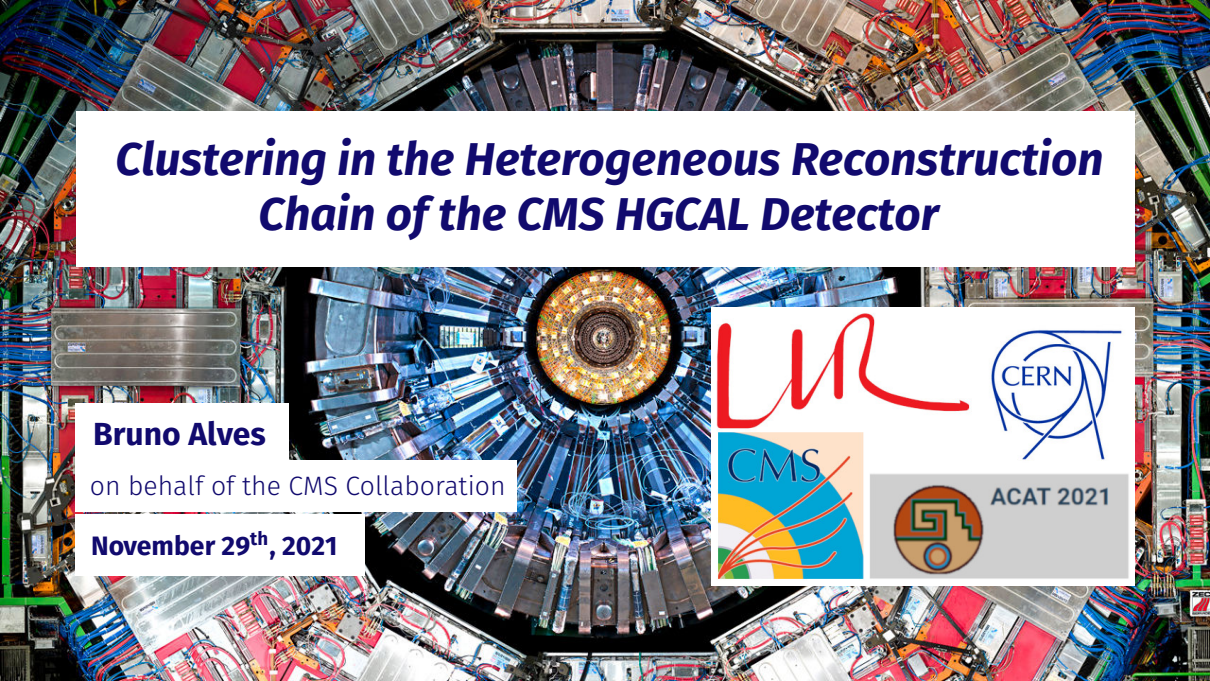
Clustering in the Heterogeneous Reconstruction Chain of the CMS HGCAL Detector



Bruno Alves

on behalf of the CMS Collaboration

November 29th, 2021

ACAT 2021

Motivation

⇒ ***High Luminosity LHC*** (HL-LHC) will start its operation in 2028

- ***large pileup (PU)***: up to 200 collisions per bunch crossing
- ***high luminosities***: 5×10^{34} (7.5×10^{34}) $\mathrm{cm}^2\,\mathrm{s}^{-1}$ for 140 (200) PU in Run4 (Run5)

⇒ The biggest contributor to CPU usage is event reconstruction (∼5% by HGCAL)

⇒ ***GPU-based reconstruction***

- ***Flexibility*** to run on HPCs and upgraded WLCG sites
- Better computing and physics performance (new algorithms)
- Profit of potentially ***favourable GPU/CPU cost*** per unit capacity

⇒ ***Heterogeneous efforts in CMS***

- ***HGCAL***: this talk covers the clustering and geometry deployment steps
- In parallel: Pixel Tracks and Vertices, ECAL/HCAL
- Example for Run3: 30 % HLT reconstruction code will be offloaded to GPUs

