IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 17, 2019, accepted May 12, 2019, date of publication May 16, 2019, date of current version May 28, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2917312

Acceleration of LSTM With Structured Pruning Method on FPGA

SHAORUN WANG, PENG LIN, RUIHAN HU, HAO WANG, (Member, IEEE), JIN HE, (Senior Member, IEEE), QIJUN HUANG, AND SHENG CHANG, (Senior Member, IEEE)
School of Physics and Technology, Wuhan University, Wuhan 430072, China

Corresponding author: Sheng Chang (changsheng@whu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61874079, Grant 61574102, and Grant 61774113, in part by the Fundamental Research Fund for the Central Universities, Wuhan University, under Grant 2042017gf0052, in part by the Wuhan Research Program of Application Foundation under Grant 2018010401011289, and in part by the Luojia Young Scholars Program.

ABSTRACT This paper focuses on accelerating long short-term memory (LSTM), which is one of the popular types of recurrent neural networks (RNNs). Because of the large number of weight memory accesses and high computation complexity with the cascade-dependent structure, it is a big challenge to efficiently implement the LSTM on field-programmable gate arrays (FPGAs). To speed up the inference on FPGA, considering its limited resource, a structured pruning method that can not only reduce the LSTM model's size without loss of prediction accuracy but also eliminate the imbalance computation and irregular memory accesses is proposed. Besides that, the hardware architecture of the compressed LSTM is designed to pursue high performance. As a result, the implementation of an LSTM language module on Stratix V GXA7 FPGA can achieve 85.2 GOPS directly on the sparse LSTM network by our method, corresponding to 681.6-GOPS effective throughput on the dense one, which shows that the proposed structured pruning algorithm makes 7.82 times speedup when only 1/8 parameters are reserved. We hope that our method can give an efficient way to accelerate the LSTM and similar recurrent neural networks when the resource-limited environment is emphasized.

INDEX TERMS FPGA, hardware acceleration, LSTM, pruning.

I. INTRODUCTION

Recurrent Neural Network (RNN) is a class of neural networks which has the capability of capturing long-range dependencies in sequential and temporal data. Long Short-Term Memory (LSTM) [1] network is one of the most popular types of RNNs, which achieves great success in various applications such as speech recognition [2], machine translation [3], scene analysis [4], etc. Since these applications demand real-time processing and low power consumption, customized hardware acceleration of LSTMs becomes an important way to improve computing speed, meeting the above requirements.

However, there are some challenges in accelerating LSTM. Due to the recurrent nature of RNNs, the computing pattern is quite complex as it needs to process temporal data and dependencies. Since RNN is fully connected, a large number of weight memory accesses is required. As a result, high computation complexity and large memory footprint become the two main barriers in LSTM accelerator design.

As demonstrated in recent works, GPUs [5]–[7], FPGAs [8]–[10] and ASICs [11], [12] are all employed as accelerator to speedup LSTMs. Among the numerous platforms, FPGA shows great potentiality of promising solution for accelerating the neural networks, as it can attain high performance with flexible reconfigurability. Many outstanding works on CNNs' acceleration [12]–[15] have revealed that FPGA can achieve high performance nearly comparable to GPU but with much less energy consumption. What's more, by the virtue of cutting-edge FPGA design methodologies such as High-level Synthesis (HLS) and OpenCL [16], it becomes easier to design complex systems on FPGAs. Under OpenCL framework, mapping a neural network to FPGA hardware is available through designing OpenCL kernels and then synthesizing them to RTL specifications. So, in this work we choose OpenCL based FPGA to explore the scheme of LSTM acceleration.

The associate editor coordinating the review of this manuscript and approving it for publication was Mauricio Silveira.

Recently, some works about accelerating RNNs are done for FPGA's high performance, low power and reconfigurability. References [8]–[10], [17] focused on exploring the parallelism for RNN but ignored the compression techniques in algorithm level. References [18]–[21] capitalized on the data sparsity of RNNs to speed up the inference procedure. The ESE [18] proposed by Han et al. emphasized on optimizing memory footprint with pruning scheme, which can compress the dense weight matrices into sparse ones. They achieved 6.2× speedup over the dense model by pruning the LSTM model to 10% non-zeros. C-LSTM [19] proposed a method that reduces the weight matrices in LSTM inference model in a spruced manner by using block circulant matrix and applying FFT algorithm to accelerate the computation-intensive circulant convolution operator. The DeltaRNN [20] explored the temporal dependency of input and activation vectors on GRU [22] RNNs by employing Delta Network Algorithm which creates sparsity in input and activation vectors. The BBS [21] proposed Bank-Balanced Sparsity to compress the weight matrices and achieved good efficiency for LSTM inference.

