Clustering in the Heterogeneous Reconstruction Chain of the CMS HGCAL Detector

Bruno Alves[1,2], Felice Pantaleo[2], Marco Rovere[2]

[1] Laboratoire Leprince-Ringuet, CNRS/IN2P3, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France
[2] CERN, 1211 Geneva 23, Switzerland

E-mail: `bruno.alves@cern.ch`

Abstract. We present an important milestone for the CMS High Granularity Calorimeter (HGCAL) event reconstruction: the deployment of the GPU clustering algorithm (CLUE) to the CMS software. The connection between GPU CLUE and the preceding GPU calibration step is thus made possible, further extending the heterogeneous chain of HGCAL's reconstruction framework. In addition to improvements brought by CLUE's deployment, new recursive device kernels are added to efficiently calculate the position and energy of CLUE clusters. Data conversions between GPU and CPU are included to facilitate the validation of the algorithms and increase the flexibility of the reconstruction. For the first time in HGCAL, conditions data are deployed to the GPU and made available on demand at any stage of the heterogeneous reconstruction. This is achieved via a new geometry ordering scheme in which physical and memory locations are connected. This scheme is successfully tested with the GPU CLUE version reported here, and is expected to have a broad range of applicability for future heterogeneous developments in CMS. Finally, the performance of the combined calibration and clustering algorithms on GPU is assessed and compared to its CPU counterpart.

1. Introduction

The High Luminosity LHC (HL-LHC) will start taking data in 2029, achieving unprecedented instantaneous luminosities of $\sim 5 \times 10^{34}\,\mathrm{cm}^2\,\mathrm{s}^{-1}$ (more than twice LHC's current value) and a pileup of up to 200. An integrated luminosity of $\sim 3\,\mathrm{ab}^{-1}$ will be reached over 10 years [1, 2].

In order to cope with the above, a major upgrade of the CMS endcap calorimeters [3, 4] is being prepared. The novel High Granularity Calorimeter (HGCAL) [2] is an extremely challenging project, requiring the development of reconstruction code capable of fully exploiting the increased granularity under the expected extreme conditions.

The biggest contributor to CPU usage is event reconstruction, of which currently $\sim 5\%$ is used by HGCAL [5]. CMS plans to port part of its reconstruction to Graphics Processing Units (GPUs), which represent one of the most promising hardware accelerator technologies on the market. GPUs are a key element when one considers taking advantage of heterogeneous architectures available on traditional and High-Performance Computing grid sites, including the upgraded Worldwide LHC Computing Grid. GPUs also promote the development of algorithms with better computing performance, and profit from a potentially favourable cost when compared to CPUs, per unit capacity. CMS is planning to adopt a heterogeneous High Level Trigger (HLT) farm already in Run 3 (2022–2025), where ∼30% of the workflow will be offloaded to GPUs (50% and 80% by the end of Run 4 and 5, respectively) [6].

This paper describes the integration of the standalone GPU CLUE (CLUstering of Energy) algorithm [11] with the CMS software (CMSSW) [10]. This effort follows the development of the CMSSW GPU based calibration of energy deposits [7], which immediately preceeds CLUE in the reconstruction chain. This work thus measures the performance of two HGCAL GPU tasks without resorting to intermediate CPU/GPU data transfers. Also, for the first time, we implement and deploy the calculation of cluster energies and positions to the device. We do so using device kernel recursion, as opposed to the local stack method chosen for CLUE. Finally, conditions data are deployed to the GPU and subsequently used by the algorithm. The implementation uses a geometry ordering scheme which minimizes memory consumption.

