VLSI Hardware Architecture of Stochastic Low-rank Tensor Decomposition

Lingyi Huang[a], Chunhua Deng[a], Shahana Ibrahim[b], Xiao Fu[b], and Bo Yuan[a]

[a]Department of Electrical and Computer Engineering, Rutgers University
[b]School of Electrical Engineering and Computer Science, Oregon State University

***Abstract*—Canonical polyadic decomposition (CPD) is one of the most popular tensor decomposition approaches used in various practical applications. Recently, AdaCPD, a doubly randomization-based reduced-complexity CPD, is proposed with exhibiting high decomposition performance. However, facing the similar challenges of all the other dense tensor-oriented decomposition methods, AdaCPD is still computation-intensive and thus poses severe challenges for practical deployment on resource-constrained platforms and real-time applications. In this paper, we propose an efficient hardware architecture for AdaCPD accelerator. With customized hardware design and memory optimization at the architecture level, the proposed AdaCPD hardware accelerator enables significant performance improvement than conventional software implementation. Evaluation results show that this customized hardware design only occupies 1.6 mm^2 area and consumes 300 mW power with CMOS 28nm technology, thereby bringing 6762 times and 1483 times reduction over the software implementation on Intel Core i9-9900K CPU in terms of area and energy efficiency, respectively.**

***Index Terms*—VLSI, hardware architecture, Large-scale tensor decomposition**

I. Introduction

Tensor decomposition, as a powerful mathematical tool that targets to explore the inherent low-rank characteristics of data, is a key technique for large-scale data processing. Among various existing tensor decomposition approaches, canonical polyadic decomposition (CPD) [1] is a very popular solution that has been widely used in many applications, such as image processing [2] and topic modeling [3]. Recently, a Block-Randomized CPD algorithm with adaptive step size, namely *AdaCPD*, has been proposed in [4]. By combining randomized block coordination descent and stochastic proximal gradient, AdaCPD enjoys simultaneous reduction in both computational cost and memory footprint.

Although AdaCPD already exhibits reduced complexity and high accuracy as compared to the conventional decomposition methods, from the perspective of practical implementation, AdaCPD is still very computation-intensive; and hence it is very challenging to deploy this technique in the time-constrained power-constrained platforms, especially for those real-time embedded applications.

To address this challenge, in this work we develop a customized hardware architecture to accelerate the execution of AdaCPD algorithm in an energy-efficient way. To be specific, in order to alleviate the performance bottleneck caused by the memory bandwidth, we propose a novel tensor storage scheme to transform random DRAM accesses to regular sequential accesses. Case study shows that our proposed hardware design achieves thousands times improvement on area and energy efficiency over the CPU-based software solution.

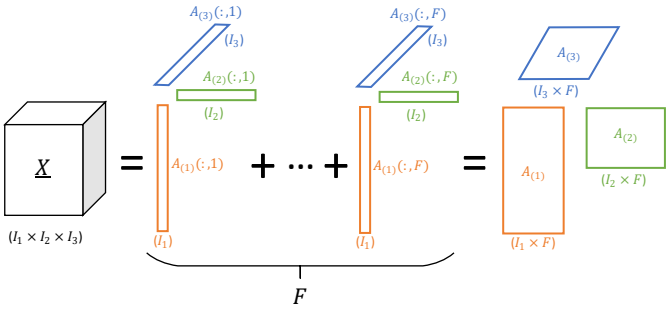

Fig. 1. Canonical Polyadic Decomposition (CPD).

The rest of the paper is organized as follows. Section II gives a brief introduction of tensor decomposition, CPD and AdaCPD algorithm. The proposed hardware architecture and the overall dataflow are presented in Section III. Section IV describes the proposed tensor storage scheme. The experimental results are reported in Section V, and Section VI draws the conclusions.

II. Algorithm Background

A. Tensors and Notations

Tensors are multidimensional arrays, and each dimension has its own coordinate system. More formally, an N-th order tensor is an element of the tensor product of N vector space [1]. Under this setting, vectors are essentially first-order tensors, which are denoted by boldface lower-case letters, (e.g. x); matrices are essentially second-order tensor, which are denoted by boldface capital letters, (e.g. X). For the high-dimensional tensor whose order is greater or equal to 3, we use boldface capital letters with underline to denote, (e.g. $\underline{\boldsymbol{X}}$).

