PSB-RNN: A Processing-in-Memory Systolic Array Architecture using Block Circulant Matrices for Recurrent Neural Networks

Nagadastagiri Challapalle[1], Sahithi Rampalli[1], Makesh Chandran[1], Gurpreet Kalsi[2], Sreenivas Subramoney[2], John Sampson[1], Vijaykrishnan Narayanan[1]

(1) Pennsylvania State University, University Park, PA, USA. (2) Processor Architecture Research Lab, Intel Labs, Bangalore, KA, India.

{nrc53, svr46, mzc88, jms1257, vijaykrishnan.narayanan }@psu.edu, {gurpreet.s.kalsi, sreenivas.subramoney}@intel.com

***Abstract*—Recurrent Neural Networks (RNNs) are widely used in Natural Language Processing (NLP) applications as they inherently capture contextual information across spatial and temporal dimensions. Compared to other classes of neural networks, RNNs have more weight parameters as they primarily consist of fully connected layers. Recently, several techniques such as weight pruning, zero-skipping, and block circulant compression have been introduced to reduce the storage and access requirements of RNN weight parameters. In this work, we present a ReRAM crossbar based processing-in-memory (PIM) architecture with systolic dataflow incorporating block circulant compression for RNNs. The block circulant compression decomposes the operations in a fully connected layer into a series of Fourier transforms and point-wise operations resulting in reduced space and computational complexity. We formulate the Fourier transform and point-wise operations into in-situ multiply-and-accumulate (MAC) operations mapped to ReRAM crossbars for high energy efficiency and throughput. We also incorporate systolic dataflow for communication within the crossbar arrays, in contrast to broadcast and multicast communications, to further improve energy efficiency. The proposed architecture achieves average improvements in compute efficiency of 44x and 17x over a custom FPGA architecture and conventional crossbar based architecture implementations, respectively.**

***Index Terms*—Recurrent neural network, Processing-in-memory, Block circulant, Fourier transform, ReRAM Crossbar.**

I. Introduction

Recurrent Neural Networks (RNNs) are the current solution of choice for providing state of the art accuracy in NLP tasks such as voice recognition, sequence to sequence translation, and question answering systems. These recurrent networks process input sequences in a sequential manner and maintain a hidden state (or memory) for each sequence that preserves useful contextual information. Long Short Term Memory Networks (LSTM) [1] and Gated Recurrent Units (GRU) [2] are the most used variants of RNNs in practice, with LSTM networks constituting 29% of all inference traffic on Google's Tensor Processing Units (TPUs) deployed in data centers [3]. Unlike the Convolutional Neural Networks (CNNs) that dominate the computer vision and image processing domains, RNNs primarily consist of fully connected layers along with the gating mechanisms to capture the contextual information in their input sequences. Considering the space and computational complexity of RNNs, with their large number of weight parameters for each layer making them both compute and memory intensive, efficient algorithmic and hardware architectures are required to enable high performance at low overheads, and these architectures will be distinct from those designed to optimize CNNs and other classes of inference solutions.

Several algorithmic techniques, such as weight pruning [4], activation thresholding (or zero skipping) [5], and block circulant based weight compression [6] have been proposed to reduce the computational and space complexity of RNNs. Owing to their large model sizes, RNNs require off chip storage (such as DRAM) and several memory accesses for fetching the weight parameters to compute units. This data movement contributes significantly to RNN power consumption [7], and many RNN optimizations focus on either reducing the total number of weights, eliding access to weights, or reducing weight movement within the memory hierarchy. Weight pruning eliminates insignificant weights that do not contribute to the training accuracy. Activation thresholding skips the computations associated with activations below a certain threshold value thereby reducing the memory accesses. Block circulant based compression imposes the block circulant property on weight matrices during training and reduces the space ($O(n^2) \rightarrow O(n)$) and computational complexity ($O(n^2) \rightarrow O(n \log n)$). C-LSTM [6] incorporated block circulant compression for RNNs in custom FPGA implementation and achieved 11x performance improvement over the custom FPGA implementation without compression [4] with minimal degradation in accuracy. Crossbar based processing-in-memory (PIM) approaches have proven very successful for CNN hardware acceleration [8]–[10] in terms of both performance and energy efficiency, and, recently, Long *et al.* [11] proposed a crossbar based accelerator for RNNs that leverages crossbar MAC capabilities to achieve 79x improvements in computing efficiency over a GPU solution. This approach maps the conventional RNN weight parameters to crossbar arrays, and input features and intermediate activations are fed as inputs to perform the dot product operation.