High-Granularity CALorimeter (HGCAL) @ CMS

Major CMS Phase2 upgrade: ***unprecedented spatial and timing resolution***

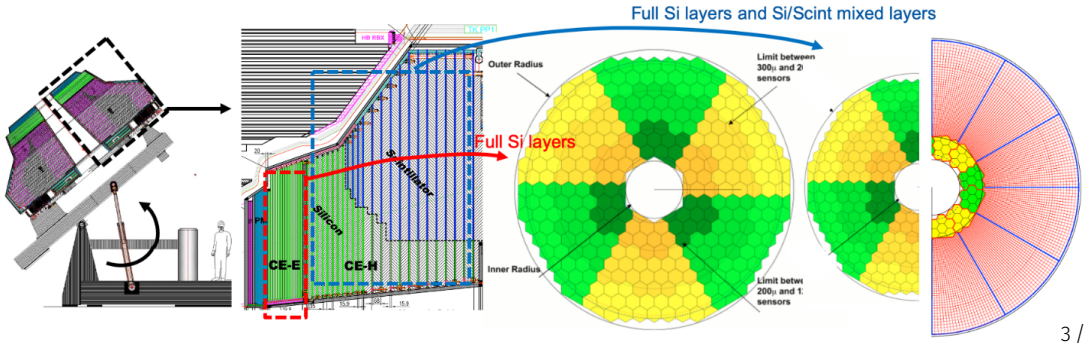
⇒ ***Silicon sensors*** (EM + HAD)

- 28 (EM) + 22 (HAD) layers, 640 m^2
- ∼30K wafers (∼6M channels)
- High and low density cells (LD/HD)

⇒ ***Plastic Scintillator + SiPM*** (HAD)

- 14 layers, 370 m^2
- ∼4K tiles (∼240K channels)

⇒ **Weight:** 200 t per endcap



CMSSW: General Concepts

⇒ HGCAL reconstruction chain is part of the ***CMS Software (CMSSW)***

⇒ A series of reconstruction steps (***modules***) make the chain. They:

- ***consume data collections*** from previous modules
- ***produce data collections*** for subsequent modules

⇒ CMSSW can handle ***multiple jobs and CPU threads***

⇒ It allows asynchronous and non-blocking offload to ***external accelerators***

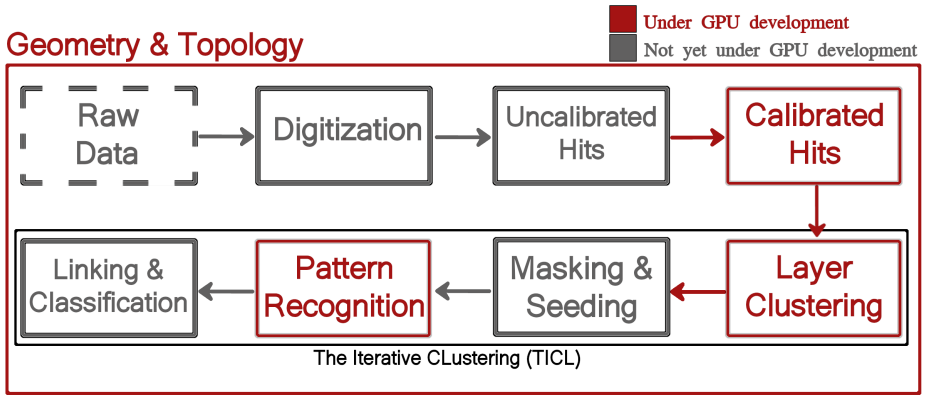
⇒ Modules may access ***conditions data***, i.e., data which are not directly linked to a particular physics event:

- Available instead per ***Interval of Validity***
- ***Detector geometry***: materials, detector element positions, ... (very complex for HGCAL: major contribution for Phase-2 simulation time)
- Other properties: alignment, calibration, ...

