2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS) | 978-1-6654-0279-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/MWSCAS54063.2022.9859440

Hardware Acceleration in Large-Scale Tensor Decomposition for Neural Network Compression

Chen-Chien Kao[1], Yi-Yen Hsieh[1], Chao-Hung Chen[2], and Chia-Hsiang Yang[1]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
[2]Industrial Technology Research Institute, Hsinchu, Taiwan

***Abstract*—A tensor is a multi-dimensional array, which is embedded for neural networks. The multiply-accumulate (MAC) operations involved in a large-scale tensor introduces high computational complexity. Since the tensor usually features a low rank, the computational complexity can be largely reduced through canonical polyadic decomposition (CPD). This work presents an energy-efficient hardware accelerator that implements randomized CPD in large-scale tensors for neural network compression. A mixing method that combines the Walsh-Hadamard transform and discrete cosine transform is proposed to replace the fast Fourier transform with faster convergence. It reduces the computations for transformation by 83%. 75% of computations for solving the required least squares problem are also reduced. The proposed accelerator is flexible to support tensor decomposition with a size of up to 512×512×9×9. Compared to the prior dedicated processor for tensor computation, this work support larger tensors and achieves a 112× lower latency given the same condition.**

***Index Terms*—Neural network compression, tensor decomposition, canonical polyadic decomposition, hardware acceleration.**

I. Introduction

A tensor is an array with multiple dimensions. It has been applied to many applications, especially for deep learning due to the structure of neural networks with multiple dimensions, contributed by the dimensions of feature map and filter (also known as kernel). Modern neural networks usually include a large amount of hyper-parameters. Multiply-accumulate (MAC) operations involved in a large-scale tensor introduces high computational complexity and makes deploying neural networks on resource-constrained devices challenging. The tensors involved in these neural networks usually feature a low-rank property [1]. This property can be leveraged to compress the neural networks, thereby reducing their computational complexity and memory usage.

The canonical polyadic decomposition (CPD) is one generalization of singular value decomposition (SVD) and it is widely used for low-rank tensor decomposition. CPD plays an essential role in deep learning field [2]. Fig. 1 illustrates the mapping between a reference neural network and the network structure through CPD. In the reference network, each feature map is convolved with multiple filters. The tensor consisting of filters can be decomposed and compressed by CPD, thereby reducing the computational complexity. The associated MAC operations and memory usage can be largely reduced accordingly. A compact network structure enables to accelerate computations for deep neural networks [2].

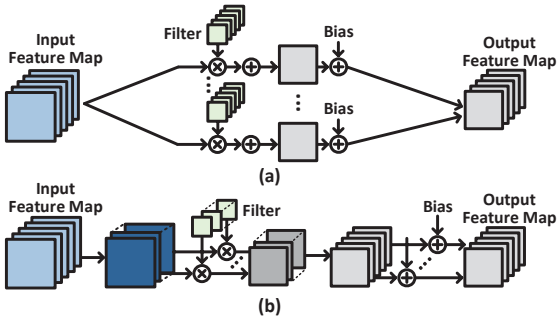

Fig. 1. (a) Reference network and (b) compressed network through CPD.

CPD computation for high-dimension tensor is time-consuming. Dedicated hardware acceleration is therefore critical. In [3], FPGA-based parallel processor is presented to compute tensor decomposition. However, it only supports decomposition for tensors with three dimensions, which is insufficient for modern neural networks. In this work, we propose a dedicated hardware accelerator that supports higher-dimensional tensor decomposition for commonly-used neural networks.

II. Canonical Polyadic Decomposition

The CPD decomposes an N-dimensional tensor $\mathbf{X}$ into a linear combination of R rank-1 outer products of matrices [4], where R is the rank of $\mathbf{X}$, which is also the minimum number of rank-1 tensors required to produce $\mathbf{X}$, As an example, for a three-dimensional tensor $\mathbf{X}$ with a size of $I \times J \times K$ and a rank of R can be expressed as

$$\begin{aligned} \mathbf{X} &= \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \\ \iff \mathbf{X}(i,j,k) &= \sum_{r=1}^{R} \mathbf{a}(i,r)\mathbf{b}(j,r)\mathbf{c}(k,r), \end{aligned} \tag{1}$$

where $\mathbf{a}_r$, $\mathbf{b}_r$, and $\mathbf{c}_r$ are the columns of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively; $\circ$ represents outer product. The CPD with low-rank approximation by $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ is formulated as an optimization problem as follows:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \left\| \mathbf{X} - \sum_{r=1}^{R} \lambda_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \right\|_F, \tag{2}$$

where λ_r is a weight for each component and $\|\|_F$ denotes the Frobenius norm.

