TPU-KNN
K Nearest Neighbor Search at Peak FLOP/s

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

This paper presents a novel nearest neighbor search algorithm achieving TPU (Google Tensor Processing Unit) peak performance, outperforming state-of-the-art GPU algorithms with similar level of recall. The design of the proposed algorithm is motivated by an accurate accelerator performance model that takes into account both the memory and instruction bottlenecks. Our algorithm comes with an analytical guarantee of recall in expectation and does not require maintaining sophisticated index data structure or tuning, making it suitable for applications with frequent updates. Our work is available in the open-source package of Jax and Tensorflow on TPU.

1 Introduction

The K-nearest neighbor (K-NN) search problem has a wide range of applications in machine learning and information retrieval systems, including image search (Jia et al., 2021; Babenko and Lempitsky, 2016), semantic textual retrieval (Liu et al., 2009; Cer et al., 2018), anomaly detection (Gu et al., 2019; Omar et al., 2013), recommendation systems (Sarwar et al., 2002; Zhao et al., 2019), as well as serving as a component for a downstream tasks (Borgeaud et al., 2021; Guu et al., 2020; Lindgren et al., 2021; Shazeer et al., 2017). Given a query, the objective of K-NN is to identify K closest datapoints from a database of finite number of data points in a vector space. The main challenge of designing a good K-NN algorithm is to compute accurate K-NN results while being computationally efficient.

Solving the K-NN problem on accelerators has emerging interests from both the academia and the industry (Johnson et al., 2021; Shanbhag et al., 2018; Zhao et al., 2020). Many accelerators can deliver hundreds of Tera Floating Point Operations Per Seconds (TFLOPS) vital to the neighbor distance computation. However, utilizing accelerators in K-NN problems is not straightforward; multiple issues in data locality, memory bandwidth, and multiple types of hardware parallelism need to be carefully considered to achieve high utilization. In this paper we extend the *roofline performance model* (Williams et al., 2009) to quantify the hardware characteristics accurately. As a result, we designed a K-NN algorithm to reach peak performance by the precise modeling of the accelerators, and our TPU implementation aligned with our predicted performance.

The main contributions of this work are:

- We extend the roofline model to address the operation throughput differences of the instructions, essential to the algorithm analysis in this paper.
- We design an approximate K-NN algorithm with recall and performance guarantees based on our proposed roofline model.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

- We conduct experiments verifying our TPU implementation of the algorithm accurately aligned with the performance model and achieves state-of-the-art speed-recall trade-offs on standard nearest neighbor search benchmarks.

2 Preliminaries

This section covers the necessary notations to work with the nearest neighbor search problem. Given a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, we let $a_{i,j}$ denote the item at the ith row and jth column of $\mathbf{A}$, and $\mathbf{a}_i$ denote the ith *row-vector* of $\mathbf{A}$. We use the matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ to abbreviate a set-representation of a database $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,2,\ldots,N}$ with N data points, where each data point $\mathbf{x}_i \in \mathbb{R}^D$ is a row vector of the matrix $\mathbf{X}$ in a D dimensional vector space. The set and matrix representation of database $\mathbf{X}$ are used interchangeably in this paper.

The K nearest neighbor search problem is stated as follows. Given a database $\mathbf{X} \in \mathbb{R}^{N \times D}$ and a query vector $\mathbf{q} \in \mathbb{R}^D$, find the subset $\mathbf{S}^* \subset \mathbf{X}$ collecting the K-closest data points to $\mathbf{q}$:

$$\boxed{{\mathbf{S}_\mathbf{q}}^* = K\text{-}\underset{\mathbf{x} \in \mathbf{X}}{\text{argmin}}\ \mathcal{D}(\mathbf{q}, \mathbf{x}),} \tag{1}$$

where $\mathcal{D}(\mathbf{x}, \mathbf{y})$ is a distance measure such as Euclidean distance $\mathcal{D}_{\ell^2}(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|_2$ or the cosine distance $\mathcal{D}_{\cos}(\mathbf{x}, \mathbf{y}) := 1 - \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|\|\mathbf{y}\|}$. A related problem is the maximum inner product search (MIPS), where the goal is to find the data points that have the highest inner products with the query:

$$\boxed{{\mathbf{S}_\mathbf{q}}^* = K\text{-}\underset{\mathbf{x} \in \mathbf{X}}{\text{argmax}}\ \langle \mathbf{q}, \mathbf{x} \rangle.} \tag{2}$$

MIPS is equivalent to the cosine similarity search when all data points are ℓ^2-normalized.

