Enabling Distance-based Addressing in Non-Volatile Memory systems

Ananth Krishna Prasad, Rajeev Balasubramonian, Mahdi Nazm Bojnordi
School of Computing, University of Utah

I. Introduction

Nearest Neighbor search (NNS) is a fundamental problem in computer science. NNS with lower dimensionsal data uses tree-based data structures such as K-D trees, where the aggressiveness of pruning can be tuned to either do exact (NNS) or approximate nearest neighbor searches (ANNS). Unfortunately, as the dimensionality of data increases, the efficiency of pruning drops sharply in case of NNS. This means that for high-dimensional data, it is often required to visit every single data point in the dataset to identify the closest k points. Linear scan is the most bandwidth-efficient approach.

It turns out that most of the present-day applications of nearest neighbors can tolerate some degree of loss in accuracy, leading to the emergence of ANNS as a solution in various database applications [1], [5], [6]. The proposed indexing algorithms for ANNS can be classified into 3 categories - Trees, Graphs, and Compression [4]. Indexing algorithms have exposed a design space trade-off between the accuracy of the ANNS and the complexity of computation. Such algorithms are less bandwidth-intensive compared to the brute force approach due to dataspace pruning. However, there are multiple sources of inefficiency in the application of such indexing algorithms to ANNS. **1)** The choice of the indexing algorithm depends on different factors such as working-set size, dimensionality, accuracy requirements, and the target distance metric. **2)** Indexing algorithms incur memory indirection in the form of pointer chasing, which impacts the processing latency. **3)** Most importantly, current applications work with **Billion-scale** data, exceeding sizes of 100s of GB. This necessitates using either distributed servers or SSDs to store the complete working set, thereby exacerbating latency inefficiency by adding factors such as SSD roundtrip delays/inter-node communication.

Given that ANNS is inherently data-parallel, and almost all of the previously mentioned competing algorithms are limited by pointer-chasing, it is a good fit for exploiting processing-in-memory (PIM) primitives. However, given the many parameters in ANNS, it is important to design a PIM framework that is general enough to target a large algorithm space. To this end, we make the following contributions:

- We propose **Distance Addressable Memory (DAM), an algorithm-hardware co-design approach to perform in-memory search space pruning** depending on a given distance metric to shortlist and access "similar" datapoints within a dataset to a given target point. The **1T1R**-based RAM performs search space pruning while keeping the data in place, thereby incurring no pointer-chasing latency overheads while maintaining high bandwidth utilization. The PIM module efficiently creates a short-list of candidates, which are then read out to the host processor to further narrow down nearest points.
- We propose an **adaptive parameterization technique, which helps build the data structure for search-space pruning on-the-fly** within memory. The parameters define the minimum distance to be searched iteratively across every single dimension. This also exposes the accuracy v/s latency design space to the users.
- Initial results across different parameters show competing performance with a State-of-the-art (SOTA) accelerator for Billion-Scale similarity search applications [2] while exposing to the user a larger design space. We also show an order of magnitude improvement over a state-of-the-art software library.

II. Distance-Addressable Memory Overview

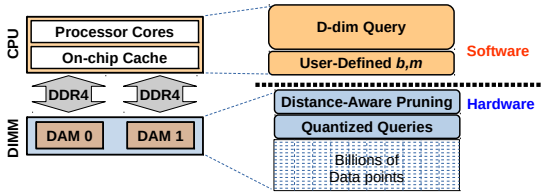

Fig. 1. Overview of the proposed system.

Figure 1 shows an overview of the proposed system. Each DIMM contains multiple DAM chips, capable of performing distance-aware pruning. The user space contains the D-dimensional queries, for which similar points are required to be found from the memory. Two user-defined parameters-b, m (defind shortly) dictate the quality of pruning. b, m are used in extracting the Quantized Query which is used in the search. The quantized query is written to the quantized query buffer (QQB) within the DIMM, which is used as the query for the current iteration of PIM. The two subsections below propose an algorithm to extract the quantized queries using b, m and thereby perform search, and explain how the algorithm is implemented using PIM primitives respectively.

A. Proposed Algorithm

The proposed algorithm for multi-dimensional data pruning is inspired by our prior work on data ranking [3] and is shown in algorithm 1. Assume a $\mathbf{D}$-dimensional dataset $\mathbf{S}$ and query $\boldsymbol{\tau}$, with n bits used to represent each dimension. The quantized query $\boldsymbol{\tau\prime}$ and dataset $\mathbf{S\prime}$ are generated by applying a right bit-shift of b on each dimension d of $\boldsymbol{\tau}$ and $\mathbf{S}$. This can be visualized as generating a (n - b) bit key ($\mathbf{p\prime}$) for each datapoint

($\mathbf{p}$). Each key corresponds to a bucket with 2^b values. m defines the number of buckets to be searched in either direction of the target bucket $\tau_{d\prime}$ within each d. All datapoints falling within the search range across all D dimensions belong to the candidate set $\mathbf{C}$.

