REDUCED-COMPLEXITY SINGULAR VALUE DECOMPOSITION FOR TUCKER DECOMPOSITION: ALGORITHM AND HARDWARE

Xiaofeng Hu, Chunhua Deng, Bo Yuan

Department of Electrical and Computer Engineering, Rutgers University

ABSTRACT

Tensors, as the multidimensional generalization of matrices, are naturally suited for representing and processing high-dimensional data. To date, tensors have been widely adopted in various data-intensive applications, such as machine learning and big data analysis. However, due to the inherent large-size characteristics of tensors, tensor algorithms, as the approaches that synthesize, transform or decompose tensors, are very computation and storage expensive, thereby hindering the potential further adoptions of tensors in many application scenarios, especially on the resource-constrained hardware platforms. In this paper, we propose a reduced-complexity SVD (Singular Vector Decomposition) scheme, which serves as the key operation in Tucker decomposition. By using iterative self-multiplication, the proposed scheme can significantly reduce the storage and computational costs of SVD, thereby reducing the complexity of the overall process. Then, corresponding hardware architecture is developed with 28nm CMOS technology. Our synthesized design can achieve 102GOPS with 1.09 mm^2 area and 37.6 mW power consumption, and thereby providing a promising solution for accelerating Tucker decomposition.

Index Terms— Tucker Decomposition, SVD, hardware architecture

1. INTRODUCTION

Tensors, as the multi-dimension generalization of matrices, are the naturally suitable and powerful mathematical tool for representing and processing high dimension data. In recent years, tensor-based computation has been widely applied in various high dimension data-demanded domains, such as signal processing [1], linear algebra [2], data mining [3], computer vision [4], and machine learning [5].

However, the superiority of tensor computation greatly depends on the high-complexity tensor algorithms. Tensor algorithms are the approaches that synthesize, transform or decompose tensors as desired. Due to the inherent large-size characteristics of tensors, even the simplest tensor algorithm, such as tensor multiplication, are very storage expensive and computation expensive, thereby limiting the potential further adoptions of tensors in many resource-constrained applications.

To address the above challenge, there are two orthogonal directions that can be explored. First, at the algorithm level, low-complexity tensor algorithms, if possible, would bring a very significant algorithm-level reduction in both space and computational complexity for tensor algorithms. Second, at the hardware level, customized hardware design of tensor algorithms, if being properly implemented, would also provide unprecedented acceleration as compared to the state-of-the-art CPU/GPU implementations [6].

Following these two directions, this paper proposes to achieve high-performance Tucker decomposition [7], as a fundamental and popular tensor decomposition approach, using an algorithm/hardware co-design way. Specifically, we first propose a new algorithm to reduce the complexity of SVD, which is the key operation in Tucker decomposition. The proposed algorithm can significantly reduce the storage costs (at least 250 times) and computational costs (at least 8 times) of SVD, and hence it also brings a huge performance increase for the overall Tucker decomposition, especially when the operated tensor has high order. Then, based on this algorithm, we further develop the corresponding hardware architecture to accelerate Tucker decomposition. With 28nm CMOS technology, our synthesized design can achieve 102GOPS with 1.09 mm^2 area and 37.6 mW power consumption, and thereby providing a promising solution for implementing Tucker decomposition.

The rest of this paper will be organized as follows: Section 2 introduces the background information of tensor and Tucker decomposition. The proposed reduced-complexity SVD algorithm is described in Section 3. Section 4 presents the performance analysis of the proposed SVD algorithm. The hardware architecture and evaluation results are presented in Section 5. Section 6 draws the conclusions.

2. BACKGROUND

2.1. Tensor and Related Notations

Tensor, or multidimensional data array, is the natural generalization of one-mode vector and two-mode matrix. In this paper, we use boldface calligraphic letter to represent tensor.

For instance, $\mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ denotes one three-mode tensor of size $I_1 \times I_2 \times I_3$. Slice is defined as a two-dimensional matrix obtained from fixing all indices but two, and Fiber is one-dimensional array by fixing all indices but one. We use boldface capital letters, for example, $\mathbf{B_{i::}}$, to denote Slice, and lower-case boldface letters to represent Fiber like $\mathbf{b_{ij:}}$.

