High-performance Hardware Architecture for Tensor Singular Value Decomposition
(Invited paper)

Chunhua Deng[1], Miao Yin[1], Xiao-Yang Liu[2], Xiaodong Wang[2], Bo Yuan[1]
[1]Department of Electrical and Computer Engineering, Rutgers University
[2]Department of Electrical Engineering, Columbia University
{chunhua.deng, miao.yin}@rutgers.edu; {xl2427, xw2008}@columbia.edu; bo.yuan@soe.rutgers.edu

***Abstract*—Tensor provides a brief and natural representation for large-scale multidimensional data by way of appropriate low-rank approximations, thus we can discover significant latent structures of complex data and generalize data representation. To date, tensor has gained tremendous success in various science and technology fields, especially in machine learning and big data applications. However, tensor computation, especially tensor decomposition, is usually expensive due to the inherent large-size characteristic of tensors, and hence would potentially hinder their future wide deployment. In this paper, we develop a hardware architecture to accelerate tensor singular value decomposition (t-SVD), which is a new tensor decomposition technique that has been successfully applied to high-dimensional data classification and video recovery. Specifically, design consideration of each key computing unit is analyzed and discussed. Then, the proposed t-SVD hardware architecture is implemented and synthesized using CMOS 28nm technology. Comparison with real-world CPU-based implementations shows that the proposed hardware accelerator is expected to provide average $14\times$ speedup on various t-SVD workloads.**

***Index Terms*—Tensor decomposition, t-SVD, hardware architecture.**

I. Introduction

As the dimensionality of data grows in the big data era, tensor becomes a cornerstone data structure in computational neuroscience, neuroinformatics, pattern recognition, signal processing and machine learning. On account of the superior representation power, tensor has been successfully applied in many practical applications, such as action recognition [1], [2], facial recognition [3], anomaly detection [4], collaborative recommendation [5], videos denoising [6], seismic data super-resolution [7] and compressive sensing [8].

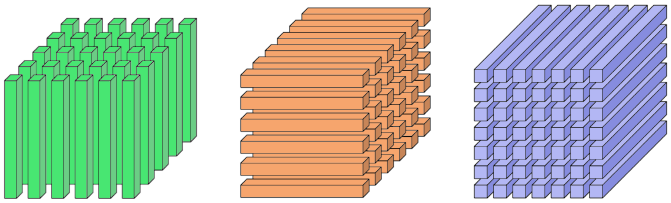
The current success of tensor in various applications highly depends on a set of tensor-specific algorithms. In particular, tensor decomposition, as a class of algorithms that factorize a tensor as the product of latent tensors, has played an important role in enabling the adoption of tensor into various domains, such as numerical linear algebra, computer vision, graph analysis etc. This is because, theoretically, tensor decomposition can aid to find inherent approximate low-rank representation of data, and hence facilitates the solutions of multilinear problems in many real-world tasks. For instance, various types of tensor decomposition approaches, such as CP [9], [10], Tucker [11], tensor train [12] and tensor ring [13], have been widely used in the neural network model compression task [14]–[18], which is a currently very active research topic in the deep learning community.

Among various types of tensor decomposition approaches, tensor singular value decomposition (t-SVD) [19] is a type of new technique that can factorize the input tensor in a matrix SVD-like way. Such a unique characteristic of t-SVD leads to many useful properties of the latent tensors, hence enabling the adoption of t-SVD in various practical applications, such as object classification [20], videos recovery [21] and indoor localization [22].

Despite the current success of tensor decomposition, from the perspective of computing, tensor decomposition is facing severe challenges. In general, tensor decomposition is very computation intensive. Although there have already been several optimized software packages available for performing tensor decomposition, the executing time, no matter on CPU or GPU, is still relatively quite long and cannot meet the real-time requirement in many scenarios. Consequently, to date the adoption of tensor decomposition in those time-sensitive applications, especially in mobile and embedded system, is still very limited.

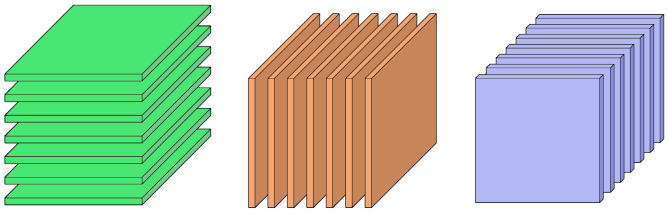
In this paper, we investigate the computing procedure of t-SVD and develop a high-performance hardware architecture to accelerate this computation-intensive tensor decomposition method. Specifically, the design consideration of each key component is carefully discussed and analyzed. A design example of t-SVD hardware architecture is implemented and synthesized using CMOS 28nm technology. Evaluation results show that the proposed t-SVD hardware architecture is expected to deliver an average $14\times$ speedup over different t-SVD workloads as compared with optimized CPU-based implementation.