In this work, focusing on high computation efficiency on resource-limited platform, a novel structured pruning method for compressing model size is proposed in algorithm level. It can generate the sparse weight matrices in one-step training process, and no repeated retraining used in ESE's pruning scheme is needed. Moreover, the distribution of the sparse weight matrices produced by our compression method is balanced, which is superior to the traditionally skewed distribution that is likely to cause unbalanced workloads among parallel compute units. As a result, our compress algorithm has the ability to prune the whole columns of weights matrices instead of removing the small weights that the common pruning schemes do. Hence, we get the regular sparse weight matrices avoiding unbalanced workloads among parallel compute units, and meanwhile the logic design of sparse matrix - vector multiplication is simplified. After a model is training well with that structured pruning method, an efficient hardware architecture is also proposed to accelerate the sparse LSTM model in FPGA platform.

Overall, the contributions of this paper are listed as:

- A novel structured pruning method is proposed to realize model compression for LSTM, which can complete the compression in training stage and directly produce balanced sparse weight matrices. It can accelerate LSTM as it reduces computation complexity because of sparse weights. And therefore, high efficiency of speedup is attained.
- An efficient hardware acceleration architecture is proposed for structured pruning. The acceleration scheme not only explores fully parallel LSTM computing but also considers coarse-grained layer pipeline.
- An OpenCL based implementation of LSTM models with structured pruning method and well-optimized architecture is presented. The results demonstrate that the proposed design can achieve high computing efficiency.

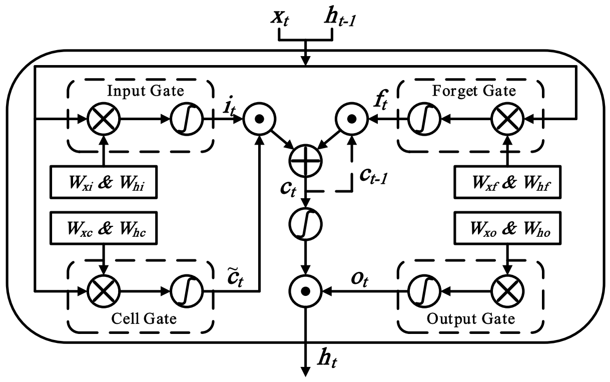

FIGURE 1. Typical long-short term memory cell architecture.

II. BACKGROUND

LSTM is one of the most widely used RNN variants. It has been proposed to deal with the well-known exploding/vanishing gradient problems. Since a gate mechanism is introduced to control the modeling of temporal sequence, it can capture long- term dependencies more easily than conventional RNNs. The architecture of a typical LSTM cell has four gates shown in Fig. 1, as the equations described as follows:

$$i_t = \delta(W_{xi}X_t + W_{hi}h_{t-1} + b_i) \quad \text{input gate} \tag{1}$$
$$f_t = \delta(W_{xf}X_t + W_{hf}h_{t-1} + b_f) \quad \text{forget gate} \tag{2}$$
$$o_t = \delta(W_{xo}X_t + W_{ho}h_{t-1} + b_o) \quad \text{output gate} \tag{3}$$
$$\tilde{c_t} = tanh(W_{xc}X_t + W_{hc}h_{t-1} + b_c) \quad \text{cell gate} \tag{4}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \quad \text{memory cell} \tag{5}$$
$$h_t = o_t \odot tanh(c_t) \quad \text{hidden state} \tag{6}$$

X_t, h_{t-1} and c_{t-1} denote input data, hidden data and internal cell state. A LSTM unit contains gate structures that learn to determine when to accept input information and update cell state. W_{xj}, W_{hj}, and b_j ($j = i, f, o, c$) are parameters to be learned during the training process. $\odot$ denotes element-wise multiplication. σ and tanh are element-wise nonlinear activation functions, where

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{7}$$

i_t, f_t and o_t are the input, forget and output gates respectively, c_t is the current state of LSTM, h_{t-1} is the previous output, X_t is the current input at time t. And matrix-vector multiplication is applied to compute $i_t, f_t, o_t, \tilde{c}_t$.

