Accelerating the DBSCAN clustering algorithm for low-latency primary vertex reconstruction at the HL-LHC

Lucas Borgna[1], Marco Barbone[2], Jiayang Cao[2], Andrew Rose[1], Alexander Tapper[1], Robert Bainbridge[1], Wayne Luk[2]

[1] Department of Physics, Imperial College London, UK.
[2] Department of Computing, Imperial College London, UK.

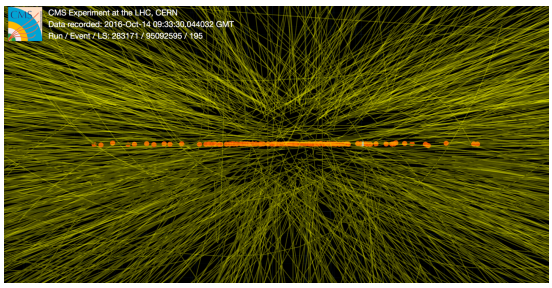
E-mail: l.borgna21@imperial.ac.uk, a.tapper@imperial.ac.uk

Abstract.

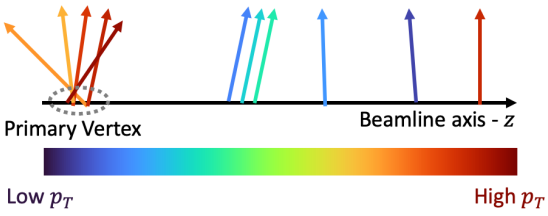
In this work we present the adaptation of the popular clustering algorithm DBSCAN to reconstruct the primary vertex (PV) at the hardware trigger level in collisions at the High-Luminosity LHC. Nominally, PV reconstruction is performed by a simple histogram-based algorithm. The main challenge in PV reconstruction is that the particle tracks need to be processed in a low-latency environment. To achieve this an accelerated version of the DBSCAN algorithm was developed to run in a Field Programmable Gate Array (FPGA). A CPU-optimized version of DBSCAN was implemented in C++ to serve as a benchmark for comparison. The CPU version of DBSCAN resulted in an average PV reconstruction latency of 93 μs , while the FPGA firmware only had a latency of 750 ns resulting in a 127x speedup. The speedup is a result of running all the input tracks in parallel, which ultimately results in high resource consumption, of up to 48.6 % of the available logic. Most of the logic was attributed to the use of sorting networks that allows for the parallel processing of the input tracks. To optimize the firmware for specific latency and resource usage constraints, the firmware can be parameterized by the number of input tracks to consider at a time.

1. Introduction

To continue the search for new physics the future High Luminosity Large Hadron Collider (HL-LHC) will ultimately reach an instantaneous luminosity of $7.5\times\ 10^{34}\text{cm}^{-2}\text{s}^{-1}$. This however also increases the average number of interactions per crossing (PU) to 200 on average [1]. These changes will bring about two main challenges for any data reconstruction algorithm to overcome. The first challenge will be to be resilient to the increased background noise and the second will be to process the larger quantity of space-points while still meeting the latency requirements. Clustering algorithms are prominent in particle physics as they can create higher level objects for physics analysis. One of the most popular clustering algorithms is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN), due to its noise resilience and clustering performance [2]. For these reasons the DBSCAN algorithm is well suited for clustering applications in the HL-LHC environment. Field Programmable Gate Arrays (FPGAs) can be used to accelerate certain algorithms and with their associated High-Level Synthesis (HLS) tools, and using a similar methodology to the one adopted for accelerating Monte Carlo simulation [3],

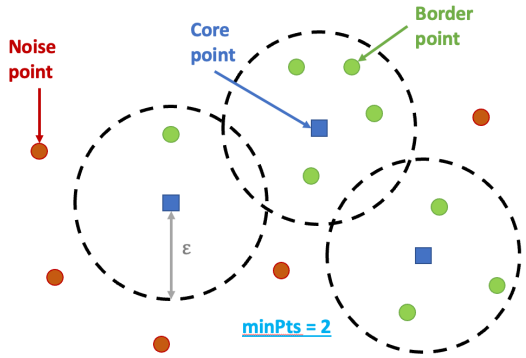


(a)