3 Related work

Exhaustively searching all pair-wise distances between the query and the entire database is compute-intensive and often infeasible on many platforms. Therefore, a problem extensively discussed in the literature (Wang et al., 2014, 2015) is to find *approximate nearest neighbors* (ANN) in exchange of speed. By convention, the quality of ANN is measured by

$$\boxed{\text{Recall} := \frac{|\mathbf{S}_\mathbf{q} \cap {\mathbf{S}_\mathbf{q}}^*|}{|{\mathbf{S}_\mathbf{q}}^*|},} \tag{3}$$

where $\mathbf{S}_\mathbf{q} \subset \mathbf{X}$ denotes the set of data points retrieved by the search method.

Compressed domain search One class of ANN approaches is to search on a lossy-compressed problem domain. These methods are composed in two steps: a) search on compressed representation[1] of the original problem to find a set of candidate data points, b) compute the distances between the query and the candidate data points to select the top-K results. Since only a subset of data points requires the exact distance computation, the overall cost is reduced.

The two steps can be composed in arbitrary ways. Locality sensitive hashing (Andoni et al., 2015; Neyshabur and Srebro, 2015) applies search followed by scoring; tree-search (Muja and Lowe, 2014; Dasgupta and Freund, 2008) applies the two steps recursively; graph-search (Malkov and Yashunin, 2018) iterates between two steps until the stopping condition is met. And the inverted file (IVF) method (Jegou et al., 2010; Babenko and Lempitsky, 2014; Baranchuk et al., 2018; Guo et al., 2020) search on subset of data points indexed by the k-means centroids.

We see that there are two major challenges with the compressed domain search:

[1]Here we mean data structures like tree, graph, locality sensitive hash etc.

- Fractional search has a poor cache reuse rate because the candidate data points for each query rarely overlaps. We show optimizing the cache usage has a huge headroom for accelerators in Section 4.2.
- Tweaking the speed-recall trade-off is data-dependent and non-trivial to tune. The key result of Beyer et al. (1999) states that the distance contrast of neighbors diminishes with increasing dimensionality (also known as the curse of high dimensionality). Furthermore, the key result of Rubinstein (2018) states that sub-linear time nearest neighbor search with high recall is impossible for Euclidean, Manhattan, or Hamming distance; otherwise, it contradicts the Strong Exponential Time Hypothesis (Impagliazzo and Paturi, 1999).

Our work takes an opposite approach to focus on machine efficiency with zero search space pruning. Moreover, since our method computes all the distances, it is immune to the curse of high dimensionality.

Accelerators In this paper, the phrase *accelerators* represents a class of specialized hardware to accelerate machine learning workloads. In particular, we are interested in the novel platforms that deliver high FLOP/s for distance computation, namely Google TPU V3, V4, Nvidia GPU V100, and A100 in our analysis and evaluation.

Modern accelerators have special computation units for matrix multiplication, providing a higher operation throughput over the regular coefficient-wise operations. The corresponding units are tensor cores in Nvidia GPUs (Markidis et al., 2018) and systolic arrays in Google TPUs (Jouppi et al., 2017; Norrie et al., 2021). Addressing these operation throughput differences is essential to our algorithm design.

While accelerators excel in parallelism, developing an efficient K-selection algorithm on accelerators is still an active research area (Monroe et al., 2011; Shanbhag et al., 2018; Johnson et al., 2021; Zhao et al., 2020). Accelerators with higher FLOP/s introduce a higher opportunity cost of computing the K-selection problem instead of the distance computation. The trend of the increasing FLOP/s in accelerators motivated us to optimize the FLOP/s usage by reducing the time required for computing K-selection.

4 Methodology

This section presents a performance model to identify non-trivial bottlenecks on multiple platforms and demonstrates some fundamental limits when designing algorithms for K-NN and related problems, and we see that the cache inefficiency of the compressed domain methods introduces a significant cost on accelerators.

We model the accelerator's runtime as executing a sequence of *computation kernels*, where each kernel is a compiled subroutine on the accelerator used by the main program on the CPU. A kernel may be composed of one or several high-level operators: Einsum, ReLU, ArgMax, etc., and each kernel can have different performance characteristics.