The rest of this paper is organized as follows. Section II introduces the background of tensor, tensor operations and t-SVD algorithm. The hardware architecture of the proposed t-SVD design is described in Section III. Section IV presents the evaluation setting and comparison results with software implementation. The conclusion is drawn in Section V.



(a) Column fibers (b) Row fibers (c) Tube fibers

Fig. 1: Fibers of a third-order tensor.



(a) Horizontal slices (b) Lateral slices (c) Frontal slices

Fig. 2: Slices of a third-order tensor.

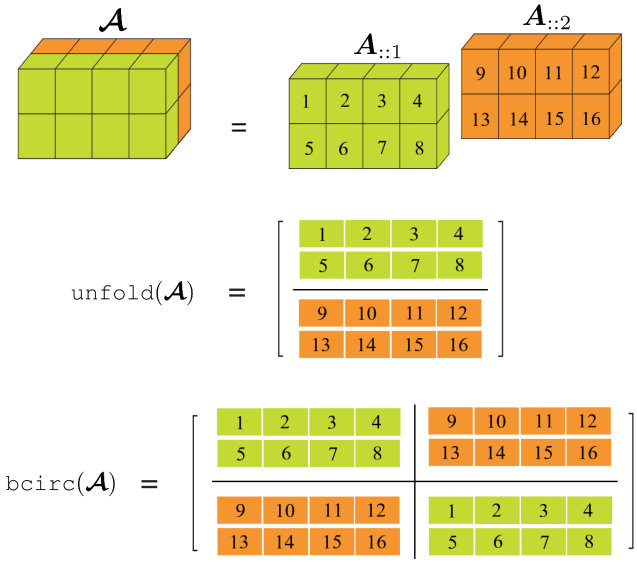


Fig. 3: `unfold` operation and the block circulant matrix of a third-order tensor.

II. Background: Tensor and t-SVD

In this section, we introduce the necessary background information for tensor and t-SVD.

A. Tensor and Related Operations

Notation. We use boldface calligraphic script letters, boldface capital letters, and boldface lower-case letters to represent tensors, matrices, and vectors, respectively. Specifically, a third-order tensor of size $n_1 \times n_2 \times n_3$ is denoted as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

Fiber and Slice. Fiber is defined as a one-dimensional array obtained from a tensor via fixing all indices but one. Column, row, and tube fibers of a third-order tensor $\boldsymbol{\mathcal{A}}$ are denoted as $\boldsymbol{a}_{:jk}$, $\boldsymbol{a}_{i:k}$ and $\boldsymbol{a}_{ij:}$ (see Fig. 1). Different from fiber, slice is defined as a two-dimensional matrices obtained from a tensor via fixing all but two indices. Horizontal, lateral, and frontal slices of a third-order tensor $\boldsymbol{\mathcal{A}}$ are denoted as $\boldsymbol{A}_{i::}$, $\boldsymbol{A}_{:j:}$ and $\boldsymbol{A}_{::k}$ (see Fig. 2). Moreover, $\boldsymbol{A}^{(\ell)}$ represents the ℓ-th frontal slice $\boldsymbol{\mathcal{A}}(:,:,\ell)$ which is a matrix.

t-DFT. $\widetilde{\boldsymbol{\mathcal{A}}}$ is defined as the frequency-domain representation of $\boldsymbol{\mathcal{A}}$. Such time-to-frequency transformation can be obtained by performing discrete Fourier transform (DFT) along the last dimension of $\boldsymbol{\mathcal{A}}$. In practice, fast Fourier transform (FFT) is used here for fast computation.

t-transpose. $\boldsymbol{\mathcal{A}}^\top \in \mathbb{R}^{n_2 \times n_1 \times n_3}$ is defined as the transpose tensor of $\boldsymbol{\mathcal{A}}$. $\boldsymbol{\mathcal{A}}^\top$ is obtained by transposing each of the frontal slices of $\boldsymbol{\mathcal{A}}$ and then reversing the order of transposed frontal slices 2 through n_3.