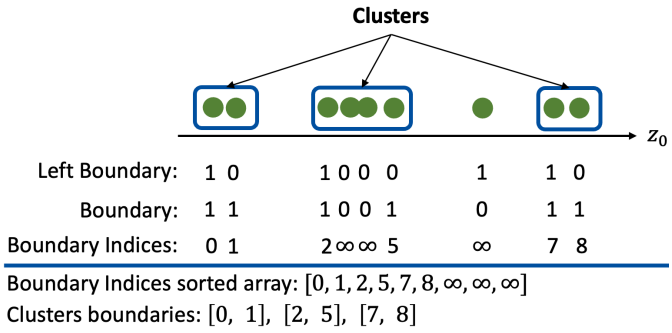


(b)

Figure 1: a) High pileup event in CMS with 130 interaction vertices [4]. b) Example of how clustering along the beamline axis can be used to determine the primary vertex.



(a)



(b)

Figure 2: a) Main components of the DBSCAN clustering algorithm. b) Illustration of the key steps in the accelerated DBSCAN algorithm.

the development time can be reduced significantly. As a test-bench the reconstruction of primary vertices is used as a 1D example.

2. Primary Vertexing at the HL-LHC

The objective of the reconstruction of primary vertices is to locate the position of the hard scattering event and find all the associated tracks that came from it. Normally tracks are complicated objects, but in this case is possible to choose a minimal representation of a track which comprises only of their z_0 position and their transverse momentum (p_T) value. The z_0 value of a track is where it intersects the beamline (z-axis) of the detector. In the HL-LHC, the CMS outer tracker will be able to provide hits above 2 GeV to the Level-1 trigger (L1). The L1 trigger is responsible for the first stage of data reduction, which is a custom hardware design based FPGAs and has a strict maximum latency of 12.5 μs to make a decision. To make this decision the L1 system will have to reconstruct tracks from the hits provided by the outer tracker, with up to 5 μs of the total latency are reserved for this reconstruction [1]. These tracks will then be processed by the primary vertexing algorithm to support the L1 trigger decision, which is challenging due to the strict time requirements. Figure 1a shows a recorded event with high pileup during 2016, which illustrates the complexity of primary vertex reconstruction is in this noisy environment.

As a baseline, the primary vertices can be reconstructed by using a histogram based approach. To do this a histogram of the tracks' z_0 coordinate is created, with each track weighted by its

p_T. A 3 bin convolution is then used on the histogram. The primary vertex location is then determined by the bin with the highest peak. This baseline approach is called "FastHisto" and it naturally provides a good proxy to determine the hard scattering event [1]. Furthermore, it has a computational complexity of $\mathcal{O}(n_{bins} + n_{tracks})$ and can be executed in an FPGA with minimal resources and latency. The downside of this method is that it is not resilient to noise, where high p_T noise tracks can overlay the tracks from the primary vertex. The DBSCAN algorithm can be used to cluster the tracks along the z_0 dimension. Then within each cluster, the p_T of each track is summed and the cluster with the highest p_T is labelled as the primary vertex. The median z_0 of all the tracks in the primary vertex cluster is then used as the z_0 of the primary vertex [5].