2.2. Unfold ,TTM and Tensor norm

Unfolding is the process to reorder the elements of a tensor into one matrix. Fig.1 shows an example of unfolding three-mode tensor. In general, the mode-d unfolding of a tensor $\mathcal{B}$ in $\mathbb{R}^{I_1 \times I_2 \ldots \times I_k}$ is a matrix $\mathbf{B}^{(d)}$ of size $I_d \times (I_1 \cdot I_2 \ldots I_{d-1} \cdot I_{d+1} \ldots I_k)$. And each tensor element $(i_1, \ldots i_{d-1}, i_d, i_{d+1} \ldots i_k)$ maps the matrix element (i_d, j) as

$$j = 1 + \sum_{q \neq d}^{k} \left((i_q - 1) \prod_{n \neq d}^{q-1} I_n \right). \qquad (1)$$

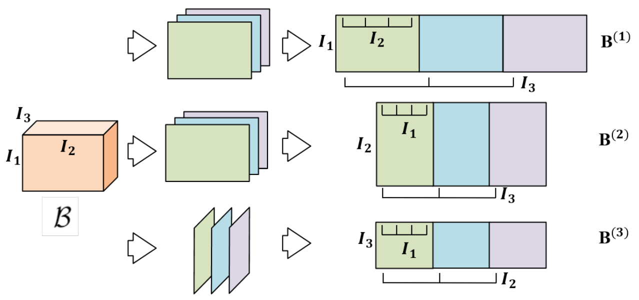


Fig. 1. Unfolding of a three-mode tensor.

Tensor times matrix (TTM) is the high-order extent of matrix multiplication. In general, the d-mode product of TTM between a given tensor $\mathcal{B} \in \mathbb{R}^{I_1 \ldots \times I_d \ldots \times I_k}$ and a matrix $\mathbf{X} \in \mathbb{R}^{J \times I_d}$ is a new tensor as

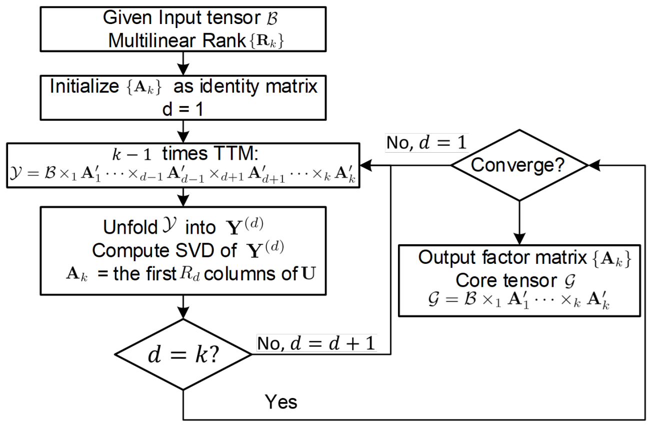
$$\mathcal{Y} = \mathcal{B} \times_d \mathbf{X} \in \mathbb{R}^{I_1 \ldots I_{d-1} \times J \ldots \times I_k} \Longleftrightarrow \mathbf{Y}^{(d)} = \mathbf{X}\mathbf{B}^{(d)}. \quad (2)$$

The procedure of TTM requires the aforementioned unfolding operation. For instance, assume we have one tensor $\mathcal{B} \in \mathbb{R}^{3 \times 2 \times 2}$ and one matrix $\mathbf{X} \in \mathbb{R}^{2 \times 3}$:

$$\mathbf{B}(:,:,1) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \mathbf{B}(:,:,2) = \begin{bmatrix} 2 & 5 \\ 4 & 3 \\ 6 & 1 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & 3 & 5 \\ 6 & 4 & 2 \end{bmatrix},$$

The TTM between them can be achieved by first unfolding $\mathcal{B}$ into $\mathbf{B}^{(1)}$:

$$\mathbf{B}^{(1)} = \begin{bmatrix} 1 & 2 & 2 & 5 \\ 3 & 4 & 4 & 3 \\ 5 & 6 & 6 & 1 \end{bmatrix}.$$



Given Input tensor $\mathcal{B}$
Multilinear Rank $\{\mathbf{R}_k\}$
Initialize $\{\mathbf{A}_k\}$ as identity matrix
d = 1
$k-1$ times TTM:
$\mathcal{Y} = \mathcal{B} \times_1 \mathbf{A}_1' \cdots \times_{d-1} \mathbf{A}_{d-1}' \times_{d+1} \mathbf{A}_{d+1}' \cdots \times_k \mathbf{A}_k'$
Unfold $\mathcal{Y}$ into $\mathbf{Y}^{(d)}$
Compute SVD of $\mathbf{Y}^{(d)}$
$\mathbf{A}_k$ = the first R_d columns of $\mathbf{U}$
$d = k?$
No, $d = d + 1$
Yes
Converge?
No, $d = 1$
Output factor matrix $\{\mathbf{A}_k\}$
Core tensor $\mathcal{G}$
$\mathcal{G} = \mathcal{B} \times_1 \mathbf{A}_1' \cdots \times_k \mathbf{A}_k'$