Tensor Operations. We next introduce several tensor operations that will be used to give a rigid definition of t-SVD. First, the operation $\texttt{unfold}(\boldsymbol{\mathcal{A}})$ is defined to take an $n_1 \times n_2 \times n_3$ tensor and returns a block $n_1 n_3 \times n_2$ matrix (see Fig. 3). Correspondingly, the `fold` undoes the operation:

$$\texttt{unfold}(\boldsymbol{\mathcal{A}}) = \begin{bmatrix} \boldsymbol{A}^{(1)} \\ \boldsymbol{A}^{(2)} \\ \vdots \\ \boldsymbol{A}^{(n)} \end{bmatrix}, \tag{1}$$

$$\texttt{fold}(\texttt{unfold}(\boldsymbol{\mathcal{A}})) = \boldsymbol{\mathcal{A}}. \tag{2}$$

In addition, the block circulant matrix of tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, which will be used in defining the key t-product operation, is an $n_1 n_3 \times n_2 n_3$ matrix (see Fig. 3) defined as follows:

$$\texttt{bcirc}(\boldsymbol{\mathcal{A}}) = \begin{bmatrix} \boldsymbol{A}^{(1)} & \boldsymbol{A}^{(n_3)} & \boldsymbol{A}^{(n_3-1)} & \cdots & \boldsymbol{A}^{(2)} \\ \boldsymbol{A}^{(2)} & \boldsymbol{A}^{(1)} & \boldsymbol{A}^{(n_3)} & \cdots & \boldsymbol{A}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{(n_3)} & \boldsymbol{A}^{(n_3-1)} & \cdots & \boldsymbol{A}^{(2)} & \boldsymbol{A}^{(1)} \end{bmatrix}. \tag{3}$$

Next we describe the definition and fast computing method of t-product, as the key tensor operation for t-SVD.

Definition 1. *[19] **t-product**. For $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_2 \times n_4 \times n_3}$, their t-product $\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ is an $n_1 \times n_4 \times n_3$ tensor defined as:*

$$\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}} = \texttt{fold}(\texttt{bcirc}(\boldsymbol{\mathcal{A}}) \cdot \texttt{unfold}(\boldsymbol{\mathcal{B}})). \tag{4}$$

It should be noted that for practical application the t-product $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ can be calculated efficiently in the frequency domain. Algorithm 1 describes such fast computing procedure using t-DFT. Here $\widetilde{\boldsymbol{\mathcal{C}}}$, as the frequency-domain representation of $\boldsymbol{\mathcal{C}}$, can be obtained via the multiplications of the each frontal slices of $\widetilde{\boldsymbol{\mathcal{A}}}$ and $\widetilde{\boldsymbol{\mathcal{B}}}$. After that, the t-product $\boldsymbol{\mathcal{C}}$ can be finally calculated by taking the inverse Fourier transform along the last dimension of $\widetilde{\boldsymbol{\mathcal{C}}}$.

Algorithm 1 Computing the t-product in the frequency domain

Input: Tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_2 \times n_4 \times n_3}$;
Output: Tensor $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{n_1 \times n_4 \times n_3}$;
1: $\widetilde{\boldsymbol{\mathcal{A}}} = \texttt{fft}(\boldsymbol{\mathcal{A}}, [\,], 3)$, $\widetilde{\boldsymbol{\mathcal{B}}} = \texttt{fft}(\boldsymbol{\mathcal{B}}, [\,], 3)$;
2: **for** $i = 1$ to n_3 **do**
3: $\quad \widetilde{\boldsymbol{\mathcal{C}}}(:,:,i) = \widetilde{\boldsymbol{\mathcal{A}}}(:,:,i)\widetilde{\boldsymbol{\mathcal{B}}}(:,:,i)$;
4: **end for**
5: $\boldsymbol{\mathcal{C}} = \texttt{ifft}(\widetilde{\boldsymbol{\mathcal{C}}})$;

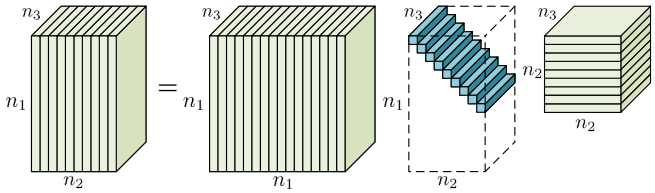


Fig. 4: The t-SVD of an $n_1 \times n_2 \times n_3$ tensor.