HGCAL Reconstruction Chain in the CMS Software (CMSSW)

⇒ The chain below has many ***modules***: from ***Raw Data*** up to ***Physics Objects***

⇒ TICL is a modular framework which iteratively builds 3D clusters from 5D (x, y, z, t, E) calorimetric data and outputs particle properties and probabilities[1]

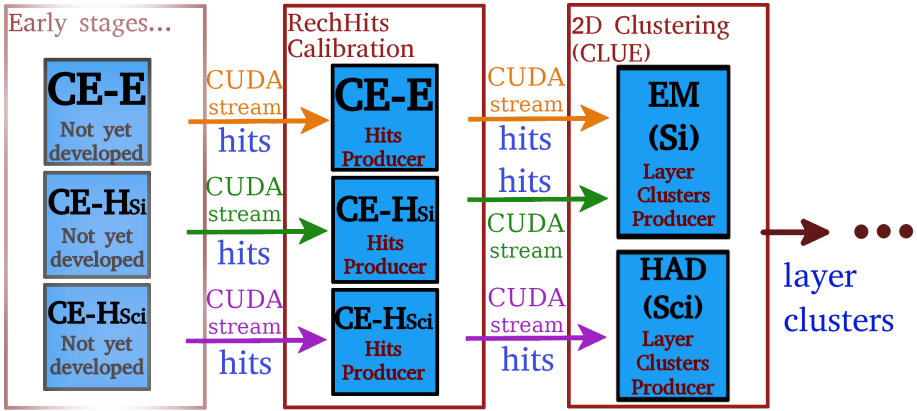


[1] see F. Pantaleo's plenary talk

Subdetectors in HGCAL's Chain

⇒ By splitting data from the same reconstruction step, modules can process it in parallel (and differently if need be):

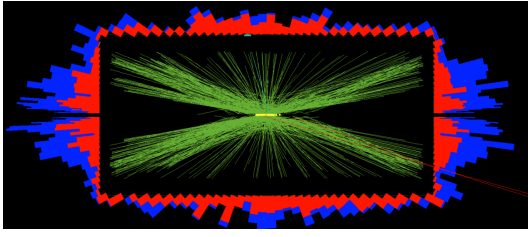
- ***Multiple subdetectors***: Silicon (***EM*≡*CE-E*+*CE-H$_{Si}$***) and Scintillator (***HAD*≡*CE-H$_{Sci}$***)
- Single ***CUDA stream*** per subdetector: ×***2 throughput*** for pixel reconstruction

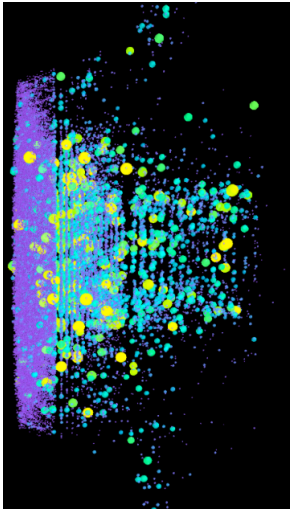


Clustering in HGCAL: High Multiplicity and Granularity Challenges

To give an idea: 78 PU in the tracker (below) / ***200 PU*** in HGCAL (right)

- ⇒ ***O(100K)*** hits per event
- ⇒ ***O(10K)*** 2D layer clusters
- ⇒ 2D clusters collections form 3D objects
- ⇒ Increased complexity: design fast and accurate algorithms, staying within the High Level Trigger PU200 time budget
  - Benefits from GPU Computing





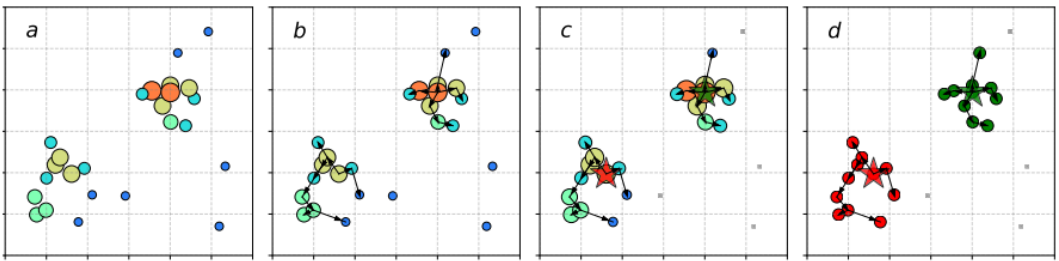
CLUstering by Energy (CLUE) in a Nutshell [1]–[3]

a), b) ***Fast and fully-parallelizable density-based clustering algorithm***: energy densities + separation distances are assigned to all hits;

c) Hits are classified as ***seeds***, ***followers*** or ***outliers*** based on a critical distances and densities (tunable parameters);

d) Clusters are built by ***traversing each seed followers' tree***, and assigning the same seed index to all its followers.