2. HGCAL and its Reconstruction Chain

HGCAL will be a sampling calorimeter. The proposed design includes an electromagnetic section of silicon sensors as active material in the first 28 layers. A hadronic section comprises 8 silicon-only layers followed by 14 silicon-scintillator hybrid layers, where the scintillation light is read out by silicon photo-multipliers. Both sections are interleaved with absorber layers. HGCAL will comprise $\sim 620\,\mathrm{m}^2$ of silicon and $\sim 400\,\mathrm{m}^2$ of plastic scintillators for a total of, respectively, ∼6 million and ∼240 thousand channels. Three subdetectors form HGCAL's hybrid detection technology: the first 28 layers made exclusively of silicon (CE-E) and the silicon and scintillator parts of the hadronic section (CE-H_{Si} and CE-$\mathrm{H}_{\mathrm{Sci}}$). The reconstruction model envisioned for HGCAL is intended to be fast and flexible, comprising a sequence of modules/stages which transform raw data into physics objects. After the initial generation, simulation, digitization [5] and calibration steps, energy deposits (hits) are clustered by CLUE, a fully-parallelizable density-based clustering algorithm [8], in order to form two-dimensional objects. In a nutshell, CLUE assigns an energy density and a separation distance to all hits, which are later used to classify each hit as either a seed, a follower (based on the hit's nearest highest density), or an outlier. Clusters are built by traversing the tree of followers of each seed, assigning the index of the seed to all its followers. This work includes the calculation of the cluster energy and cartesian positions, which are computed in the device (section 3.1). In addition, a heterogeneous approach for navigating through the detector's geometrical/topological information is devised and used within CLUE (section 3.2).

3. Methodology

The CPU and GPU workflows available in CMSSW follow the same structure (fig. 1). For the GPU case, the "Device to Host", "D→H" and "SoA to Legacy" modules guarantee a robust validation and increase the framework's flexibility. The implementation of all GPU code relies on Nvidia's CUDA [9]. Conditions data are deployed from the CPU to the GPU and remain available in the GPU during an "interval of validity". Algorithms might run in parallel for the CE-E, CE-H_{Si} and CE-$\mathrm{H}_{\mathrm{Sci}}$ subdetectors, using different CUDA streams, as is the case for the hit calibration and clustering stages.

A GPU standalone version of CLUE [11] is currently available and being actively used for testing and optimization purposes. In the following we list the improvements and the features brought by deploying CLUE to CMSSW:

- **Data Formats:** The standalone version is currently using Structure of Arrays (SoAs) where each array performs its own independent memory management. We instead allocate a single SoA memory block for all arrays, thus decreasing the time dedicated to device and host memory allocations (often a bottleneck [12]) and initialization, and improving spatial memory locality. Allocations are made in multiples of CUDA's *warpSize* (currently

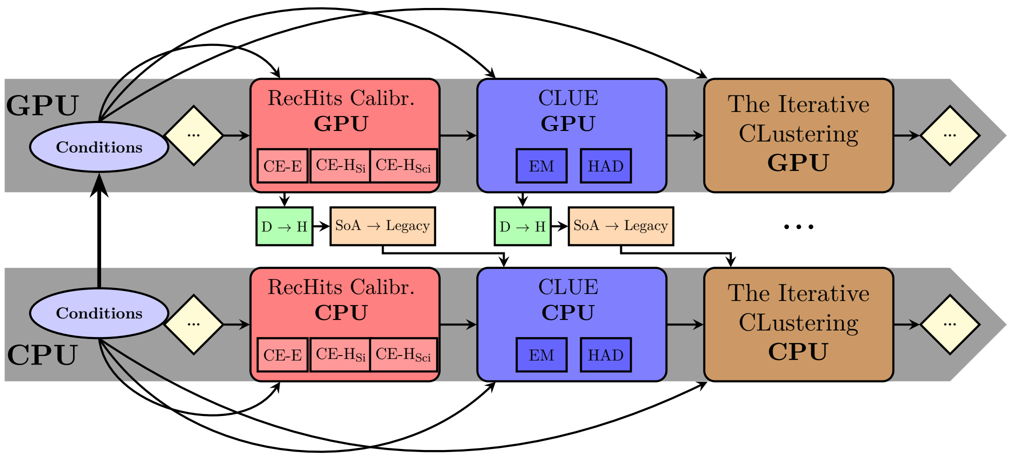

Figure 1. Partial side-by-side view of the HGCAL CPU and GPU reconstruction chains. Each subdetector is asssigned to a different CUDA stream. Data conversion modules (where "D→H" corresponds to "Device to Host") perform GPU to CPU data legacy conversion. Conditions data are deployed from the CPU to the GPU and remain available in the GPU during an interval of validity. The CE-E and CE-H$_{\mathrm{Si}}$ are considered separately for the hit calibration and jointly during the clustering stage (EM). CE-H$_{\mathrm{Sci}}$ and HAD refer to the same hadronic section.

