Similarity Search with Tensor Core Units

Thomas D. Ahle[1] and Francesco Silvestri[2(✉)]

[1] IT University and BARC, Copenhagen, Denmark
thdy@itu.dk
[2] University of Padova, Padova, Italy
silvestri@dei.unipd.it

Abstract. Tensor Core Units (TCUs) are hardware accelerators developed for deep neural networks, which efficiently support the multiplication of two dense $\sqrt{m} \times \sqrt{m}$ matrices, where m is a given hardware parameter. In this paper, we show that TCUs can speed up similarity search problems as well. We propose algorithms for the Johnson-Lindenstrauss dimensionality reduction and for similarity join that, by leveraging TCUs, achieve a $\Omega(\sqrt{m})$ speedup up with respect to traditional approaches.

Keywords: Similarity search · Tensor core units · Dimensionality reduction · Similarity join · Locality sensitive hashing

1 Introduction

Several hardware accelerators have been introduced to speed up deep neural network computations, such as Google's Tensor Processing Units [13] and NVIDIA's Tensor Cores [16]. The most important feature of these accelerators is a hardware circuit to efficiently compute a small and dense matrix multiplication between two $\sqrt{m} \times \sqrt{m}$ matrices, where m is a given hardware parameter. On modern chips m can be larger than 256 [13]. Matrix multiplication is indeed one of the most frequent operations in machine learning, and specialized hardware for supporting this operation can significantly reduce running times and energy requirements [12]. We refer to these accelerators as *Tensor Core Units* (TCUs).

Recently, several studies have been investigating how to use TCUs in other domains. For instance, TCUs have been used for scanning and prefix computations [10], linear algebra primitives like matrix multiplication and FFT [9,15], and graph problems [9]. The key designing goal when developing TCU algorithms is to decompose the problem into several small matrix multiplications of size $\sqrt{m} \times \sqrt{m}$, which are then computed on the accelerator. Such algorithms also imply fast external memory algorithms, though not the other way around, since the matrix multiplication chip can be seen as a restricted cache [9].

This work was partially supported by UniPD SID18 grant, PRIN17 20174LF3T8, MIUR "Departments of Excellence".

S. Satoh et al. (Eds.): SISAP 2020, LNCS 12440, pp. 76–84, 2020.
https://doi.org/10.1007/978-3-030-60936-8_6

The goal of this paper is to show that TCUs can also speed up similarity search problems. As case studies, we propose TCU algorithms for the Johnson-Lindenstrauss dimensionality reduction and for similarity join. In both cases, our results improve the performance by a factor $\sqrt{m}$ with respect to state of the art approaches without hardware accelerators.

We analyze our algorithms on the (m, τ)-*TCU model*, which is a computational model introduced in [9] and capturing the main hardware features of TCU accelerators. In the (m, τ)-TCU model, it is possible to compute the matrix multiplication between two matrices of size $\sqrt{m} \times \sqrt{m}$ in time τ, where m and τ are given parameters. In a traditional machine, without accelerators, we have $\tau = \Theta(m^{3/2})$.[1] In contrast, with TCUs, we have $\tau = O(m)$ (i.e., input size complexity) or even sublinear time under some assumptions.

The *Johnson-Lindenstrauss (JL)* dimensionality transform reduces the dimension of a vector $x \in \mathbb{R}^d$ to roughly $k = \varepsilon^{-2} \log(1/\delta)$ while preserving its norm up to a factor $1 \pm \varepsilon$ with probability at least $1 - \delta$. It is an important primitive in many learning algorithms, since it dramatically reduces the number of trained variables, while preserving important characteristics of the feature vectors, such as their pairwise inner products. The JL transform can be represented as a multiplication of the input vector $x \in \mathbb{R}^d$ by a $k \times d$ matrix. This naively takes time $\Omega(dk)$. In this paper we use recent breakthroughs in dimensionality reduction techniques, combined with TCU's to reduce the time to $O(dk/\sqrt{m} + d + k^2 \log^3 \frac{d}{k})$. This is significant, since TCUs typically cut a factor $\sqrt{m}$ off matrix-matrix multiplication, but here we cut $\sqrt{m}$ off *matix-vector* multiplication! When $\sqrt{m} \geq k$ our dimensionality reduction takes time linear in the input dimension. This improves upon even the famous "Fast Johnson Lindenstrauss" transform [6], which takes time $\Omega(d \log d + k^{2+\gamma})$ for any $\gamma > 0$ [7], or $\Omega(d \frac{\log d}{\log m})$ with TCU optimized FFT [9].