⇒A ***standalone version*** (no CMSSW) is available for testing and optimizations



Intermezzo: Structure of Arrays (SoAs) as Data Format

⇒ SoAs improve access to global memory and exploit CPU vectorization

⇒ Device data uses the SoA format

- takes advantage of ***memory coalescing*** and ***warp alignment***

⇒ CMS is currently investigating novel CPU SoA-like formats (eg. AoSoA):

- decrease/eliminate the time for SoA ↔ Legacy conversions (next slide)
- potentially ***accelerate CPU code***

```
//Structure of Arrays
struct pointlist3D {
  float x[N];
  float y[N];
  float z[N];
};
struct pointlist3D points;
float get_point_x(int i) {
    return points.x[i]; }
```

```
//Array of Structures
struct point3D {
  float x;
  float y;
  float z;
};
struct point3D points[N];
float get_point_x(int i) {
    return points[i].x; }
```

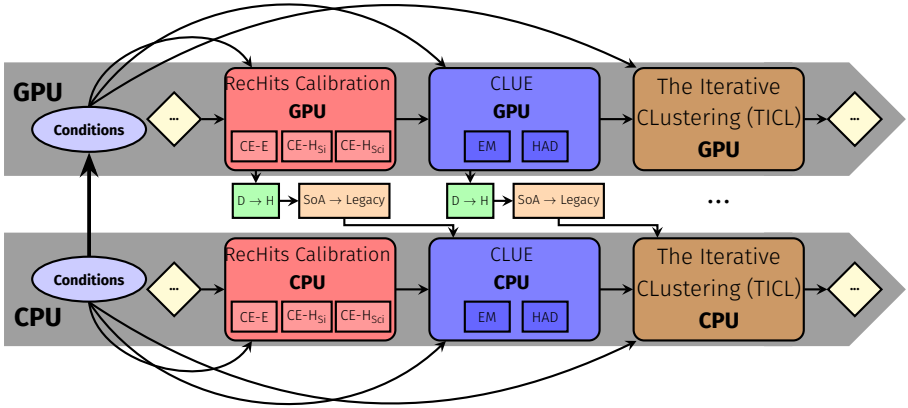
Heterogeneous HGCAL Reconstruction Flowchart

⇒ Different modules run on CPUs and GPUs, where conditions are deployed

⇒ ***GPU → CPU data conversion modules*** bring flexibility and ease validation:

- Modules for data conversions: Device-Host (D→H) and SoA to CPU Legacy

⇒ The diagram is simplified: jobs and CPU threads increase complexity



Benefits Accrued from Porting CLUE to CMSSW

⇒ ***General:***
- ✓ Integration with other reconstruction steps
- ✓ Custom caching allocator
- ✓ Use the full HGCAL geometry
- ✓ Linking modules in the GPU
- ✓ Bonus #1: Cluster energy and position calculation in the device
- ✓ Bonus #2: Conditions data deployment to the GPU

⇒ ***Data Format:*** Single SoA memory block
- ✓ Increased spatial locality
- ✓ Smaller allocation and initialization time
- ✓ Warp data alignment

⇒ ***Dynamic memory allocations:***
- ✓ Allocate depending on event size (further optimizations are being explored)

⇒ ***Subdetectors*** (Silicon vs Scintillator changes the algorithm):
- ✓ Exploit subdetectors parallelism

Bonus #1: Clusters Energy and Position Calculation

⇒ A ***reduced cluster SoA is introduced*** to forward the information to TICL:

- *hit-level* → *cluster-level*
- Its defining quantities are ***energies*** and ***positions***

⇒ Calculations in the GPU are parallelized at cluster-level:

- ***Energy:*** sum the energy of all hits in a cluster
- ***X and Y positions*** ($Z \in \{x, y\}$, E_i is the energy of hit i, and W_0 is a tunable parameter)***:***

$$Z = \frac{\sum Z_i W_i}{\sum W_i}, \quad W_i = W_0 + \log\left(\frac{E_i}{\sum E_i}\right)$$

⇒ Both imply ***traversing trees*** (or chains) ***of followers***

- The CPU version instead flattens the tree before the calculation

Bonus #1: Local Stack & Loop or Function Recursion?