III. PROPOSED STRUCTURED PRUNING ALGORITHM

In this work, we propose a structured pruning algorithm that can remove whole columns of weight matrix. Compared with the methods pruning small value weights, pruning the entire columns of redundant weights has some advantages. In one hand, the sparse weight matrix produced by our method can

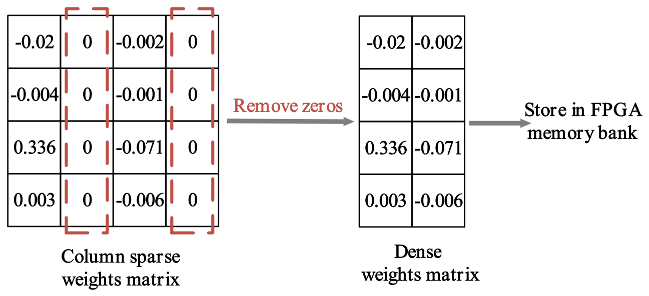

FIGURE 2. Store column sparse weight matrix with dense format.

be stored as a dense matrix, while ordinary sparse matrix needs to be stored in particular formats, like CSC, CRC et al., which consume more memory storage at the same sparsity. Fig. 2 gives the weights storage strategy as the form of dense matrix in FPGA memory bank. On the other hand, thanks to the regularity of our sparse matrix, it's not necessary to design complex logic for sparse matrix vector multiplication. It can work almost the same as dense one in addition with input sparse encode.

Our structured pruning algorithm directly acts on the weights in the process of network training, described by the following formulation:

$$F\left(W_{ij}\right)=\begin{cases}0, & S_j\leq C_w\\ K_wW_{ij}, & S_j>C_w\end{cases}\tag{8}$$

where F represents the function of pruning weights, W_{ij} denotes the weight located at the spatial position on *i-th*row and *j-th* column of the original weight matrix. And S_j is the sum of the absolute value of weights in *j-th* column, as shown in (9). C_w is the critical value to prune the weights away. S_j is calculated as (10).C_w is not a pre-defined constant and is self-adaptive along with the training iterations. C_w which is related to the sparsity and S_j can be chosen as the *n-th* maximum value of S_j according to the pruning rate. For example, if the weight matrix has 100 columns and the pruning rate is 0.8, then C_w is the 20'th maximum value of S_j . Since the weights change every iteration, C_w is not a constant value because of the change of S_j.

$$S_j=\sum_{i=0}^{n}|W_{ij}|\tag{9}$$

$$K_w=\frac{S_j-C_w}{S_j}\tag{10}$$

If the sum of absolute value of weights in the same column is smaller than a threshold, then the whole column weights are pruned away. Specifically, apply the compression algorithm to the weights first, and get the pruned weights $W^{'}$. Then start LSTM unit's forward computation with the sparse weight matrix $W^{'}$ instead of the original ones, replacing the W in (1) $\sim$ (4) with $F(W)$ in brief.

During the training procedure the weights are updated usually in the way described in (11) for a standard neural network. While back propagation is modified to (12), introducing the structured pruning algorithm. Where l denotes the learning rate, and $\frac{\partial\varepsilon}{\partial W_{ij}}$ is the partial derivate of weights which can be written in the form of (13) based on the chain rule. From (8), we can get $\frac{\partial F(W_{ij})}{\partial W_{ij}}$ as (14). In order to avoid the correspond weight maintaining constant 0 once $S_j < C_w$ at some iterations, we set $\frac{\partial F(W_{ij})}{\partial W_{ij}}=1$, so (11) can be rewritten as (12). It doesn't change the direction of gradient convergence when $\frac{\partial F(W_{ij})}{\partial W_{ij}}=1$. So, it will not cause the failure of network convergence. The computation of $\frac{\partial\varepsilon}{\partial W_{ij}{'}}$ is the same as the original LSTM network. The training process has no much difference compared to a standard one. Only the weights need to be processed by the structured pruning algorithm every time before start LSTM layer and the back propagation following (12).