The *Similarity Join* on two sets P and Q of n points each in $\mathbb{R}^d$, asks us to find all pairs $(x, y) \in P \times Q$ whose distance is below a given threshold r (i.e., all near pairs). Similarity join occurs in numerous applications, such as web deduplication and data cleaning. As such applications arise in large-scale datasets, the problem of scaling up similarity join for different metric distances is getting more important and more challenging. Exact similarity join cannot be faster than brute force [4], but by leveraging Locality Sensitive Hashing (LSH), we will develop a TCU approximate algorithm that, under some assumptions, finds all pairs in expected time $O((\frac{n}{\sqrt{m}})^{\rho}(\frac{|P \bowtie_r Q| d}{\sqrt{m}} + n))$, where $|P \bowtie_r Q|$ is the number of near pairs. When $\tau = O(m)$, the TCU algorithm exhibits a $\Omega(\sqrt{m})$ speedup with respect to traditional approaches (even those based on LSH).

[1] Fast matrix multiplication algorithms require $O(m^{\omega/2})$ time with $\omega \in [2, 3]$, [8], but they exhibit poor experimental performance than traditional $\Theta(m^{3/2})$ algorithms.

2 Preliminaries

2.1 The TCU Model

(m, τ)-TCU model is a RAM model with an instruction to multiply two dense matrices of size $\sqrt{m} \times \sqrt{m}$ in time τ, where m and τ are given parameters depending on the underline platform.[2] It is reasonable to assume that $\tau = O(m)$, that is matrix multiplication takes linear time: indeed, on TCUs, the cost of the operation is upper bounded by the time for reading/writing the $\sqrt{m} \times \sqrt{m}$ matrices, while the cost of the $m^{3/2}$ elementary products is negligible due to the high level of parallelism inside TCU accelerators (e.g., systolic array). Moreover, under some conditions on high bandwidth connections, we might have τ to be even sublinear (e.g., $O(\sqrt{m})$). We recall a result from [9] that will be used later:

Theorem 1. *Let A and B be two matrices of size $p \times r$ and $r \times q$ with $p, r, q \geq \sqrt{m}$, then there exists an algorithm for computing $A \cdot B$ on a (m, τ)-TCU model in time $O(prqm^{-3/2}\tau)$.*

2.2 Johnson-Lindenstrauss Dimensionality Reduction

We say a distribution over random matrices $M \in \mathbb{R}^{k \times d}$ is a (ε, δ)-Johnson-Lindenstrauss (JL) distribution, if we have $\Pr[|\|Mx\|_2 - 1| \leq \varepsilon] \geq 1 - \delta$ for all unit vectors $x \in \mathbb{R}^d$ In this section we will note some definitions and lemmas related to building and combining random matrices in ways related to JL distributions. The first property was introduced by Kane and Nelson [14]:

Definition 1 (JL-moment property). *We say a distribution over random matrices $M \in \mathbb{R}^{k \times d}$ has the (ε, δ, p)-JL-moment property, when $E[\|Mx\|_2^2] = 1$ and $\left(E\left[\left|\|Mx\|_2^2 - 1\right|^p\right]\right)^{1/p} \leq \varepsilon\delta^{1/p}$ for all $x \in \mathbb{R}^d$, $\|x\|_2 = 1$.*

A distribution with the (ε, δ, p)-JL-moment property is (ε, δ)-JL because of Markov's inequality: $\Pr[|\|Mx\|_2 - 1| > \varepsilon] \leq \mathrm{E}\left[\left|\|Mx\|_2^2 - 1\right|^p\right] / \varepsilon \leq \delta$.