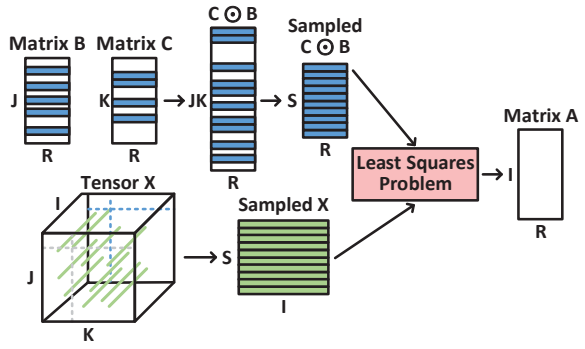

Fig. 2. Illustration of RCPD on a tensor X.

A. Alternating Least Squares

The alternating least (ALS) squares algorithm computes the CPD of tensors with low-rank approximation [3], [6]. ALS first initializes two matrices ($\mathbf{B}$ and $\mathbf{C}$ here). It then updates the other matrix ($\mathbf{A}$) at once while fixing others by solving a least squares problem [5] as follows

$$\mathbf{A}^T = (\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}(\mathbf{C} \odot \mathbf{B})^T\mathbf{X}_u, \tag{3}$$

where $*$ is the Hadamard product, $\odot$ is the Khatri-Rao product, and $\mathbf{X}_u$ is the unfolded matrix of $\mathbf{X}$, given by

$$\mathbf{X}_u = [vec(\mathbf{X}(1,:,:)) \ ... \ vec(\mathbf{X}(I,:,:))]. \tag{4}$$

One of the other two matrices $(\mathbf{B}, \mathbf{C})$ is updated in a similar way by fixing the other two matrices. This procedure repeats until the three matrices converge. The most computationally intensive part for the ALS algorithm is to compute $(\mathbf{C} \odot \mathbf{B})^T\mathbf{X}_u$, which is called metricized tensor times Khatri-Rao product (MTTKRP) [5]. The required number of multiplications for MTTKRP is $JKR + IJKR$.

B. Randomized CPD

To minimize the computational complexity of MTTKRP, randomized CPD (RCPD), which is a sampling-based method for least squares problem, is presented in [7]. RCPD only samples S (which is smaller than JK) rows of $\mathbf{X}$ and $(\mathbf{C} \odot \mathbf{B})$ to implement the Khartri-Rao product, yielding a great reduction in complexity scale from JK to S. It is shown that $10R \times \log R$ is sufficient for RCPD. Fig. 2 shows an example of RCPD on $\mathbf{X}$ and the corresponding Khatri-Rao product through random sampling.

However, random sampling sometimes leads to error in biasedly sampled rows, which may cause rank deficiency and non-convergence of RCPD. Thus, a *mixing* strategy is commonly utilized for RCPD by randomly flipping the sign of each row followed by a mixing operation. The fast Fourier transform (FFT) is adopted for mixing in [7].

C. Comparison between ALS and RCPD

To measure the performance of the CPD algorithms, *fit* [7] and *score* [8] are commonly-used metrics. The closer the values of *score* and *fit* to 1, the better the performance. The experimental setting of RCPD with tensor mixing is based on [7]. The test dataset includes low-rank tensors $\mathbf{X}$ with a size of 512×512×9×9 [1], [2] and a rank of 16. Fig. 3 and Table I show the performance of ALS and RCPD. As can be seen, RCPD achieves a higher $score$ and a higher fit than ALS with much less execution time (on Intel i7-8700 CPU) due to its lower computational complexity. Thus, RCPD is adopted in this work.

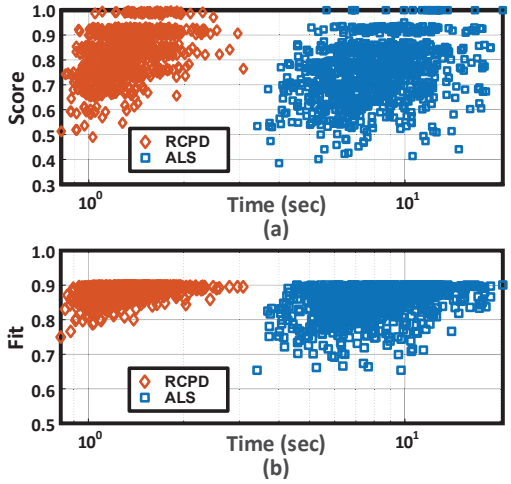

Fig. 3. Performance in (a) *score* and (b) *fit* vs. execution time for ALS and RCPD.

