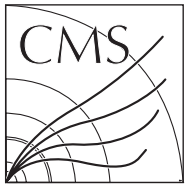
Available on CMS information server

CMS CR -2025/066



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

25 February 2025 (v3, 28 February 2025)

Heterogeneous reconstruction of hadronic particle flow clusters with the Alpaka Portability Library

Andrea Bocci, Abhishek Das, Kenichi Hatakeyama, Florian Lorkowski, Felice Pantaleo, Wahid Redjeb, Jonathan Samudio, Mark Saunders for the CMS Collaboration

Abstract

In response to increasing data challenges, CMS has adopted the use of GPU offloading at the High-Level Trigger (HLT). However, GPU code is often hardware specific, and increases the maintenance burden on software development. The Alpaka (Abstraction Library for Parallel Kernel Acceleration) portability library offers a solution to this issue, and has been implemented into the CMS software (CMSSW) for use online at HLT. A portion of the final-state particle candidate reconstruction algorithm, Particle Flow, represented a target for increased performance through parallel operation. We discuss the port of hadronic Particle Flow clustering to Alpaka, and the validation of physics and performance at HLT for 2024 data taking.

Presented at *CHEP2024 27th International Conference on Computing in High Energy and Nuclear Physics*

Heterogeneous reconstruction of hadronic particle flow clusters with the Alpaka Portability Library

Andrea Bocci[1], Abhishek Das[2], Kenichi Hatakeyama[3], Florian Lorkowski[4], Felice Pantaleo[1], Wahid Redjeb[5] Jonathan Samudio[3], Mark Saunders[3],
on behalf of the CMS collaboration

[1]CERN, European Organization for Nuclear Research, Meyrin, Switzerland
[2]University of Notre Dame, Notre Dame, Indiana, USA
[3]Baylor University, Waco, Texas, USA
[4]Deutsches Elektronen-Synchrotron, Hamburg, Germany
[5]RWTH Aachen University, III. Physikalisches Institut A, Aachen, Germany

Abstract. In response to increasing data challenges, CMS has adopted the use of GPU offloading at the High-Level Trigger (HLT). However, GPU code is often hardware specific, and increases the maintenance burden on software development. The Alpaka (Abstraction Library for Parallel Kernel Acceleration) portability library offers a solution to this issue, and has been implemented into the CMS software (CMSSW) for use online at HLT. A portion of the final-state particle candidate reconstruction algorithm, Particle Flow, represented a target for increased performance through parallel operation. We discuss the port of hadronic Particle Flow clustering to Alpaka, and the validation of physics and performance at HLT for 2024 data taking.

1 Introduction

As the LHC increases collision rates during Run 3 and projects higher pileup scenarios for the HL-LHC era, the CMS experiment [1] has adopted the use of GPU acceleration in the High-Level Trigger (HLT) system [2]. The HLT uses a slimmed set of subdetector information to make trigger decisions on data. The timing budget of HLT is limited to 500 ms per event during Run 3 [3] and developments on improved reconstruction algorithms must account for this. The Alpaka portability library [4] was introduced in CMSSW to offer a hardware-agnostic approach to GPU source code and also provides simplified maintainability across hardware back-ends. The Particle Flow (PF) algorithm [5] uses information from all CMS subdetectors to perform global event reconstruction and produce final-state particle candidates in each event. One particular step, the formation of clustered energy deposits in the hadronic calorimeter (HCAL), was previously ported to a CUDA-only implementation [6] and has been transitioned to Alpaka for use online at HLT.

The PF workflow is outlined in Fig. 1. Particle Flow uses reconstructed calorimeter hits (rechits) and tracking information from the tracker and muon systems as inputs. The resulting outputs are PF clusters and PF tracks, respectively. These elements are then linked together to form PF blocks which are used in the creation of the final-state particle candidate objects. Prior to the CUDA version, the formation of hadronic PF clusters was one of the most computationally intensive tasks during the PF reconstruction at HLT. Following the latest guidelines

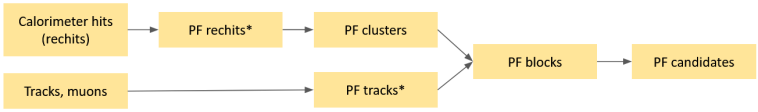


Figure 1. Diagram of the general PF workflow in CMSSW

for GPU algorithms in CMS, the hadronic PF clustering was ported to the Alpaka portability library.

