Maximizing System Performance by Balancing Computation Loads in LSTM Accelerators

Junki Park, Jaeha Kung, Wooseok Yi and Jae-Joon Kim
Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea
{junkipark, jhkung, iyohoyi, jaejoon}@postech.ac.kr

***Abstract*—The LSTM is a popular neural network model for modeling or analyzing the time-varying data. The main operation of LSTM is a matrix-vector multiplication and it becomes sparse (spMxV) due to the widely-accepted weight pruning in deep learning. This paper presents a new sparse matrix format, named CBSR, to maximize the inference speed of the LSTM accelerator. In the CBSR format, speed-up is achieved by balancing out the computation loads over PEs. Along with the new format, we present a simple network transformation to completely remove the hardware overhead incurred when using the CBSR format. Also, the detailed analysis on the impact of network size or the number of PEs is performed, which lacks in the prior work. The simulation results show 16∼38% improvement in the system performance compared to the well-known CSC/CSR format. The power analysis is also performed in 65nm CMOS technology to show 9∼22% energy savings.**

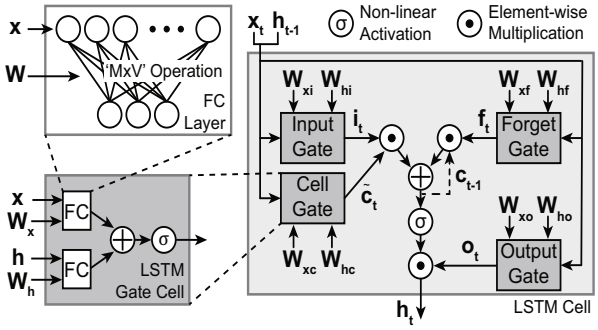

Fig. 1. Detailed illustration of a standard Long Short-Term Memory (LSTM) cell. It consists of four independent gate cells where each cell requires two MxV operations.

I. Introduction

Recurrent Neural Networks (RNNs) have shown promising results in dealing with sequential data. Due to the capability of learning data history, RNNs were applied to machine translation [1], speech recognition [2], user behavior learning [3], video captioning [4], to name a few. This advancement in RNN research was achieved by the development of its variant, i.e. Long Short-Term Memory (LSTM) [5], to overcome training difficulties in standard RNNs [6]. As LSTM shows significantly higher accuracy than the standard RNN, several LSTM accelerators were recently demonstrated [7], [8], [9].

In LSTM, a large portion of computation is Matrix-Vector multiplication (MxV) as its memory cell consists of eight MxV operations, equivalently eight fully-connected (FC) layers (Fig. 1). When designing the LSTM hardware accelerator, the performance bottleneck comes from the required memory bandwidth for providing weight matrices to on-chip processing elements (PEs). The weight values are not re-used in LSTM (or FC layer), which differs from the usage of weight kernels in Convolutional Neural Networks (CNNs) [10]. This makes it impossible to reduce memory access latency or energy consumption by re-using weights loaded on chip.

To alleviate such bottleneck in data fetching, model compression is performed in recent deep learning hardware platforms [9], [11], [12]. The model compression is mostly done by weight pruning and/or quantization. As a result of the weight pruning, the weight matrix may become sparse and the dominant MxV operation becomes 'sparse MxV (spMxV)'. To avoid wasting clock cycles in fetching zero elements in the weight matrix, the pruned weight matrix needs to be stored in sparse matrix format, such as Compressed Sparse Row (CSR) format. The CSR format stores only non-zero elements and pointers to indicate positions in the matrix. As will be discussed in Section II-B, the existing sparse matrix formats suffer from load imbalance across PEs resulting in reduced system performance.

This paper presents a new sparse matrix format, called *Compressed and Balanced Sparse Row (CBSR)*, considering PE load balancing to maximize the system performance of LSTM accelerators. The experiments were carried over two different benchmarks (three different LSTM networks); speech recognition and language modeling. Our analysis shows that the system performance, equivalently, the total number of cycles to complete a single inference run, is improved by 26% on average by using the proposed CBSR format. In turn, it leads to 15% of energy saving on average in the LSTM accelerator.