Given a sequence of kernels k_i, we let W_i denotes the total amount of work and P_i denotes the operational speed. Our goal is to estimate the total time of a program:

$$t = \sum_i \frac{W_i}{P_i}. \tag{4}$$

In the following example, we focus on the MIPS problem. Let $\mathbf{Q} \in \mathbb{R}^{M \times D}$ and $\mathbf{X} \in \mathbb{R}^{N \times D}$ denote the queries and the database, the runtime of a generic approximate-MIPS program can be modeled as

$$\boxed{t = \frac{\lambda W_{\mathcal{D}}}{P} + \mathcal{O}(\text{Auxiliary}) \geq \frac{\lambda W_{\mathcal{D}}}{P},} \tag{5}$$

where $W_{\mathcal{D}}$ denotes the total FLOPs required for searching the entire database, and λ denotes the *search fraction*. We note that P varies by algorithm and platform. Traditionally, compressed domain search methods minimize λ but sacrifice cache efficiency. Our method use an alternative route to optimize P instead.

Table 1: Hardware specifications for the generalized roofline model

Name	π (TFLOP/s)	β (GB/s)	γ (TCOP/s)
GPU V100	125	900	15.7
GPU A100	312	1555	19.5
TPU V3	126	858	4.0
TPU V4	274	1144	4.3

4.1 Instruction throughput-aware roofline model

This subsection describes how we model the kernel-dependent performance P on multiple platforms with a small extension of the roofline model.

The *classic roofline model* (Williams et al., 2009) is a function of machine peak performance π measured in FLOP/s, machine peak memory bandwidth β measured in bytes/s, and arithmetic intensity I_{MEM} expressed as the ratio of floating-point operations performed to data movement (FLOP/byte). The model states the performance is bounded by $P \leq \min(\pi, \beta \times I_{\text{MEM}})$.

We desire to model kernels that has a mixture of floating point operations accelerated by dedicated hardware as well as other coefficient-wise operations. The coefficient-wise operations are abbreviated as COPs. Almost every non matrix multiplication operations are COPs, including vectorized add, multiply, compare, conditional-move, etc. We use the symbol γ for peak COP/s on platforms, and define the instruction throughput intensity I_{COP} as the ratio between the number FLOPs and the number of COPs performed in a kernel (FLOP/COP). The attainable performance of a kernel is bounded by:

$$\boxed{P \leq \min \begin{cases} \pi \\ \beta \times I_{\text{MEM}} \\ \gamma \times I_{\text{COP}}. \end{cases}} \tag{6}$$

The statement is self-explanatory because the inadequate resources impede the kernel throughput. Table 1 lists the properties of selected accelerators for our analysis[2]. The roofline model is commonly used in accelerator profiling tools but not as frequently discussed in algorithm designs. The following sections show how the model prevents pitfalls due to the hardware constraints.

4.2 The memory bandwidth bound

This subsection demonstrates how to evaluate if a kernel hits the memory bandwidth wall. We associate the distance computation with three levels of BLAS (Dongarra et al., 1990). Level 1 BLAS describes vector operations on non-consecutive memory access, such as computing distances while traversing through a graph. Level 2 BLAS represents scoring a query with consecutively stored data points. Level 3 BLAS expresses batched query-database distance computation, often used in brute-force scoring.

Compressed domain searches are either level 1 or 2 BLAS due to the cache inefficiency. It has $\mathcal{O}(1)$ memory arithmetic intensity because the number of FLOPs is proportion to the bytes read. Combining (5) and (6) we have the following remark:

Remark 1. *Distance computations in compressed domain searches are memory bandwidth bounded. In our model, the runtime is lower bounded by:* $t \geq \mathcal{O}\left(\lambda W_{\mathcal{D}}/\beta\right)$.

To estimate the memory arithmetic intensity for level 3 BLAS, we continue to use $\mathbf{Q} \in \mathbb{R}^{M \times D}$ and $\mathbf{X} \in \mathbb{R}^{N \times D}$ for denoting queries and database. In many K-NN applications N and M are much greater than D. The corresponding memory arithmetic intensity is:

$$I_{\text{MEM}} = \frac{2MND}{4MN + o(MN)} \approx \frac{D}{2}. \tag{7}$$

[2]Readers can find these numbers from the accelerators' specification sheets.

