GPU acceleration of the ATLAS calorimeter clustering algorithm

Nuno Fernandes[1,2] on behalf of the ATLAS Collaboration

[1] LIP - Laboratório de Instrumentação e Física Experimental de Partículas,
Av. Prof. Gama Pinto, 2, Complexo Interdisciplinar (3is), 1649-003 Lisboa, Portugal
[2] Departamento de Física, Instituto Superior Tećnico, Universidade de Lisboa,
Av. Rovisco Pais, 1049-001 Lisboa, Portugal

E-mail: nuno.dos.santos.fernandes@cern.ch

Abstract. Given the upcoming High-Luminosity LHC Upgrade, the performance requirements for the trigger systems associated with the LHC experiments will increase due to the larger volume of data to be processed. One of the possibilities that the ATLAS Collaboration is evaluating for upgrading the software-based portion of its trigger system is the use of Graphical Processing Units as hardware accelerators. The present work focuses on the GPU acceleration of the Topological Clustering algorithm, which is used to reconstruct calorimeter showers by grouping cells according to their signal-to-noise ratio. A more GPU parallelizable version of the Topological Clustering, called Topo-Automaton Clustering, was implemented within AthenaMT, the software framework of the ATLAS trigger, and its results were compared to those of the standard CPU algorithm to ensure physical validity is maintained. Time measurements suggest an average improvement of the event processing time by a factor between 3.5 and 5.5 (depending on the kind of the event), though less than 20% of that time corresponds to the algorithm itself, suggesting that the main bottleneck lies in data transfers and conversions.

ATL-DAQ-PROC-2022-002
22 February 2022

1. Introduction

The Large Hadron Collider (LHC) is currently the largest particle accelerator in the world, and one of the most useful instruments for Particle Physics due to both the high energies and high luminosities that can be attained. To further increase its capabilities, it is undergoing the High-Luminosity Upgrade (HL-LHC), which should be completed by Run 4 (2028), increasing the instantaneous luminosity by a factor of $5-8$ (up to $\mathcal{L} \simeq 8 \times 10^{34}$ cm^{-2}s^{-1}) and the average number of collisions per bunch crossing by a factor of 8 (up to $\langle \mu \rangle \simeq 200$), with respect to the respective nominal values.

The ATLAS experiment [1] is one of the two general-purpose detectors at the LHC. This means that its trigger system, which is responsible for determining which events are committed to storage for later offline analysis, must take into account a variety of physical signals. Given the ensuing High-Luminosity LHC Upgrade, several improvements must be made to this system for it to be able to cope with the increased collision rate and the increased computational cost of processing each event.

The ATLAS trigger has two stages, with the first being based on custom electronics and the second being executed in software running on commodity hardware, integrated in the AthenaMT framework [2, 3], a multi-threaded version of Athena [4]. An interesting possibility for improving the computing power available at this second stage is the use of hardware accelerators, which can execute certain operations (hence, certain parts of event reconstruction) faster and/or more

efficiently than the Central Processing Units (CPU) which are more commonly employed in computational tasks [5]. One of the more widespread varieties of hardware accelerator in use today is the Graphics Processing Unit (GPU), which offers large-scale parallelism in terms of the number of threads available for processing, at the cost of requiring special attention to the memory access patterns and branching behaviour of the operation to be executed by each thread.

The present work is focused on the porting to GPUs of Topological Clustering [6], and in particular of a more parallelizable variation of it called Topo-Automaton Clustering [7], the algorithm which is used to reconstruct three dimensional calorimeter showers produced by the interactions between the particles that come out of the collision and the material in the detector.

2. The Algorithms

2.1. Topological Clustering

In Topological Clustering [6], the calorimeter cells are grouped according to the signal-to-noise ratio of the energy deposited during the event, taking into account the noise due to both the electronic read-out systems and the pile-up.

Cells are classified as *seed*, *growing*, *terminal* or *invalid* cells according to three thresholds, the *seed threshold*, T_{seed}, the *growing threshold*, T_{grow} and the *terminal threshold*, T_{terminal}, with $T_{\text{seed}} > T_{\text{grow}} \geq T_{\text{terminal}}$. The seed cells will be the origin of the clusters, which are the starting point of the algorithm. The clusters are evaluated in order of descending signal-to-noise ratio of their initial seed cell, by adding all the valid neighbouring cells that are not already part of another cluster and by merging the clusters that border each other through a seed or growing cell, which corresponds to adding all the cells to the cluster originated by the cell with the highest signal-to-noise ratio. This procedure is then applied to the growing cells that were added to the clusters at the previous step, in an iterative process, until there are no more seed or growing cells to consider. At this point, the algorithm has arrived at the final clusters, whose physical properties (energy, transverse energy, pseudo-rapidity, azimuthal angle) may be calculated from the cells that were assigned to them.