The main contributions can be summarized as follows:

1. We propose **a new sparse matrix format (CBSR)** to minimize load imbalance in PEs for faster spMxV operation. By simply changing the data format, the throughput of LSTM accelerator can be boosted by 16∼38% over the well-known CSC/CSR format.

2. We present a simple **network transformation**, as a pre-processing, to remove the memory interface overhead incurred when using CBSR format. This pre-processing has negligible effect on the run-time of LSTM accelerators as it is required only once before the model deployment.

3. We perform **detailed analysis** by comparing various sparse matrix formats in a wide spectrum of LSTM/FC layers which differ in size. Prior work lacks studying the impact of sparse matrix formats in LSTM system performance.

978-3-9819263-0-9/DATE18/©2018 EDAA

	0	1	2	3	4	5	6	7
0		A				B	C	
1				D	E			
2			F					
3	G	H		I				
4		J				K		L
5							M	
6					N			O
7							P	

CSR						
PE0						
CLK	0	1	2	3	4	5
Value	A	B	C	J	K	L
PE1						
CLK	0	1	2			
Value	D	E	M			
PE2						
CLK	0	1	2			
Value	F	N	O			
PE3						
CLK	0	1	2	3		
Value	G	H	I	P		

CSC						
PE0						
CLK	0	1	2	3	4	5
Value	A	J	B	K	C	L
PE1						
CLK	0	1	2			
Value	D	E	M			
PE2						
CLK	0	1	2			
Value	F	N	O			
PE3						
CLK	0	1	2	3		
Value	G	H	I	P		

CISR [13]						
PE0						
CLK	0	1	2	3	4	
Value	A	B	C	N	O	
PE1						
CLK	0	1	2	3		
Value	D	E	M	P		
PE2						
CLK	0	1	2	3		
Value	F	J	K	L		
PE3						
CLK	0	1	2			
Value	G	H	I			

CBSR						
PE0						
CLK	0	1	2	3		
Value	A	B	C	F		
PE1						
CLK	0	1	2	3		
Value	G	H	I	M		
PE2						
CLK	0	1	2	3		
Value	J	K	L	P		
PE3						
CLK	0	1	2	3		
Value	D	E	N	O		

Fig. 2. The comparison between different sparse matrix formats in mapping computations to on-chip processing elements (PEs). Conventional sparse matrix formats suffer from load imbalance deteriorating the system performance of a LSTM accelerator.

II. Preliminary

A. Long Short-Term Memory (LSTM)

The standard RNN can process the sequence information of any length, but can only learn short history due to the vanishing/exploding gradient problem during training [6]. The LSTM, which was introduced to solve this problem, keeps updating and forgetting the information within the cell [5]. This is done by using multiple gates in the LSTM cell (Fig. 1). The computation within each cell can be decsribed by

$$\begin{aligned} i_t &= \delta\left(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i\right) \\ \tilde{c_t} &= tanh\left(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c\right) \\ f_t &= \delta\left(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f\right) \\ o_t &= \delta\left(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o\right) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \\ h_t &= o_t \odot tanh\left(c_t\right) \end{aligned} \tag{1}$$

where i_t is an input gate, $\tilde{c_t}$ is a cell gate, f_t is a forget gate, and o_t is an output gate. The cell state (c_t) is updated by forgetting some history (c_{t-1}) and accepting some new information ($\tilde{c_t}$). Note that there are eight weight matrices which dominate the memory space and the number of computations.

B. Sparse Matrix Format

To efficiently store/load a sparse matrix, various sparse matrix formats have been proposed. The most widely used formats are Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC). They are very similar except their encoding directions; CSR encodes in rows while CSC encodes in columns. The CSR (or CSC) format stores only non-zero values with their corresponding column (or row) indices. Also, the extents of rows (or columns) are stored to identify when the row (or column) changes. However, there is a load imbalancing problem when these formats are used to map computations to hardware [9]. Recently, an alternative format, named Compressed Interleaved Sparse Row (CISR) was proposed to solve the parallelization and load imbalance problems of CSR/CSC format [13]. A computation for the next row is assigned to a PE which finishes the computation first among the PEs. However, CISR may not be fully optimized as it assigns the new row to a PE without considering the size of load of the PE.