In this work, we propose a crossbar based hardware architecture for RNNs using block circulant-based compression

This work was supported in part by Semiconductor Research Corporation (SRC) Center for Brain-inspired Computing Enabling Autonomous Intelligence (C-BRIC).

incorporating systolic dataflow. In contrast to prior work, such as C-LSTM [6], that uses a high level automated synthesis for mapping block circulant RNNs to FPGAs, we emphasize and address the microarchitectural challenges in incorporating block circulant compression with a PIM crossbar architecture. To the best of our knowledge, this is the first effort to map block circulant based RNNs to a crossbar architecture. Block circulant based compression reduces both space and computational complexity with minimal degradation in accuracy without altering the network structure [6], [12]. However, RNNs with block circulant weight matrices cannot be readily mapped onto existing crossbar architectures as they comprise Fourier transform operations, block wise element-element multiplications and result accumulations from different blocks in the complex domain. We map the Fourier transform and block circulant operations to crossbar based in-situ MAC operations by efficient weight mapping and incorporate support for complex arithmetic. Existing PIM architectures operate crossbar arrays as massively parallel compute units by broadcasting the input operands and intermediate results. In contrast, inspired by successful systolic array custom hardware architectures for neural networks [3], [13], we implement systolic dataflow among the crossbar arrays to avoid global data transfers. Finally, we show the performance and energy efficiency benefits of our proposed architecture contrasted with that of existing custom FPGA architecture [4], [6], crossbar architecture [11], CPU, and GPU implementations for LSTM and GRU variants of RNNs on TIMIT [14], HAR [15], and NLP [16] benchmarks.

The key contributions of the paper include:

- We propose a new PIM architecture for RNNs, *PSB-RNN*, that leverages block-circulant matrices, systolic communication among PIM processing elements, and ReRAM MAC operations to provide both speedups and efficiency improvements for RNN inference. We characterize the power, area, throughput and energy efficiency of a SystemVerilog RTL implementation of PSB-RNN in 32 nm technology. The proposed architecture achieves 44x and 17x improvements in compute efficiency over a custom FPGA RNN acceleration architecture and a conventional crossbar-based RNN architecture, respectively.
- In support of PSB-RNN, we develop a new approach to map block circulant compression based RNNs to ReRAM crossbar PIM architectures with end-to-end implementation of matrix-vector multiplication of complex elements. We map the Fourier transform and element wise arithmetic operations in the complex domain in block circulant matrix operations to crossbar based MAC operations to leverage their high energy efficiency and massive parallelism.
- We incorporate systolic dataflow in the crossbar based PIM architectures alleviating the need for complex interconnect architectures and global communications, such as the broadcast and multicast functionalities common in several prior inference accelerator designs.

II. Background

A. Recurrent Neural Networks

In Recurrent Neural Networks, the neurons are part of a recurrent connection such that they store temporal information by maintaining an internal state (memory). Unlike feed-forward networks, which process each input independently, recurrent neural networks process a sequence of inputs using the internal state through a series of gating mechanisms and activation functions. LSTM and GRU are the widely used RNN variants. LSTM networks read, write and erase the data in the cell through some regulatory gates. We use the Google LSTM cell [17] as the reference for our architecture evaluation. Equation (1) lists the operations involved in an LSTM cell.