It must be pointed out that, according to this algorithm, the terminal cells are added to the first cluster that can reach them. Given the order of evaluation of the clusters, this means they will belong to the cluster which can reach them in fewer iterations and, within those that reach them at the same iteration, to the one originated by the seed cell with the highest signal-to-noise ratio.

2.2. Topo-Automaton Clustering

A direct implementation of the Topological Clustering algorithm as-is on GPUs would likely result in little to no performance gains. To ensure proper ordering of the operations, it requires maintaining two lists of cells to be evaluated, one at the current and other at the next iteration, which does not provide for a very GPU-friendly memory access pattern. It is also highly non-trivial to parallelize the cluster growing process in itself due to the assignment of terminal cells, which, as explained, explicitly relies on the order in which the seed and growing cells are considered to achieve correct results.

An alternative way to implement this cluster growing procedure with greater possibility of parallelization could be through the Topo-Automaton Clustering algorithm, which employs a cellular automaton to propagate *tags* through the cells to uniquely identify them as belonging to a given cluster. The tag of each cell at every step of the propagation only depends on its current tag and those of its neighbours, such that all the cells can be evaluated in parallel. Furthermore, the aforementioned lists of cells are no longer required, which allows for better memory access patterns on a GPU.

The most straightforward way of implementing the algorithm would be requiring that the ordering of the tags that correspond to each cluster reflects the ordering of the signal-to-noise ratio of the seed cell which originated them, with a special tag value, comparing lower than any cluster tag, being reserved to mark the cells that are not part of any cluster. With this, tag propagation consists in taking the maximum between the current tag of the cell and the tag which is being propagated to it. This operation can be done in parallel for all valid neighbours, in particular by

considering a list of all the pairs of cells eligible for tag propagation, which correspond to all the pairs of neighbouring valid cells with at least one seed or growing cell. This tag propagation is repeated, iteratively, until a stable state is reached, that is, until further iterations would result in no changes to the cell tags. That state contains the final clusters, whose properties can be then calculated, also on the GPU, from the cells that have the tag which uniquely corresponds to each cluster.

As mentioned, the assignment of the the terminal cells in the Topological Clustering depends explicitly on the order of evaluation. However, given the architecture of a GPU and the parallelism model it offers, the order in which the threads are executed is unspecified, which would lead to non-deterministic results if the same criterion was kept. Explicitly calculating the distance between the terminal cells and the clusters' seed cells, in order to replicate the effects of that criterion, would also be impractical, especially on a GPU, so an approximate solution was adopted instead: the terminal cells are assigned to the neighbouring cluster that has the highest signal-to-noise ratio of its initial seed cell (and, hence, the highest tag) among all clusters that border the terminal cell in question. This allows for deterministic terminal cell assignment easily implemented by taking the maximum between the terminal cell's tag and all the neighbouring cells' tags. In section 4, the impact of this change on the physical significance of the results will be discussed.

This alternative criterion has a secondary advantage. As the assignment of the terminal cells is only dependent on the final tag of the neighbouring cells, the iterative part of the algorithm can be reduced to considering only propagation between growing and seed cells, with the assignment of the terminal cells being done in a final step afterwards. This enables some performance gains by avoiding the branching of the code that would otherwise be necessary to prevent tags from being propagated from the terminal cells to their neighbours.

Though previous prototypes included an explicit sorting step to ensure the correct ordering of the tags, as discussed in [8], the signal-to-noise ratio in itself can be leveraged to provide an ordering with the desired properties while allowing the sorting to be skipped. In particular, the implementation uses 64-bit tags that have in their most significant 32 bits the signal-to-noise ratio of the corresponding seed cell and in the 32 least significant bits an index which corresponds uniquely to the cluster, which facilitates the calculation of the final cluster properties.