The comparison of load balancing between different sparse matrix formats is shown in Fig. 2. For both CSC and CSR format, a dot product is assigned to PE in row basis. The only difference is the encoding direction which impacts loading of input values (*not a scope of this paper*). As shown in Fig. 2, CISR format distributes the load better than CSC/CSR format. In the following section, a new sparse matrix format, named *Compressed and Balanced Sparse Row (CBSR)*, is presented. The proposed matrix format maximizes the load balancing with simple changes in CSR format.

III. Compressed and Balanced Sparse Row (CBSR)

A. CBSR Format Generation

Let us assume 4 PEs in a LSTM accelerator. In practice, the number of PEs is determined by the external memory bandwidth (refer to Section IV). For a given (sparse) weight matrix as in Fig. 3, there are five steps for a pre-processing on a server/host. These CBSR generation steps incur negligible run-time overhead as it is done only once for a given LSTM network. Note that the format generation is performed on every layer in the LSTM network with the same procedure.

Step 1: The first step is to extract the number of non-zero elements in each row for analyzing the computation load. The number of non-zero elements is denoted as 'row length (R_len in Fig. 3)'. 'R_id' is simply the index of each row in the weight matrix. When counting non-zero elements, we change the matrix structure from the one obtained by TensorFlow [14] (Fig. 4). The logic behind this will be discussed in *Step 2*.

Step 2: This is the most important step in CBSR as this simple change in the format facilitates the even distribution of computation loads to PEs. The row indices of the matrix are sorted in descending order with respect to 'R_len'. It makes PEs to first process row vectors with larger number of non-zero elements. Then, it becomes highly probable that each PE will handle a row vector with similar 'R_len' (*balanced computation mapping*). Meanwhile, the row sorting changes the order of dot products, equivalently the order of hidden neurons (h_t). It should be handled properly to avoid memory overhead during the cell state (c_t) update (refer to *Step 5*).

	0	1	2	3	4	5	6	7
0		A				B	C	
1				D	E			
2			F					
3	G	H		I				
4		J				K		L
5							M	
6					N			O
7							P	

Step 1

R_id	R_len
0	3
1	2
2	1
3	3
4	3
5	1
6	2
7	1

Step 2

R_id	R_len
0	3
3	3
4	3
1	2
6	2
2	1
5	1
7	1

Step 3

PE	R_id	R_len
0	0	3
1	3	3
2	4	3
3	1	2
3	6	2
0	2	1
1	5	1
2	7	1

Step 4

	CLK 0				CLK 1			
PE_id	0	1	2	3	4	5	6	7
Value	A	G	J	D	B	H	K	E
Col_id	1	0	1	3	5	1	5	4
R_len	3	3	3	2				
	CLK 2				**CLK 3**			
PE_id	8	9	10	11	12	13	14	15
Value	C	I	L	N	F	M	P	O
Col_id	6	3	7	4	2	6	6	7
R_len				2	1	1	1	

Step 5

Ori	Swit
0	0
1	3
2	4
3	1
4	6
5	2
6	5
7	7

Fig. 3. The process of generating Compressed and Balanced Sparse Row (CBSR) format.

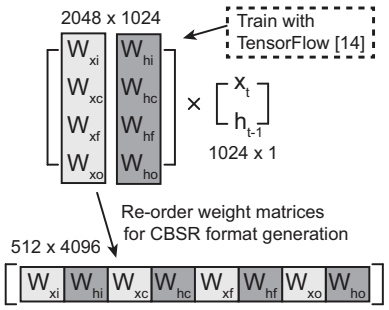

Fig. 4. The conversion of weight structure for CBSR format generation. The example shows a LSTM layer having 512 hidden neurons with 512 input neurons (either from the external input or the previous LSTM layer). It allows the LSTM hardware to update each gate in parallel.

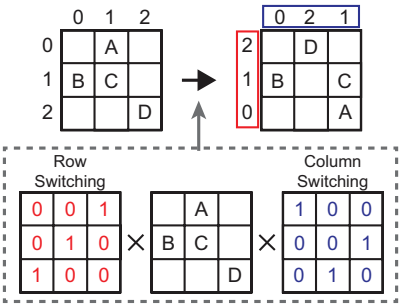

Fig. 5. A simple weight transformation by using elementary matrices for row and column switching (*Step 5*). This simple step completely removes the memory overhead when using CBSR format.