Fig. 2. The computation flowchart of the HOOI.

and then the one-mode TTM product $\mathcal{Y} = \mathcal{B} \times_1 \mathbf{X} \Longleftrightarrow \mathbf{X}\mathbf{B}^{(1)}$ is as follows:

$$\mathbf{Y}(:,:,1) = \begin{bmatrix} 35 & 44 \\ 28 & 40 \end{bmatrix}, \mathbf{Y}(:,:,2) = \begin{bmatrix} 44 & 19 \\ 40 & 44 \end{bmatrix}.$$

Tensor norm is a natural extent of matrix or vector norm. The norm of a tensor $\mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \ldots \times I_k}$ is typically the Frobenius norm defined below:

$$\|\mathcal{B}\| = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \ldots \sum_{i_k=1}^{I_k} b_{i_1 i_2 \ldots i_k}^2} \qquad (3)$$

2.3. Tucker Decomposition

Tucker decomposition approximates an original tensor $\mathcal{B}$ as the multiplication between a small core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \ldots \times R_k}$ and a set of matrices $\mathbf{X}_k$ in each mode:

$$\mathcal{B} \approx \mathcal{G} \times_1 \mathbf{X}_1 \times_2 \mathbf{X}_2 \ldots \times_k \mathbf{X}_k, \qquad (4)$$

where $\mathbf{X_k} \in \mathbb{R}^{R_k \times I_k}$ is the factor matrix, which is usually orthogonal. R_k is the number of components (columns) in each factor matrix, also called multilinear rank of $\mathcal{B}$.

In general there are two major implementation methods for Tucker decomposition: the high-order SVD (HOSVD) [2], and the High Order Orthogonal Iteration (HOOI) [8]. Since the HOSVD is not optimal in giving the best fit of data [9], we focus on the HOOI in this paper. The computation flowchart of the HOOI is presented in Fig. 2.

3. REDUCED-COMPLEXITY SVD ALGORITHM

As shown in Fig. 2, the process of HOOI consists of two main steps: $k-1$ times TTMs and one Matrix SVD. Since the implementation of TTM is usually straightforward, computing SVD is the bottleneck in Tucker decomposition. In this section, inspired by the idea in Andersson[10] and Zhan[11]

Algorithm 1 Reduced-complexity SVD for Tucker Decomposition

```
Input: matrix A, leading vector number N, iteration times T
Output: leading left singular vector U
1: $\mathbf{P}_1 = \mathbf{A}\mathbf{A}'$
2: while $i < N$ do
3:     while $j < T$ do
4:         $\mathbf{P}_i^j = k \cdot \mathbf{P_i}^{j-1} \times \mathbf{P}_i^{j-1}$
5:     end while
6:     $U(:,i) \approx \dfrac{\mathbf{P^i}(:,1)}{\|\mathbf{P^i}(:,1)\|}$
7:     $\mathbf{P}_{i+1} = (\mathbf{I}_{N_t} - U_{:,i}U_{:,i}^{'})\mathbf{P}_i$
8: end while
9: return U
```

that computing only the dominant singular vectors instead of the whole SVD, we propose a reduced-complexity SVD algorithm.

Specifically, different from prior work that utilizes Gram-Schmidt process [11] or Givens rotation [12][13] to guarantee the orthogonal property of each vector, we derive perfect orthogonal singular vector by the simple matrix multiplication. Algorithm 1 describes the general procedure of our approach. Here the key idea is to only compute matrix multiplication, instead of computing singular values, right singular vector matrix and orthogonalization process, to derive leading left singular vectors, which will be used in other steps of HOOI later. Specifically, after computing the first vector u_1, the correlated component of u_1 in $\mathbf{P}_1$: $\sigma_1^2 u_1 u_1'$ should be eliminated to derive next component u_2. Since the orthogonal vector has two following properties:

$$\sum_{i=1}^{N_t} u_i u_i' = \mathbf{I}_{N_t} \tag{5}$$

$$u_i u_j' = 0, i \neq j. \tag{6}$$

The elimination process (Line7 in Algorithm 1) can be performed as below

$$\mathbf{P}_2^0 = (\mathbf{I}_{N_t} - u_1 u_1')\mathbf{P}_1^0. \tag{7}$$

Then we can apply the same process to derive all the needed leading vectors.