⇒ Different approach: ***device function recursion*** instead of CLUE stack + loop

- ✓ adds clarity, reduces algorithm allocated memory
- ⊗ potential function stack frame growth

⇒ Profiling required!

```
void func() {
    buffer[stackSize] = allocate();
    while(!buffer.empty()) {
        i = buffer.back();
        buffer.pop_back();

        for(auto fl : i.followers()) {
            fl.clusterID = i.clusterID;
            buffer.push_back(fl);
        }
    }
}
func();
```

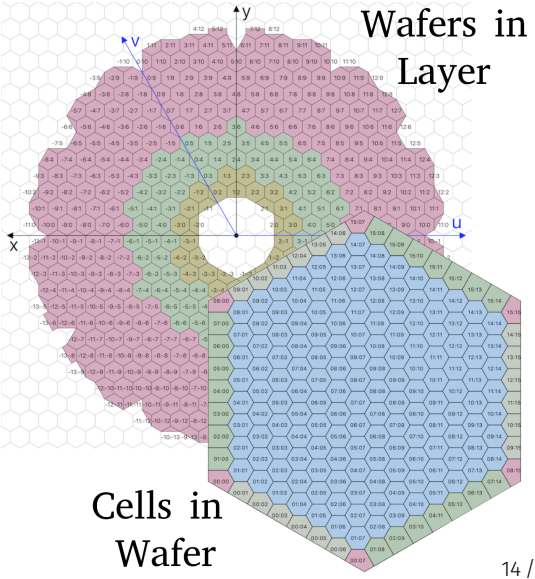
Cluster ID assignment with stack + loop

```
void func(&vars, &followers) {

    for(auto fl : followers()) {

        //energy and positions
        some_calculation(vars);

        func(vars, fl.followers());
    }
}

vars = set_variables();
func(vars, followers);
```

Calculations with function recursion

Second Intermezzo: Detector IDs

- ⇒ CMS ***detector elements*** are encoded in 32 bit ***IDs***, part of which reserved to HGCAL (layers, wafers, cells, ...)
- ⇒ An ***ID*** gives access to all geometry-related information for a detector element
  - *x* and *y* positions required by CLUE!
- ⇒ The ***ID*** class is ported to the GPU
  - Device kernels are now able to access its information



Bonus #2: Geometry Deployment to the GPU

- ⇒ ***Remove host-device transfers*** of conditions data between modules
- ⇒ Data available to all reco chain via a ***single transfer per Interval of Validity***
- ✓ Avoids slow and recurrent CPU → GPU data transfers
- ✓ Reduced memory consumption of modules (including CLUE) by avoiding directly storing conditions data
- ✓ Makes data transfers faster, since they are ***now done in parallel***:
  - Only the IDs of detector elements are transferred serially
  - From the GPU-converted ID all geometry information can be retrieved
  - The (x, y) positions GPU filling is embarassingly parallel

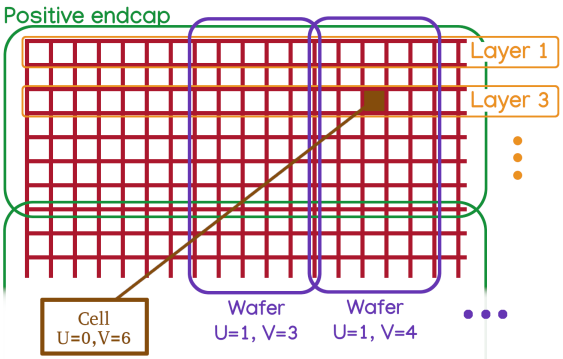
Bonus #2: Geometry Deployment and Access from the GPU

⇒ A ***fast***, ***user-friendly*** and ***generalizable*** mechanism to deploy conditions

⇒ Filling loop follows an intuitive ***geometrical order***

Loop over (Silicon example):