32 threads), and memory is accessed such that coalesced reads are possible (potential throughput increase of one order of magnitude [12]). Padding is added to the end of each array to ensure data alignment with the warps. The memory management is done using CMSSW's caching allocator scheme which is heaviliy based on Nvidia's CUB [13] and which reuses memory regions over multiple events as long as the data can be accomodated.

- **Data Input:** The input is now received from the preceding hit calibration stage [7] in the GPU, avoiding additional host-device transfers present in the standalone version.
- **Data Output:** The output, instead of being *hit-level* and stored in a text file, is converted to a *cluster-level* data format in order to be propagated to the next reconstruction steps. Cluster level information must therefore be calculated and stored (section 3.1).
- **Dynamic Memory Allocations:** The CMSSW version now allocates memory dynamically according to the number of hits in each physics event and in each subdetector. Further optimization is possible and currently being pursued (see section 5).
- **Conditions data:** In CMSSW the detector geometry must be considered and conditions are ported to the GPU (section 3.2).
- **Subdetectors:** Subdetectors are assigned to different CUDA stream, where multiple reconstruction steps can run sequentially.

3.1. Calculation of cluster energy and positions
The calculation of the energy and cartesian x and y positions of the clusters follows the CMSSW CPU approach [8]. A GPU thread is assigned to each cluster. The energy of a cluster is calculated by summing the energy of all its hits. The positions are estimated by using a logarithmic-weighted truncated sum:

$$\beta = \frac{\sum \beta_i W_i}{\sum W_i}, \quad W_i = W_0 + \log\left(\frac{E_i}{\sum E_i}\right) \geq 0 \tag{1}$$

where β stands for x or y, E_i is the energy of hit i, and W_0 is a parameter set to 2.9, which controls the smallest fraction of energy that a cell can have and still contribute to the position measurement (2.9 corresponds to a minimum energy fraction of 5.5%).

For both the algorithm above and CLUE, a method is needed to traverse the tree of followers of a given seed. CLUE opted for defining a local fixed-size stack where, together with a loop, items are added and removed (left side of fig. 2). This method has the side effect of allocating more memory than required. As an alternative, we used device kernel recursion (right of fig. 2). This approach is general, and could be applied to the CPU algorithm as well. Recursion is often used for tree traversal since it adds clarity, and in this case reduces the amount of algorithm-related allocated memory. However, it can also lead to uncontrolled function stack frame growth, effectively using more memory, especially for deeply nested trees. Future measurements will establish which method is more performant in terms of memory consumption and throughput.

```
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();
```

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

  for(auto fl : followers()) {

    //energy and positions
    some_calculation(vars);

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

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

Figure 2. Pseudo-code displaying the two methods employed to traverse all followers in a cluster: local stack plus a loop (*left*) and device function recursion (*right*), avoiding a local stack, where "&" refers to pass-by-reference.

3.2. Deployment of conditions data to the GPU

HGCAL will have approximately 6 million detector elements; a fast and straightforward mechanism to transfer conditions data to the GPU is needed. An achievement of this work is to completely remove host-device transfers of conditions data between modules, making them available to all modules of the reconstruction chain via a single transfer per interval of validity, per CPU job and per available device. This approach offers several advantages:

- Alleviates module (including CLUE) memory consumption, by avoiding the storage of conditions data;
- Avoids slow and recurrent CPU$\rightarrow$GPU data transfers;
- Makes data transfers faster, as *they are now done in parallel*. Indeed, conditions data are mapped via a CMSSW indexing class which allows to retrieve information for each particular detector element. Once all indexes have been transferred (serially), conditions (sensor positions, for instance) can be filled in the GPU in parallel, as they are independent from one another.
- Simplifies the implementation of future GPU reconstruction modules, given that all geometry information is already accessible on the device.