Alpaka is a C++ portability library for the development of parallel algorithms that provides a common interface for targeting different CPU and GPU hardware and software backends. The hierarchal parallelism structure is similar to CUDA providing developers with an easier transition from CUDA to Alpaka based code. The goal for CMS is to establish a single maintainable source code that can be run on a variety of hardware architectures close to their native performance. To this end, several reconstruction algorithms have been implemented with Alpaka and have been used at HLT. Before the introduction of Alpaka, each algorithm required duplicate implementations, one written in C++ and one written in CUDA, and the CMSSW framework required explicit switching points for using the CUDA version only when an NVIDIA GPU was detected. This complication led to difficult development, as numerous files had to be tracked and changes propagated between the backend specific algorithms. Alpaka simplifies the development structure to a single set of source files which are translated upon compilation to the backends supported by the framework. Moreover, with Alpaka, in the case where no accelerator hardware is detected the framework defaults to serial CPU operation. Thus all Alpaka developments targeting GPU parallelism function serially as a backup.

2 The formation of hadronic PF clusters

The general PF clustering algorithm is the same across the hadronic and electromagnetic calorimeters (HCAL and ECAL), however only the PF clustering of HCAL rechits has been ported to Alpaka during the 2024 data taking period. The two processes targeted are the translation from HCAL rechits to PF rechits, and the subsequent formation of HCAL PF clusters. This is described in detail below and in [5].

Creation of the PF rechits:

HCAL local rechits are taken as inputs to create the PF rechit objects. An energy threshold cut is applied to the rechits, varying based on detector conditions across the calorimeter geometry. Each PF rechit stores the identification of its 8 nearest neighbors in a square grid which is a key feature needed in the cluster formation. The collection of PF rechits are then used as the primary input for the PF clustering algorithms.

Seeding of the PF Clusters:

PF clusters within an event are initially seeded from the local energy maxima rechits. Within a set of neighbors, the highest energy rechits are selected as seeds under the condition that they also pass an additional noise threshold. The thresholds are calorimeter dependent and kept up to date in the configuration by reading from an online conditions database.

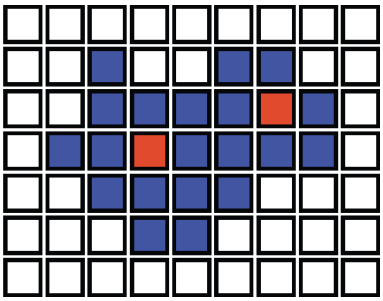


Figure 2. Diagram representing a topological cluster containing multiple seeds, shown in red. Each seed will become its own PF cluster.

Topological clustering of PF Rechits:

After the seeding algorithm, the rechits are processed into topological clusters within the same depth segmentation of the calorimeter. The topological clusters are formed via the neighbor information of rechits, and all rechits with common sides are associated with the same topological cluster as in Fig. 2. Multiple seeds are allowed per topological cluster and these will serve as the basis for the energy sharing step of PF clustering.

Energy sharing of PF Clusters:

The primary feature of the PF clustering implementation is the energy sharing algorithm, where individual rechits can share a fraction of their energy between multiple PF clusters. This is accomplished by a Gaussian mixture model, based on the distance of a rechit to the cluster seed. The PF rechits, j, then contribute a fraction of their energy, f_{ji} to a cluster, i, according the model:

$$f_{ji} = \frac{A_i e^{-\left(\vec{c}_j - \vec{\mu}_i\right)^2/(2\sigma)^2}}{\sum_{k=1}^{N} A_k e^{-\left(\vec{c}_j - \vec{\mu}_k\right)^2/(2\sigma)^2}}, \tag{1}$$

$$A_i = \sum_{j=1}^{M} f_{ji} E_j, \vec{\mu}_i = \sum_{j=1}^{M} f_{ji} E_j \vec{c}_j. \tag{2}$$

where M is the number of rechits in the topological cluster and N is the multiplicity of seeds. E_j and c_j are energy and position of the rechit, and A_i and μ_i are the energy and position of the cluster. σ is the assumed Gaussian width given as a configuration parameter to the algorithm, typically given a value of 10. Through iteration, the calculated fraction of energy for each rechit influences the final position of the cluster until the model converges.