In addition, if the orders of dot products in different weight matrices (e.g. $[\mathbf{W_{xi}},\mathbf{W_{hi}}]$ and $[\mathbf{W_{xc}},\mathbf{W_{hc}}]$) are not matched, the state update at each gate will not be synchronized. To maximize the throughput of LSTM hardware, it is necessary to allow same row switching for all weight matrices within the layer. This is done by a simple conversion in matrix structure during CBSR format generation (Fig. 4).

Step 3: This step determines the PE index to which a dot product related to each row vector in the converted weight matrix is assigned. Each result corresponds to the state of one hidden neuron (h_t). Note that h_t becomes the input to the next LSTM layer as well (Fig. 6). If the number of rows in the matrix is greater than the number of PEs, as in most cases, each PE needs to cover multiple rows in CBSR. The next row with the most non-zero elements is assigned to the PE with the least computation loads. That is why PE3 gets the 5th row ('R_id' = 6) in the example shown in Fig. 3. This protocol makes the computation loads of all PEs as similar as possible.

Step 4: In this step, CBSR format is generated by using the table obtained in *Step 3*. The format is generated for each weight matrix pair (e.g. $[\mathbf{W_{xi}},\mathbf{W_{hi}}]$) of each gate that will be fetched in parallel. The data format is arranged in a sequential manner by placing one non-zero element to each PE. Initially, the first non-zero element of a row assigned to each PE will be placed (A for PE0, G for PE1, J for PE2, D for PE3). Then, a column index of each entry is stored in the format, which is similar to CSR format. After all non-zero elements in a row are scheduled, the next row assigned to that PE_id is placed. The row length, 'R_len', is also stored in the format to count the number of computations required to complete a single dot product. The data size of 'R_len' is much smaller than 'Value' or 'Col_id' (<0.5%).

Step 5: *Network transformation*. This step will be explained in detail in Section III-B.

B. Network Transformation

Due to the change in the order of dot products, it is required to store 'R_id' along with 'R_len' to identify the address of each output update (state of a hidden neuron h_t). It incurs memory access overhead at each inference run (loading the list of 'R_id') and requires on-chip memory that stores entire 'R_id' for each layer. Thus, we present a *simple network transformation* which completely removes such memory overhead caused by using the proposed CBSR format.

The network transformation is done by multiplying an elementary matrix which either changes the order of rows or columns for a given weight matrix (Fig. 5). The entire process is explained in Fig. 6. The *Step 2* in CBSR format generation re-orders the rows in a concatenated weight matrix $[\mathbf{W_x}, \mathbf{W_h}]$. This has the same effect of changing the order of hidden neurons in that LSTM layer. Effectively, the order of columns in the following matrices has to be modified to account for the change in hidden neurons. As shown in Fig. 6, the output of LSTM layer 'l' serves as 1) h_{t-1} by recurrent connection ($\mathbf{W_h}$) and 2) x_t by feedforward connection ($\mathbf{W_x}$) to the next LSTM layer '$l+1$'. The order of external inputs in the first layer never changes.

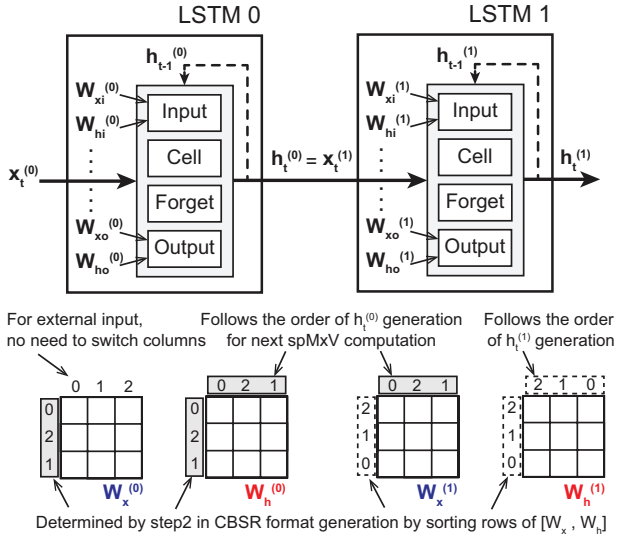

Fig. 6. An illustration of the required matrix transformation due to the row switching from CBSR format generation.