A final word must be said regarding the merging of the clusters. While the rules laid out for tag propagation ensure that, after a sufficient number of iterations, the largest tag is propagated to all the cells of the merged clusters, the inclusion of an explicit merging step where all the cells of the merged clusters change their tag has been measured to be more performant [8].

The data representation must be carefully considered to better leverage the architecture of a GPU. In particular, the substitution of the usual arrays-of-structures, where each object is kept as an entry in an array, for structures-of-arrays, where each variable of the object corresponds to a separate array, enables a better utilization of the several levels of caching available within the GPU. This is applicable, above all, to the description of the cluster geometry, namely in terms of the coordinates of the cells, and to the final cluster properties. Besides these, on GPU memory, one must keep the array of cell tags and the list of valid pairs of cells for tag propagation, as well as an auxiliary table for the explicit merge step, which holds the tag that the cells of every cluster must have after the merges took place.

3. Structure of the Implementation

There have been some previous prototype demonstrations [7, 9] of the possibility of employing GPU acceleration to the Topo-Automaton Clustering algorithm. They employed a client-server architecture that isolated the implementation of the algorithm from interfacing with the GPU. While a speed-up by a factor between 1.3 and 2 was achieved, about 17% of the time was spent in inter-process communication due to the architecture. As such, for the current implementation, a simple, linear structure was adopted, with the GPU being interfaced directly from within the algorithm.

To minimize data conversions and transfers to and from the GPU, an additional approximation is

made in regards to the CPU implementation: the noise associated with the energy measurements is assumed to remain constant throughout the run. The evaluation of the effects of this approximation on the resulting clusters will be made in section 4. The description of the geometry of the calorimeter, including the neighbourhood relations between the cells and their positions, also needs to be setup on GPU memory before event processing can commence, typically at the start of the run.

The processing of each event can be broken down into three stages:

(i) **Preparing the Data**: read the energy measurements at each cell, write them in the GPU-friendly struct-of-array representation and send it to the GPU.

(ii) **Running the Algorithm**: on the GPU,
   (a) Calculate the signal-to-noise ratio of each cell and initialize the tags to the appropriate value.
   (b) Create the list of eligible cell pairs for tag propagation.
   (c) Grow the clusters by propagating the tags following the algorithm outlined in 2.2.
   (d) Calculate the final cluster properties by accumulating the cell energies and positions weighted by the absolute. energy

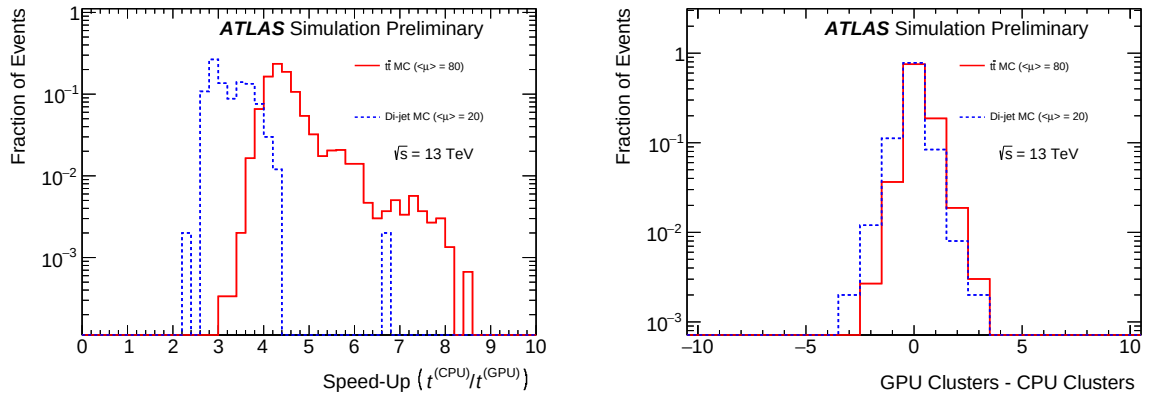
(iii) **Packaging the Results**: receive the array of cell tags and the final cluster properties from the GPU and construct the appropriate AthenaMT data structures to represent the final clusters in a way that is compatible with the rest of the software.