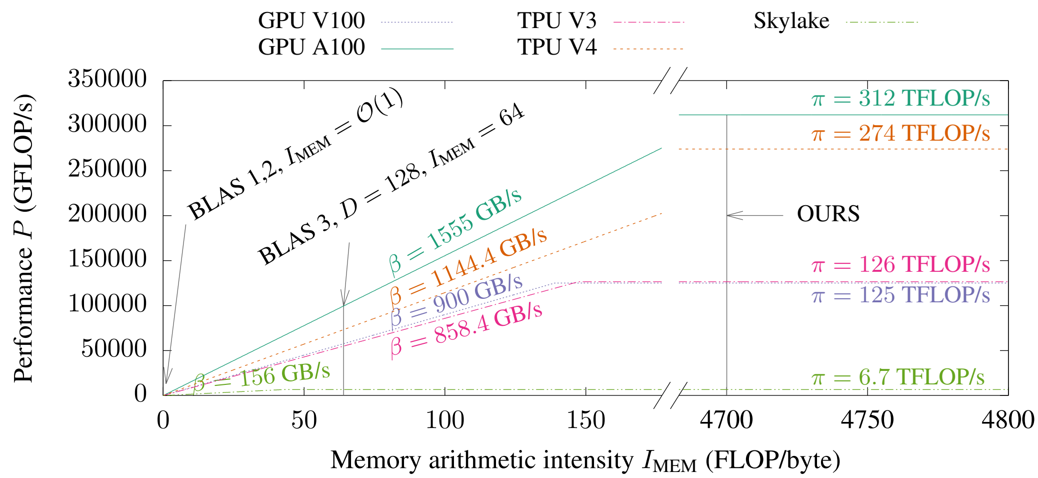

Figure 1: Memory rooflines of accelerators and a dual-sockets Intel skylake machine as a baseline. Each colored line denotes the maximum performance a platform could achieve, and each vertical line represents the memory arithmetic intensity of an algorithm. The intersections of the lines show the maximum performance of an algorithm could achieve on a platform. We label three levels of BLAS kernels and our algorithm described in Section 5.

The largest term in the denominator of (7) is the $4MN$ bytes of the query-database distances. We omit the insignificant terms and refer readers to (Golub and Van Loan, 2013, Section 1.5.4) for a comprehensive review on memory transfers in block matrix multiplications.

Figure 1 shows that the distance scoring kernels of different BLAS levels can easily hit the memory bandwidth wall. In order to attain high performance, we designed our algorithm to aggregate the results within the kernel to avoid writing the $\mathcal{O}(MN)$ bytes into memory.

4.3 The instruction bandwidth bound

The use of COPs (non matrix multiplication instructions) introduce another slowdown. We let C denotes the number of COPs used per dot-product score in a kernel equipped with COPs and matrix multiplication instructions. There are $M \times N$ dot-product scores, so the total COPs used in a kernel is CMN. To prevent hitting the COPs bandwidth wall, we must satisfy:

$$I_{\text{COP}} = \frac{2\cancel{MN}D}{C\cancel{MN}} \geq \frac{\pi}{\gamma}, \tag{8}$$

$$\Rightarrow C \leq \frac{2D \times \gamma}{\pi}. \tag{9}$$

The number of COPs we can afford in the kernels is scarce. We take $D = 128$ as an example and substitute it into (9). We can only use 4 coefficient-wise instructions per dot-product for TPU V4, and 16 for GPU A100. We conclude with the following remark:

Remark 2. *Exact and generic K-selection algorithm cannot be efficiently implemented with the coefficient-wise operations for the selected platforms (GPU V100, A100, TPU V3 and V4).*

Because of Remark 2, we develop an approximate approach to achieve the peak performances.

5 Algorithm

Our algorithm consists of two kernels:

1. PartialReduce kernel computes the distances and partially aggregate the results from $M \times N$ distances to $M \times L$ distances with original indices.

Algorithm 1: PartialReduce for MIPS

Input: $\mathbf{Q} \in \mathbb{R}^{M \times D}$ Batch queries
Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$ Database
Input: 2^W Bin size
Output: $\mathbf{V} \in \mathbb{R}^{M \times L}$ Top-K values
Output: $\mathbf{A} \in \mathbb{N}^{M \times L}$ Top-K indices

for $i \leftarrow 1$ **to** M **do**
  for $j \leftarrow 1$ **to** N **do**
    $y_{i,j} \leftarrow \langle \mathbf{q}_i, \mathbf{x}_j \rangle$;
    $l \leftarrow$ `ShiftRight`(j, W) ; `/* Unrolled and does not cost COP */`
    $b \leftarrow y_{i,j} > v_{i,l}$; `/* COP 1: Vectorized compare */`
    $v_{i,l} \leftarrow$ **if** b **then** $y_{i,j}$ **else** $v_{i,l}$; `/* COP 2: Vectorized conditional move */`
    $a_{i,l} \leftarrow$ **if** b **then** j **else** $a_{i,l}$; `/* COP 3: Vectorized conditional move */`
  end
end

2. ExactRescoring kernel is an *optional* kernel that aggregates the final top-K results. The complexity is $\mathcal{O}(ML\log^2(L))$ by a bitonic sort followed by a truncation.