Algorithm 1 function MultiDimDataPrune(b, m)

$\boldsymbol{\tau} \leftarrow$ query, $\mathbf{S} \leftarrow$ {all datapoints}, $\mathbf{D} \leftarrow$ {all dims}
$\mathbf{C} \leftarrow \varnothing$, $\boldsymbol{\tau\prime} \leftarrow \boldsymbol{\tau} >> b$, $\mathbf{S\prime} \leftarrow \mathbf{S} >> b$
for all $p\prime \in \mathbf{S\prime}$ **do**
  for all $d \in D$ **do**
    if $|p_{d\prime} - \tau_{d\prime}| \leq m$ **then**
      $\mathbf{C} \leftarrow \mathbf{C} \cup \{p\}$

B. Proposed Hardware, Control, and Results

The proposed memory array is illustrated with a small example in Figure 2. The array considers a configuration of $n = 3$ and shows 3 3-dimensional datapoints, with $b = 1$. There are $D = 3$ entries within the QQB, with each entry having $2m + 1$ different masks (m bins in each direction of the target-bin, plus the target-bin), and the QQB is shared across different arrays. The rightmost $b = 1$ bits within each QQB entry are masked out, denoted by bitlines outside the search range. The address mapping ensures that every datapoint within an array is bit-aligned. The search in-hardware proceeds in a dimension-serial data-parallel fashion. Each dimension search proceeds in a bit-serial fashion. The bitline is activated based on the QQB mask corresponding to the specific bit-location, which reads the value into the selectline. A comparison occurs between the corresponding bit-location within the QQB key and the seleclines across all rows. If a specific bit-location has a mismatch in a row, the stream detection logic deactivates the row and excludes it from consideration in future iterations. Hence, the stream detection logic ensures that the select bit stays 1 only if one of the $2m+1$ bins searched for each dimension returns a one, across all dimensions. This ensures that the hardware follows the same search algorithm as explained in the previous section.

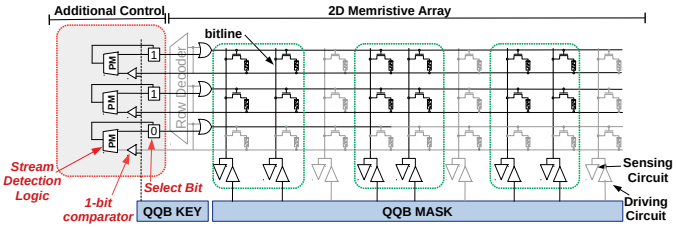

Fig. 2. Proposed Memory Array.

After the search is performed across all dimensions, the select bits denote the rows containing the datapoints forming our candidate set $\mathbf{C}$. The select bits are accumulated into a count of the number of active rows present in memory, which is then read by the on-chip controller through a RAM read request to a special address. The memory controller then issues reads corresponding to the total count of active rows in the memory. Such read requests are unaware of the exact location of active rows forming C. A priority encoder employed within the memory selects the data to be read out for each read request, similar to [3]. The unused commands as part of the DDR protocol for NVMs are re-purposed (without protocol changes) to enable the two key operations required within DAM - (a) write Quantized target values $\boldsymbol{\tau\prime}$ to QQB (b) Read final count of candidates from the Memory. The software maintains a priority queue which stores the top-k candidate points from C. The proposed memory is also extensible to newer technologies such as CXL.

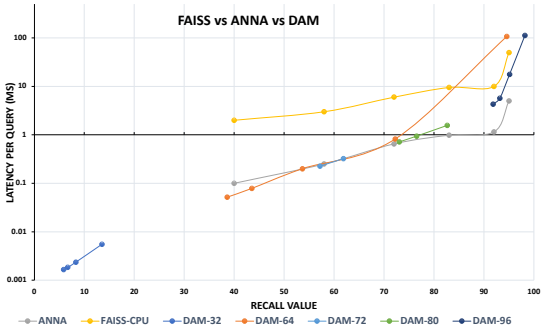

Fig. 3. Results of Proposed Approach

The results for BIGANN-1B, a 128-dimension $uint8$-based dataset is shown in Figure 3. The graph presents two baselines-ANNA [2] (H/W accelerator) and FAISS (software on a CPU). Accuracy (or Recall) here is defined as the portion of true top 100 datapoints identified by the ANN algorithm. Each datapoint corresponds to a different configuration. The different trendlines of DAM correspond to different values of minimum range ($2^b \times m$) to be searched wrt each dimension. The different datapoints within each trendline correspond to different combinations of b, m used to achieve the same minimum range. On average, DAM performs similarly to ANNA across the best configurations, while giving the same benefits as ANNA over the FAISS baseline.

While we were unable to improve upon ANNA's latency, we note that DAM has several other advantages. Our approach is more scalable than ANNA due to the in-place nature of computation. It allows tunable b and m that enable iterative ANN without having to rebuild data structures, as is required in the baselines. The current iteration of DAM is limited by its inefficient select-bit based pruning mechanism. As part of follow-up work, we are exploring modifications to hardware that allow DAM to apply additional levels of pruning, rather than just a bit-based binary pruning.

REFERENCES

[1] N. Hasanzadeh and Y. Forghani, "Improving the accuracy of m-distance based nearest neighbor recommendation system by using ratings variance." *Ingénierie des Systèmes d'Information*, vol. 24, no. 2, 2019.

[2] Y. Lee, H. Choi, S. Min, H. Lee, S. Beak, D. Jeong, J. W. Lee, and T. J. Ham, "Anna: Specialized architecture for approximate nearest neighbor search," pp. 169–183, 2022.

[3] A. K. Prasad, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi, "Memristive data ranking," in *2021 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2021.

[4] S. J. Subramanya, F. Devvrit, H. Simhadri, R. Krishnawamy, and R. Kadekodi, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," *Advances in Neural Information Processing Systems*, vol. 32, pp. 13 771–13 781, 2019.

[5] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk, "Approximate nearest neighbor negative contrastive learning for dense text retrieval," *arXiv preprint arXiv:2007.00808*, 2020.

[6] J. Zhang, Y. Yao, and B. Deng, "Fast and robust iterative closest point," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.