Each gate of the LSTM cell: *forget* (f_t), *input* (i_t), *cell* (g_t) and *output* (o_t) gates, process the input (x_t) and the previous hidden state (h_{t-1}) using the corresponding weight matrices. The processed output is element-wise added to the corresponding bias and the peephole connections obtained by multiplying diagonal matrices (W_{fc}, W_{ic}, W_{oc}) with previous cell state (c_{t-1}). GRUs have similar operations as LSTMs. The difference is GRUs have only two gates: *update* (z_t) and *reset* (r_t) that are computed by applying the input and previous hidden state to the weight matrices. The gate vectors are passed to activation units and fused to get the final output. In both LSTM and GRU architectures the MAC operation is the most compute intensive operation.

$$\begin{aligned} f_t &= \sigma(W_f\,[x_t, h_{t-1}] + W_{fc} c_{t-1} + b_f) \\ i_t &= \sigma(W_i\,[x_t, h_{t-1}] + W_{ic} c_{t-1} + b_i) \\ o_t &= \sigma(W_o\,[x_t, h_{t-1}] + W_{oc} c_{t-1} + b_o) \\ g_t &= \sigma(W_g\,[x_t, h_{t-1}] + b_g), \quad c_t = f_t \odot c_{t-1} + i_t \odot g_t \\ m_t &= o_t \odot \tanh(c_t), \quad h_t = \phi(W_{hm} m_t + b_h) \end{aligned} \tag{1}$$

B. Block Circulant Compression

In a block-circulant matrix, the rows (columns) in a particular block of matrix are formed by right rotation of their previous row (column). The block circulant property is used to speed-up matrix operations such as multiplication and eigen value decomposition [18]. Matrix-vector multiplication (MVM) with a block circulant matrix simplifies to circular convolution, which can be efficiently computed using discrete Fourier transform (DFT). Also, imposing the block circulant property on a matrix reduces space complexity from $O(n^2)$ to $O(n)$ for a block of dimension n. Consider a matrix W of size $N \times M$ partitioned into circulant blocks B_{ij} of size $n \times n$, where $i \epsilon \{1..p\}$, $j \epsilon \{1..q\}$ and $p = \frac{N}{n}$, $q = \frac{M}{n}$. Figure 1 illustrates the block partitioning of a matrix such that each block is circulant. A matrix-vector multiplication can

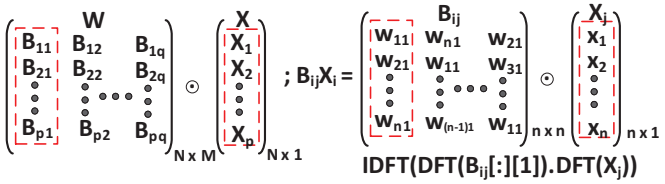

Fig. 1. Matrix vector multiplication operation with block circulant matrices

be achieved by performing block matrix-vector multiplication. Since each block is circulant, the result vector R of operation WX can be calculated as:

$$R_i = \sum_{j=1}^{q} F^{-1}[F(B_{ij}[:][1]) \cdot F(X_j)] \quad (2)$$

where $i \in \{1, \ldots, p\}$, X_j is the corresponding partitioned input. Here, F represents a discrete Fourier transform (DFT) and F^{-1} represents Inverse DFT (IDFT). Thus, an $n \times n$ matrix-vector multiplication can be reduced to performing DFT on the first column of a circulant block and the input vector followed by element wise multiplication and IDFT operations. Thus, the compute complexity per block reduces from $O(n^2)$ to $O(n \log n)$ [19].

This block circulant property can be imposed on the weight matrices of neural networks during training for compression and computational complexity reduction [6], [12], [19]. Compared to other compression techniques such as exploiting sparsity [20] and connection pruning [21], block circulant matrices offer structured compression that maintains the regular network structure.

C. ReRAM Crossbar

ReRAM based crossbar arrays can be configured to perform several in-situ MAC operations in parallel leveraging the current summation technique [8]. When an input voltage is applied to each crossbar cell through a digital to analog converter (DAC), current passes through the cell resistors. This current from each cell is summed along the bit-line according to Kirchoff's Law which is equivalent to the sum of products of the applied voltage and the cell resistance. The resultant analog current from each bitline is held in sample and hold units (S&H) and then fed sequentially to an analog to digital converter (ADC).

The number of bits per crossbar cell is limited by technology factors. Also higher number of bits per cell needs high precision ADCs. Hence, to support the MAC operation, the operands will be interleaved across contiguous bitlines and the partial products will be accumulated using shift and add units (Figure 2).