The PartialReduce kernel is where most of the time and compute takes place. See Algorithm 1 for an outline of the algorithm. We collect top-1 distances from the L non-overlapping bins of size 2^W for each query, resulting high arithmetic intensities:

$$I_{\text{MEM}} \approx \mathcal{O}\left(\min\left(M, N\right)\right), \tag{10}$$

$$I_{\text{COP}} = \frac{2\cancel{MN}D}{C\cancel{MN}} = \frac{2D}{C}. \tag{11}$$

We show these arithmetic intensities can achieve high performance on real world database in section 6.1. See Appendix A.3 for the detailed expansion of the algorithm and how the arithmetic intensities are derived.

5.1 Recall estimation

This section shows the PartialReduce kernel can achieve high recall with good speed. We reformulate our problem in terms of balls and bins. We have K balls representing the top-K distances that are thrown into L bins. The location of each ball is chosen independently and uniformly at random. We let $\mathbf{Z}$ denotes the random variable of the number of balls that does not share the bin with other balls. Following the recall definition (3) we have:

$$\text{Recall} = \frac{\mathbf{Z}}{K}, \tag{12}$$

which is a standard Birthday problem:

$$\mathbb{E}[\text{Recall}] = \frac{\mathbb{E}[\mathbf{Z}]}{K} = \left(\frac{L-1}{L}\right)^{K-1}. \tag{13}$$

Our goal is to find the minimal L such that the expected recall is greater equals to the target recall r. Finding L is simple because (13) is invertible in the natural range $0 < r < 1$.

$$\boxed{\mathbb{E}[\text{Recall}] \geq r \Rightarrow L \geq \frac{1}{1 - r^{1/(K-1)}} \approx \frac{K-1}{1-r}.} \tag{14}$$

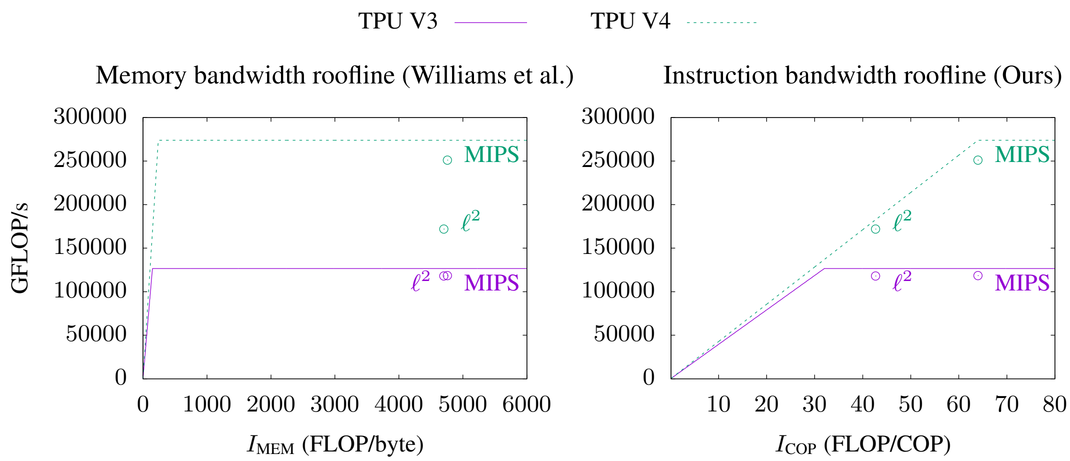

Figure 2: Roofline plots for MIPS and ℓ^2 search benchmarks using the PartialReduce kernel. The colored lines denotes the attainable performance derived from Table 1. The figure on the left shows none of the benchmark is memory bandwidth limited. The figure on the right shows that our model gives a much tighter bound for ℓ^2 on TPU V4. See also Appendix A.5 for detailed deviation of the numbers.

The approximation in (14) follows from Appendix A.4. Since L is at the order of K, and in most applications $K \ll N$, the cost of the ExactRescoring kernel is amortized out. Thus we affirm the claim that our method attains high performance with an analytical recall guarantee. $\square$

6 Evaluation

In this section, we show that our proposed algorithm and implementation is near the hardware limit and leads to superior performance over the baselines of similar recalls. We applied our algorithm to two datasets from the public ANN benchmarks (Aumüller et al., 2020). In our first evaluation, we compares the measured FLOP/s to the theoretical peak governed by the proposed refinement of the roofline model (6), proclaiming our implementation is reaching the hardware peak performance. In the second benchmark, we compare the end-to-end performance with competitive baselines with pre-tuned parameters. We plot each algorithm's speed-recall curve and show ours achieves the state-of-the-art.