Position information is stored according to HGCAL's geometry: two endcaps, subdetectors, layers, and silicon cells (or tiles for the hadronic section). A single SoA-like memory block is allocated. The same specific order is later exploited on the GPU side to retrieve the correct information, mapping detector elements to their memory locations ($O(1)$ lookup complexity). The procedure keeps data transfer to a minimum while ensuring spatial locality: conditions of adjacent detector elements will often be located very close in memory and be requested at close moments in time. A given set of detector element coordinates (layer, cell, ...) points to a unique location in GPU memory, where a GPU user can access the relevant element's information.

4. Validation and Performance

Both validation and performance measurements are executed starting from three uncalibrated CPU collections of hits (one per subdetector). The hits are transferred to the GPU, are calibrated using techniques described in [7], and are directly forwarded to CLUE. Finally, GPU data in SoA format is converted back to the Legacy CPU format (as in fig. 1).

For the validation, the CPU-only and the GPU steps just described are run over the same input dataset. The clusters thus obtained are compared in terms of their underlying quantities: no differences are found for hit energy densities and distances while minor discrepancies, currently under investigation, are observed for cluster energies and positions.

Performance is assessed using a T4 Nvidia and V100 GPU, and an Intel(R) Xeon(R) Silver 4114 CPU (40 logical cores), using a custom benchmarking tool [14]. Several pileup scenarios are assessed: 0, 50, 100, 140 and 200. We measure the throughput, defined as the number of events processed per second. Throughput measurements are performed using 4 and 8 jobs for the CPU and 1 job for GPU, with 10 CPU threads each and 400 events, skipping the first 100 to mitigate GPU initialization costs. 512 threads per block are tested in the GPU. Larger values were tested with no performance benefit. The measurements include initial uncalibrated hits data transfer from the CPU to the GPU. A single interval of validity was considered[1].

The measurements in fig. 3 show the throughput for multiple CPU and GPU scenarios (left) and the obtained speedups (right), defined as the throughput of a GPU scenario divided by the most performant CPU throughput. We stress that the comparison is being made between a full CMS node and a single GPU. The GPU memory usage observed varied significantly depending on specific algorithm parameters, such as the maximum number of seeds allowed, the maximum number of followers per seed and the size of the local stack (see fig. 2), which can be tuned for different pileup values. The GPU throughput did not change significantly when considering device-to-host data transfers, conversions to legacy formats and the kernel discussed in section 3.1, suggesting the bottleneck lies in the CLUE algorithm itself.

We note that the final chain will include multiple modules accessing data directly in the GPU; memory transfers and allocations will therefore not be requested on a per-module basis, significantly reducing their impact on the overall throughput.

5. Conclusions and Next Steps

This work presents the second HGCAL heterogeneous algorithm to be fully integrated within CMSSW after hit calibration [7]. It describes the algorithm's porting to CMSSW and the GPU implementation of additional, non embarassingly-parallel cluster-related steps. Preliminary performance measurements were made, where the GPU throughput includes data conversions and transfers which will not be present in the final version of the reconstruction chain.

The implementation of the clustering algorithm described in this work will be soon followed by additional developments in HGCAL: using the `alpaka` [15]abstraction library, using RAPIDS [16]

[1] In CMS, intervals of validity last on average some hours, with some exceptions on the order of minutes (eg., calorimeter gains). Even if sensor positions would change once a minute (orders of magnitude faster than what is currently expected), the offline reconstruction rate of 5kHz would imply a change every $\sim$300k events only.