$$W_{ij}=W_{ij}+l\frac{\partial\varepsilon}{\partial W_{ij}}\tag{11}$$

$$W_{ij}=W_{ij}+l\frac{\partial\varepsilon}{\partial W_{ij}{'}}\tag{12}$$

$$\frac{\partial\varepsilon}{\partial W_{ij}}=\frac{\partial\varepsilon}{\partial W_{ij}{'}}\bullet\frac{\partial W_{ij}{'}}{\partial W_{ij}}=\frac{\partial\varepsilon}{\partial W_{ij}{'}}\bullet\frac{\partial F(W_{ij})}{\partial W_{ij}}\tag{13}$$

$$\frac{\partial F(W_{ij})}{\partial W_{ij}}=\begin{cases}0\\ K_w\end{cases}\tag{14}$$

After finishing the training procedure, the dense weight matrix W and sparse weight matrix $W^{'}$ are got. In the inference stage, $W^{'}$ is imported into the LSTM units for computing. The pruning algorithm we proposed can prune the weights along with the training iterations rather than pruning the weights after training. By this way, the repeated retraining process is eliminated, and the efficiency is enhanced. Besides that, since $W^{'}$ is the result of the network converging to the optimal point in training, the accuracy of network is ensured.

IV. HARDWARE METHODOLOGY

In this section, the hardware implementation scheme of LSTM acceleration based on the proposed structured pruning method is presented. By designing massive parallel processing elements combining with coarse-grained layer pipeline strategy, hardware's potential for high-performance computation is well brought out.

A. OVERALL LSTM ARCHITECTURE

The overall LSTM unit architecture designed for the entire model is shown in Fig. 3. There are three main components, SpIn, MxV and Act, respectively. The SpIn is responsible for generating the sparse input vector for MxV compute unit. The MxV executes the matrix-vector multiplication (MVM) using the sparse output of SpIn. It contains four parallel channels to complete matrix-vector multiplication for $i_t, f_t, o_t, \tilde{c_t}$ synchronously. The Act realizes the action of computing i_t, $f_t, o_t, \tilde{c_t}$, through the activation function based on the results from MxV, and further calculates c_t, h_t for the current timestep. The architecture and the function of each block are presented subsequently in detail.

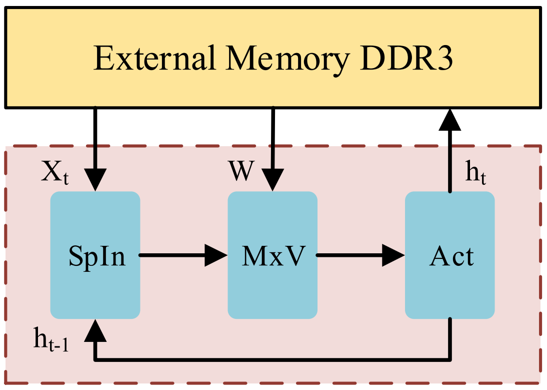

FIGURE 3. The overall hardware architecture.

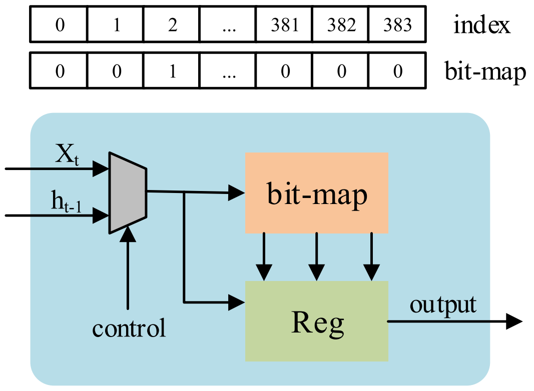

FIGURE 4. The structure of the SpIn.

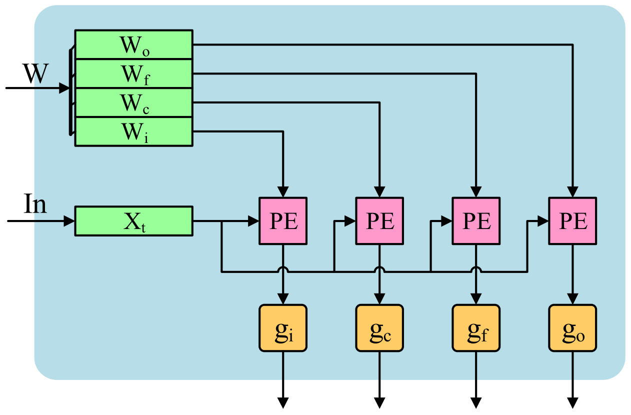

FIGURE 5. The architecture of MxV unit.