An interesting property of the JL Moment Property is related to the tensor product of matrices. The tensor (or Kronecker) product between two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{k \times \ell}$ is defined as below. In particular, if we take the tensor product $I_k \otimes A$, where I_k is the $k \times k$ identity matrix, we get a $km \times kn$ block matrix with A on the diagonal:

$$A \otimes B = \begin{bmatrix} A_{1,1}B & \cdots & A_{1,n}B \\ \vdots & \ddots & \vdots \\ A_{m,1}B & \cdots & A_{m,n}B \end{bmatrix}, \quad I_k \otimes A = \begin{bmatrix} A & 0 & \cdots & 0 \\ 0 & A & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A \end{bmatrix}.$$

[2] The model in [9] is slightly different, and we use here a simplified version for the clarity of exposition.

The tensor product relates to the JL-moment property by the following simple lemma from [1]:

Lemma 1 (JL Tensor lemma). *For any matrix, Q, with (ε, δ, p)-JL moment property, $I_k \otimes Q$ has (ε, δ, p)-JL moment property.*

By the simple property $A \otimes B = (I \otimes B)(A \otimes I)$ this lemma allows studying the JL properties of general tensor products, as long as we can also handle matrix products. The following generalization of the JL Moment Property will be key to doing exactly that:

Definition 2 ((ε, δ)-Strong JL Moment Property). *Let $\varepsilon, \delta \in [0, 1]$. We say a distribution over random matrices $M \in \mathbb{R}^{m \times d}$ has the (ε, δ)-Strong JL Moment Property, when $E\left[\|Mx\|_2^2\right] = 1$ and $\left(E\left[\left|\|Mx\|_2^2 - 1\right|^p\right]\right)^{1/p} \leq \frac{\varepsilon}{e}\sqrt{\frac{p}{\log 1/\delta}}$, for all $x \in \mathbb{R}^d$, $\|x\|_2 = 1$ and all p such that $2 \leq p \leq \log 1/\delta$.*

Note that the (ε, δ)-Strong JL Moment Property implies the $(\varepsilon, \delta, \log 1/\delta)$-JL Moment Property, since then $\varepsilon\delta^{1/p} = \varepsilon/e$. Similarly, having the $(\varepsilon\sqrt{2/e}, \delta, p)$-JL-moment property for all $p \in [2, \log 1/\delta]$ implies the Strong JL Moment Property, since $\delta^{1/p} \leq \frac{1}{\sqrt{2e}}\sqrt{\frac{p}{\log 1/\delta}}$.

The key workhorse is the following lemma by Ahle and Knudsen [2]. Note that the original lemma required the $(\varepsilon/(C_0\sqrt{k}), \delta)$-Strong JL Moment Property, but a quick scan of the proof shows that $(\varepsilon/(C_0\sqrt{i}), \delta)$-Strong suffices.

Lemma 2 (JL Product lemma). *There exists a universal constant C_0, such that, for any constants $\varepsilon, \delta \in [0, 1]$ and positive integer $k \in \mathbb{Z}_{>0}$. If $M^{(1)} \in \mathbb{R}^{d_2 \times d_1}, \ldots, M^{(k)} \in \mathbb{R}^{d_{k+1} \times d_k}$ are independent random matrices satisfying the $(\varepsilon/(C_0\sqrt{i}), \delta)$-Strong JL Moment Property, then the matrix $M = M^{(k)} \cdot \ldots \cdot M^{(1)}$ has the (ε, δ)-Strong JL Moment Property.*

Intuitively this says that combining k JL reductions, we don't get an error of εk, as we would expect from the triangle inequality, but only $\varepsilon\sqrt{k}$, as we would expect from a random walk.