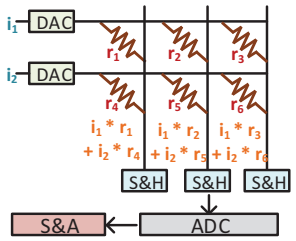

Fig. 2. Dot product operation in crossbar

III. Hardware Architecture

This section discusses the architecture of our proposed crossbar-based processing-in-memory systolic array (SA) architecture for RNNs, *PSB-RNN*. Supporting RNN inference comprises accelerating MVM operation (gating mechanisms), activation functions (gating mechanisms), and scalar arithmetic (fusing different gate outputs). Figure 3 illustrates the overall dataflow architecture of the RNN inference operation. The most data and compute intensive operation in the RNN is matrix-vector multiplication (MVM).

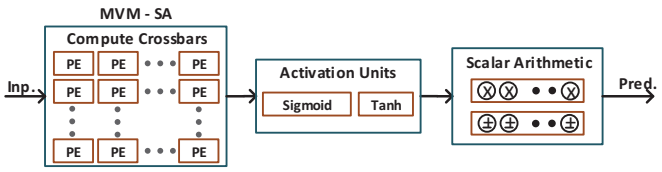

Fig. 3. PSB-RNN dataflow overview

We map the MVM operations in RNN inference to crossbar based MAC operations. The input feature vectors, along with the previous hidden state vectors, are multiplied with the weight matrices of different gates which are fused together to get the output vector. We impose the block circulant property on the weight matrices to reduce the space and computational complexity. In the block circulant based approach, MVM is decomposed into discrete Fourier transform (DFT), element wise multiplications and accumulation of partial products from different blocks. The DFT values (frequency domain values) of weight matrix elements are pre-calculated and mapped to crossbar arrays. Only the input feature vectors and hidden state vectors need to be converted to frequency domain in runtime through DFT. We map the DFT operations, element wise multiplications, and accumulation operations to MAC operations in crossbar, resulting in an end-to-end crossbar based processing engine (PE) for block circulant MVM operation.

A. Discrete Fourier Transform Architecture

The Fast Fourier Transform (FFT) is the most widely used algorithm for performing the DFT operation. The FFT algorithm decomposes DFT operation into a series of sparse multiplications reducing the computational complexity as shown in Figure 4(a). The sparse matrices in each stage of FFT operation (including the input reordering stage) can be fused together to form a single dense matrix (FFT weight matrix) (Figure 4(a)).

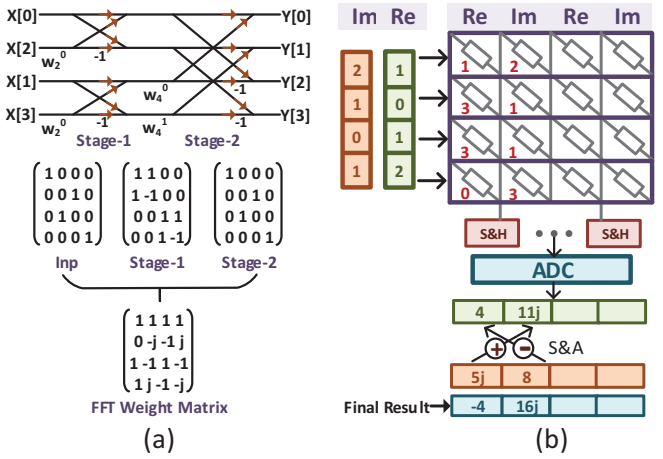

Fig. 4. (a) FFT operation, (b) Complex arithmetic in PSB-RNN crossbar PE

The FFT matrix elements can be mapped to crossbar cells and the FFT operation can be performed as MVM between the FFT matrix and input vectors. Similarly, the inverse-FFT (IFFT) operation can also be formulated as an MVM operation. The FFT matrix elements are complex (contains

both real and imaginary parts), Figure 4(b) illustrates the PSB-RNN crossbar processing element (PE) architecture for MVM operation with complex operands (both input vector, and matrix elements are in complex form). The real and imaginary parts of each matrix element are mapped to adjacent columns of crossbar. The real and imaginary parts of the input elements are also fed sequentially as inputs. Consider two operands $a + jb$ and $c + jd$. In the first cycle, a is fed as input resulting in partial products r_1, r_2. In the next cycle, b is fed as input resulting in r_3, r_4. The final product $e + if$ is $e = r_1 + r_4$ and $f = r_2 - r_3$ (Figure 4(b)). We store the partial results in output buffers before merging them together to form the final product.