B. Canonical Polyadic Decomposition (CPD)

CPD is one of the most popularly used tensor decomposition methods. In general, CPD aims to represent an order-N tensor

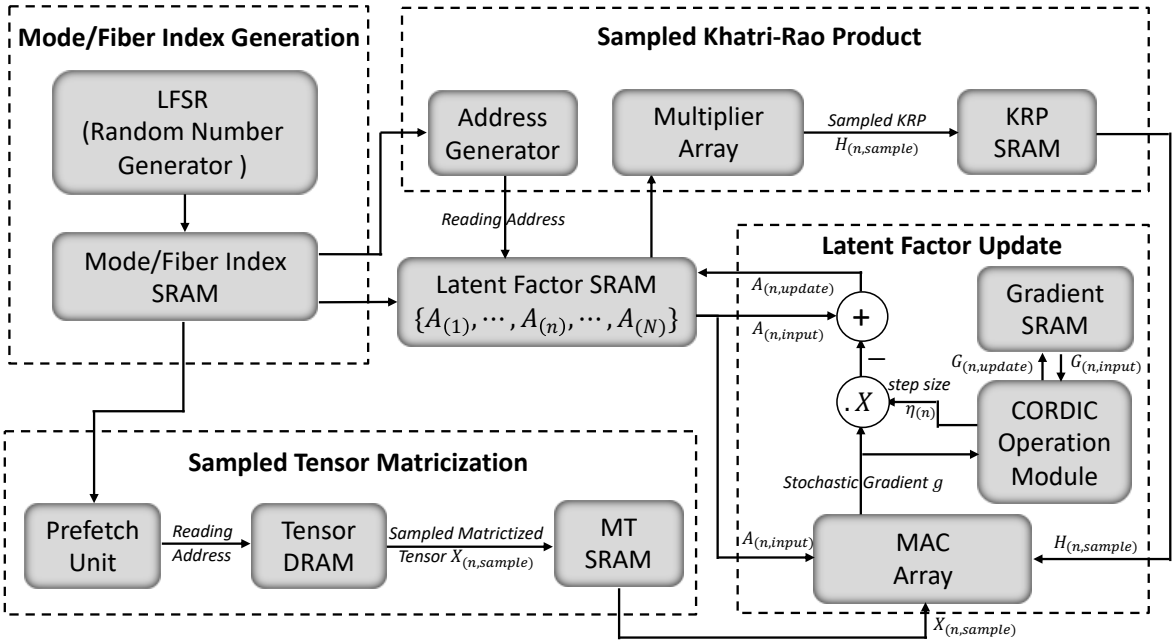

Fig. 2. The overall architecture of AdaCPD hardware accelerator.

$\underline{\boldsymbol{X}}$, which has a size of $I_1 \times I_2 \times .... \times I_N$, as the sum of rank-one components:

$$\underline{\boldsymbol{X}} = \sum_{f=1}^{F} A_{(1)}(:,f) \circ A_{(2)}(:,f) \circ ... \circ A_{(N)}(:,f), \quad (1)$$

where "$\circ$" denotes the outer product, $A_{(n)}$ is an $I_n \times F$ matrix as *mode-n latent factor*, and F is the minimal integer that satisfies Eq. (1). Fig. 1 shows an example of the CP-format order-3 tensor $\underline{\boldsymbol{X}}$ represented with three matrices (order-2 tensors) $\{\mathrm{A}_{(1)}, A_{(2)}, A_{(3)}\}$.

C. Tensor Matricization

The tensor matricization operation, or matrix unfolding of a tensor, is very important for designing tensor factorization algorithms [4]. Essentially, the tensor matricization reorders the elements of the original tensor into a matrix. In this paper, we are mainly interested in the mode-n tensor matricization operation as:

$$\underline{\boldsymbol{X}}(i_1,...,i_N) = X_{(n)}(j,i_n), \quad (2)$$

where $\underline{\boldsymbol{X}}(i_1,...,i_N)$ is the original N-th order tensor, $X_{(n)}$ is a $J_n \times I_n$ matrix defined as the matricized tensor, where $j = 1+\sum_{k=1,k\neq n}^{N}(i_k-1)J_k$, and $J_k = \prod_{m=1,m\neq n}^{k-1} I_m$ [1]. Then, Eq. 1 can be reformulated as:

$$X_{(n)} = H_{(n)}A^{\top}. \quad (3)$$

Notice that here $H_{(n)}$, as a $J_n \times F$ matrix, is defined as:

$$H_{(n)} = A_{(1)} \odot A_{(n-1)} \odot A(n+1) \odot ... \odot A_{(N)} = \odot_{i=1,i\neq n}^{N} A_{(i)}, \quad (4)$$

where $\odot$ denotes Khatri-Rao product.

