Accelerating DBSCAN Algorithm with AI Chips for Large Datasets

Zhuoran Ji and Cho-Li Wang
Department of Computer Science
The University of Hong Kong
August 11, 2021
DBSCAN

- DBSCAN is a powerful clustering areas
  - E.g., text clustering, astronomy, geography, image processing

- Advantage:
  - No need to specify the # of clusters to be found
  - Can detect cluster with any shape

- But: high computational complexity
  - distance between every two points
AI Chips

- Driven by DL, many AI chips are proposed
  - Nvidia GPU, Huawei Ascend, Alibaba Hanguang, Google TPU
- Their marquee feature is TCUs (Tensor Core Units): dedicated matrix multiplication hardware units
- Provide very efficient MM algorithm
  - RTX3090: 71 vs 35.6 TFLOPS
- However, other components, such as vector units and scalar units are usually
  - less powerful
  - highly specialized for AI apps
Motivation

• Distance metric for DBSCAN:
  • cosine similarity, dot product, Euclidean distance

• Cosine similarity:

\[
d_{i,j} = \frac{P_i \cdot P_j}{\|P_i\| \|P_j\|} = \frac{\sum_{k=1}^{dim} P_i[k]P_j[k]}{\sqrt{\sum_{k=1}^{dim} P_i[k]^2} \sqrt{\sum_{k=1}^{dim} P_j[k]^2}}
\]

\( l2 \) normalized

\[
d_{i,j} = \frac{\sum_{k=1}^{dim} P_i[k]P_j[k]}{1 \times 1} = P_i \cdot P_j
\]

• Distance matrix:

\[
\begin{bmatrix}
  d_{[F_0,0]} & \cdots & d_{[F_0,n]} \\
  \vdots & \ddots & \vdots \\
  d_{[F_m,0]} & \cdots & d_{[F_m,n]}
\end{bmatrix} =
\begin{bmatrix}
P_{F_0} \\
\vdots \\
P_{F_m}
\end{bmatrix} \times
\begin{bmatrix}
P_0^T \\
\vdots \\
P_n^T
\end{bmatrix}
\]

• Natural to use TCUs to accelerate distance calculation
Challenge: Identify $\epsilon$-neighbors on AI chips

- However, identify and count $\epsilon$-neighbors on AI chips is challenging
  - involve compare-and-select (CMPSEL) operation
  - weakly supported by many AI chips

- Some AI chips do have dedicated ReLU units
  - $y = x > 0 \ ? \ x : 1$
  - BUT, not allow an arbitrary value for the positive input

procedure IDENTIFY_NEIGHBORS(dis_matrix)

```
adj[i][j] = dis_matrix[i][j] ≤ \epsilon \ ? \ 1 : 0
numNeighbors[i] = REDUCE(+, adj[i][0:j])
isCore[i] = numNeighbors[i] ≥ minPts ? true : false
isNeighbor[j] = REDUCE(||, adj[0:i][j], isCore[i])
return isNeighbor[0:j]
```
end procedure
Challenge: Device Memory is Limited

- GPUs and other AI chips’ device memory: 10x GBs
- The size of the datasets can be 100x GBs, exceeding the capacity
- Space complexity
  - Dataset: $O(DN)$
  - Adjacency matrix: $O(MN)$
- $N$ is the # of points in the dataset
Identify $\epsilon$-neighbors on AI chips

- The essence of $\epsilon$-neighbor identification is to find a mapping function so that
  - $\#$ of $\epsilon$-neighbors $\leq \text{REDUCE}(\text{op}_1, \text{MAP}(f, \text{distance}[i][0:j]))$
  - whether $j$ is an $\epsilon$-neighbors $\leq \text{REDUCE}(\text{op}_2, \text{MAP}(f, \text{distance}[0:i][j]))$