3 Porting to Alpaka

A CUDA version of the clustering algorithm was previously developed, and the parallelism structure of Alpaka allows for mostly direct translation. However, to properly support parallel operations CMSSW has adopted the use of structure-of-arrays (SoA) data formats which are packaged into "portable collections". The portable collection implementation in the framework allows for minimal memory copies between the host (CPU) and device (GPU). The portable collection will remain on the GPU memory until a piece of code downstream calls for the collection on the host. A limitation of this framework is that the size of the portable

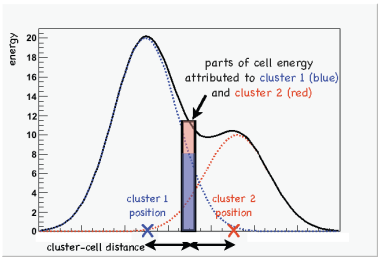


Figure 3. Example of energy sharing between two clusters.

collection must be declared on the host before being used in kernel running on GPU. In general the recommendation is to declare the largest possible size for the SoA even if in most processed events it will be relatively sparse. This can lead to a misuse of GPU memory in a case like those of the PF clusters, where the size of the rechit fraction array varies greatly. To better manage the memory usage, the clustering kernel was split in two with an intermediate, runtime copy of the exact SoA size needed passed back to the host. The performance impact in the memory transaction is minimal and the memory usage is much more conservative allowing additional Alpaka code to execute on the GPU without encountering memory bottlenecks.

Exploiting parallelism in the creation of PF rechits and PF clusters is mostly straightforward. In the case of the PF rechits the thresholds are applied asynchronously thanks to the simplicity of the operation. Each rechit is assigned to a GPU thread where all the associated information is calculated and stored before moving to the PF clustering steps. The seeding of PF clusters operates similarly, as threshold operations are computationally simple. Given the CUDA implementation, porting to Alpaka was straightforward.

Topological clustering of the PF rechits is aided by an efficient GPU connected-component labeling algorithm, ECL-CC [7]. This algorithm was initially developed for CUDA and thus is highly parallelized. The use in PF clustering is the first implementation of ECL-CC using Alpaka, and translated almost directly using Alpaka's functionality in CMSSW. The key features are the "intermediate pointer jumping" and "hooking" operations. ECL-CC uses graph data as input to label the vertices, v, and associated edges from the neighbors, (u, v). Intermediate pointer jumping in essence is a path-halving technique which exploits the GPU parallelism and reduces the number of times the algorithm must traverse any given path of connected components. Edge processing, also known as the hooking operation, includes functionality for high degree vertices using thread, warp, and block level granularity depending on the number of vertices. However, in the formation of the topological clusters the grid-like nature of the neighbors constitutes a vertex degree of 8 and only thread level granularity is needed. Occasionally when updating the edge information the algorithm can create data races between threads as the primary operation is synchronization free. These data races are benign due to an inherently atomic write to shared memory, and that each update is equally valid regardless of the thread. Saving on synchronization time increases the speed of ECL-CC and no issues arise from this.

The bulk of development focused on the energy sharing and PF cluster calculations. The core PF clustering algorithm remains the same, but is optimized where available. In Alpaka, a tiered computation is done based on the size of the cluster. Clusters with single seeds are processed quickly using an individual GPU thread per rechit to calculate the cluster position and energy. Depending on the size of larger clusters, GPU shared or global memory is used

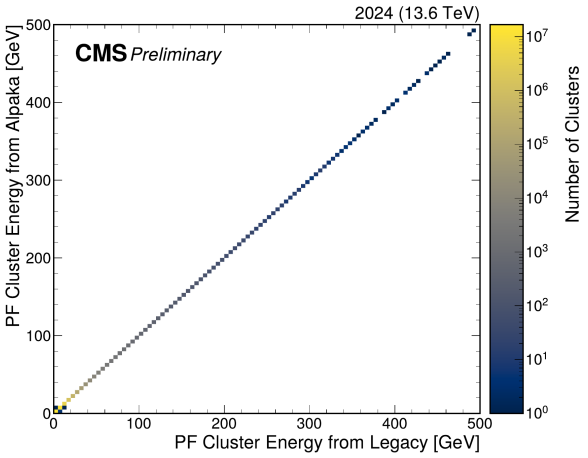


Figure 4. Comparisons of reconstructed HCAL PF cluster energies with Legacy CPU and Alpaka GPU clustering algorithms as measured in Run 3 2024 data

and the clusters are processed iteratively. In the case of exotically large clusters a slower global memory only kernel is available as a backup.