B. Systolic Array Architecture

The weight matrices of the RNN gating mechanisms are preprocessed and mapped to compute crossbar arrays connected in a systolic dataflow fashion for performing MVM operations. As discussed in section II-B, we impose the block circulant property on the weight matrices. In order to simplify the mapping of block circulant weights to the crossbar arrays, we rearrange the weights as shown in Figure 5. The MVM operation in RNNs with block circulant weight matrices can be decomposed into four steps: 1. Computing FFT for input vectors and previous hidden state, 2. Block-wise element-element multiplication, 3. Accumulation of partial products from adjacent blocks (in the horizontal direction), 4. IFFT operation to get the final MVM result.

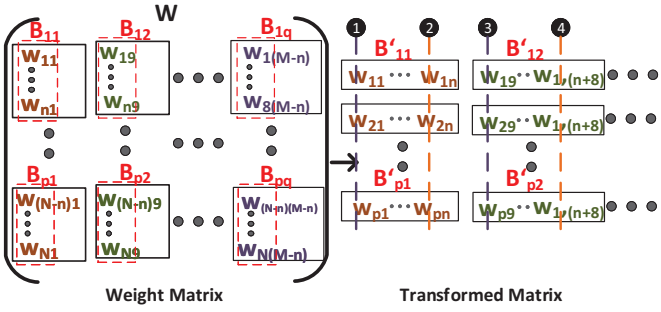

Fig. 5. Weight matrix re-ordering

The weight layout (mapping) across the compute crossbars is orchestrated to facilitate the systolic dataflow movement among the compute crossbars. Figure 6 illustrates the overall systolic array architecture, weight layout, and different compute elements for a block of size $n \times n$. We consider 128×128 2-bit crossbars. 32-bit complex weights are mapped to the crossbars. Thus, each wordline has 8 weight elements. The set of weight elements which share common input i.e. the elements across the columns of B'_{ij} are mapped to a common wordline. For example, columns ❶ and ❷ in Figure 5 are mapped to the first and eighth wordlines of the first row of crossbars respectively. The weight elements from different blocks that need to be accumulated to form the final product are mapped to a single column of crossbar to facilitate the accumulation operation through the crossbar current summation mechanism. For example, columns ❶ and ❸ in Figure 5

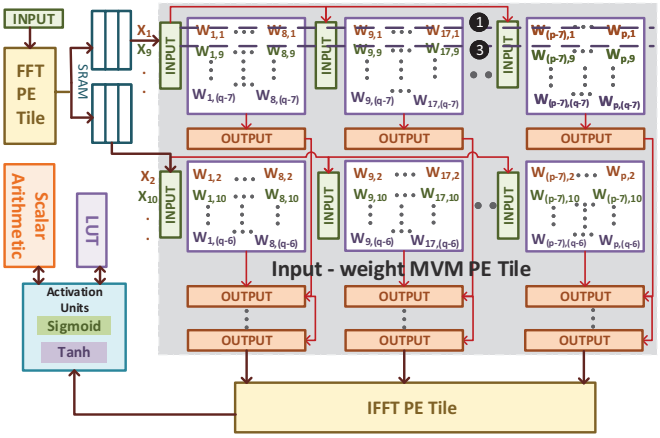

Fig. 6. Overview of the PSB-RNN systolic array architecture

are mapped to the first two rows of the first (systolic array) row of crossbars. The inputs are fed sequentially in a systolic manner to the crossbar compute array, i.e. inputs X_1, X_9 (Figure 6) are fed to the first crossbar in the first cycle and in the next cycle they will be moved to the second crossbar and further down the array sequentially in the horizontal direction. The accumulated results from each crossbar are transferred to other MVM crossbars and IFFT crossbars for further processing in the vertical direction (Figure 6). The weight elements of different gate matrices (input, cell etc.) are interleaved across adjacent columns of crossbars to feed the activation units and scalar arithmetic units down the pipeline for performing further operations as discussed in Section II-A in a pipelined manner with minimum buffering. For supporting negative weight elements, we incorporate the 4's complement method introduced in Fujiki *et al.* [22] and feed 2-bit inputs through DAC to crossbar arrays to support the 4's complement based MAC. This method does not require storing the positive and negative weights in separate crossbars, thereby reducing the storage requirements. The maximum number of wordlines being activated for a compute operation is set to 16 to alleviate the effect of per-cell current accumulation on inference accuracy as suggested in Yang *et al.* [9]. Limiting the number of active wordlines per compute operation also reduces the ADC precision requirements and improves compute access latency (15ns for 128x128 crossbar) [9].