- We can then generalize the $\epsilon$-neighbors identification kernel as

```
procedure IDENTIFY_NEIGHBORS(dis_matrix)
  adj[i][j] = dis_matrix[i][j] \leq \epsilon \? 1 : 0
  numNeighbors[i] = REDUCE(+, adj[i][0:j])
  isCore[i] = numNeighbors[i] \geq \text{minPts} \? \text{true} : \text{false}
  isNeighbor[j] = REDUCE(||, adj[0:i][j], isCore[i])
return isNeighbor[0:j]
```

```plaintext
indicators[i][j] = f(distance[i][j], \epsilon)
internalNum[i] = \text{REDUCE}(\text{op}_1, indicators[i][0:j])
isCore[i] = \text{CHECK_CORE_POINT}(internalNum[i], \text{minPts})
isNeighbor[j] = \text{REDUCE}(\text{op}_2, indicators[0:i][j], isCore[i])
return isNeighbor[0:j]
```
The key of the mapping function is to
- differentiate the behaviors of $\varepsilon$-neighbors and non-$\varepsilon$-neighbors during reduction
- unify the behavior of the points within the same category
- For example, the unit step function maps
  - $\varepsilon$-neighbors to 1
  - non-$\varepsilon$-neighbors to 0
  - satisfies the two requirement
Mapping Function $f_{-2.0/\pm0.0}$

- $f_{-2.0/\pm0.0} = ((\text{distance} - \varepsilon) \& 0x8000 | 0x3c00) - 1.0$

- In the form of instructions:

**Algorithm 3 Mapping function $f_{-2.0/\pm0.0}$**

1. $v_1 = \text{distance} - \varepsilon$ \(\triangleright\) $\varepsilon$ - similarity, if measured as similarity
2. $v_2 = \text{reinterpret } v_1 \text{ as a 16-bits binary}$
3. $v_3 = v_2 \& 0x8000$ \(\triangleright\) $\varepsilon$-neighbors = $0x8000$, others = $0x0000$
4. $v_4 = v_3 \text{ | } 0x3c00$ \(\triangleright\) $\varepsilon$-neighbors = $0xbc00$, others = $0x3c00$
5. $v_5 = \text{reinterpret } v_4 \text{ as a float16}$
6. $\text{indicator} = v_5 - 1.0$

**IEEE Numbers**

- $\varepsilon$-neighbor: $\varepsilon - 2.0$
- non-$\varepsilon$-neighbor: $\varepsilon + 2.0$

1. Differentiate the behaviors of $\varepsilon$-neighbors and non-$\varepsilon$-neighbors during reduction
2. Unify the behavior of the points within the same category

**Special Cases**

- $+0.0$ and $-0.0$: Non-similar behaviors for addition and subtraction
- $0.0$: Special case for multiply and division

**Mapping Function $f_{-2.0/\pm0.0}$**

- $f_{-2.0/\pm0.0} = ((\text{distance} - \varepsilon) \& 0x8000 | 0x3c00) - 1.0$
- In the form of instructions:

```plaintext
Algorithm 3 Mapping function $f_{-2.0/\pm0.0}$

1: $v_1 = \text{distance} - \varepsilon$  \hspace{1em} $\triangleright \varepsilon$ - similarity, if measured as similarity
2: $v_2 = \text{reinterpret } v_1 \text{ as a 16-bits binary}$
3: $v_3 = v_2 \& 0x8000$  \hspace{1em} $\triangleright \varepsilon$-neighbors = 0x8000, others = 0x0000
4: $v_4 = v_3 \mid 0x3c00$  \hspace{1em} $\triangleright \varepsilon$-neighbors = 0xbc00, others = 0x3c00
5: $v_5 = \text{reinterpret } v_4 \text{ as a float16}$
6: $\text{indicator} = v_5 - 1.0$
```

$\varepsilon$-neighbor: $\pm\varepsilon - 2.0$

| $0xC000$ & $0x8000$ & $0xbc00$ & $-2.0$ |
| 0x$4000$ & 0x$0000$ & 0x$3c00$ & 0.0 |

non-$\varepsilon$-neighbor: $\varepsilon + 2.0$

| $0xC000$ & $0x8000$ & $0xbc00$ & $-2.0$ |
| 0x$4000$ & 0x$0000$ & 0x$3c00$ & 0.0 |
Mapping Function $f_{-128/0}$