B. Tensor Singular Value Decomposition (t-SVD)

In this subsection we describe the definition of t-SVD and the computing procedure. To prepare the rigid definition of t-SVD, three types of special structured tensors need to be described first:

Definition 2. *[19]* ***Identity tensor****. The identity tensor $\boldsymbol{\mathcal{I}} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ is a tensor whose first frontal slice is the $n_1 \times n_1$ identity matrix and all other frontal slices are zero.*

Definition 3. *[19]* ***f-diagonal tensor****. A tensor is called f-diagonal if each frontal slice is a diagonal matrix.*

Definition 4. *[19]* ***Orthogonal tensor****. A tensor $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ is orthogonal if*

$$\boldsymbol{\mathcal{Q}}^\top * \boldsymbol{\mathcal{Q}} = \boldsymbol{\mathcal{Q}} * \boldsymbol{\mathcal{Q}}^\top = \boldsymbol{\mathcal{I}}. \tag{5}$$

Then, the tensor singular value decomposition of an input tensor can be defined as follows:

Definition 5. *[19]* ***t-SVD.*** *For $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, the tensor singular value decomposition (t-SVD) of $\boldsymbol{\mathcal{A}}$ is defined as*

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{U}} * \boldsymbol{\mathcal{S}} * \boldsymbol{\mathcal{V}}^\top,$$

where $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$ are orthogonal tensors of size $n_1 \times n_1 \times n_3$ and $n_2 \times n_2 \times n_3$, respectively, and $\boldsymbol{\mathcal{S}}$ is a rectangular f-diagonal tensor of size $n_1 \times n_2 \times n_3$.

Fig. 4 illustrates the t-SVD of a third-order tensor. From this figure it is seen that t-SVD can be viewed as the generalization and extension of conventional matrix SVD in the high dimension scenario.

Because t-product is a basic component operation for defining t-SVD, the computation procedure of t-SVD can also be efficiently performed in the frequency domain. Algorithm 2 describes the overall computing steps for performing t-SVD on a third-order tensor. Here `SVD` means the matrix SVD and `conj` returns the conjugate element in MATLAB notation.

Algorithm 2 Computing Procedure of t-SVD

Input: Tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$;
Output: $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$, $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{n_2 \times n_2 \times n_3}$;
1: $\widetilde{\boldsymbol{\mathcal{A}}} = \texttt{fft}(\boldsymbol{\mathcal{A}}, [\,], 3)$;
2: **for** $i = 1$ to $\lceil \frac{n_3+1}{2} \rceil$ **do**
3: $\quad [\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \texttt{SVD}(\widetilde{\boldsymbol{\mathcal{A}}}(:,:,i))$;
4: $\quad \widetilde{\boldsymbol{\mathcal{U}}}(:,:,i) = \boldsymbol{U}$;
5: $\quad \widetilde{\boldsymbol{\mathcal{S}}}(:,:,i) = \boldsymbol{S}$;
6: $\quad \widetilde{\boldsymbol{\mathcal{V}}}(:,:,i) = \boldsymbol{V}$;
7: **end for**
8: **for** $i = \lceil \frac{n_3+1}{2} \rceil + 1$ to n_3 **do**
9: $\quad \widetilde{\boldsymbol{\mathcal{U}}}(:,:,i) = \texttt{conj}(\widetilde{\boldsymbol{\mathcal{U}}}(:,:,n_3 - i + 2))$;
10: $\quad \widetilde{\boldsymbol{\mathcal{S}}}(:,:,i) = \widetilde{\boldsymbol{\mathcal{S}}}(:,:,n_3 - i + 2)$;
11: $\quad \widetilde{\boldsymbol{\mathcal{V}}}(:,:,i) = \texttt{conj}(\widetilde{\boldsymbol{\mathcal{V}}}(:,:,n_3 - i + 2))$;
12: **end for**
13: $\boldsymbol{\mathcal{U}} = \texttt{ifft}(\widetilde{\boldsymbol{\mathcal{U}}}, [\,], 3)$;
14: $\boldsymbol{\mathcal{S}} = \texttt{ifft}(\widetilde{\boldsymbol{\mathcal{S}}}, [\,], 3)$;
15: $\boldsymbol{\mathcal{V}} = \texttt{ifft}(\widetilde{\boldsymbol{\mathcal{V}}}, [\,], 3)$;

Specifically, for a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, the first step of t-SVD is to calculate the frequency-domain representation $\widetilde{\boldsymbol{\mathcal{A}}}$ by taking discrete Fourier transform along the third dimension. Then, the second step is to decompose the first $\lceil \frac{n_3+1}{2} \rceil$ frontal slices via matrix SVD. Notice that here the SVD for the rest part can be automatically obtained by using the conjugate symmetry property. Finally, the t-SVD component tensors can be calculated by performing inverse fast Fourier transform to recover the representation in the time domain.

III. Hardware Architecture

In this section, we develop a hardware architecture for t-SVD to map the computing procedure described in Algorithm 2 to hardware computing fabric.