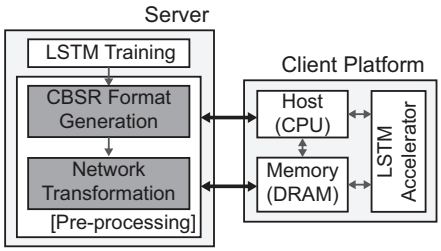

Fig. 7. The generation of CBSR format for LSTM accelerators is performed on the server as a pre-processing. The pre-processing happens only once right after the training is done for a specific application, e.g. speech recognition.

With this simple network transformation, we are able to compute dot products in sequence without knowing the initial order of rows. Note that the proposed CBSR format generation *including the network transformation* requires pre-processing time. However, this is an one-time process performed on the server right after the training is done. Thus, the run-time overhead, incurred by CBSR format preparation, does not affect the inference time on a client platform (Fig. 7).

IV. LSTM Hardware Architecture

Similar to other machine learning accelerators, the system architecture of LSTM hardware consists of three main components: off-chip memory (DRAM), on-chip data registers, and spatial array of PEs. The overall architecture of the LSTM accelerator is shown in Fig. 8. To design the LSTM hardware architecture, we first need to determine the specification of the off-chip memory. For our simulation, we assume DDR4-3200 providing 25GB/s per channel at maximum.

By using CBSR format, the data required for one PE at each clock cycle is 3.75Byte (30bits): 16bits for the weight and 14bits for the 'Col_id'. Input '$\mathbf{x_t}$' is fetched to input register as a small page (e.g. 512B or 1KB) whenever there is a page miss. With these assumptions, the number of PEs (nPE) can be determined by satisfying

$$\frac{3.75Byte \times nPE}{Clk_{PE}} \leq 25GB/s, \qquad (2)$$

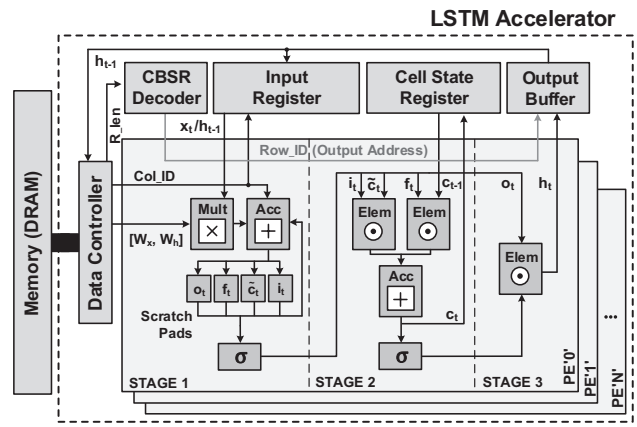

Fig. 8. The overall architecture of a LSTM accelerator using CBSR format.

where Clk_{PE} represents the clock period of PEs. To maximize the system throughput, it is natural to put as many PEs as possible that satisfies Eq. (2). Thus, we assume 32 PEs per channel in the LSTM accelerator; 128 PEs with four channels.

Each PE consists of scratch pads for intermediate gate values, multiplier (Mult), accumulator (Acc), nonlinear function unit (σ), and element-wise multiplier (Elem). The operation of PE to compute Eq. (1) divides into three stages: 1) the state update of each gate (i_t, $\tilde{c}_t$, f_t, o_t), 2) the update of LSTM cell state (c_t), and 3) the state update of hidden neurons (h_t). For the computation in *Stage 1*, the input data '$\mathbf{x_t}$ or $\mathbf{h_{t-1}}$' is retrieved from the input register addressed by 'Col_id'.

To handle spMxV operation more efficiently, an on-chip decoder for CBSR format is needed. The input to the CBSR decoder is 'R_len' and the decoder figures out when to end the current dot product by decrementing the counter by one. When the counter reaches zero, the next dot product (row) is assigned for the computation. Thanks to the network transformation, the output address to update the state of hidden neurons is sequential. Thus, we can simply use a counter that increments by word size (gray line in Fig. 8).