The MVM operation results are passed to activation units and scalar arithmetic units to fuse the outputs of different gates for final output prediction and hidden state calculation. Sigmoid and tanh are the most widely used activation functions in the RNNs. For sigmoid operation, we implement a piecewise linear approximation approach [23]. The arithmetic operations such as multiply/add in the linear approximation of sigmoid function are replaced by simple digital gate transformations between input and output, thus reducing the complex non-linear sigmoid operation to mere combinational logic. A range lookup table [24] approach is used for implementing the tanh operation. It uses range addressable decoding for the lookup table which reduces the lookup table storage requirements without incurring any degradation in accuracy.

TABLE I
PARAMETERS OF THE PSB-RNN CROSSBAR PEs

Component	Configuration	Area ($mm^2 \times 10^{-3}$)	Power (mW)
ReRAM crossbar	128x128, 2-bits/cell number: 1	0.025	0.3
DAC	2-bit, number: 16	0.0025	0.0008
ADC	6-bit, 1.2GSps number: 4	2.35	2.57
S&H	number: 128	0.0039	0.0025
S&A	number: 1	2.24	0.24
Input Buffer	width: 16, number: 1	0.11	0.0047
Output Buffer	128 bytes	0.1	0.068
1 PE Total		**4.8**	**3.18**

TABLE II
PARAMETERS OF THE PSB-RNN ARCHITECTURE

Component	Configuration	Area ($mm^2 \times 10^{-3}$)	Power (mW)
FFT Output SRAM	512 bytes	0.4	0.272
Interconnect	bus width: 16	6.2	0.0207
Adder	number: 16	10	2.272
Multiplier	number: 96	147	56.16
Sigmoid	number: 16	5.3	0.205
Crossbar PEs	number: 800	3840	2544
Tanh	number: 4	2.8	0.118
Total Chip		**4.01** mm^2	**2.692 W**

IV. RESULTS & ANALYSIS

In this section, we analyze the computational efficiency of the proposed architecture in comparison with other state of the art architectures.

A. Baseline Architectures

The baseline LSTM and GRU networks are implemented using Tensorflow. The performance of the baseline networks on CPU and GPU is analyzed using the Tensorflow profiler. CPU and GPU power consumption are measured using Intel RAPL [25] and Nvidia-SMI [26], respectively. We used an Intel Xeon Bronze 3104 CPU (14 nm) operating at 1.7 GHZ and an Nvidia Titan V GPU (12 nm) operating at 1.2 GHz for the performance evaluation. We ensure no user-level process is active during the performance and power analysis and we also subtract out measured system idle power before comparing against the power of our accelerator design for a fairer comparison.

To compare against other accelerator approaches, we use C-LSTM [6], a custom FPGA-based block circulant RNN architecture and ReRAM RNN [11], a custom ReRAM based crossbar architecture for conventional RNN as our hardware accelerator baseline architectures. C-LSTM synthesizes LSTM designs onto an FPGA using an automated synthesis framework. In contrast, our approach emphasizes the micro-architectural aspects of mapping block circulant-based RNNs to obtain better compute efficiency. ReRAM RNN maps conventional RNNs to crossbars. In contrast, we map block circulant RNN to crossbar PEs in a systolic dataflow architecture to leverage the compression benefits. C-LSTM experiments show that block circulant compression incurs minimal accuracy degradation compared to conventional RNN. PSB-RNN is functionally equivalent to the C-LSTM implementation. Consequently, we do not perform additional accuracy analysis in this work as the accuracy is identical to that of C-LSTM.

