DBSCAN Software

Firmware plans

Lucas Santiago Borgna
DBSCAN Recall

- Last time I compared DBSCAN with Fast Histo
  - Results shown [here](#)
  - This was done with python/ sklearn

- DBSCAN efficiency = 78.16 %

- Fast Histo efficiency = 74.95 %

- Time to move into firmware space!
DBSCAN firmware + software

- The starting point for this is the code from the MSc project
  - https://github.com/DiamonDinoia/AcceleratedDBSCAN

- I’ve validated the ”Software” implementation to see if it does something sensible.
  - Compared it to what I would get with sklearn/mlpack in python using the same truth data.
  - The algorithm pseudocode is described in the appendix
DBSCAN firmware results (from report)

• Original implementation of the firmware
• DBSCAN kernel performance = 0.726 μs
  – 100 MHz clock, VU9P

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Available</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>423605</td>
<td>1182240</td>
<td>35.83</td>
</tr>
<tr>
<td>FFs</td>
<td>1298871</td>
<td>23644780</td>
<td>54.93</td>
</tr>
<tr>
<td>BRAMs</td>
<td>364</td>
<td>2160</td>
<td>16.85</td>
</tr>
<tr>
<td>DSPs</td>
<td>0</td>
<td>6840</td>
<td>0</td>
</tr>
</tbody>
</table>
DBSCAN firmware + software, main issues

• There is one key issue with the current implementation of the software
  • The software/firmware is limited to a maximum of 232 input tracks
    – Theoretical maximum of 1665
    – Runs on all tracks in parallel, which is good for speed but poor for resources.
    – The software/firmware is not parametrized to control the level of parallelism.

• Additionally, the software is also only works if the minimum number of points in the cluster is 2, changing this would require a whole re-write.
  – We’ve seen that this is the optimal choice for DBSCAN though.
DBSCAN firmware + software, plan

- To solve this key issue, I plan to parametrize the software/algorithm by the number of input tracks to consider.
  - This will help to tune the algorithm for a desired level of latency and FPGA resources.
  - Makes it more widely applicable and comparable to other solutions.
DBSCAN primer

Clusters boundaries = [0, 1], [2, 5], [7, 8]

Boundary Indices sorted array = [0, 1, 2, 5, 7, 8, ∞, ∞, ∞]
DBSCAN parametrized

• Due to the sorting of $z_0$, the process can be parametrized so that the input tracks are analyzed in batches by only needing to consider the track before and after the batch.

• This is what I’m currently working on at the software level (C++).

• Almost done, just fixing some edge cases.
SwiftHEP plans, 23/24 of March @ Durham

- Introduce the primary vertexing problem
- Introduce the DBSCAN
- Show the comparison between DBSCAN and FastHisto
- Explain the accelerated DBSCAN implementation
- Show FPGA latency vs resources results
- Show FPGA physics performance benchmarks
Summary

- Working on a parametrizable version of the accelerated DBSCAN
  - Close, need to work out a few kinks.

- Looking to determine the latency vs resource curve for FPGA implementation

- Will plan to present these results in SwiftHEP@Durham
Appendix
\texttt{Algorithm 2: minPts = 2 DBSCAN algorithm pseudocode}

def DBSCANTwoMinPts\( (P, \epsilon) \):
    /* sort according to the \( z_0 \) positions of the tracks */
    Sort\( (P) \)
    for \( i \leftarrow 0 \) to \( P.\text{size} \) do
        isLeftBoundaries\( [i] \leftarrow P[i].z0 - P[i-1].z0 \leq \epsilon \)
    end
    /* a point is a boundary point if it is a left boundary (not noise) or right boundary */
    for \( i \leftarrow 0 \) to \( P.\text{size} \) do
        isBoundaries\( [i] \leftarrow (\text{isLeftBoundaries}[i] \text{ and not isLeftBoundaries}[i+1]) \text{ or (isLeftBoundaries}[i+1] \text{ and not isLeftBoundaries}[i]) \)
    end
    /* find the indices of boundaries */
    for \( i \leftarrow 0 \) to \( P.\text{size} \) do
        if isBoundaries\( [i] \) then
            boundaryIndices\( [i] \leftarrow i \)
        else
            boundaryIndices\( [i] \leftarrow \infty \)
        end
        Sort\( (\text{boundaryIndices}) \)
    end
    /* calculate the vertices and the sum of the \( p_T \) of the tracks associated with the vertices */
    for \( i \leftarrow 0 \) to \( P.\text{size} \) by \( 2 \) do
        vertices\( [i] \leftarrow \text{CalculateVertex}(\text{boundaries}[i], \text{boundaries}[i+1]) \)
    end
    /* sort according to the sum of the \( p_T \) */
    Sort\( (\text{vertices}) \)
    return vertices