V. Experimental Results

A. LSTM Benchmarks

As test benchmarks, we use speech recognition (TIMIT [15]) and language modeling (Penn Treebank [16]). To check the impact of different pruning results on the load balancing, we simulate two networks for speech recognition. All simulated LSTM networks have two hidden layers between input and output layers. The details of simulated LSTM networks are provided in Table I. Each network is trained and tested with TensorFlow [14]. For the evaluation metric, Label Error Rate (LER) is used for speech recognition and perplexity is used for language modeling. For both cases, the lower metric means a well-trained LSTM network.

After the training is done, weight pruning is performed with hard thresholding. The pruning is a well-known approach to reduce data size in evaluating deep networks [9], [11]. In Table I, the sparsity of each weight matrix after pruning in each LSTM network is reported (10%∼24%). Also, the accuracy

TABLE I
THE DETAILS ON TESTED LSTM NETWORKS AFTER TRAINING AND CBSR FORMAT GENERATION

Benchmark	Dataset	Layer	Matrix Size	Sparsity [%]	$\Delta\sigma_{clk}$ over CSC/CSR [%]	Compressed Data Size [KB]	Pre-processing Time [s]	Evaluation (pre-, post-) pruning
Speech Recognition	TIMIT (39-450-450-62)	LSTM0	450x1956	24.09	-79.9	699.49	18	LER (27.3%, 27.3%)
		LSTM1	450x3600	10.70	-67.5	572.49	14	
		FC_Layer	62x450	23.47	-18.8	18.08	1	
	TIMIT (39-512-512-62)	LSTM0	512x2204	22.72	-95.2	848.81	24	LER (25.6%, 25.6%)
		LSTM1	512x4096	11.58	-98.8	830.45	23	
		FC_Layer	62x512	23.51	-42.9	22.82	1	
Language Modeling	Penn Treebank (1500-1500-1500-10000)	LSTM0	1500x12000	11.19	-74.0	7136.29	573	Perplexity (78.66, 76.74)
		LSTM1	1500x12000	10.77	-93.6	6865.24	549	
		FC_Layer	10000x1500	10.67	-98.4	5865.54	1243	

of the LSTM network may change due to pruning and we set the threshold not to sacrifice the accuracy. For language modeling, the quality of the trained model got even better after the pruning. After pruning, each weight matrix becomes sparse and we convert it to a sparse matrix format (CSR, CSC, CISR or CBSR). The compressed data size is very similar between different formats and that of CBSR is reported in Table I.

B. Performance Improvement by Load Balancing

With the presented benchmarks, we evaluate the performance of a LSTM accelerator using different sparse formats. As explained in Section IV, we assume DDR4-3200 as an external memory providing 25GB/s of bandwidth per channel. Having four memory channels in the client platform allows the LSTM accelerator to keep 128 PEs. Throughout the experiment, we assume 128 PEs operating at 200MHz in updating LSTM cell states.

The run-time of the LSTM accelerator is evaluated by the worst clock cycles consumed by PEs. Even if a PE finishes the scheduled computations, it has to wait for other PEs with more computations to finish too. The cycle-level simulator reports the total clock cycles required for each PE to complete the state update of each layer. The normalized run-time per layer for a single inference is shown in Fig. 9. Note that the LSTM accelerator runs on the input sequence (multiple inferences). The level of load balancing can be checked by the standard deviation of the required clock cycles of PEs (σ_{clk}). The reduction of σ_{clk} is reported in the 6th column of Table I.

For the last FC layer in TIMIT dataset, there is no performance gain as the number of rows is smaller than the number of PEs (no balancing opportunity). However, it has negligible effect on the total run-time of the entire network as the computations are dominated by the first two LSTM layers. For the second LSTM layer in TIMIT-512 network, the performance improvement is significant when using CBSR over CSC (or CSR). When we compare the histogram of non-zero elements per row between LSTM0 and LSTM1 layers (Fig. 10), the distribution of LSTM1 layer is wider and has two peaks (unbalanced). It indicates that the pruning outcome affects the efficiency of load balancing.