2.3 Locality Sensitive Hashing

Much of recent work on similarity search and join has focused on Locality Sensitive Hashing: at a high level, similar points (i.e., with distance $\leq r$) are more likely to collide than far points (i.e., with distance $\geq cr$ for a given approximation factor c). Formally, an LSH is an (r, cr, p_1, p_2)-sensitive hashing scheme:

Definition 3. *Fix a distance function $D : \mathbb{U} \times \mathbb{U} \to \mathbf{R}$. For positive reals r, c, p_1, p_2, and $p_1 > p_2, c > 1$, a family of functions $\mathcal{H}$ is* (r, cr, p_1, p_2)-sensitive *if for uniformly chosen $h \in \mathcal{H}$ and all $x, y \in \mathbb{U}$:*

- *If $D(x, y) \leq r$ then $Pr[h(x) = h(y)] \geq p_1$;*
- *If $D(x, y) \geq cr$ then $Pr[h(x) = h(y)] \leq p_2$.*

We say that $\mathcal{H}$ is monotonic if $Pr[h(x) = h(y)]$ is a non-increasing function of the distance function $D(x, y)$.

LSH schemes are characterized by the $\rho = \log_{p_2} p_1$ value, with $\rho \in [0, 1]$: small values of ρ denote LSHs that well separate near points from far points. Term c is the approximation factor.

3 Dimensionality Reduction

We will describe a construction of a matrix $M \in \mathbb{R}^{k \times d}$ which is (ε, δ)-JL as described in the preliminaries, and for which there is an efficient algorithm for computing the matrix vector product Mx on a TCU. We first give a general lemma describing the construction, then show how it applies to TCUs:

Lemma 3. *Let $T(a, b, c)$ be the time for multiplying two matrices of size $(a \times b)$ and $(b \times c)$. For a constant $C > 0$ and for any $d, \varepsilon, \delta > 0$, there exists a matrix $M \in \mathbb{R}^{k \times d}$, with $k = \lceil C\varepsilon^{-2} \log 1/\delta \rceil$, such that $|\|Mx\|_2 - \|x\|_2| \leq \varepsilon \|x\|_2$ for any $x \in \mathbb{R}^d$ with probability $1 - \delta$ (i.e., M is (ε, δ)-JL). The multiplication Mx can be computed in time $\sum_{i=1}^{\ell} T(ik, \zeta ik, \zeta^{\ell - i})$ for any $\zeta > 1$ and ℓ such that $\zeta^\ell = d/k$.*

Note that, depending on the speed of the rectangular matrix multiplication, it might be beneficial to pick different values for ζ.

Proof. We define the JL transformation by the following matrix:

$$M = (I_{r_\ell} \otimes A_\ell) \cdots (I_{r_1} \otimes A_1) \in \mathbb{R}^{r_m k_\ell \times r_1 c_1},$$

where $r_1, \ldots, r_\ell$ is a sequence of positive integers, I_r is the $r \times r$ identity matrix, and $A_1, \ldots, A_{\ell-1}$ are independent $k_i \times c_i$ matrices, where A_i has the $(\varepsilon/(C_0\sqrt{i}), \delta)$-Strong JL Moment Property (SJLMP). By Lemmas 1 and 2 we get that the tail $(I_{r_{\ell-1}} \otimes A_{\ell-1}) \cdots (I_{r_1} \otimes A_1) \in \mathbb{R}^{r_m k_\ell \times r_1 c_1}$ has the $(\varepsilon/\sqrt{C_0}, \delta)$-SJLMP. We further assume A_ℓ has the $(\varepsilon/(\sqrt{2C_0}), \delta)$-SJLMP. Again by Lemmas 1 and 2 we get that M has the (ε, δ)-SJLMP, and thus M is a JL reduction as wanted.

Next we prove the running time of the matrix-vector multiplication. The key is to note that $I \otimes A$ is the "block identity matrix" with A copied along the diagonal. The following figure should give some some intuition:

$$(I_{r_i} \otimes A_i)x = \underset{\text{blocks}}{{}^{r_i}} \left\{ \begin{bmatrix} \overbrace{A_i}^{c_i} & & \\ & A_i & \\ & & A_i \end{bmatrix}^{k_i} x \simeq A_i \left\{ \begin{bmatrix} x_1 & \ldots & x_{r_i} \end{bmatrix}^{c_i} = \begin{bmatrix} y_1 & \ldots & y_{r_i} \end{bmatrix}^{k_i} \right\} \right\}$$

By splitting x into r_i blocks, the multiplication $(I_{r_i} \otimes A)x$ corresponds to reducing each block of x by identical JL matrices. Repeating this process for a logarithmic number of steps, we get the complete dimensionality reduction.