4. Preliminary Results

The results that follow were obtained on a machine running CentOS 7.9.2009 equipped with an AMD EPYC 7552 CPU and a Tesla V100S GPU, using `CUDA` 10.2, GCC 8.3 and version 22.0.24 of AthenaMT. Two different Monte Carlo simulated samples were used: $t\bar{t}$ **events**, with $\mu = 80$ collisions per bunch crossing, corresponding to events where a top quark-antiquark pair has been formed and there is at least one lepton in the final state, which tend to be quite dense in terms of the number and size of the clusters, and **di-jet events**, with $\mu = 20$ collisions per bunch crossing, corresponding to events where a quark-antiquark or a gluon-antigluon pair has originated two jets of particles. In the plots, $t\bar{t}$ events will be represented as solid or dotted red lines, while di-jet events will be represented by dashed and dash-dotted blue lines. Dotted and dash-dotted lines are used to present results for the standard CPU implementation of Topological Clustering, with the solid and dashed lines corresponding to the Topo-Automaton Clustering.

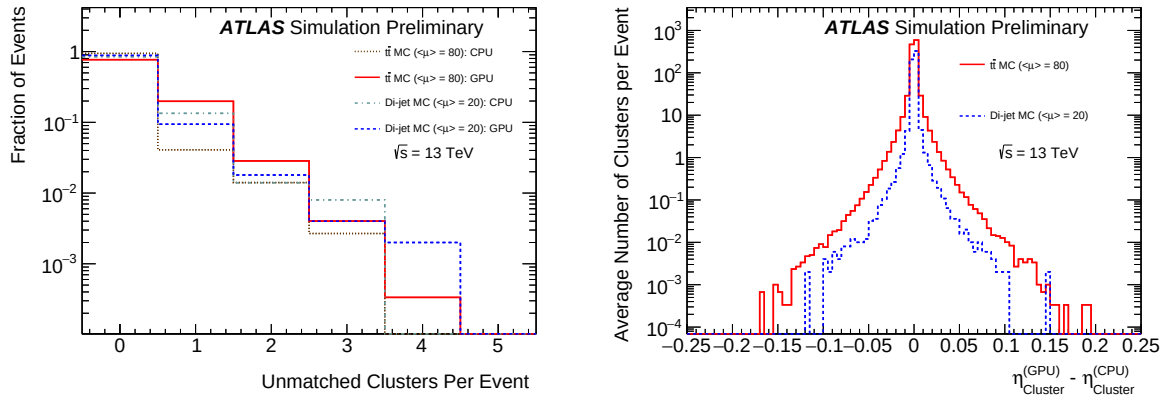
The most significant results from the present work are the measurements for the speed-up factor gained from GPU acceleration, measured as the ratio between the total time necessary to process a single event in the standard CPU implementation of Topological Clustering and the total time necessary to process that same event in the GPU-accelerated implementation of Topo-Automaton Clustering, as shown in Figure 1a. The average speed-up factor is 3.5 for di-jet events and 5.5 for $t\bar{t}$ events. However, less than 20% of the time is actually spent on the algorithm, with the remaining 80% being taken by the data transfers and especially the data conversions.

Figure 1b shows the differences between the total number of clusters per event that were identified by each implementation. To perform an accurate comparison between the implementations, a one-to-one correspondence between the clusters identified by each of them must be found, a process which will be designated matching henceforth. Matching is done through a variant of the Gale-Shapley algorithm [11], ensuring that the clusters that have the most cells in common between both implementations are paired with each other. Figure 2a presents the number of clusters that cannot be matched according to this procedure. For most of the events, all clusters are matched, with a few events having between 1 and 4 unmatched clusters, a small difference when compared to the average number of clusters per event (about 1200 for $t\bar{t}$ events and 550 for di-jet events). These differences were found to be a result of the constant noise approximation used in the GPU implementation.



(a) (b)

Figure 1: Distributions of the speed-up factor obtained from GPU acceleration (left) and of the differences between the number of clusters identified by the GPU and the CPU implementation (right), for $t\bar{t}$ (solid red line) and di-jet events (dashed blue line). Plots available at [10].



(a) (b)

Figure 2: Left: number of clusters that are identified by the GPU (solid and dotted lines) or CPU (dashed and dash-dotted lines) implementations in each event and that have not been successfully matched to a cluster in the other implementation, for $t\bar{t}$ (red lines) and di-jet events (blue lines). Right: differences between the pseudo-rapidity of matched clusters as reconstructed by the GPU and as reconstructed by the CPU implementation, for $t\bar{t}$ (solid red line) and di-jet events (dashed blue line). Plots available at [10].