The total run-time required to run the entire network for each benchmark is shown in Table II. Compared to the well-known CSC/CSR format, the system throughput is improved by 16∼38%. In [9], the balance-aware pruning (lossy approach) is proposed for CSC format, yet it incurs retraining overhead to check validation accuracy after each pruning. As both CISR and CBSR balance out the computation loads

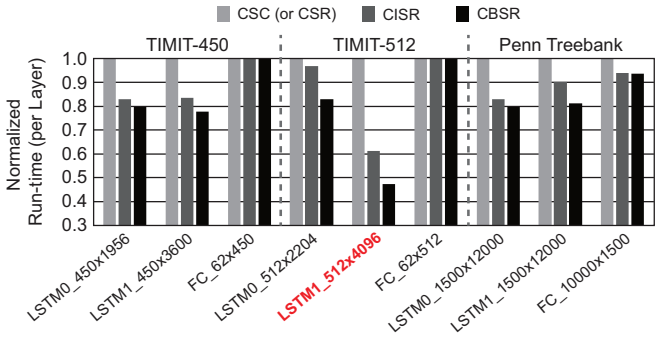

Fig. 9. The normalized run-time (per layer) of LSTM accelerators using different sparse matrix formats.

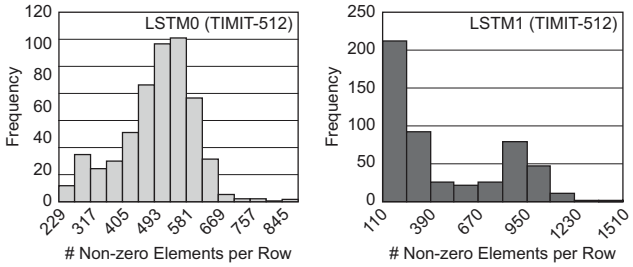

Fig. 10. The histogram of the number of non-zero elements per row in LSTM layers of TIMIT-512 network.

without allowing any computation error, they are more reliable and require significantly less pre-processing time. Between the two, CBSR balances the computation loads better which results in 4.5∼13.4% speed-up (refer to Table II). This is because CISR blindly assigns the next row to a PE that completed its prior computations.

C. Power Analysis

The early completion of inference reduces the power consumption as well. The total dynamic power of PEs consumed by the useful computations is same for all matrix formats. If the computation loads are not balanced, however, some PEs consume the leakage power and the global clock network needs to be toggled to run the remaining PEs. We assume that a single PE can locally gate the clock signal to avoid switching within the PE. Thus, the total energy consumption of an accelerator is computed by

$$E_{LSTM} = T_{clk} \cdot \sum_{i=1}^{nPE} \{(nWorst - nCycles_i) \cdot (P_{leak} + P_{clk}) + nCycles_i \cdot P_{tot}\}, \quad (3)$$

where T_{clk} is the clock period, $nCycles_i$ is the number of clock cycles of PE$_i$, $nWorst$ is the maximum of $nCycles_i$

TABLE II
THE PERFORMANCE OF COMPUTING ENTIRE NETWORK (SINGLE INFERENCE) USING DIFFERENT SPARSE MATRIX FORMATS

Total Run-time [μs] (Improvement over CSC/CSR)			
Benchmark	CSC/CSR [9]	CISR [13]	CBSR
TIMIT-450	22.46 (-)	19.64 (-13%)	18.13 (-19%)
TIMIT-512	35.25 (-)	26.72 (-24%)	22.01 (-38%)
Penn Treebank	261.87 (-)	231.27 (-12%)	219.51 (-16%)

(last ending PE), P_{leak} is the leakage power, P_{clk} is the clock network power, and P_{tot} is the total power consumed by 128 PEs and the clock network. The LSTM accelerator as shown in Fig. 8 is designed and synthesized in 65nm CMOS technology. The power is estimated after PnR and the energy consumption is reported in Table III. The result shows that the LSTM hardware using CBSR can save energy consumption by 9∼22% over CSC/CSR (and by 2∼7% over CISR).

VI. DISCUSSION

The experimental results in Section V were obtained by assuming 128 PEs in the LSTM accelerator. The memory system with higher bandwidth (e.g. HMC, HBM or Wide I/O) allows more PEs in the LSTM accelerator. Thus, we simulated the LSTM system with higher number of PEs (256 PEs) for the same benchmarks. The simulation result shows that the load balancing with CBSR format allows higher speed-up (Fig. 11). For TIMIT dataset (an average-sized LSTM network), however, the performance improvement may become less for CISR or CBSR. This is because only two rows are mapped to each PE (with 256 PEs) and the chance of balancing the loads is lowered. Still, CBSR shows better trend than CISR as it accounts for the load of the next row assignment (check TIMIT-450 result). With 256 PEs, CBSR performs 25∼30% better than CSC/CSR and 10∼19% better than CISR. In terms of the energy consumption, the benefit also increases when using CBSR format (refer to Table III). It is improved by 15∼27% over CSC/CSR and 6∼18% over CISR.