6.1 Comparison with the theoretical peak

This section shows that our refined roofline model (6) captures additional performance characteristic over the classic roofline model, and demonstrates our kernels are having near optimal performances. We select the Glove[3] (Pennington et al., 2014) and Sift[4] (Jegou et al., 2010) datasets from the ANN benchmarks. Their corresponding distances are the cosine distance and the Euclidean distance. See the code snippets in Appendix A.1 and A.2.

See Figure 2, the colored lines represent machines' max performances, and the dots represent each benchmark with its measured FLOP/s. The classic roofline on the left shows that our in-cache aggregation strategy has a large memory arithmetic intensity (∼4,700) exceeding the memory bandwidth ridge points π/β. However, it is difficult to diagnose why the Euclidean distance search does not perform well on TPU V4 from the classic roofline plot.

Fortunately, when combined with the instruction bandwidth roofline we can tell the performance regression is caused by hitting the coefficient-wise operation throughput wall. Therefore we affirms

[3]Released in Apache license 2.0.
[4]Released in CC0 public domain.

the claim that our MIPS solution is reaching the peak FLOP/s, and our Euclidean distance search solution is meeting the compute bound on TPU V4 and attaining the peak FLOP/s on TPU V3.

6.2 Recall-speed benchmark

To evaluate the effectiveness of the K-NN algorithm in a realistic setting, we adopted the methodology of public ANN benchmarks (Aumüller et al., 2020) to compare the end-to-end performance against other methods. The typical ANN benchmarks are only performed on a single platform. However, it is non-trivial to either port our TPU algorithm to GPU or vice versa. Alternatively, we selected the following GPUs with parity in peak performance to TPU (Table 1).

We select the Faiss GPU (Johnson et al., 2021) implementation as our baseline. Faiss provides three algorithms: Flat, IVF-flat, and IVF-PQ. The Flat algorithm performs a brute-force search, and the IVF-Flat and IVF-PQ algorithms corresponds to the inverted file method with and without the product quantization (Jegou et al., 2010; Johnson et al., 2021). We use the repository's suggested inverted file size (16384) in the IVF methods.

Figure 3 shows our performance significantly outperforms competing methods in the high recall regions. We highlight that our method has a consistent recall-speed trade-off over different datasets, because our recall only rely on the order statistics instead of the information encoded in the compression domain search methods, which may vary by the datasets. Since our method scores all the pair-wise distances, our method is immune from the curse of high dimensionality.

7 Discussion and future work

We limit our experiments and discussion to single-chip accelerator K-NN performance of dense vectors. Accelerators performance on sparse vectors follow a completely different paradigm due to random memory access. Our algorithm can be naturally extended to multi-chip or distributed settings to handle billion scale datasets through Tensorflow's `tf.distribute` (Abadi et al., 2015) or Jax's `jax.pmap` (Bradbury et al., 2018) programming interfaces.

It is also possible to use our operations in conjunction with other strategies, including dimension reduction, quantization and tree search (Van Der Maaten et al., 2009; Jegou et al., 2010; Wang et al., 2014), because many compressed domain search methods use brute-force distance computation on its auxiliary data structures before performing the fractional search. We note that heterogeneous architectures with off-HBM storage such as host-RAM or even SSD (Chen et al., 2021; Jayaram Subramanya et al., 2019; Ren et al., 2020) are great starting points for future research.

Finally, while our refinement of the roofline model handles kernel with mixture of instructions that have different throughput, there are many studies that have extended the roofline model to related topics in recent years: GPU warp instruction roofline (Ding and Williams, 2019), time-based roofline (Wang et al., 2020), roofline for multiple cache tiers (Yang et al., 2021), and energy rooflines (Choi et al., 2013; Lopes et al., 2017). Readers may refer to these models for some analysis that are left out, such as the auxiliary work in (5).