TABLE I. AVERAGE CPD PERFORMANCE

Algorithm	Avg. Score	Avg. Fit	Avg. Execution Time (s)
ALS	0.75	0.85	8.32
RCPD	0.83	0.88	1.34

III. System Design

For the mixing operation, the FFT-based mixing strategy converts a real-valued tensor into a complex-valued one, which introduces more memory usage for storing the imaginary-valued parts. The complex-valued nature of FFT also asks for higher computational complexity. In this work, a mixing method that combines the real-valued Walsh-Hadamard transform (WHT) and discrete cosine transform (DCT) is proposed to replace FFT. Fig. 4 shows $score$ and fit of the FFT and WHT-DCT schemes, respectively. It can be seen that the WHT-DCT scheme has a similar convergence behavior to the FFT counterpart. The average number of iterations of the WHT-DCT and the FFT schemes are 23.2 and 23.9, respectively, for convergence. The computational complexity for transformation is reduced by 83%. Furthermore, 75% computational complexity is reduced for solving the following least squares problem.

Fig. 5 shows the architecture of the proposed accelerator that implements RCPD with WHT-DCT mixing. The accelerator contains a mixing engine, an index sampler, a least squares (LS) solver, and a normalizer. The mixing engine implements the mixing strategy with WHT-DCT for RCPD. The index sampler generates random indices to sample rows and the corresponding addresses for data access. The LS solver is responsible for solving the LS problem through an MAC array

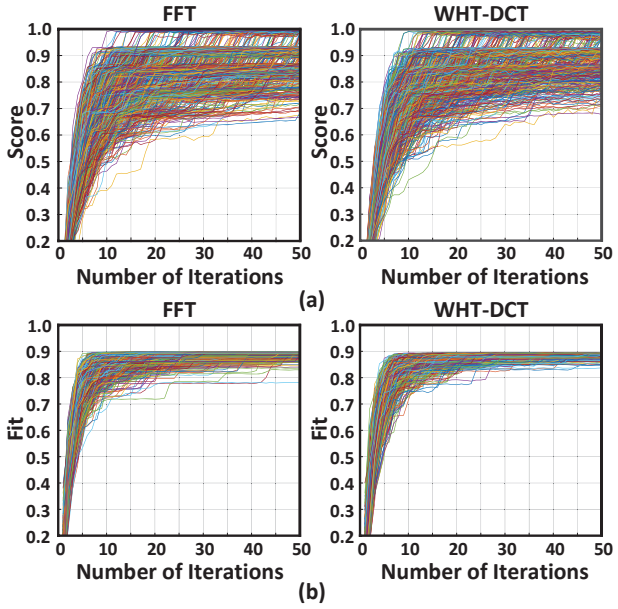

Fig. 4. Performance comparison in (a) *score* and (b) *fit* for RCPD.

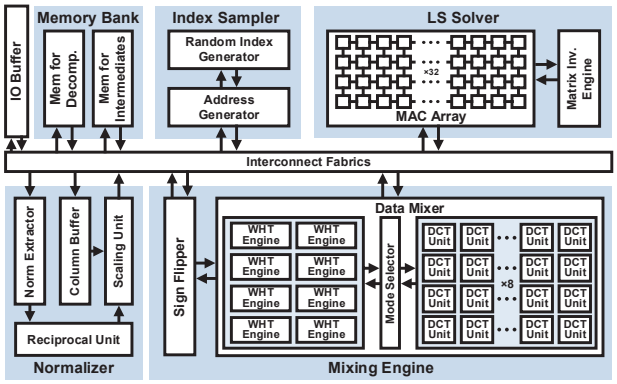

Fig. 5. Hardware architecture of the proposed tensor decomposition accelerator.

and a matrix inversion engine. The normalizer is deployed for column-wise normalization by scaling the vectors computed by the LS solver. The memory bank stores the intermediate data for the LS solver and the CPD outputs.