To make sure the matrix sizes match up, we have

$$d = r_1 c_1, \quad r_1 k_1 = r_2 c_2, \quad r_2 k_2 = r_3 c_3, \quad \ldots, \quad r_{\ell-1} k_{\ell-1} = r_\ell c_\ell, \quad r_\ell k_\ell = k.$$

We will define $k = \lceil C\varepsilon^{-2} \log 1/\delta \rceil$, $k_{i<\ell} = ik$, $k_\ell = k$, $c_1 = k\zeta$, $c_{i>1} = \zeta k_{i-1}$, $r_i = \zeta^{\ell-i}$ and $\ell = \frac{\log(d/k)}{\log \zeta}$ such that $c_1 r_1 = k\zeta^\ell = d$. The constant C depends on the constant of the JL lemma we use for the individual A_i, but in general $10C_0^2$ will suffice, where C_0 is the constant of Lemma 2.

Recall the assumption that rectangular multiplication takes time $T(a, b, c)$, and hence the ith step thus takes time $T(k_i, c_i, r_i)$. Adding it all up we get

$$\sum_{i=1}^{\ell} T(k_i, c_i, r_i) = T(k, \zeta k(\ell - 1), 1) + \sum_{i=1}^{\ell-1} T(ik, \zeta k \max(1, i-1), \zeta^{\ell-i})$$

which is then upper bounded by $\sum_{i=1}^{\ell} T(ik, \zeta ik, \zeta^{\ell-i})$. The claim follows.

By the above theorem and by using the matrix multiplication algorithm of Theorem 1, we get the following theorem (see the full version [5] for the proof).

Theorem 2. *For any $d, \varepsilon, \delta > 0$, there exists a (ε, δ)-JL matrix $M \in \mathbb{R}^{k \times d}$ such that the product Mx can be computed in time $O((dk + k^2\sqrt{m} \log^3 \frac{d}{k})\, \tau m^{-3/2})$, on the (m, τ)-TCU model, assuming $k \geq \sqrt{m}$.*

In particular for $\tau = O(m)$ it takes time $O(dk/\sqrt{m} + k^2 \log^3 \frac{d}{k})$. If $\sqrt{m} > k$ we can "pad" the construction by increasing k to $\sqrt{m}$ and simply throw away the unneeded rows. The running time is then $O(d + k^2 \log^3 \frac{d}{k})$. We observe that if $\tau = O(m)$ and d dominates k^2, then we get time $O(dk/\sqrt{m}))$, which improves a factor $\sqrt{m}$ over a standard application of the standard JL transform in the case of dense vectors, and for $m \approx k$ this even improves upon the so-called "Fast JL transform" [6].

Finally, we note the following extra properties of the construction:

1. In the case of sparse vectors, where many blocks of x are empty, we can skip them in the computation.
2. The computation can be easily parallelized, with different blocks of x being reduced on different machines. Our construction also implies a $O(dk/\sqrt{m})$ upper bound in the external memory model.
3. Our construction improves upon the standard matrix-vector multiplication for JL, even in the RAM model, by using the Coppersmith-Winograd method for fast matrix multiplication. In particular we can do JL in time $dk^\varepsilon + k^{2+\varepsilon}$ if matrix multiplication takes time $n^{2+\varepsilon}$.
4. The construction works with any distribution of matrices that have the Strong JL Moment Property. This means we can use random ± 1 matrices or even ε-Sparse JL matrices.