A. Overall Hardware Architecture

Fig. 5 shows the overall architecture of the proposed design. Here the input tensor that needs to be decomposed is stored in the working memory, which contains multiple memory banks. The number of the memory banks N is identical to the maximum setting of n_3. Such memory partition arrangement is to improve data access parallelism for the tube fiber that is to be processed. After the current tube fiber has been read from the memory, it is sent to 1D FFT module to perform domain transform. Notice that here the FFT is performed in an in-place style that its output will be written back to the working memory. Once all the tube fibers are transformed to the frequency domain, they will be read again from working memory to the SVD module for executing the SVD operation. Notice that according to Algorithm 2, each frontal slice of the transformed tensor needs to be decomposed; hence the proposed SVD module is equipped with multiple SVD computing units (SCUs) for parallel processing. Also, as it will be analyzed in Section III-C, the computation of t-SVD involves

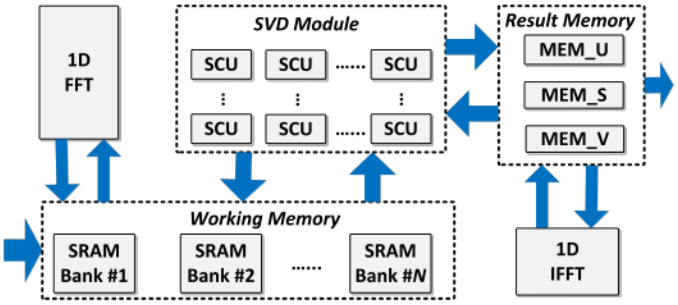


Fig. 5: The overall hardware architecture.

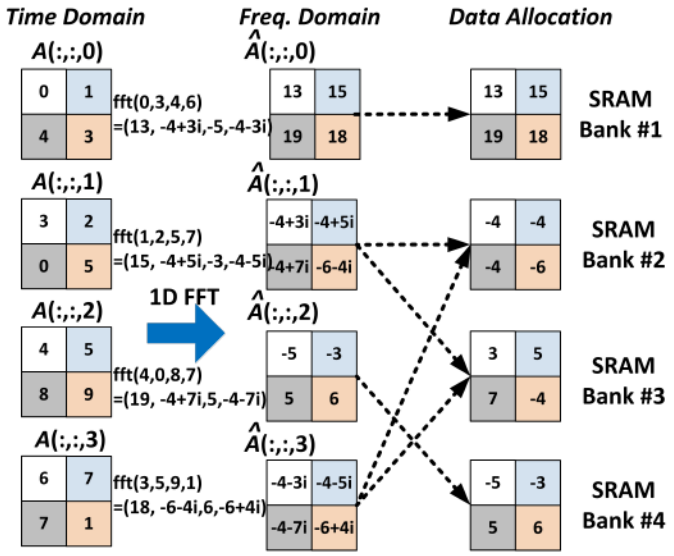


Fig. 6: Example of 1D-FFT for t-SVD across 4 memory banks.

with the continuous update of the final results; therefore, besides extensively communicating with the working memory, the SVD module is also designed to interact with result memory as well. Finally, all the tube fibers of the result tensors are transformed back to the time domain via an 1D IFFT module and stored back to the result memory. Next, we describe the details of each component module of the hardware architecture.

B. The FFT Module

As described in Algorithm 2, the t-SVD algorithm performs FFT operation on the tube fibers of the input tensor. Fig. 6 shows an example transform procedure for a $2 \times 2 \times 4$ tensor. Here the matrices, no matter in the time or frequency domain, represent the frontal slices that are physically stored in different memory banks individually. Also, across different frontal slices, the values with the same color represent the entries belonging to the same tube fiber.

It should be noted that in the proposed design the FFT is computed in an in-place style. Specifically, because only a single FFT hardware is allocated as the underlying computing fabric, each time only one tube fiber is read from the working memory and is transformed. After each time of FFT, the calculated frequency-domain representation is stored back to the addresses where the time-domain representation stores. Notice that as shown in Fig. 6, since the input tensor is real-valued, the transformed tube fiber in the frequency domain exhibits conjugate symmetry, thereby remaining the same memory requirement though the transformed tube fiber is complex-valued now. For instance, for the tube fiber [0 3 4 6], its frequency-domain representation contains a conjugate pair $(-4 + 3\mathrm{i}, \ -4 - 3\mathrm{i})$. Accordingly, for the physical data allocation, a specific memory address calculator is designed to ensure that the real and imaginary parts of each calculated complex number are stored in the adjacent memory banks. This well-designed address calculator also ensures that each complex number will be correctly reconstructed from multiple memory banks when demanded in the subsequent SVD operation.

Besides, to support different shapes of input tensor, the FFT hardware is designed in a reconfigurable way to be compatible with different shapes of input tensor. To facilitate this reconfigurable design, the FFT hardware is implemented in a purely serial way, where only one complex multiplier is allocated as the computing resource. Notice that because the SVD computation is the bottleneck for the overall computing procedure, such serial design on FFT does not affect the overall timing performance significantly.