B. Area and Power Model

The PSB-RNN architecture is implemented and functionally verified in SystemVerilog RTL. The area and power parameters for the digital components are obtained by doing RTL synthesis in Synopsys 32 nm technology for operation at a 650 MHz frequency using the Synopsys Design Compiler. We activate only up to 16 wordlines in each compute operation depending on the block partition size. Hence, a 6-bit ADC is sufficient for the compute operation. We use ReRAM crossbar, S&H unit designs proposed in ISAAC [8] and ADC, DAC designs proposed in Yang *et al.* [9] in PSB-RNN. We model these analog compute components using behavioral RTL and corresponding latency, area and power parameters are annotated from the reference designs for system level area, power and performance analysis. Note that these designs are reported to be compatible with the 32 nm technology. Table I summarizes the area and power parameters for a single PSB-RNN crossbar PE and Table II summarizes the total chip configuration, area, and power.

C. Performance and Efficiency Evaluation

We evaluate the performance and efficiency of PSB-RNN for LSTM and GRU networks on TIMIT, NLP, and HAR benchmarks. We use Google LSTM network with 1024 hidden states for TIMIT, and conventional LSTM and GRU networks with 128 hidden states for the NLP and HAR benchmarks. The network configurations and benchmark suite are the same as those used to evaluate existing state of the art RNN accelerators, so as to enable fair comparisons. It should be noted that the number of active crossbars varies depending on the block size and network parameters and that we incorporate power gating mechanism to turn off inactive crossbars. Table III shows the comparison of the PSB-RNN architecture with CPU, GPU and C-LSTM [6] for block sizes of 8 and 16. PSB-RNN achieves $\sim$2e+6$\times$, and $\sim$71.7e+3$\times$ improvements in compute efficiency (FPS/W) over block circulant CPU and GPU implementations respectively. It should be noted that frame means an input sample for the RNN benchmarks. Compared to C-LSTM [6], our architecture achieves a $\sim$44$\times$ improvement in compute efficiency. This superior compute efficiency stems from the efficient mapping of block circulant MVM operations to low-energy crossbar MAC operation and efficient pipelining with systolic dataflow.

Table IV compares the performance of PSB-RNN (block size = 8) with CPU, GPU and a ReRAM RNN accelerator [11] for LSTM and GRU networks on NLP and HAR benchmarks. PSB-RNN architecture achieves an average $\sim$88e+3$\times$, $\sim$70+e3$\times$, and $\sim$17$\times$ improvement in compute efficiency (FPS/W) over CPU, GPU, and ReRAM RNN respectively for LSTM and GRU networks (without block circulant compression) on NLP and HAR benchmarks. These improvements can be attributed to adopting block circulant compression for weight matrices. Also, as explained in section III-B, limiting

TABLE III
PERFORMANCE COMPARISON OF PSB-RNN ARCHITECTURE WITH C-LSTM [6] AND CPU, GPU IMPLEMENTATIONS

Parameter	FFT-8				FFT-16			
	CPU	**GPU**	**C-LSTM [6]**	**PSB-RNN**	**CPU**	**GPU**	**C-LSTM [6]**	**PSB-RNN**
Latency (μs)	21.9E+4	3.84E+3	16.7	6.12	13E+4	3.49E+3	9.1	2.64
Frames per Second (FPS)	5	260	179.7E+3	378.8E+3	8	287	330.3E+3	1388.9E+3
Power (W)	24	35	22	2.6	24	35	23	1.4
Compute Efficiency (FPS/W)	0.2	7.4	8E+3	145.5E+3	0.3	8.2	14.3E+3	1006E+3
Technology (nm)	14	12	20	32	14	12	20	32

TABLE IV
PERFORMANCE COMPARISON OF PSB-RNN WITH RERAM-RNN [11], CPU AND GPU IMPLEMENTATIONS

Benchmarks	**NLP**		**HAR**	
FPS (x1000)	**LSTM**	**GRU**	**LSTM**	**GRU**
CPU	5.80	2.75	0.07	0.07
GPU	9.40	5.20	0.12	0.08
ReRAM RNN	68	88	14.60	19.80
PSB-RNN	1041.67	1041.67	115.74	115.74
FPS/W	**LSTM**	**GRU**	**LSTM**	**GRU**
CPU	278.44	138.39	1.74	2.15
GPU	261.11	144.44	3.34	2
ReRAM RNN	102272	140294	23556	33433
PSB-RNN	2631400	3013731	292505	334857

the number of active wordlines per compute reduces the ADC precision requirements and improves the compute latency.