D. AdaCPD Algorithm

AdaCPD is a Block-Randomized CPD algorithm with adaptive step size. In general, the key idea of AdaCPD is to update the latent factors $A_{(n)}$ using a doubly stochastic procedure instead of working with the entire dataset. To be specific, at the beginning of the r-th iteration, AdaCPD randomly samples a mode index n, where $1 \leq n \leq N$, and a set of mode-n fibers indices $\mathcal{F}_n$. Then the gradient $G^{(r)} \in \mathbb{R}^{(I_1+...+I_N)\times F}$ can be estimated as:

$$G^{(r)} = [(G_{(1)}^{(r)})^T, ..., (G_{(N)}^{(r)})^T]^T, \quad (5)$$

where:

$$G_{(n)}^{(r)} = \frac{1}{|\mathcal{F}_n|}(A_{(n)}^{(r)} H_{(n)}^T(\mathcal{F}_n,:)H_{(n)}(\mathcal{F}_n,:) - X_{(n)}^T(\mathcal{F}_n,:)H_{(n)}(\mathcal{F}_n,:)). \quad (6)$$

where $X_{(n)}(\mathcal{F}_n,:)$ is defined as the sampled matricized tensor $X_{(n,sample)}$, and $H_{(n)}(\mathcal{F}_n,:)$ is defined as the sampled Khartri-Rao Product $H_{(n,sample)}$.

To determine the step size $\eta_{(n)}^{(r)}$ of the current iteration, Adagrad strategy [5] is used as:

$$\eta_{(n)}^{(r)} \leftarrow \frac{\eta}{(\sum_{t=1}^{r}[G_{(n)}^{(r)}]^2)^{1/2}}. \quad (7)$$

Then, the corresponding latent factor $A_{(n)}$ can be updated as:

$$A_{(n)}^{(r+1)} \leftarrow Prox_{h_n}(A_{(n)}^{(r)} - \eta_{(n)}^{(r)} \odot G_{(n)}^{(r)}), \quad (8)$$

where

$$A_{(n')}^{(r+1)} \leftarrow A_{(n')}^{(r)}, n' \neq n. \quad (9)$$

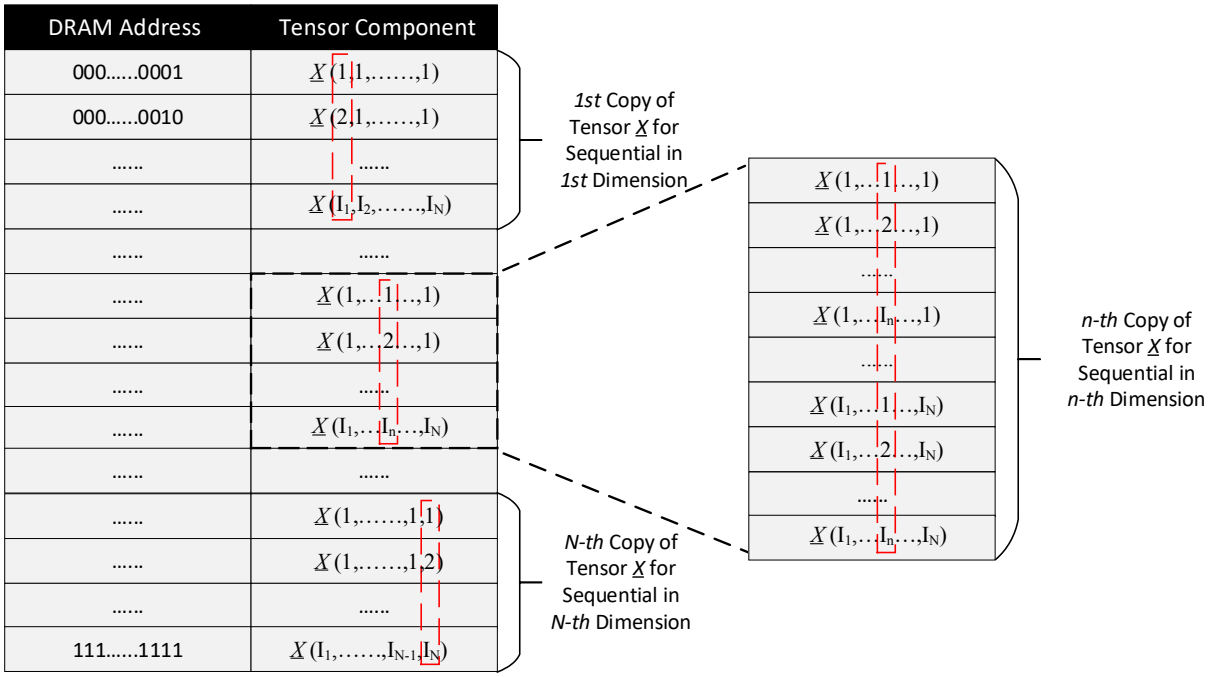

Fig. 3. Proposed tensor storage scheme in DRAM.