Analysing the differences between the clusters that do match between both implementations, the first conclusion is that more than 99.9% of the clusters have exactly the same growing and seed cells, but some differences remain in the terminal cells due to the different criteria used for their assignment. Figure 2b shows the differences in the reconstructed pseudo-rapidity, η, of the final clusters, according to each implementation. Performing the same comparison for other cluster properties, such as the energy or the transverse energy, would lead to the same conclusion: despite the slight differences in the terminal cell assignment, the final cluster properties are very similar between both implementations. In particular, defining the distance between the matched clusters in the η-ϕ plane as $\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$, more than 99.9% of the clusters are within $\Delta R < 0.07$.

5. Conclusions

The present work aimed to explore the possibility of employing GPU acceleration within the ATLAS trigger, anticipating the stricter performance requirements posed by the High-Luminosity LHC Upgrade.

The feasibility of employing GPU acceleration for the Topological Clustering algorithm used within the calorimeter reconstruction has been succesfully demonstrated, with a promising speed-up having been measured in comparison to the standard CPU implementation, on average by a factor of $\sim$3.5 for di-jet events and $\sim$5.5 for $t\bar{t}$ events.

The validation studies of the implementation show that, despite there being some discrepancies that arise mainly from the differences in the terminal cell assignment between the standard CPU implementation and the GPU-accelerated version, the cluster properties are reconstructed sufficiently similarly in both implementations for the purposes of the trigger, and could be further improved by future algorithmic changes.

Another important conclusion is the fact that, in the present implementation, only 20% of the total event processing time is spent on the execution of the algorithm in itself. This means that future development efforts focused on reducing the overheads associated with data transfers and conversions would probably have the most significant impact in the efficiency of the implementation. This can be achieved, for example, by including other steps of the calorimeter reconstruction also on GPUs, allowing the reuse of data already stored in GPU memory, or by considering some possible changes to the data structures currently used within AthenaMT so that they would be more GPU-friendly, thereby reducing the overheads associated with data conversion.

Acknowledgments

This work was funded through Portuguese national funds by FCT – Fundação para a Ciência e Tecnologia, I. P., under the CERN/FIS-PAR/0002/2019 project. The generous disponibilization by INCD of a remote server environment for testing must also be acknowledged.

References

[1] The ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider *J. Instrum.* **3** S08003

[2] The ATLAS Collaboration 2021 Performance of Multi-threaded Reconstruction in ATLAS Tech. rep. CERN Geneva URL: `https://cds.cern.ch/record/2771777`

[3] Legget C et al, on behalf of the ATLAS Collaboration 2017 AthenaMT: upgrading the ATLAS software framework for the many-core world with multi-threading *J. Phys. Conf. Ser.* **898** 042009

[4] The ATLAS Collaboration 2021 The ATLAS Collaboration Software and Firmware Tech. rep. CERN Geneva URL: `https://cds.cern.ch/record/2767187`

[5] The ATLAS Collaboration 2017 Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System URL: `https://cds.cern.ch/record/2285584`

[6] The ATLAS Collaboration 2017 Topological Cell Clustering in the ATLAS Calorimeters and its Performance in LHC Run 1 *Eur. Phys. J. C* **77**

[7] Conde-Muíño P, on behalf of the ATLAS Collaboration 2017 Multi-Threaded Algorithms for GPGPU in the ATLAS High Level Trigger *J. Phys. Conf. Ser.* **898** 032003

[8] Fernandes N 2021 *Development of GPU-Accelerated Trigger Algorithms for the ATLAS Experiment at the LHC* Master's thesis defended 16 Nov 2021 URL: `https://cds.cern.ch/record/2790668`

[9] Delgado A and Emeliyanov D, on behalf of the ATLAS Collaboration 2016 ATLAS Trigger Algorithms for General Purpose Graphics Processor Units *2016 IEEE Nuclear Science Symp., Medical Imaging Conf. and Room-Temperature Semiconductor Detector Workshop (NSS/MIC/RTSD)* pp 1–6

[10] The ATLAS Collaboration HLT Calorimeter Public Results URL: `https://twiki.cern.ch/twiki/bin/view/AtlasPublic/HLTCaloPublicResults`

[11] Gale D and Shapley L S 1962 College Admissions and the Stability of Marriage *Am. Math. Mon.* **69** 9–15 ISSN 00029890, 19300972