4 Similarity Join

We now study the similarity join problem: given two sets P and Q of n points each in $\mathbb{R}^d$ and a distance function $D : \mathbb{R}^d \to R_0^+$, compute the set $P \bowtie_r Q = \{(x, y) : x \in P, y \in Q, D(x, y) \leq r\}$. We consider distance functions that can be

computed with an inner product on a suitable transformation of the two points: a distance function D is an *ip-distance* is there exist two functions $f, g : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that $D(x, y) = f(x) \cdot g(y)$ for each pair $x, y \in \mathbb{R}^d$ For the sake of simplicity, we assume $d' = \Theta(d)$. Notable examples of ip-distances are Hamming, squared L_2 distance, and cosine similarity: for Hamming, $f(x) = (x_0, 1 - x_0, x_1, 1 - x_1, \ldots, x_{d-1}, 1 - x_{d-1})$ and $g(x) = (1 - y_0, y_0, 1 - y_1, y_1 \ldots, 1 - y_{d-1}, y_{d-1})$; for the squared L_2 distance, $f(x) = (x_0^2, 1, -2x_0, x_1^2, 1, -2x_1 \ldots, x_{d-1}^2, 1, -2x_{d-1})$ and $g(x) = (1, y_0^2, y_0, 1, y_1^2, y_1 \ldots, 1, -y_{d-1}^2, y_2)$; for cosine similarity, $f(x) = g(x) = x/||x||_2$.

The simplest way to exploit TCUs is a brute force approach, where all pair distances are computed. As ip-distance computations can be translated into inner products, we can reduce the similarity join problem to a simple matrix multiplication between two $n \times d'$ matrices F_P and G_Q: F_P and G_Q are the matrices representing, respectively, the sets $\{f(p), \forall p \in P\}$ and $\{g(q), \forall q \in Q\}$. By exploiting TCUs, we can compute $P \cdot Q^T$ in time $O(dn^2 m^{-3/2} \tau)$.

A more efficient approach uses LSH for reducing the number of candidate pairs for which we have to compute distances. The proposed algorithm finds all $P \bowtie_r Q$ pairs in expectation, but it can be easily modified to return all near pairs with high probability by running $O(\log n)$ instances of the algorithm and merging the results.

The standard LSH approach for similarity join (see e.g. [11,17]) partitions the points in $P \cup Q$ into buckets using an (r, cr, p_1, p_2)-sensitive monotone LSH. A brute force algorithm is then used for searching similar pairs within each bucket. The procedure is repeated L times with independent LSHs to guarantee that all near pairs are found. The LSH is usually set so that $p_2 = 1/n$, which implies that each point collides once (in expectation) with a point at distance larger than cr (i.e., a far point), while L is set to $\tilde{O}\left(p_1^{-1}\right) = \tilde{O}\left(p_2^{-\rho}\right) = \tilde{O}\left(n^{\rho}\right)$ to guarantee that each near pair is found once (in expectation).

As for similarity join in the external memory model [17], we can improve the performance in the TCU model by increasing the value of p_2 (i.e., by allowing for more collisions between far points), which implies that the number L of repetitions decreases since $L = p_1^{-1} = \tilde{O}\left(p_2^{-\rho}\right)$. We observe that a TCU unit can multiply two matrices of size $\sqrt{m'} \times \sqrt{m'}$ in a TCU(m, τ) in τ time for each $m' \leq m$, and we exploit this fact by increasing the number of collisions with far points. We set $p_2 = m^{3/2}/(\tau n)$: each point collides in expectation with at most $m^{3/2}/\tau$ far points, but the overhead due to the respective inner products do not dominate the running time.

As an LSH is usually given as a black box $\mathcal{H}'$ with fixed probability values p_1' and p_2', we can get the desired probability $p_2 = m^{3/2}/(\tau n)$ by concatenating $k = \log_{p_2'} p_2$ hash functions. However, if k is not an integer, the rounding gives $L = O(n^{\rho} p_1^{-1})$. A more efficient approach has been recently proposed in [3] that uses L_{high} hash tables by concatenating $\lceil k \rceil$ LSHs $\mathcal{H}'$, and L_{low} hash tables by concatenating $\lfloor k \rfloor$ LSHs $\mathcal{H}'$, and where $L = L_{low} + L_{high} = O(n^{\rho} p_1^{-(1-\rho)})$. The right values of L_{low} and L_{high} depend on the decimal part of k.

We have the following result (see the full version [5] for the proof).