C. The SVD Module

As shown in Fig. 5, the component unit of SVD module is SCU that performs SVD operation on one matrix (Line 3 in Algorithm 2). Consider the inherent parallelism existed in the multiple executions of Line 3; multiple SCUs are allocated in the SVD to decompose different matrices, namely the frontal slices of the input tensor, independently. Fig. 7 shows the inner architecture of each SCU unit. Here the SVD computation in the proposed SCU is based on the one-sided Jacobian SVD (JSVD) [23], which contains four computation steps. Accordingly, the proposed SCU contains four computing blocks, namely inner product unit (IPU), trigonometric unit (TRU), Rotator and post-processing unit (PPU).

Inner Product Unit (IPU). In the first step of SVD, three parameters, α, β and γ, need to be calculated for any two columns of the to-be-decomposed matrix $\boldsymbol{A}$ as follows:

$$\begin{aligned} \alpha &= \boldsymbol{A}_{:,i}^T \boldsymbol{A}_{:,i} \\ \beta &= \boldsymbol{A}_{:,j}^T \boldsymbol{A}_{:,j} \\ \gamma &= \boldsymbol{A}_{:,i}^T \boldsymbol{A}_{:,j}. \end{aligned} \tag{6}$$

To support such computation, IPU is used to calculate the inner product of two column vectors. Besides this major function, IPU is also used to determine whether the two column vectors are already orthogonal or not. If so, there is no need to use TRU and Rotator to make these two columns become orthogonal again. In general, such decision is based on the comparison between $|\gamma|$ and $\epsilon\sqrt{\alpha\beta}$, where ϵ is the error tolerance parameter that is used to control the tradeoff between decomposition accuracy and computational cost. In IPU, if

$|\gamma|$ is smaller than $\epsilon\sqrt{\alpha\beta}$, then it means that the current two columns read from memory banks are already orthogonal, and hence the computations in TRU and Rotator can be skipped.

Trigonometric Unit (TRU). When the values of γ do not satisfy the requirements of the aforementioned early stopping, TRU is used to perform the diagonalization of the two currently being processed columns. Specifically, the target of TRU is to find an angle θ which satisfies:

$$\begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}^T \begin{bmatrix} \alpha & \gamma \\ \gamma & \beta \end{bmatrix} \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} = \begin{bmatrix} d_{11} & \\ & d_{22} \end{bmatrix}. \tag{7}$$

According to [24], θ can be calculated as follows:

$$\begin{aligned} \rho &= \frac{\beta - \alpha}{2\gamma} \\ t &= \frac{\text{sign}(\rho)}{|\rho| + \sqrt{1+\rho^2}} \\ cos\theta &= \frac{1}{\sqrt{1+t^2}} \\ sin\theta &= t\cos\theta. \end{aligned} \tag{8}$$

In order to execute Eqn. (8), one multiplier and one divider are used in TRU to perform necessary arithmetic operations. More importantly, a CORDIC hardware [25] is used here for calculating the square root. Notice that though division and square rooting are computation-expensive operations, their corresponding latency is still relatively short as compared with the time-consuming matrix multiplication in the Rotator.

Rotator. After TRU sends the calculated $sin\theta$ and $cos\theta$ to the Rotator, the Rotator first constructs a rotation matrix $\boldsymbol{B}$ with rotation angle θ as follows:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{I} & & & \\ & cos\theta & & sin\theta \\ & & \boldsymbol{I} & \\ & -sin\theta & & cos\theta \\ & & & & \boldsymbol{I} \end{bmatrix}. \tag{9}$$

Then, we can rotate matrix $\boldsymbol{A}$ by simply multiplying rotation matrix $\boldsymbol{B}$:

$$\boldsymbol{A} = \boldsymbol{AB}. \tag{10}$$

Besides, Rotator also rotates one output matrix $\boldsymbol{V}$ with the similar way:

$$\boldsymbol{V} = \boldsymbol{VB}, \tag{11}$$

where $\boldsymbol{V}$ is initialized as identity matrix $\boldsymbol{I}$. Notice that here the matrix multiplication-based rotations for both $\boldsymbol{B}$ and $\boldsymbol{V}$ are performed for each calculated θ. Therefore, this procedure is very time consuming and is the most computation-intensive part of the entire t-SVD computation in Algorithm 2.

