of unrelated template errors, with the one meaningful line buried in the middle. Fixing the include order inside the layout header is not sufficient: every translation unit that reaches the SoA headers has to be corrected. The failure mode is also asymmetric — a build that never instantiates a view will not fire the assertion at all, so an Eigen column can go silently unverified.

The back-pointer design deserves comment because it generalises. The obstacle to an SoA is rarely the scalar fields; it is the members whose length varies per object. Flattening them into one block with an ownership column trades a per-object indirection for a single contiguous array, which is both what the GPU wants to read and what a parallel kernel wants to iterate over. The same pattern applied unchanged to four different members of the Trackster, which suggests it is the right shape rather than a convenience for one case.

Finally, converting pattern recognition exposed a constraint that is easy to miss when the conversion is described as a change of layout. The interface had to change as well. An output parameter of growable type and an allocate-once collection are not interchangeable, and no amount of care inside the function body hides that: the sizes have to be known before the allocation, which means the clustering has to have run, which means the collection cannot be handed in from outside. Layout and interface are coupled, and the second is where the real design work turned out to be.

6 Limitations and Future Work

6.1 Limitations

- **The pattern recognition conversion has not been executed.** It compiles for the serial backend, and runtime verification sits with the CMS supervision after the integration build the working area targeted was retired from the shared software distribution.
- **The TICL plugins are built for the serial backend only**, which is the existing configuration of the branch and not a limitation introduced by this work. The GPU build of those plugins is a separate step.
- **No AMD hardware was available**, so that backend is verified only to the extent that it compiles.
- **No performance measurement was made**, so no claim of a speed-up is made anywhere in this report.
- **The remaining consumers of the AoS structure were left untouched.** The old structure is referenced by roughly seventy files across ten packages, and because it is a class with methods while the SoA is a view with column accessors, every call site changes shape rather than being a textual substitution. Scoping the work to pattern recognition was a decision taken with the supervisor at the start of that phase.

6.2 Future work

1. Execute the converted pattern recognition, retaining the row-count assertion as a regression guard.

2. Build the TICL plugins for the CUDA backend and move the property computation into an Alpaka kernel, which the three-pass structure was written to allow.

3. Migrate the remaining consumers of the AoS structure, after which the old class can be deleted; no compatibility converter is planned.

4. Benchmark the SoA path against the existing CPU implementation on representative events, which is the first point at which a speed-up can honestly be claimed.

7 Conclusion

The Trackster of the CMS TICL framework was converted from an Array of Structs to a Structure of Arrays layout using the CMSSW SoA templates and the Alpaka portability layer, including host and device collections and a back-pointer design for its variable-length members. The layout is verified by execution: data written on the host, transferred to an NVIDIA GPU and brought back is unchanged in every field.

The layout was then applied to the pattern recognition step of TICL, which required changing the interface from an output parameter to a return value because a Structure of Arrays must know its sizes before it is allocated. That conversion builds, and its runtime verification is in the hands of the CMS supervision after the loss of the local build environment.

The general lesson of the project is the one that governed the way it was carried out: the arrangement of data in memory is a design decision on the same footing as the choice of algorithm, and code that compiles is not code that is correct.

Acknowledgements

I thank KAUST Academy for the opportunity, Felice Pantaleo for hosting the project, Simone Balducci for day-to-day supervision, and the TICL group at CERN.

References

[1] CMS Collaboration. The phase-2 upgrade of the cms endcap calorimeter. Technical Report CERN-LHCC-2017-023, CMS-TDR-019, CERN, Geneva, 2017.

[2] Erik Zenker, Benjamin Worpitz, René Widera, Axel Huebl, Guido Juckeland, Andreas Knüpfer, Wolfgang E. Nagel, and Michael Bussmann. Alpaka – an abstraction library for parallel kernel acceleration. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 631–640, 2016.

[3] CMS Collaboration. CMSSW: CMS offline software. `https://github.com/cms-sw/cmssw`. Accessed August 2026.

[4] Marco Rovere, Ziheng Chen, Antonio Di Pilato, Felice Pantaleo, and Christopher Seez. CLUE: A fast parallel clustering algorithm for high granularity calorimeters in high-energy physics. *Frontiers in Big Data*, 3:591315, 2020.