- $f_{-128/0} = (\text{distance} - \varepsilon) \& 0x8000$
- Same with the first three steps of $f_{-2.0/\pm 0.0}$
- In $f_{-2.0/\pm 0.0}$
  - $+0.0$ and $-0.0$ have similar behaviors for addition and subtraction
  - $0.0$ is special for multiply and division
- How about we interpret it as int8

\[
\begin{align*}
\text{ε-neighbor: } & \varepsilon - 2.0 & \varepsilon - 2.0 & \text{reinterpret as 16-bit bin} & 0xC000 & 0x8000 (-0.0) & \text{reinterpret as 2 int8} & (-128, 0) \\
\text{non-ε-neighbor: } & \varepsilon + 2.0 & 2.0 & 0x4000 & 0x8000 (0.0) & 0x0000 (0.0) & (0, 0)
\end{align*}
\]
Mapping Function $f_{-128/0}$

- Easy to determine whether a point is a $\varepsilon$-neighbor
  - i.e., $op_2 = \|$
- However, cannot count $\varepsilon$-neighbors by integer addition
- The value "$-128" is the maximum negative value for int8
- Vector units usually need the inputs and the outputs to have the same data type
  - Overflow almost always occurs
- Two intuitive method
  - Convert the data type to int32
    - e.g., static_cast
    - Casting is costly, reducing int32 is expensive than reducing int16
  - Reduce the magnitude by vector operation
    - e.g., right shift, integer division, or exponential function
    - weakly supported or costly than $f_{-2.0/\pm0.0}$
• Mixed-precision MMA
  • One of the most novel features of TCUs
• Input matrices in low-precision data types
• Accumulates the result in high-precision data types
• E.g.,
  • Nvidia TCUs support FP16 → FP32 and INT4/8 → INT32
  • Huawei TCUs support FP16 → FP32 and INT8 → INT32
• Thus, we use TCUs to accumulate the mapped values
Mini Batch DBSCAN

- DBSCAN can process the data in a mini-batch manner
  - much like the deep neural network training
  - but high data transfer overhead

```plaintext
1: for batch_i in # of mini-batches do
2:   transfer dataset[batch_i] to device mem
3:   calculate the distance matrix disMat[batch_i]
4:   identify \( \varepsilon \)-neighbors adjMat[batch_i]
5:   accumulate the # of \( \varepsilon \)-neighbors of adjMat[batch_i]
6:   transfer adjMat[batch_i] to host mem
7: determine corePoints with # of \( \varepsilon \)-neighbors
8: for batch_i in # of mini-batches do
9:   transfer adjMat[batch_i] to device mem
10: merge \( \varepsilon \)-neighbors of corePoints to isNeighbor[batch_i]
11: transfer isNeighbor[batch_i] to host mem
```
Mini Batch DBSCAN: Ping Pong Buffer

- Ping-Pong buffers can overlap- ping computation and data transfer
  - time should be comparable
  - otherwise, overlap is negligible.

- Distance matrix calculation
  - computation & data transferring time is comparable
- BUT, for merging ε-neighbors
  - overlap is negligible

1. for batch_i in # of mini-batches do
2.   transfer dataset[batch_i] to device mem O(DN)
3.   calculate the distance matrix disMat[batch_i] O(MDN)
4.   identify ε-neighbors adjMat[batch_i] O(NM)
5.   accumulate the # of ε-neighbors of adjMat[batch_i]
6.   transfer adjMat[batch_i] to host mem O(NM)
7.   determine corePoints with # of ε-neighbors
8. for batch_i in # of mini-batches do
9.   transfer adjMat[batch_i] to device mem O(MN)
10. merge ε-neighbors of corePoints to isNeighbor[batch_i]
11. transfer isNeighbor[batch_i] to host mem
Speculative Execution