A. Mixing Engine

The mixing engine contains a sign flipper and a data mixer. The sign flipper generates random numbers to decide whether the sign of each row of the tensor is flipped or not. The data mixer is designed to perform the mixing operation. It includes 8 WHT engines and 32 DCT units to implement WHT and DCT, respectively. A mode selector is designed to determines the type of transformation for tensor mixing based on the value of the dimension: WHT is used when the value of the dimension is a power of two; DCT is adopted when the value of the dimension is not a power of two. If the value of the dimension is a product of aforementioned cases, a combined WHT-DCT method is utilized. Fig. 6 shows the architecture of the WHT engine, in which four variable-length single-path delay-feedback (SDF) processing elements are deployed followed by a radix-4 (R4) butterfly unit. The architecture of the variable-length SDF recessing elements is similar to that of the SDF FFT unit composed of radix-2 (R2) butterfly units [9], but compute-intensive multipliers can be simplified since the coefficients for WHT are ± 1. Fig. 7 shows the architecture of the DCT unit, in which the matrix-vector multiplications are computed in a column-wise fashion.

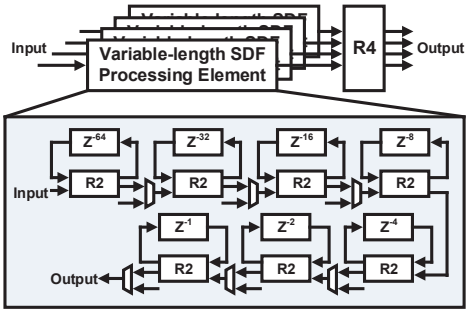

Fig. 6. Architecture of the Walsh-Hadamard transform (WHT) engine.

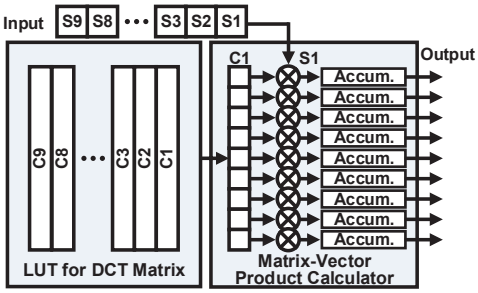

Fig. 7. Architecture of the DCT unit.

B. Index Sampler

The index sampler is used to generate random numbers and to generate the corresponding addresses for sampling rows for solving the LS problem. Fig. 8 shows the details of the index sampler, which consists of a random index generator and an address generator. Multiple pseudo-random number generators are included in the random index generator for sampling multiple dimensions in parallel. For each pseudo-random number generator, an array of linear-feedback shift registers (LFSRs) are used to generate the required random number. The dynamic range of a random number depends on the value of the dimension. If the value of the dimension equals to a power of two, the random number is generated by LFSRs directly. Otherwise, the random number is generated by a look-up table and the address is from LFSRs. For the case that the value of the dimension is the product of the aforementioned cases, a combination of the above two schemes is adopted to generate the random number within the required range.

C. LS Solver and Normalizer

The LS solver is composed of an MAC array and a matrix inversion engine. The MAC array is used to perform the matrix

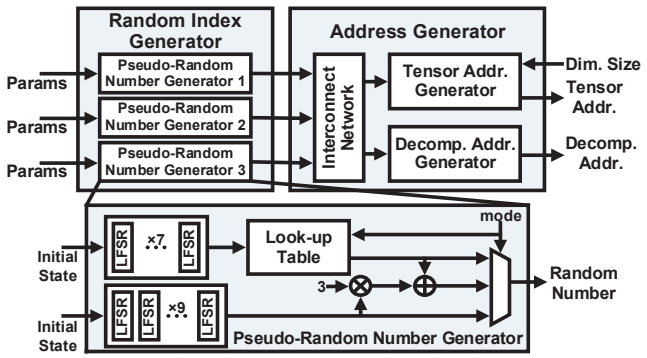

Fig. 8. Architecture of the index sampler.

TABLE II. SUPPORTED TENSOR SIZE AND CONFIGURATION

Max Tensor Size ($N_i \times N_o \times F \times F$)	512×512×9×9
No. of Input Feature Maps (N_i)	2^n (n = 2 to 9), $3{\times}2^m$ (m = 2 to 7)
No. of Output Feature Maps (N_o)	2^n (n = 2 to 9)
Filter Size ($F \times F$)	3×3, 5×5, 8×8, 9×9
Max Rank	16

operations for solving the LS problem, including computing the Khatri-Rao product and associated matrix product. The matrix inversion engine is used to compute the pseudo inverse matrix, which is the solution of the LS problem. The design of [10] that includes coordinate rotation digital computers (CORDICs) is adopted to implement matrix inversion efficiently. The utilization of the CORDICs can be up to 100% to maximize the throughput. The normalizer is designed to implement the column-wise normalization.