Notice that at the initialization stage the initial matrix $\mathbf{P}_1$ is:

$$\mathbf{P}_1 = \mathbf{A}\mathbf{A}' = k \cdot \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}' = k \cdot \sum_{i=1}^{N_t} \sigma_i^2 u_i u_i' \tag{8}$$

where k is the arbitrary non-zero coefficient and u_i represents the i-th column of $\mathbf{U}$. In order to avoid overflow, we can normalize all the matrix's elements by k, and it can be easily implemented in hardware by right shift. Since the exact value of every element on intermediate matrix is not needed at the whole computing process, this normalization can be executed in every multiplication iteration without jeopardizing the overall accuracy but alleviating the burden of memory.

Compared with the traditional methods such as Jacobi rotation [13], our proposed approach has unique benefits that it eliminates the computation and buffer of multiple large intermediate matrix like $\mathbf{B}^{(d)} \in \mathbb{R}^{I_k \times \prod_{i \neq k} R_i}$ and $\mathbf{U}^{(d)} \in \mathbb{R}^{I_k \times I_k}$, which is storage-intensive and computing-intensive. Instead, we only need to buffer and iteratively self-multiply one small intermediate matrix $\mathbf{P} = \mathbf{B}^{(d)'} \times \mathbf{B}^{(d)} \in \mathbb{R}^{I_k \times I_k}$ during the entire procedure. Consequently, the required memory cost as well as computational costs can be significantly reduced, especially for large-mode tensor.

4. SVD PERFORMANCE ANALYSIS

In this section we evaluate and analyze the performance of our proposed SVD algorithm. Specifically, convergence speed, computational cost and memory cost will be discussed.

4.1. Convergence Speed

We set up our experiment in MATLAB R2018b platform, considering a three-mode tensor generated by Gaussian distribution and corrupted with noise varying 10 percent of the tensor element. We compare our method, both in floating point and fixed point precision with the popular parallel single-side Jacobi rotation method [12] in double-floating point precision. The experiment results are shown in Fig.3. It is seen that our proposed approach has much faster convergence speed than single-sided Jacobi rotation method. This verifies their theoretical difference: our method can converge at the speed of $O\left(|\frac{\sigma_i}{\sigma_{i+1}}|^{2^n}\right)$; while single-side Jacobi rotation method only has quadratic convergence speed [9].

4.2. Computational and Memory Costs Analysis

Computational Cost. The computational cost analyzed here is measured in term of multiplication-accumulation operation.

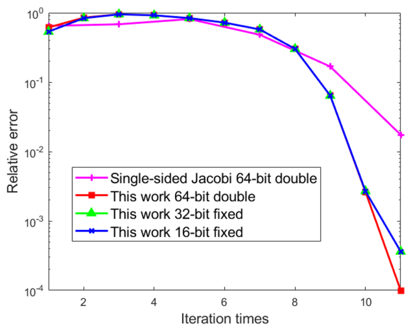


Fig. 3. Comparison on convergence speed.

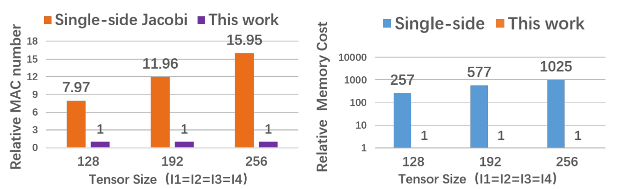


Fig. 4. Comparison on computational and memory costs. Left: Computational cost. Right: Memory cost.

For the single-sided Jacobi rotation method, in each SVD iteration, $I_k(I_k-1)/2$ pairs of orthogonalizations will be performed, and each orthogonalization processes $(I_k+\prod_{i\neq k} R_i)$ elements, so the entire SVD in this case takes roughly:

$$T_{jac} = I_k(I_k-1)(\frac{I_k+\prod_{i\neq k} R_i}{p}) \times T_{ite} \tag{9}$$

operations, where p is the degree of parallelism, and T_{ite} is the number of SVD iterations with a typical value of 15 to get acceptable relative error.

For our approach, since we only need to derive R_k leading vectors, therefore the whole SVD takes roughly

$$T_{mul} = R_k \times (\frac{I_k^3}{p}) \times T_{ite} \tag{10}$$

operations, which is much fewer than the single-sided Jacobi rotation method (Eqn. (10) vs Eqn. (9)).

Memory Cost. For single-sided Jacobi rotation method, it requires large buffer to store at least two large intermediate matrices: $\mathbf{B}^{(d)}$ of size $I_k \times \prod_{i\neq k} R_i$ and $\mathbf{U}$ of size $I_k \times I_k$. On the other hand, our approach only needs to buffer one small intermediate matrix $\mathbf{P}$ of size $I_k \times I_k$. Consequently, by applying iterative self-multiplication strategy, the required buffer size to store intermediate matrix is significantly reduced.