- The control flow depends on # of neighbors: calculated with a loop over mini-batches

```python
1: for batch_i in # of mini-batches do
2:    transfer dataset[batch_i] to device mem
3:    calculate the distance matrix disMat[batch_i]
4:    identify ϵ-neighbors adjMat[batch_i]
5:   accumulate the # of ϵ-neighbors of adjMat[batch_i]
6:    transfer adjMat[batch_i] to host mem
7: determine corePoints with # of ϵ-neighbors
8: for batch_i in # of mini-batches do
9:    transfer adjMat[batch_i] to device mem
10:   merge ϵ-neighbors of corePoints to isNeighbor[batch_i]
11:    transfer isNeighbor[batch_i] to host mem
```

Control Dependency
Speculative Execution

• The control flow depends on # of neighbors: calculated with a loop over mini-batches
• kernels are executed in two loops, even if deal with the same piece of data

1. for batch_i in # of mini-batches do
2. transfer dataset[batch_i] to device mem
3. calculate the distance matrix disMat[batch_i]
4. identify ε-neighbors adjMat[batch_i]
5. accumulate the # of ε-neighbors of adjMat[batch_i]
6. transfer adjMat[batch_i] to host mem
7. determine corePoints with # of ε-neighbors
8. for batch_j in # of mini-batches do
9. transfer adjMat[batch_i] to device mem
10. merge ε-neighbors of corePoints to isNeighbor[batch_i]
11. transfer isNeighbor[batch_i] to host mem
Speculative Execution

- The neighborhood relation is usually symmetric
  - a point has many $\varepsilon$-neighbors, its $\varepsilon$-neighbors usually also have many $\varepsilon$-neighbors
- Based on this, we propose a speculative merging strategy
- Aggressively assumes all processed frontiers are core points
  - speculatively merges their $\varepsilon$-neighbors in the first loop

1: for batch_i in # of mini-batches do
2:   transfer dataset[batch_i] to device mem
3:   calculate the distance matrix disMat[batch_i]
4:   identify $\varepsilon$-neighbors adjMat[batch_i]
5:   accumulate the # of $\varepsilon$-neighbors of adjMat[batch_i]
10:  merge $\varepsilon$-neighbors of frontiers :: isNeighbor[batch_i]
11:  transfer isNeighbor[batch_i] to host mem
Speculative Execution

- The neighborhood relation is usually symmetric
  - A point has many \( \varepsilon \)-neighbors, its \( \varepsilon \)-neighbors usually also have many \( \varepsilon \)-neighbors
- Based on this, we propose a speculative merging strategy
  - Aggressively assumes all processed frontiers are core points
    - Speculatively merges their \( \varepsilon \)-neighbors in the first loop
- A correct one: avoids transferring the adjacency matrix
- An incorrect one: incurs \((M - 1) \times N\) bitwise operation
  - Arithmetic is much cheaper than PCIe data transferring
Speculative Execution

- Another bottleneck is initializing clusters
- Computes the distance vector between the potential seed (a single point) and the whole dataset
  - transfer the whole dataset to device mem
  - use TCU to compute matrix-vector multiplication
    - APIs impose limitation, at least $16 \times 16 \times 16$
  - Low arithmetic intensity, low TCU utilization
- Thus, speculatively explore the initial points when constructing new cluster(s)
Speculative Execution

• Speculatively explores many unvisited points rather than one
• A call to `explore_points` is reduced if
  • one of these points is not a core point
  • any two belong to different clusters
• More formally
  • $x$ non-core points and $y$ different clusters reduce $x + y - 1$ times calls to `explore_points`
• How many points are explored speculatively?
  • explore 16 points incurs no extra cost, but poor tiling strategy
  • thus, explore as many as possible until exhausting on-chip memory
Evaluation