III. Hardware Architecture Design

In this section we describes the customized hardware architecture that can accelerate the execution of AdaCPD in an energy-efficient way. To be specific, in order to perform efficient mapping from algorithm to hardware, we first generalizes the computing steps of AdaCPD algorithm to four main operations:

- *Mode/Fiber Index Generation*
- *Sampled Tensor Matricization*
- *Sampled Khartri-Rao Product*
- *Latent Factor Update*

Execution Order. Fig. 2 shows the overall hardware architecture of our proposed AdaCPD hardware, which maps the corresponding four main operations and performs the entire execution based on the following scheme: The *Mode/Fiber Index Generation* is executed first. After that, the two operations, the *Sampled Tensor Matricization* and the *Sampled Khartri-Rao Product*, are simultaneously performed. Then, the *Latent Factor Update* is operated, and the results are sent back to latent factor SRAM for the use of the next iteration.

Overall Dataflow. At the beginning of each iteration, a linear-feedback shift register (LSFR) generates sets of random numbers as the mode index n and the fiber indices $\mathcal{F}_n$, and then it sends them into the Mode/Fiber Index SRAM. Specifically, for the N-th order tensor, we need one mode index and N fiber indices for each iteration. Here the mode index n is a single integer, which is randomly sampled from 1 to N, and each fiber index $\mathcal{F}_n$ is a vector with the length of J_n, in which each entry is bounded by I_n.

Once all the desired indices are obtained, the address generator and the prefetch unit will produce the corresponding reading addresses that will be used for the sampled Khatri-Rao product operation and the sampled tensor matricization operation. To be specific, with the generated reading address from the prefetch unit, the sampled matricized tensor (MT), $X_{(n,sample)}$, will be loaded row by row from the off-chip meomory, the Tensor DRAM, and then they are stored to the corresponding on-chip memory, the MT SRAM. Meanwhile, another on-chip memory, the Latent Factor SRAM, will be accessed to obtain the sampled rows of the latent factors, which will be later sent to the multiplier array to perform element-wise multiplication. The calculated sampled Khatri-Rao product (KRP), $H_{(n,sample)}$, is stored at the KRP SRAM. Finally, the latent factor sampled by the mode index will be updated, and such specific *Latent Factor Update* operation can be performed with a three-step process:

- Step 1: $X_{(n,sample)}$ and $H_{(n,sample)}$, together with the to-be updated latent factor $A_{(n,input)}$, will be sent to the multiply-accumulate (MAC) array to calculate the stochastic gradient g for the current iteration.
- Step 2: Benefited from the historically accumulated square of stochastic gradients stored at the Gradient SRAM, the step size of the current iteration $\eta_{(n)}$ is then calculated by an individual CORDIC operation module.
- Step 3: Finally, the updated latent factor $A_{(n,update)}$ is written back to the Latent Factor SRAM.

IV. Proposed Tensor Storage Scheme

Challenge on DRAM Access. Considering in practice tensor decomposition is usually used to process very large-

scale tensor data, which typically contains millions of elements, we assume our proposed hardware architecture is equipped with the off-chip DRAM to store the large-size input data $\underline{\boldsymbol{X}}$. Consequently, from the perspective of hardware implementation, a unique challenge for the proposed AdaCPD accelerator is the random addressing for the memory access during tensor matricization phase. To be specific, during the tensor matricization procedure, a large amount of DRAM access with random reading address are required. However, as indicated in [6], the random access is relatively slower than the sequential access in DRAM, thereby causing significant performance degradation to the overall AdaCPD hardware.

Proposed Solution. To overcome this challenge, we propose a new tensor storage scheme in DRAM. As shown in Fig. 3, in DRAM we store N copies of the N-way tensor $\underline{\boldsymbol{X}}$ to make the components are located sequentially in different dimensions (from 1 to N). Recall that as described in Eq. (2), the mode-n tensor matricization operation is essentially the process of reordering the N-th order tensor $\underline{\boldsymbol{X}}(i_1, ..., i_N)$ into a $J_n \times I_n$ matrix (the matricized tensor). Here we can obverse that the n-th row of the matricized tensor $X_{(n)}$ is actually sequentially sampled from the tensor $\underline{\boldsymbol{X}}$ in the dimension n. Therefore, being consistent with this inherent procedure, our proposed tensor storage scheme can make the large number of DRAM accesses in tensor matricization phase become sequential. Based on our experiment, the resulting memory bandwidth can be increased by up to 30 times.