The above analysis is further evaluated on different sizes of tensors. We assume the decomposition rate I_k/R_k is fixed $4:1$, and the degree of parallelism p is set as 1 all the time. Fig.4 shows the normalized computational and memory costs compared with the single-sided Jacobi rotation method. Here our approach is normalized as 1. From this figure it can be seen that our proposed approach achieves 8.5-17$\times$ reduction in computational cost and 257-1025$\times$ reduction in memory costs.

5. HARDWARE ARCHITECTURE AND PERFORMANCE

Based on the HOOI algorithm for Tucker decomposition in Fig.2 and reduced-complexity SVD in Algorithm 1, we develop the hardware architecture for Tucker decomposition engine (TDE). Fig.5 shows the overall hardware architecture of the proposed design. Here the main part of the computing engine is the matrix matrix multiplication engine (MMME). This is because considering the main computation of Tucker decomposition is TTM and SVD and both of them can be implemented by matrix matrix multiplication. In our design these two types of computations are mapped on the same hardware resources. Specifically, here the MMME is configured with 32 $\times$ 4 multiplier-accumulator (MAC) unit. Such arrangement allows this architecture to support different kinds of configurations to fully utilize the MAC computing resources. The data movement of all the MACs is orchestrated by a main controller. Regarding dataflow, the MMME receives row data from the matrix SRAM, and it receives column data from the tensor SRAM. To reduce the DRAM bandwidth requirement and save the power, we configure the tensor SRAM as 512 KB and the matrix SRAM as 64 KB to ensure that most of the data can be accessed from SRAM. Besides the MMME, each column of tensor SRAM has a tensor norm calculator (TNC) for the use of Line6 in Algorithm1.

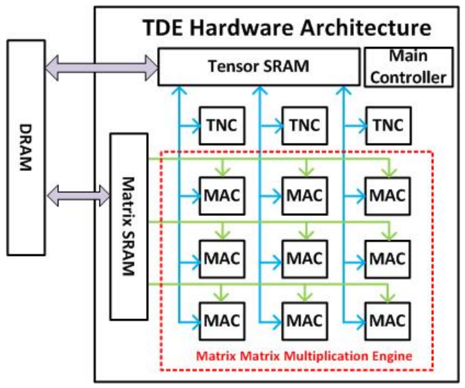


Fig. 5. Hardware architecture of Tucker decomposition engine.

The proposed architecture is developed and synthesized with 28nm CMOS technology. The overall area is is 1.09 mm^2 with power consumption as 37.6 mW. Under clocking frequency as 400MHz, the TDE can provide 102GOPS computing power, which is very promising to accelerate Tucker decomposition procedure.

6. CONCLUSION

In this paper, we propose a reduced-complexity SVD algorithm for Tucker decomposition. Compared with the conventional approach, our proposed method has much less computational and memory costs. Based on this method, a hardware architecture for Tucker decomposition is developed. Evaluation results show that the hardware design is very attractive for Tucker decomposition acceleration.

7. REFERENCES

[1] Lieven De Lathauwer and Bart De Moor, "From matrix to tensor: Multilinear algebra and signal processing," in *Institute of Mathematics and Its Applications Conference Series*. Citeseer, 1998, vol. 67, pp. 1–16.

[2] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[3] Volker Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[4] M Alex O Vasilescu and Demetri Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," in *European Conference on Computer Vision*. Springer, 2002, pp. 447–460.

[5] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[6] Brett W. Bader and Tamara G. Kolda, "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping," *ACM Transactions on Mathematical Software*, vol. 32, no. 4, pp. 635–653, Dec. 2006.

[7] Ledyard R Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[8] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle, "On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[9] Tamara G Kolda and Brett W Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[10] Claus A Andersson and Rasmus Bro, "Improving the speed of multi-way algorithms:: Part i. tucker3," *Chemometrics and intelligent laboratory systems*, vol. 42, no. 1-2, pp. 93–103, 1998.

[11] Cheng-Zhou Zhan, Yen-Liang Chen, and An-Yeu Wu, "Iterative superlinear-convergence svd beamforming algorithm and vlsi architecture for mimo-ofdm systems," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 3264–3277, 2012.

[12] James Demmel and Krešimir Veselić, "Jacobis method is more accurate than qr," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 4, pp. 1204–1245, 1992.

[13] Richard P Brent and Franklin T Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 1, pp. 69–84, 1985.