Post-Processing Unit (PPU). After the Rotator finishes the rotation of $\boldsymbol{A}$, the PPU utilizes the updated $\boldsymbol{A}$ to calculate the output matrices $\boldsymbol{S}$ and $\boldsymbol{U}$ as follows:

$$\begin{aligned} \boldsymbol{S}_{i,i} &= \|\boldsymbol{A}_{:,i}\|_2 \\ \boldsymbol{U}_{:,i} &= \boldsymbol{A}_{:,i}/\boldsymbol{S}_{i,i}, \end{aligned} \tag{12}$$

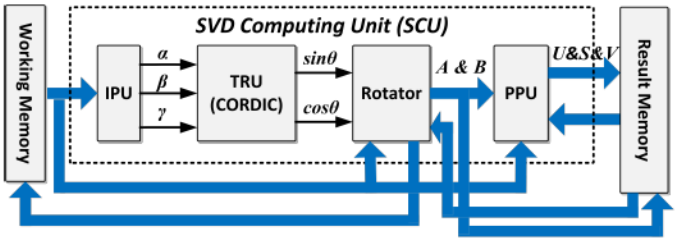


Fig. 7: The hardware architecture of SVD computing unit.

where $\boldsymbol{S}_{i,i}$ is the i-th diagonal element of diagonal matrix $\boldsymbol{S}$. Notice that as shown in Fig. 5 and Fig. 7, the output matrices $\boldsymbol{U}, \boldsymbol{S}$ and $\boldsymbol{V}$ are stored in the result memory consisting of individual memorys MEM_U, MEM_S and MEM_V, respectively. Consider that the calculation of these three matrices also needs the information of rotated matrix $\boldsymbol{A}$; PPU, similar to the Rotator, communicates with both working memory and result memory.

D. The IFFT Module

The IFFT module transforms the frequency-domain slices, in the format of matrices $\boldsymbol{U}, \boldsymbol{S}$ and $\boldsymbol{V}$, back to the time domain, and then outputs the three result tensors $\widetilde{\boldsymbol{\mathcal{U}}}$, $\widetilde{\boldsymbol{\mathcal{S}}}$, and $\widetilde{\boldsymbol{\mathcal{V}}}$. Here similar to Fig. 6, the IFFT module is a 1D version and performs on the tube fibers of the tensor. Also, consider the computational similarity between FFT and IFFT; the IFFT module has the same hardware architecture to the FFT module with extra components for input reordering and output normalization.

IV. Evaluation Results

Hardware Configuration. Based on the proposed hardware architecture, we design an example t-SVD hardware accelerator. This example design can support the input tensor $\boldsymbol{\mathcal{A}}$ with size up to 64×64×32. For the overall design, SRAM is the most resource-consuming part. Specifically, 32 4096×32 SRAMs are used to store the input tensor $\boldsymbol{\mathcal{A}}$. The output tensors $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$ are stored in 64 4096×32 SRAMs in a distributed way. In addition, since $\boldsymbol{S}$, as the frontal slice of the output tensor $\boldsymbol{\mathcal{S}}$, is a diagonal matrix, the entire $\boldsymbol{\mathcal{S}}$ is stored into a smaller SRAM with size 2048×32. Therefore, with 32-bit fixed-point quantization scheme, the entire SRAM consumption is 1.51MB.

EDA Tools and Synthesis Results. Based on the adopted hardware configuration, we then develop an RTL model and synthesize it at 1GHz clock frequency using Synopsys Design Compiler with CMOS 28nm library to obtain the area and power reports. In overall, the area consumption and power consumption of the entire design are 3.47 mm^2 and 319.4 mW, respectively.

Comparison with Software Implementation. We further investigate the potential acceleration of the proposed hardware design over software implementation. Here the baseline software implementation is based on the optimized MATLAB t-SVD code. For fair comparison, we use many MATLAB internal functions to maximize the speed of software solution.

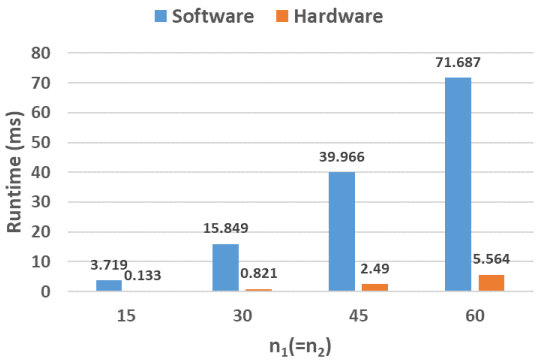


Fig. 8: The runtime comparison between software and hardware implementations when fixing $n_3 = 32$.