8 Conclusion

Accelerator-based machine learning has become the mainstream in academics and industries. However, the performance characteristics of accelerators are counter-intuitive and difficult to program. In this paper, we propose a roofline-based complexity analysis framework to discuss the optimality of the algorithms without low-level optimization details: unrolling factors, batch window sizes, vectorization, and systolic array scheduling, which are platform-dependent and lengthy to read. We demonstrated several examples of inferring the hardware performance limits by simply addressing the kernel's total FLOPs, byte transferred, and the number of coefficient-wise instructions used. Our refined model foreshadowed non-trivial performance regression caused by the coefficient-wise instructions bandwidth. We took it into account to design a new algorithm for K-NN and achieved peak performance on TPU. Finally, our experiments showed that our method outperformed state-of-the-art baselines on platforms with similar performance characteristics, which are known to be hard to beat.

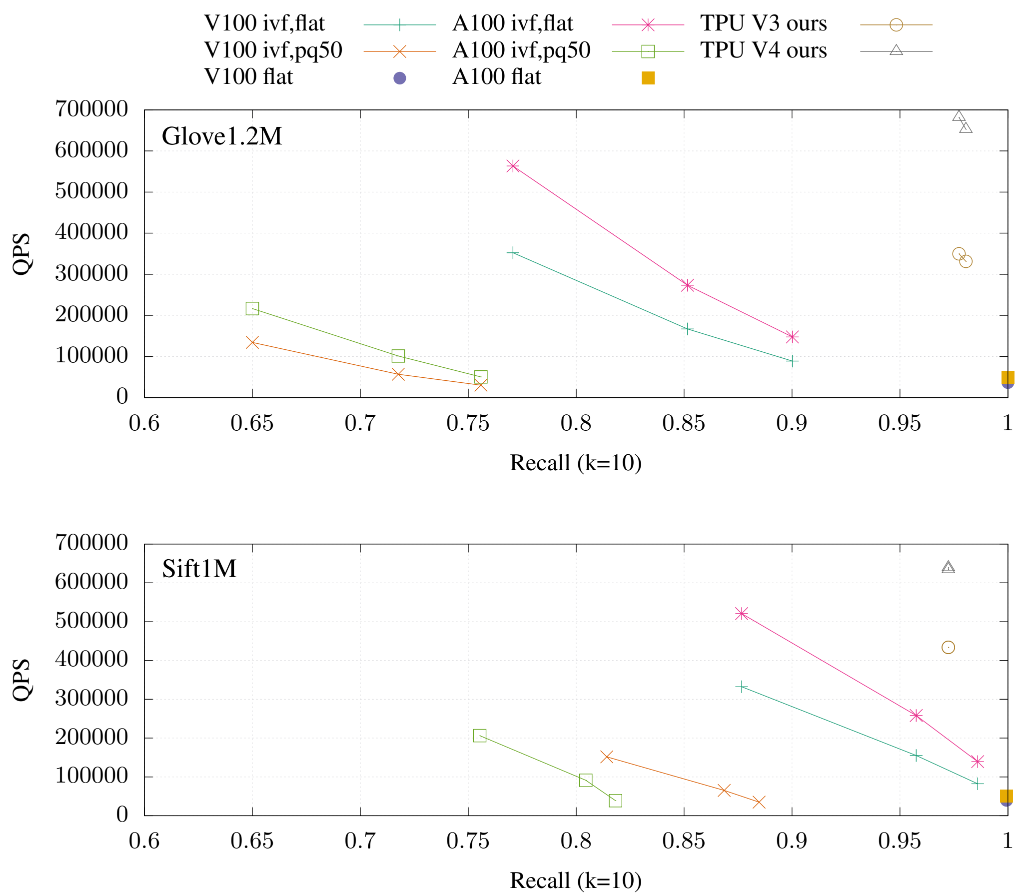

Figure 3: Speed-recall trade-off on Glove1.2M and Sift1M. Up and to the right the better. The GPU methods (ivf-flat, ivf-pq, and flat) are released by Faiss (Johnson et al., 2021). For each ivf* benchmark, the search fractions are $\lambda = \{0.24\%, 0.61\%, 1.22\%\}$. We note that the recall differences between datasets with similar ivf search configurations is a known problem asserted by Rubinstein (2018).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., and Schmidt, L. (2015). Practical and optimal lsh for angular distance. *Advances in neural information processing systems*, 28.

Aumüller, M., Bernhardsson, E., and Faithfull, A. (2020). Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374.

Babenko, A. and Lempitsky, V. (2014). The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence*, 37(6):1247–1260.

Babenko, A. and Lempitsky, V. (2016). Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2055–2063.

Baranchuk, D., Babenko, A., and Malkov, Y. (2018). Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 202–216.

Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., Driessche, G. v. d., Lespiau, J.-B., Damoc, B., Clark, A., et al. (2021). Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Chen, Q., Zhao, B., Wang, H., Li, M., Liu, C., Li, Z., Yang, M., and Wang, J. (2021). Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems*, 34:5199–5212.