IV. Experimental Verification

Table II shows the supported tensor sizes and configurations, based on the settings for neural network compression [1], [2]. The maximum dimension is 4. The maximum values of respective dimensions are 512, 512, and (9×9) for input feature map, output feature map, and filter. The proposed tensor decomposition accelerator is designed in a 40nm CMOS technology. It integrates 4.46M logic gates according to the synthesis estimate. The estimated power consumption is 298.8 mW at a clock frequency of 250 MHz from a 0.9V supply. The average latency of the accelerator for computing CPD is 17.8 ms for a tensor size of 512×512×9×9 with a rank of 16, achieving a 75× higher acceleration than a high-end CPU (Intel i7-8700). Table III shows the performance comparison with the FPGA tensor processor (on Xilinx XCVU9P-L2FSGD2104E) presented in [3]. The number of iterations is set to an equal value to make a fair comparison. The latency is normalized by the tensor size and rank to make a fair comparison [11]. Under this condition, this work achieves a 112× lower latency compared to [3].

V. Conclusions

This work presents an CPD accelerator for neural network compression. RCPD is adopted since it achieves better performance than ALS with less computational complexity, which allows for large-scale tensor decomposition. Walsh-Hadamard transform and discrete cosine transform are proposed for tensor mixing, reducing the computational complexity for transformation by 83%. It also leads to a computational reduction for the solving LS problem by 75%, when compared to tensor mixing with FFT. Compared to previous FPGA-based tensor processor for CPD, this work supports larger tensors and achieves a 112× faster speed given the same condition. This work demonstrates a promising solution for efficient neural network compression on large-scale tensors.

TABLE III. COMPARISON WITH PRIOR ART

	TVLSI'21 [3]	This Work
Platform	FPGA	ASIC
Technology (nm)	14/16	40
CPD Algorithm	ALS	RCPD
Tensor Size	40×40×1200	512×512×9×9
Rank	18	16
Freq (MHz)	250	250
Power (mW)	3636	298.8
Latency (cycle)	50M (est.)	4.46M
Normalized Latency*	1.45	0.013

* Normalized Latency = Latency × $\frac{1}{\text{Tensor Size}\times\text{Rank}}$.

VI. Acknowledgement

This work is supported by the Industrial Technology Research Institute (ITRI) and the Ministry of Science and Technology (MOST) of Taiwan. The authors also thank the Taiwan Institute Research Institute (TSRI) for technical support on chip design.

References

[1] A.-H. Phan, *et al.*, "Stable low-rank tensor decomposition for compression of convolutional neural network," *European Conference on Computer Vision*, pp. 522–539, 2020.

[2] V. Lebedev, Y. Ganin, *et al.*, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," *Proc. Int. Conf. Learn. Represent (ICLR)*, pp. 838–845, May 2015.

[3] W.-P. Huang, R. C. C. Cheung and H. Yan, "An efficient parallel processor for dense tensor computation," *IEEE Trans. Very Large Scale Integration (TVLSI) Systems*, vol. 29, no. 7, pp. 1335–1347, Jul. 2021.

[4] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009.

[5] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis and C. Faloutsos, "Tensor Decomposition for Signal Processing and Machine Learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551-3582, July, 2017.

[6] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the PARAFAC model," *Comput. Stat. Data Anal.*, vol 50, pp. 1700-1734, 2006.

[7] C. Battaglino, G. Ballard, and T. Kolda, "A practical randomized CP tensor decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 39, no. 2, pp. 876-901, 2018.

[8] B. W. Bader, T. G. Kolda, *et al.*, (April 2021) Matlab tensor toolbox Version 3.2.1, [Online] Available: https://www.tensortoolbox.org/

[9] T.-D. Chiueh, P.-Y. Tsai, and I.-W. Lai, *Baseband Receiver Design for Wireless MIMO-OFDM Communications*, Hoboken, NJ, USA: Wiley, 2012.

[10] Y.-Y. Hsieh, Y.-C. Lin, C.-H. Yang, "A 96.2nJ/class Neural Signal Processor with Adaptable Intelligence for Seizure Prediction," *Int. Solid-State Circuits Conference (ISSCC)*, pp. 506-507, 2022.

[11] P. Comon, X. Luciani, and A. L. F. de Almeida, "Tensor decompositions alternating least squares and other tales," *J. Chemometrics*, vol. 23, no. 7–8, pp. 393–405, Jul. 2009.