V. CONCLUSION

Recurrent Neural Networks (RNNs) are a widely used class of deep neural networks for speech processing and natural language processing tasks. This paper presents a crossbar based processing-in-memory architecture for RNN inference, PSB-RNN, and evaluates the performance and efficiency of the architecture for three widely used RNN benchmarks. The key benefits of PSB-RNN derive from our combination of incorporating block circulant based weight matrix compression for RNNs to reduce the computational and storage complexity and developing efficient techniques for mapping the complex domain arithmetic support needed for block circulant matrix vector multiplication operations to an efficient crossbar-based architecture. PSB-RNN achieves a $\sim$44$\times$ improvement in computational efficiency compared to the state of the art block circulant LSTM FPGA implementation for the TIMIT benchmark and achieves a $\sim$17$\times$ improvement in computational efficiency compared to the conventional crossbar based architecture for LSTM and GRU networks on NLP and HAR benchmarks. The latter result confirms the value of supporting block-circulant computation within crossbar-PIM architectures and the former demonstrates the potential of a PIM-based block-circulant architecture compared with prior designs.

REFERENCES

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *EMNLP*, pp. 1724–1734, 2014.

[3] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, pp. 1–12, 2017.

[4] S. Han *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *FPGA*, pp. 75–84, 2017.

[5] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "DeltaRNN: A Power-efficient Recurrent Neural Network Accelerator," in *FPGA*, pp. 21–30, 2018.

[6] S. Wang *et al.*, "C-LSTM: Enabling Efficient LSTM Using Structured Compression Techniques on FPGAs," in *FPGA*, pp. 11–20, 2018.

[7] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *ISSCC*, pp. 10–14, 2014.

[8] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, pp. 14–26, 2016.

[9] T.-H. Yang *et al.*, "Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks," in *ISCA*, pp. 236–249, 2019.

[10] J. Lin, Z. Zhu, Y. Wang, and Y. Xie, "Learning the sparsity for reram: Mapping and pruning sparse neural network for reram based accelerator," in *ASPDAC*, pp. 639–644, 2019.

[11] Y. Long, T. Na, and S. Mukhopadhyay, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Trans. on VLSI Systems*, vol. 26, no. 12, pp. 2781–2794, 2018.

[12] C. Ding *et al.*, "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-circulant Weight Matrices," in *ISCA*, pp. 395–408, 2017.

[13] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *ISCA*, pp. 367–379, 2016.

[14] J. S. Garofolo *et al.*, "DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM," 1993.

[15] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *ESANN*, 2013.

[16] R. Atienza, "LSTM by example using Tensorflow." [online]. Available: https://github.com/roatienza/Deep-Learning-Experiments. [Accessed: 10- Sep- 2019].

[17] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *INTERSPEECH*, 2014.

[18] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Springer-Verlag, 2001.

[19] Y. Cheng *et al.*, "An Exploration of Parameter Redundancy in Deep Networks with Circulant Projections," in *ICCV*, pp. 2857–2865, 2015.

[20] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *ISCA*, pp. 27–40, 2017.

[21] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," in *NIPS*, pp. 1135–1143, 2015.

[22] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *ASPLOS*, pp. 1–14, 2018.

[23] H. Amin, K. M. Curtis, and B. R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," *IEE Proceedings - Circuits, Devices and Systems*, vol. 144, no. 6, pp. 313–317, 1997.

[24] K. Leboeuf, A. H. Namin, R. Muscedere, H. Wu, and M. Ahmadi, "High Speed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function," in *ICCHT*, vol. 1, pp. 1070–1073, 2008.

[25] Intel Corporation, "Intel vtune amplifier performance profiler." [online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe. [Accessed: 26- Jun- 2019].

[26] Nvidia Corporation, "Nvidia system management interface." [online]. Available: https://developer.nvidia.com/nvidia-system-management-interface. [Accessed: 26- Jun- 2019].