4 Performance Results

The port was validated for the Alpaka GPU and Alpaka CPU serial backends against the legacy clustering in both physics performance and timing at the HLT. Figure 4 shows the cluster energy comparison between the Alpaka GPU and legacy CPU outputs. We see good agreement where energy discrepancies greater than 1% occur in just 0.00001 % of HCAL PF clusters.

Figure 5 shows the performance of the port for a variety of configurations on an HLT computing node. The full HLT processing chain is run and by using Alpaka PF clustering on GPU we see a 2.5% speedup under the typical HLT configuration. Looking at just particle flow the CPU only timing at HLT averages 46.1 ms, or 7.1% of the overall HLT timing. By switching the the GPU algorithms, particle flow timing reduces to an average of 33.2 ms showing significant improvement.

5 Conclusion

The Alpaka port of hadronic PF clustering has proven to have a significant impact on the event processing time at HLT while retaining the same physics results as the original CPU only algorithm. The use of Alpaka specifically has lightened the maintenance burden as well due to the extensive hardware compatibility of the library. This hardware compatibility additionally opens up many options for balancing price and performance when upgrading the HLT computing farm, or offline computing sites as well.

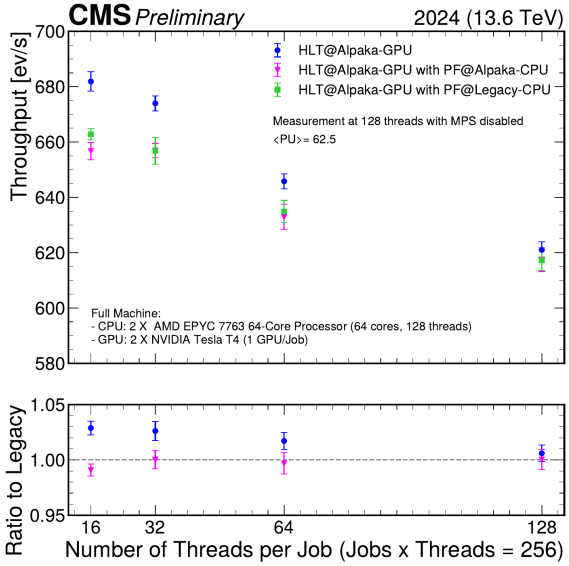


Figure 5. The event throughput of a CMS HLT configuration employed during the 2024 data-taking period. Each measurement runs the configuration on 40,000 events of proton-proton collision data from 2024 at an average pileup of 62.5. The blue points represent the event throughput achieved by executing the HLT with all the available heterogeneous modules on GPU. In contrast, the magenta ones depicts the event throughput when using the Alpaka-CPU version of PFRechit and PFCluster. The green points showcase the event throughput when utilizing the legacy version of PFRechit and PFCluster on CPU. Notably, the plot demonstrates a 2.5% speedup in HLT performance when utilizing 8 jobs with 32 threads each (standard data-taking HLT settings).

References

[1] CMS Collaboration, The CMS Experiment at the CERN LHC, JINST **3**, S08004 (2008). 10.1088/1748-0221/3/08/S08004

[2] CMS Collaboration, The CMS trigger system, JINST **12**, P01020 (2017). 10.1088/1748-0221/12/01/P01020

[3] CMS collaboration, Development of the cms detector for the cern lhc run 3, Journal of Instrumentation **19**, P05064 (2024). 10.1088/1748-0221/19/05/P05064

[4] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer, W.E. Nagel, M. Bussmann, Alpaka – An Abstraction Library for Parallel Kernel Acceleration, in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2016), https://github.com/alpaka-group/alpaka/

[5] CMS Collaboration, Particle-flow reconstruction and global event description with the CMS detector, JINST **12**, P10003 (2017). 10.1088/1748-0221/12/10/P10003

[6] CMS Collaboration, Heterogeneous Reconstruction of Hadronic Particle Flow Clusters with Alpaka Portability Library (2024), http://cds.cern.ch/record/2898660.

[7] J. Jaiganesh, M. Burtscher, A high-performance connected components implementation for gpus (2018). 10.1145/3208040.3208041