V. Evaluation

To demonstrate the effectiveness of the proposed architecture, we develop an design example for AdaCPD hardware accelerator. Here the user case is for decomposing a low-rank tensor with $N = 3$ and $I_1 = I_2 = I_3 = 100$. $|\mathcal{F}_n|$ is chosen as 18 and the integer $F = 10$. For the configuration of the hardware implementation, 128 16-bit multipliers are allocated with 16 KB on-chip SRAM. The overall hardware architecture is implemented using Verilog HDL, and the RTL model is then synthesized using Synopsys Design Compiler with 28nm CMOS technology. The synthesis reports show our design example consumes 1.6 mm^2 area and 300 mW power under 800 MHz clock frequency.

Fig. 4 compares the performance between our customized hardware implementation and CPU-based software implementation. Here the evaluated CPU platform is Intel Core i9-9900K with 3.6 GHz clock frequency. It is seen that the specialized hardware accelerator can achieve 6762 times and 1483 times improvement over CPU-based software implementation with respect to area efficiency and energy efficiency, respectively.

VI. Conclusion

In this paper we propose and design a specialized hardware architecture for AdaCPD algorithm for tensor decomposition. A DRAM tensor storage scheme is proposed to alleviate the huge cost caused by the large number of DRAM accesses during the tensor matricization operation. Compared with CPU-based software implementation, the customized hardware architecture provides significant improvement. Future work will be conducted towards applying the proposed hardware architecture to several emerging applications, such as tensor decomposition-based deep neural network compression [7] [8] [9] [10] as well as extending it to a more general tensor decomposition engine [7] [8] [9] [11].

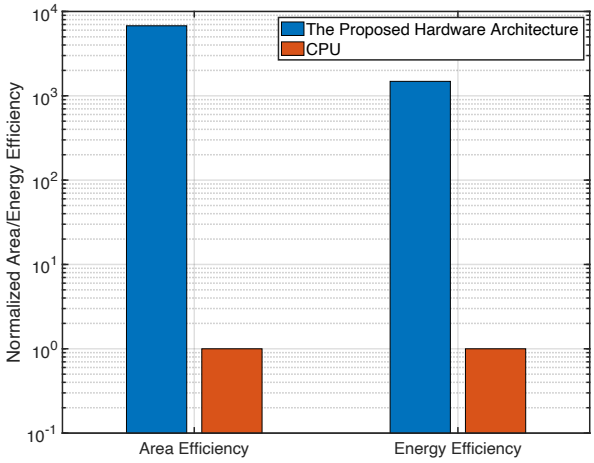

Fig. 4. The comparison with CPU-based implementation.

VII. Acknowledgement

This work is partially funded by National Science Foundation Award CNS-1932370, ECCS-1808159 and ECCS-2024058.

References

[1] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[2] Charilaos I Kanatsoulis, Xiao Fu, Nicholas D Sidiropoulos, and Wing-Kin Ma. Hyperspectral super-resolution: A coupled tensor factorization approach. *IEEE Transactions on Signal Processing*, 66(24):6503–6517, 2018.

[3] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of machine learning research*, 15:2773–2832, 2014.

[4] Xiao Fu, Shahana Ibrahim, Hoi-To Wai, Cheng Gao, and Kejun Huang. Block-randomized stochastic proximal gradient for low-rank tensor factorization. *IEEE Transactions on Signal Processing*, 68:2170–2185, 2020.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[6] Arm Holdings. Amba axi and ace protocol specification, 2021.

[7] Miao Yin, Siyu Liao, Xiao-Yang Liu, Xiaodong Wang, and Bo Yuan. Towards extremely compact rnns for video recognition with fully decomposed hierarchical tucker structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12085–12094, 2021.

[8] Miao Yin, Yang Sui, Siyu Liao, and Bo Yuan. Towards efficient tensor decomposition-based dnn model compression with optimization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10674–10683, 2021.

[9] Miao Yin, Huy Phan, Xiao Zang, Siyu Liao, and Bo Yuan. Batude: Budget-aware neural network compression based on tucker decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[10] Chunhua Deng, Fangxuan Sun, Xuehai Qian, Jun Lin, Zhongfeng Wang, and Bo Yuan. Tie: Energy-efficient tensor train-based inference engine for deep neural network. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 264–278, 2019.

[11] Chunhua Deng, Miao Yin, Xiao-Yang Liu, Xiaodong Wang, and Bo Yuan. High-performance hardware architecture for tensor singular value decomposition: Invited paper. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6, 2019.