1. Two endcaps;
2. Three subdetectors;
3. Layers;
4. Square grid to cover all wafers;
5. Silicon cells (HD and LD).

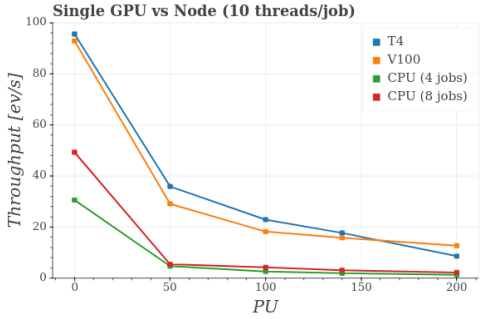


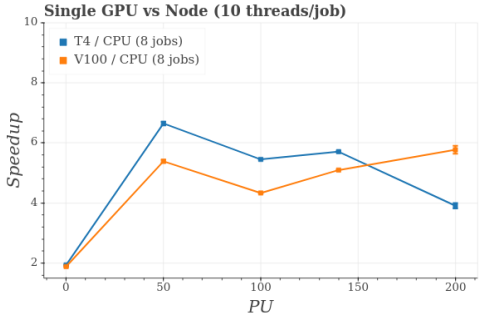
⇒ Given an ID, its memory location in the GPU is immediately found ($O(1)$).

⇒ A GPU user ***only needs to know the IDs*** of the elements it is interested in.

Performance of RecHit Calibration [4] + CLUE: Throughput and Speedup

⇒ ***Full*** Intel(R) Xeon(R) Silver 4114 with 40 logical cores vs. ***Single GPU*** (T4-16GB or V100-32GB), 10 CPU threads per job, 512 GPU threads per block





⇒ The speedup peaks between ***5*** and ***6*** for PU140 (Run4) and PU200 (Run5)

⇒ Additional measurements allows to conclude that ***data conversion modules and recursion functions do not affect the throughput***: CLUE is the bottleneck

Conclusions & Next Steps

- The ***CLUE standalone algorithm*** has been ***integrated in the CMS software***
- ***Conditions data have been deployed to the GPU*** with a new approach
- Performance measurements show ***speedup benefits***
- This work paves the way for future HGCAL accelerator developments:
  - ⇒ Facilitates the ***deployment of the clustering to multiple accelerators*** using abstraction libraries (*eg.* alpaka, see W. Redjeb's poster);
  - ⇒ Several optimizations are being studied: ***reducing memory footprint*** and ***decreasing the number and size of allocations***;
  - ⇒ Currently investigating ***novel SoA-like formats***
  - ⇒ Port the ***full HGCAL reconstruction*** into GPUs

Back-up

Acknowledgments

Bruno Alves thanks the following institutions for the funding provided:

- Fundação para a Ciência e Tecnologia (FCT): SFRH/BEST/150186/2019
- Organisation européenne pour la recherche nucléaire (CERN)

References (1/1)

[1] M. Rovere, Z. Chen, A. Di Pilato, F. Pantaleo, and C. Seez, "CLUE: A Fast Parallel Clustering Algorithm for High Granularity Calorimeters in High Energy Physics,", 2020. arXiv: *2001.09761*. [Online]. Available: *http://arxiv.org/abs/2001.09761*.

[2] Z. Chen, A. Di Pilato, F. Pantaleo, and M. Rovere, "GPU-based Clustering Algorithm for the CMS High Granularity Calorimeter," EPJ Web of Conferences, vol. 245, p. 05 005, 2020. DOI: *10.1051/epjconf/202024505005*.

[3] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," Science, vol. 344, no. 6191, pp. 1492–1496, 2014, ISSN: 10959203. DOI: *10.1126/science.1242072*.

References (2/2)

[4] B. Alves, A. Bocci, M. Kortelainen, F. Pantaleo, and M. Rovere, "Heterogeneous techniques for rescaling energy deposits in the CMS Phase-2 endcap calorimeter," EPJ Web of Conferences, vol. 251, p. 04 017, 2021. DOI: 10.1051/epjconf/202125104017.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was som
unprocessed data that should have been added to the final page this extra pag
has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away,
because LaTeX now knows how many pages to expect for this document.