3. Accelerating DBSCAN for primary vertexing

The behaviour of DBSCAN algorithm changes based on the hyperparameters used. Different hyperparameters lead to different optimisations. Hence, the key in accelerating the DBSCAN algorithm is to first focus on its two main hyperparameters. The first is called ε, which represents the maximum distance a neighbouring point can be to be considered part of the same cluster. The second is the *minPts*, which is the minimum number of points needed for it to be considered as a cluster. These parameters are illustrated in Figure 2a. In the context of HL-LHC primary vertex reconstruction, the optimal set of parameters is $minPts = 2$ and $\varepsilon = 0.15$ [cm]. Fortunately, because of the value of $minPts = 2$, it is possible to ignore inner loops within the algorithm. The second key to accelerate DBSCAN is to first sort the tracks along the z_0 dimension as it allows for tracks to be processed in parallel, which will be exploited by the FPGA. A bitonic sort method is used to reduce the latency of the sorting [6]. Moreover, a prefix sum method is used to pre-compute in parallel the p_T sums of the clusters [7]. A brief summary of the computations needed is shown in Algorithm 1, which has been designed to have all of the *for-loops* execute in *parallel* in the FPGA hardware. The algorithm only requires two inputs the tracks and the ε parameter. In Line 1, the tracks are sorted along the z_0 axis. The first loop in Line 2 is used to determine if the the tracks represents the left boundary of a cluster. A track is considered to be a left boundary if it is within a distance of ε to the next track along the z_0. The second loop in Line 4 is used to determine the right boundaries of each cluster. Once the left and right boundaries are identified a vector of the indices of boundaries is created in the loop in Line 8 and where there is not a boundary the value is filled with ∞. This is done so that the sorting by the index in Line 13 pairs the left and right boundaries together. This then maps the indices of tracks to where the clusters begin and end. The loop in Line 14 uses those indices to compute the median z_0 and p_T sum for each vertex. Using the median statistic here is critical as the clusters can have skewed z_0 distributions. Lastly, the vertices are sorted by p_T in Line 16 so that the vertex with the highest p_T value can be identified as the primary vertex. An illustration of how the boundaries are used to identify the clusters is shown in Figure 2b.

Algorithm 1: minPts=2 accelerated DBSCAN

input : tracks, ε
output: vertices

Sort(tracks) ▷ Sorted by z_0
for $i \leftarrow 0$ ***to*** *tracks.size* **do**
  $isLeftBoundaries[i] \leftarrow tracks[i].z_0 - tracks[i-1].z_0 \leq \varepsilon$
for $i \leftarrow 0$ ***to*** *tracks.size* **do**
  $leftEdge \leftarrow isLeftBoundaries[i]$ **and not** $isLeftBoundaries[i+1]$
  $rightEdge \leftarrow$ **not** $isLeftBoundaries[i]$ **and** $isLeftBoundaries[i+1]$
  $isBoundaries[i] \leftarrow leftEdge$ **or** $rightEdge$
for $i \leftarrow 0$ ***to*** *tracks.size* **do**
  if $isBoundaries[i]$ **then**
    $boundaryIndices[i] \leftarrow i$
  else
    $boundaryIndices[i] \leftarrow \infty$
Sort($boundaryIndices$) ▷ Sorted by i
for $i \leftarrow 0$ ***to*** *tracks.size* ***by*** 2 **do**
  $vertices[i] \leftarrow$ CalculateVertex($boundaries[i]$, $boundaries[i+1]$) ▷ median z_0, Σp_T
Sort($vertices$) ▷ Sorted by p_T

4. Testbench for Accelerated DBSCAN

To obtain a benchmark of how the accelerated DBSCAN algorithm performs a CPU version using C++ was developed. The FPGA implementation was done using a Xilinx VU9P FPGA within the Maxeler Dataflow Engine (DFE) setup. These results were obtained using a clock frequency of 100 MHz. In the DFE the tracks are fed from the CPU to the FPGA via a PCI-e bus. In this case the latency of the PCI-e transfer is not a concern since in the L1 environment the tracks would be fed directly to the FPGA via optical fibre connections. Unfortunately, due to the FPGA resource constraints a firmware with 1665 tracks cannot be synthesised. To understand the performance of the firmware a maximum number of tracks of 232 was used. Table 1 shows the resource usage of the accelerated DBSCAN firmware with 232 input tracks. The firmware results in 48.6% of the available logic being used up. As shown in Table 1 the main consumer of these resources are the sorting networks. A single sorting network, responsible of sorting the 232 tracks along z_0 uses up 11.2 % of the available logic. Another sorting network is used to sort the *boundaryIndices*, which also has a size of 232. Lastly, to identify the primary vertex, a final sorting network is used to sort the vertices in decreasing p_T order, which has a reduced size of $232/minPts = 116$. This means that roughly 58 % of the used logic is going towards the three sorting networks. The latency to process a single event is shown in Table 2. The latency of the CPU optimized version was 93 ms, while the FPGA version achieved a latency of 0.73 ms, representing a speedup of 127×. This indicates that the FPGA acceleration is necessary to reach the latency constraints of the L1 trigger system.