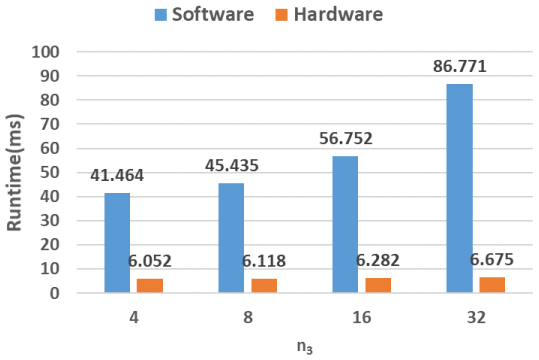


Fig. 9: The runtime comparison between software and hardware implementations when fixing $n_1 = n_2 = 64$.

For instance, for SVD computation we use MATLAB built-in function instead of our own Jacobian-based SVD code, which is 10 times slower than MATLAB's official optimized function. In overall, the entire MATLAB implementation runs on the computer equipped with 22nm 2.5GHz Intel Core i7-4710HQ CPU with 6MB cache and 8GB main memory.

Fig. 8 and Fig. 9 show the comparisons between the real-world runtime for MATLAB implementation and estimated runtime for our hardware accelerator with different n_1, n_2 and n_3. Here without loss of generality we set $n_1 = n_2$ in our experiments. From these two figures it is seen that the hardware accelerator is estimated to provide a significant speedup ($7\times \sim 28\times$) over software implementation on different t-SVD tasks. In average, the proposed hardware accelerator is expected to provide around $14\times$ speedup as compared with the CPU-based implementation.

V. Conclusion

This paper proposes a high-performance hardware architecture for t-SVD. The design consideration of each key computation component is discussed and analyzed. Compared with CPU-based software implementation, the proposed hardware architecture is expected to provide average $14\times$ speedup on different shapes of input tensor.

References

[1] P. Koniusz, A. Cherian, and F. Porikli, "Tensor representations via kernel linearization for action recognition from 3d skeletons," in *European Conference on Computer Vision*. Springer, 2016.

[2] C. Jia and Y. Fu, "Low-rank tensor subspace learning for rgb-d action recognition," *IEEE Transactions on Image Processing*, vol. 25, 2016.

[3] D. Tao, Y. Guo, Y. Li, and X. Gao, "Tensor rank preserving discriminant analysis for facial recognition," *IEEE Transactions on Image Processing*, vol. 27, 2017.

[4] S. Li, W. Wang, H. Qi, B. Ayhan, C. Kwan, and S. Vance, "Low-rank tensor decomposition based anomaly detection for hyperspectral imagery," in *International Conference on Image Processing (ICIP)*. IEEE, 2015.

[5] Y. Hu, X. Yi, and L. S. Davis, "Collaborative fashion recommendation: A functional tensor factorization approach," in *Proceedings of the 23rd ACM International Conference on Multimedia*. ACM, 2015.

[6] F. Jiang, X.-Y. Liu, H. Lu, and R. Shen, "Efficient multi-dimensional tensor sparse coding using t-linear combination," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[7] S. Liao, X.-Y. Liu, F. Qian, M. Yin, and G.-M. Hu, "Tensor super-resolution for seismic data," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.

[8] L.-H. Lim and P. Comon, "Multiarray signal processing: Tensor decomposition meets compressed sensing," *Comptes Rendus Mecanique*, vol. 338, 2010.

[9] R. A. Harshman *et al.*, "Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis," 1970.

[10] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition," *Psychometrika*, vol. 35, 1970.

[11] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, 1966.

[12] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, 2011.

[13] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor ring decomposition," *arXiv preprint arXiv:1606.05535*, 2016.

[14] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[15] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[16] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, 2015.

[17] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[18] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR, 2017.

[19] M. E. Kilmer and C. D. Martin, "Factorization strategies for third-order tensors," *Linear Algebra and its Applications*, vol. 435, 2011.

[20] M. E. Kilmer, K. Braman, N. Hao, and R. C. Hoover, "Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, 2013.

[21] Z. Zhang, G. Ely, S. Aeron, N. Hao, and M. Kilmer, "Novel methods for multilinear data completion and de-noising based on tensor-svd," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[22] X.-Y. Liu, S. Aeron, V. Aggarwal, X. Wang, and M.-Y. Wu, "Adaptive sampling of rf fingerprints for fine-grained indoor localization," *IEEE Transactions on Mobile Computing*, vol. 15, 2015.

[23] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "The singular value decomposition: Anatomy of optimizing an algorithm for extreme scale," *SIAM Review*, vol. 60, 2018.

[24] R. P. Brent, F. T. Luk, and C. Van Loan, "Computation of the singular value decomposition using mesh-connected processors," Cornell University, Tech. Rep., 1982.

[25] J. R. Cavallaro and F. T. Luk, "Cordic arithmetic for an svd processor," *Journal of Parallel and Distributed Computing*, vol. 5, 1988.