VII. CONCLUSION

This paper presents a new sparse matrix format (CBSR) that maximizes the throughput of LSTM accelerators without allowing extra hardware overhead. In addition, a simple network transformation, which eliminates the memory overhead incurred by the CBSR format, is presented. The proposed format effectively distributes the computation loads over PEs, which results in 16∼38% of speed-up and 9∼22% of energy saving compared to the well-known CSC/CSR format. Lastly, we predict even more improvement in performance using the CBSR if the improved input reuse characteristic is considered. Detailed analysis remains as our future work.

ACKNOWLEDGEMENT

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the "ICT Consilience Creative Program" (IITP-R0346-16-1007) supervised by the IITP(Institute for Information & communications Technology Promotion) and SK hynix Inc.

TABLE III
THE ENERGY CONSUMPTION OF COMPUTING ENTIRE NETWORK USING DIFFERENT SPARSE MATRIX FORMATS

Benchmark	128 PEs [mJ]			256 PEs [mJ]		
	CSC	CISR	CBSR	CSC	CISR	CBSR
TIMIT-450	0.050	0.044	0.043	0.055	0.050	0.040
TIMIT-512	0.068	0.058	0.053	0.077	0.068	0.062
Penn Treebank	0.61	0.56	0.55	0.65	0.59	0.56

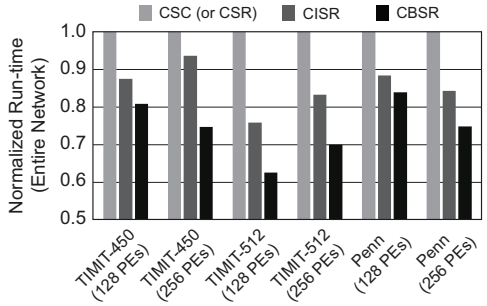

Fig. 11. The impact of the number of available PEs on the efficiency of the proposed CBSR format.

REFERENCES

[1] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *abs/1406.1078*, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[2] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *Proc. IEEE Intl. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649, March 2013.

[3] M. Köck and A. Paramythis, "Activity sequence modelling and dynamic clustering for personalized e-learning," *User Modeling and User-Adapted Interaction*, vol. 21, no. 1, pp. 51–97, April 2011.

[4] H. Yu *et al.*, "Video paragraph captioning using hierarchical recurrent neural networks," *abs/1510.07712*, 2015. [Online]. Available: http://arxiv.org/abs/1510.07712

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] S. Hochreiter *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[7] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan 2017, pp. 629–634.

[8] P. Ouyang, S. Yin, and S. Wei, "A fast and power efficient architecture to parallelize LSTM based RNN for cognitive intelligence applications," in *Proc. Design Autom. Conf. (DAC)*, June 2017, pp. 63:1–63:6.

[9] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse lstm on fpga," in *Proc. ACM/SIGDA Intl. Symp. Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 75–84.

[10] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 52, no. 1, pp. 127–138, Jan 2017.

[11] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. Intl. Symp. Computer Architecture (ISCA)*, 2016, pp. 243–254.

[12] A. See, M. Luong, and C. D. Manning, "Compression of neural machine translation models via pruning," *abs/1606.09274*, 2016. [Online]. Available: http://arxiv.org/abs/1606.09274

[13] J. Fowers *et al.*, "A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication," in *IEEE Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 36–43.

[14] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," *abs/1603.04467*, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[15] J. S. Garofolo *et al.* (1993) TIMIT acoustic-phonetic continuous speech corpus. [Online]. Available: https://catalog.ldc.upenn.edu/ldc93s1

[16] A. Taylor, M. Marcus, and B. Santorini, "The Penn Treebank: an overview," in *Treebanks*. Springer, 2003, pp. 5–22.