Choi, J. W., Bedard, D., Fowler, R., and Vuduc, R. (2013). A roofline model of energy. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 661–672. IEEE.

Dasgupta, S. and Freund, Y. (2008). Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546.

Ding, N. and Williams, S. (2019). *An instruction roofline model for gpus*. IEEE.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. S. (1990). A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17.

Golub, G. H. and Van Loan, C. F. (2013). *Matrix computations*. JHU press.

Gu, X., Akoglu, L., and Rinaldo, A. (2019). Statistical analysis of nearest neighbor methods for anomaly detection. *Advances in Neural Information Processing Systems*, 32.

Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., and Kumar, S. (2020). Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. (2020). Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.

Impagliazzo, R. and Paturi, R. (1999). The complexity of k-sat. In *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference)(Cat. No. 99CB36317)*, pages 237–237. IEEE Computer Society.

Jayaram Subramanya, S., Devvrit, F., Simhadri, H. V., Krishnawamy, R., and Kadekodi, R. (2019). Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems*, 32.

Jegou, H., Douze, M., and Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Jia, C., Yang, Y., Xia, Y., Chen, Y.-T., Parekh, Z., Pham, H., Le, Q., Sung, Y.-H., Li, Z., and Duerig, T. (2021). Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, pages 4904–4916. PMLR.

Johnson, J., Douze, M., and Jégou, H. (2021). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12.

Lindgren, E., Reddi, S., Guo, R., and Kumar, S. (2021). Efficient training of retrieval models using negative cache. *Advances in Neural Information Processing Systems*, 34.

Liu, T.-Y. et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331.

Lopes, A., Pratas, F., Sousa, L., and Ilic, A. (2017). Exploring gpu performance, power and energy-efficiency bounds with cache-aware roofline modeling. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 259–268. IEEE.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.

Markidis, S., Der Chien, S. W., Laure, E., Peng, I. B., and Vetter, J. S. (2018). Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 522–531. IEEE.

Monroe, L., Wendelberger, J., and Michalak, S. (2011). Randomized selection on the gpu. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 89–98.

Muja, M. and Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240.

Neyshabur, B. and Srebro, N. (2015). On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning*, pages 1926–1934. PMLR.

Norrie, T., Patil, N., Yoon, D. H., Kurian, G., Li, S., Laudon, J., Young, C., Jouppi, N., and Patterson, D. (2021). The design process for google's training chips: Tpuv2 and tpuv3. *IEEE Micro*, 41(2):56–63.

Omar, S., Ngadi, A., and Jebur, H. H. (2013). Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, 79(2).

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ren, J., Zhang, M., and Li, D. (2020). Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory. *Advances in Neural Information Processing Systems*, 33:10672–10684.

Rubinstein, A. (2018). Hardness of approximate nearest neighbor search. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pages 1260–1268.

Sarwar, B. M., Karypis, G., Konstan, J., and Riedl, J. (2002). Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, pages 291–324. Citeseer.

Shanbhag, A., Pirk, H., and Madden, S. (2018). Efficient top-k query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1557–1570.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Van Der Maaten, L., Postma, E., Van den Herik, J., et al. (2009). Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13.

Wang, J., Liu, W., Kumar, S., and Chang, S.-F. (2015). Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 104(1):34–57.

Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*.

Wang, Y., Yang, C., Farrell, S., Zhang, Y., Kurth, T., and Williams, S. (2020). Time-based roofline for deep learning performance analysis. In *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, pages 10–19. IEEE.

Williams, S., Waterman, A., and Patterson, D. (2009). Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76.

Yang, C., Wang, Y., Kurth, T., Farrell, S., and Williams, S. (2021). Hierarchical roofline performance analysis for deep learning applications. In *Intelligent Computing*, pages 473–491. Springer.

Zhao, W., Tan, S., and Li, P. (2020). Song: Approximate nearest neighbor search on gpu. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1033–1044.

Zhao, Z., Hong, L., Wei, L., Chen, J., Nath, A., Andrews, S., Kumthekar, A., Sathiamoorthy, M., Yi, X., and Chi, E. (2019). Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 43–51.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4 for how we model the hardware limitations and Section 6.1 for real world evaluations.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 5.1, we formulate the problems in terms of the classic balls into bins.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section 5.1 and Appendix A.4.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix A.1 and A.2
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Table 1 and 2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 6.1.
   (b) Did you mention the license of the assets? [Yes] See the footnotes in Section 6.1.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]