5. Conclusions and future work

The results indicate that the DBSCAN algorithm can be accelerated by the FPGA hardware. In the case of the primary vertex reconstruction with up to 232 input tracks a speedup of 127× was observed. The speedup is attributed to the high degree of parallelization that is enabled by pre-processing the input tracks with a sorting network. The downside of using the sorting operation is that is necessary to utilize a large area of the FPGA resources, preventing the algorithm to be synthesized for 1665 input tracks.

Resource	Use	Total	Usage [%]	Sort only [%]
Logic Utilization	1725099	3546720	48.6	11.2
LUTs	422811	1182240	35.8	7.3
FFs	1302288	2364480	55.1	13.2
DSP	0	6840	0	0
BRAM18	611	4320	14.1	8.3
URAM	118	960	12.3	8.1

Table 1: FPGA resources usage of the accelerated DBSCAN firmware using 232 input tracks. The sort only column shows how much of the resources are used up by sorting the tracks along the z_0 axis. The chip used here was the Xilinx VU9P.

Hardware	CPU	FPGA
Execution time [μs]	93	0.73

Table 2: Average execution time for both the CPU optimized and FPGA optimized accelerated DBSCAN to process events with up to 232 tracks. The FPGA optimized version represents a 127× speedup over the CPU optimized version. The FPGA was running with a conservative 100 MHz clock.

In the future the DBSCAN firmware can be modified to process tracks in batches, which would avoid having large sorting networks. By doing this the resource usage can be minimized at the cost of increased latency, which would allow the firmware to be tuned for each application. This modification would come with additional overheads that would needed to be treated with care. The first is that processing tracks in batches would require an overlap checking procedure to merge clusters. This process is quite costly as it has a complexity of $\mathcal{O}(N_{tracks}(N_{tracks}-1)/2)$. Moreover, within each batch noise points cannot be ignored and will only be excluded in the merging procedure, which will require additional resources to account for. Lastly, because of the batching procedure, there will not be a fully sorted vector of tracks for the median z_0 calculation. This however can potentially be overcome by switching to a weighted mean statistic.

References

[1] Tumasyan A *et al.* (CMS) 2017 The Phase-2 Upgrade of the CMS Tracker

[2] Ester M, Kriegel H P, Sander J and Xu X 1996 A density-based algorithm for discovering clusters in large spatial databases with noise KDD'96 (AAAI Press) p 226–231

[3] Barbone M, Howard A, Tapper A, Chen D, Novak M and Luk W 2023 *Journal of Physics: Conference Series* **2438** 012023 URL `https://dx.doi.org/10.1088/1742-6596/2438/1/012023`

[4] Collaboration C and Mc Cauley T 2016 Collisions recorded by the CMS detector on 14 Oct 2016 during the high pile-up fill URL `https://cds.cern.ch/record/2231915`

[5] Cieri D 2018 Development of a Level-1 Track and Vertex Finder for the Phase II CMS experiment upgrade presented 26 Feb 2018 URL `http://cds.cern.ch/record/2317060`

[6] Peters H, Schulz-Hildebrandt O and Luttenberger N 2010 Fast in-place sorting with cuda based on bitonic sort *Parallel Processing and Applied Mathematics* ed Wyrzykowski R, Dongarra J, Karczewski K and Wasniewski J (Berlin, Heidelberg: Springer Berlin Heidelberg) pp 403–410 ISBN 978-3-642-14390-8

[7] Blelloch G E 2004 URL `https://kilthub.cmu.edu/articles/journal_contribution/Prefix_sums_and_their_applications/6608579`