- Evaluate on both Nvidia GPUs and Huawei Ascend 310

<table>
<thead>
<tr>
<th>Device</th>
<th>Architecture</th>
<th>TCU Size</th>
<th>TCU throughput</th>
<th>Memory Type</th>
<th>Memory Size</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nvidia RTX 2080ti</td>
<td>Turing</td>
<td>4 x 4 x 4</td>
<td>53.8 TFLOPS</td>
<td>GDDR6</td>
<td>11 GB</td>
<td>CUDA + WMMA</td>
</tr>
<tr>
<td>Nvidia RTX 3090</td>
<td>Ampere</td>
<td>4 x 8 x 4</td>
<td>71.0 TFLOPS</td>
<td>GDDR6X</td>
<td>24 GB</td>
<td>CUDA + WMMA</td>
</tr>
<tr>
<td>Huawei Ascend 310</td>
<td>DaVinci</td>
<td>16 x 16 x 16</td>
<td>11.0 TFLOPS</td>
<td>LPDDR4X</td>
<td>8 GB</td>
<td>TIK&amp;TBE DSL [16]</td>
</tr>
</tbody>
</table>

- Datasets are in 128-dimensional space
  - # of points: 256K – 8M
    - sizes range from 100MB to 10GB
  - # of clusters: 0 to 4096
    - 0 indicates all points are noise points
    - D-NORMAL: datasets with a reasonable number of clusters (i.e., 1 – 4096)
    - D-NOISE: dataset without any cluster
Evaluation: Mini Batch Framework

- Compared with CUML library
- NV-CUDA: use CUDA core to compute distance matrix
- NV-TCU: use TCU to compute distance matrix

- Small dataset: AC-DBSCAN is slightly slower than CUML
- Large dataset: AC-DBSCAN is much faster than CUML
On D-8M: NV-TCU has 19.43x throughput of CUML on RTX 3090
  - CUML use 11.1GB device memory, mem error on RTX 2080ti
  - Tensor core vs CUDA core
    - 2.6x speedup on RTX 3090
  - Better scalability
    - device memory usage can be controlled by the size of the mini-batch
Evaluation: Mapping Functions

- HA-CPU: deploy $\varepsilon$-neighbor identification to CPU
- HA-CMPSEL: CMPSEL func in TBE DSL
- HA-$f_2.0/\pm 0.0$, HA-$f_{128}/0$: proposed method
- "(SP)" suffix indicates speculative execution is enabled

![Graph showing execution time in $10^3$ sec for different scenarios.](image)
Evaluation: Mapping Functions

- Portability:
  - Our mapping functions are supported by almost all AI chips
  - No need to deploy computation to CPU
  - Avoid transferring distance matrix to host mem
    - much larger than adjacency matrix
  - Vectorized bitwise op is usually faster than CMPSEL on CPUs
  - 16.51x and 17.88x throughput
• Performance Portability:
  • The performance is also portable
    • i.e., have notional performance on different AI chips.
    • The hardware implementations of bitwise AND/OR are about the same.
  • In contrast, the CMPSEL operation can be implemented in various ways.
• No need to fine-tune the performance
• 23.1% and 29.0% higher throughput of HA-CMPSEL
Evaluation: Speculative Execution

- Speculative merging speedup
  - NV-TCU by 16.4% on D-NORMAL
  - HA-f\textsubscript{-128/0} by 15.8% on D-NORMAL
- Speculative initialization avoid very long execution time caused by abnormal datasets (e.g., D-NOISE)
  - reduces 93.9% on Nvidia GPUs
  - reduces 99.3% on Ascend 310
Future Work

• Extend AC-DBSCAN to multiple AI chips and distributed system
  • the mini-batch manner -> good scalability
  • rethink the bottleneck
    • large capacity SSD is common
    • GPUDirect can transfer SSD -> GPU directly
  • save the whole dataset in each machine’s SSD and assign works based on computation?

• Integrating AC-DBSCAN with
  • accelerating index structures
  • approximation algorithms
Conclusion

- This paper presented AC-DBSCAN, a DBSCAN algorithm designed for AI chips.
  - 2.61× throughput by deploying distance calculation to Ampere TCUs
  - With high portability, our ε-neighbor identification kernels can be executed on almost all AI chips
    - 16.20× higher throughput than deploying ε-neighbor identification to CPUs.
  - HA-\(f_{-128/0}\) reduces the execution time by 29.0% compared with HA-CMPSEL.
  - The speculative execution further reduces the execution time by 15.1% on D-NORMAL and 99.0% on D-NOISE.
THANK YOU
Q & A