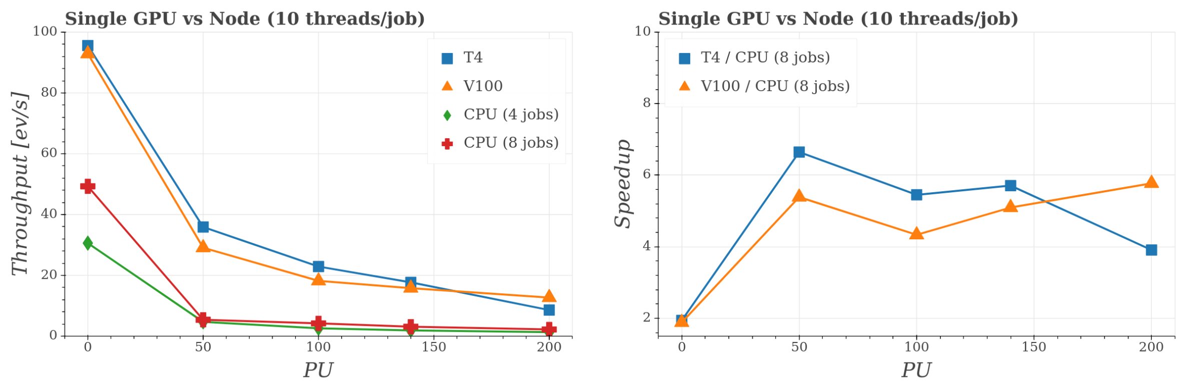

Figure 3. *Left*: Throughput measurements of CLUE as a function of pileup, considering 4 and 8 jobs for the CPU and always 1 job for the GPU, with 10 CPU threads each. The GPU throughput was measured using 512 threads per block. Vertical error bars are included but are generally too small to be seen. *Right*: Speedups obtained after dividing the GPU throughput measurements by the corresponding most performant (8 jobs) CPU measurements.

for memory footprint reduction and investigating novel SoA-like formats. The final goal of the heterogeneous efforts is to perform the entire HGCAL reconstruction in the GPU.

References

[1] The CMS Collaboration 2017 The Phase-2 Upgrade of the CMS Tracker (Technical Design Report) https://cds.cern.ch/record/2272264
[2] The CMS Collaboration 2017 The Phase-2 Upgrade of the CMS Endcap Calorimeter (Technical Design Report) https://cds.cern.ch/record/2293646
[3] CMS Collaboration 2008 JINST **3** S08004
[4] Contardo D, Klute M, Mans J, Silvestris L and Butler J 2015 Technical Proposal for the Phase-II Upgrade of the CMS Detector https://cds.cern.ch/record/2020886
[5] The CMS Collaboration 2021 Evolution of the CMS Computing Model towards Phase-2. https://cds.cern.ch/record/2751565
[6] Badaro, Gilbert *et al* 2021 The Phase-2 Upgrade of the CMS Data Acquisition *EPJ Web Of Conferences* **251** 04023
[7] Alves B, Bocci A, Kortelainen M, Pantaleo F. and Rovere M. 2021 Heterogeneous techniques for rescaling energy deposits in the CMS Phase-2 endcap calorimeter *EPJ Web Of Conferences* **251** 04017
[8] Rovere, M., Chen, Z., Di Pilato, A., Pantaleo, F. & Seez, C. CLUE: A Fast Parallel Clustering Algorithm for High Granularity Calorimeters in High Energy Physics 2020 http://arxiv.org/abs/2001.09761
[9] Vingelmann P and Fitzek F 2021 NVIDIA CUDA release: 11.2.152 https://developer.nvidia.com/cuda-toolkit
[10] Jones C, Paterno M, Kowalkowski J, Sexton-Kennedy L and Tanenbaum W 2006 The New CMS Event Data Model and Framework. *Proceedings Of International Conference On Computing In High Energy And Nuclear Physics (CHEP06).*
[11] Chen Z, Di Pilato A, Pantaleo F and Rovere M Standalone CLUE Algorithm on GPU and CPU (https://gitlab.cern.ch/kalos/clue) Accessed: 2021-11-17
[12] NVIDIA 2021 CUDA C++ Programming Guide https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html Accessed: 2021-02-17
[13] NVIDIA CUB. 2016 https://nvlabs.github.io/cub/ Accessed: 2021-11-17
[14] Bocci A. Patatrack. https://patatrack.web.cern.ch/patatrack/ Accessed: 2021-05-12
[15] Matthes A, Widera R, Zenker E, Worpitz B, Huebl A and Bussmann, M 2017 Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the Alpaka library arXiv 1706.10086 https://arxiv.org/abs/1706.10086
[16] Rapids Development Team 2018 RAPIDS: Collection of Libraries for End to End GPU Data Science https://rapids.ai