Theorem 3. *Given two sets $P, Q \subset R^d$ of n points, with $n, d \geq \sqrt{m}$, a threshold value $r > 0$, and an (r, c, p_1, p_2)-sensitive monotone LSH, then the set $P \bowtie_r Q$ for an ip-distance can be computed on a TCU(m, τ) in expected time:*

$$O(p_1^{\rho-1}(n\tau m^{-3/2})^{\rho}\left(\frac{|P \bowtie_r Q|\tau}{m^{3/2}} + n\right) + \tau m^{-3/2}|P \bowtie_{cr} Q|).$$

When $\tau = O(m)$, there are at least $n\sqrt{m}$ near pairs, and the number of pairs with distance in $[r, cr]$ is at most linear with the number of near pairs (which happens in several datasets [17]), the cost is $O(p_1^{\rho-1}(n/\sqrt{m})^{\rho}|P \bowtie_r Q|/\sqrt{m})$, a factor at least $\sqrt{m}$ faster than an LSH solution without TCU (e.g., $O(p_1^{\rho-1}n^{\rho}|P \bowtie_r Q|)$).

5 Conclusion

In this paper, we have investigated from a theoretical point of view how to exploit TCU accelerators for similarity search problems, showing a $\Omega(\sqrt{m})$ improvement over algorithms for traditional architectures. As future work, we plan to experimentally evaluate our algorithms on common TCU accelerators, such as the GPU Nvidia Tesla.

References

1. Ahle, T.D., et al.: Oblivious sketching of high-degree polynomial kernels. In: Proceedings of the 40th Symposium on Discrete Algorithms (SODA), pp. 141–160 (2020)
2. Ahle, T.D., Knudsen, J.B.: Almost optimal tensor sketch. arXiv preprint arXiv:1909.01821 (2019)
3. Ahle, T.D.: On the problem of p_1^{-1} in locality-sensitive hashing. In: Proceedings of the 13th International Conference on Similarity Search and Applications (SISAP) (2020)
4. Ahle, T.D., Pagh, R., Razenshteyn, I., Silvestri, F.: On the complexity of inner product similarity join. In: Proceedings of the 35th Symposium on Principles of Database Systems (PODS), pp. 151–164 (2016)
5. Ahle, T.D., Silvestri, F.: Similarity search with tensor core units. arXiv preprint arXiv:2006.12608 (2020)
6. Ailon, N., Chazelle, B.: Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In: Proceedings of the 38th Symposium on Theory of computing (STOC), pp. 557–563 (2006)
7. Ailon, N., Liberty, E.: Fast dimension reduction using rademacher series on dual BCH codes. Discr. Comput. Geom. **42**(4), 615 (2009)
8. Alman, J.: Limits on the universal method for matrix multiplication. In: Proceedings of the 34th Computational Complexity Conference (CCC), vol. 137, pp. 12:1–12:24 (2019)
9. Chowdhury, R., Silvestri, F., Vella, F.: Brief announcement: a computational model for tensor core units. In: Proceedings of the 32nd Symposium on Parallelism in Algorithms and Architectures (SPAA) (2020)

10. Dakkak, A., Li, C., Xiong, J., Gelado, I., Hwu, W.M.: Accelerating reduction and scan using tensor core units. In: Proceedings of the International Conference on Supercomputing (ICS) (2019)
11. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of VLDB'99, pp. 518–529 (1999)
12. Jouppi, N.P., Young, C., Patil, N., Patterson, D.A.: A domain-specific architecture for deep neural networks. Commun. ACM **61**(9), 50–59 (2018)
13. Jouppi, N.P., et al.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th International Symposium on Computer Architecture (ISCA), pp. 1–12 (2017)
14. Kane, D.M., Nelson, J.: Sparser Johnson-Lindenstrauss transforms. J. ACM (JACM) **61**(1), 1–23 (2014)
15. Lu, T., Chen, Y.F., Hechtman, B., Wang, T., Anderson, J.: Large-scale discrete fourier transform on TPUs (2020)
16. Nvidia Tesla V100 GPU architecture. http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
17. Pagh, R., Pham, N., Silvestri, F., Stöckel, M.: I/O-efficient similarity join. Algorithmica **78**(4), 1263–1283 (2017)