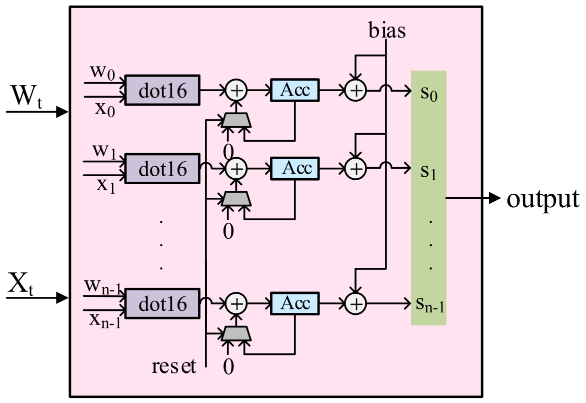

FIGURE 6. The architecture of PE.

1) SPARSE INPUT ENCODING

After pruning, the sparse weight matrix is got, in which the most columns of weights are zero. In order to reduce the memory cost, only the nonzero part of weights is loaded to FPGA memory bank, as Fig. 2 demonstrates. So before executing the sparse matrix - vector multiplication, the effective elements are extracted from the entire input vector to generate the new input vector for MxV. This process of sparse input encoding can be easily realized by designing a bit-map look-up table. '1' indicates that the data under the current index is valid, while '0' indicates invalid. The structure of SpIn is shown in Fig. 4. According to the bit-map, the valid data are stored in registers. After the data flow of X_t, h_{t-1}, passes through the block finished, the valid data are sent to MxV for computation. Hence, the sparse matrix - vector multiplication is transformed into the dense one with sparse input encode.

2) MATRIX - VECTOR MULTIPLICATION

The computation of the LSTM unit is defined as (1) –(6), and its core is the matrix -vector multiplication applied to generate i_t, f_t, o_t,$\tilde{c}_t$. Through the optimization of the matrix -vector multiplication, good performance can be attained.

Fig. 5 shows the optimized matrix -vector multiplication process. Considering the independence of i_t, f_t, o_t, $\tilde{c}_t$ calculations, they can be processed in parallel. As shown in Fig. 5, four channels are applied to transfer W_i, W_f,W_o, W_c respectively, forming a daisy chain for MVM computing. The pattern employed on MVM calculation is data pipeline and parallel computation, which can provide high efficiency. Each process element (PE) completes N times dot product accumulations of 16 elements vector every clock and 4 PEs work simultaneously to calculate g_i, g_f, g_o, g_c. The structure of PE is shown in Fig. 6. Each PE products a N elements vector by working N times dot product in parallel, making total 64N times synchronous multiplication accumulation (MAC) operations every clock.

3) ACTIVATION FUNCTION

In LSTM unit, two activation functions, Sigmoid and tanh, are employed as (7). We can see that they are both complex exponential functions, which are not hardware friendly. Like the strategy applied in [9], [19], piecewise linear functions are also utilized in this paper to approximate them. In detail, the Sigmoid and tanh functions are approximated by the piece-wise linear functions with 22 segments, and the error rates are less than 1% compared with the original functions. The piece wise linear functions can be represented as $y = kx + b$ and only the associate parameters of k and b need to be stored, which is beneficial to reduce the implementation complexity and resource consumption.

Since the dependence between i_t, f_t,o_t, $\tilde{c}_t$ and c_t, h_t, several stages are needed. As shown in Fig. 7, Act has 3 pipeline

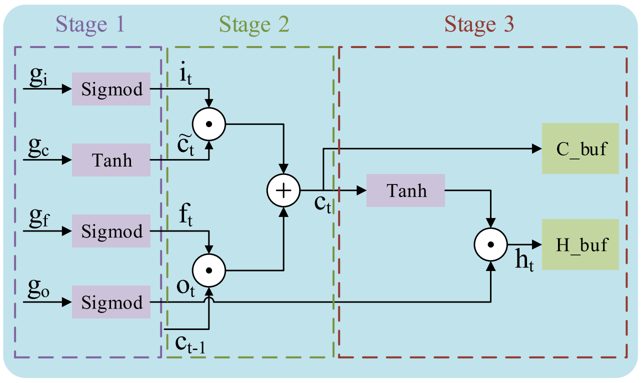

FIGURE 7. Data flow of Act module.

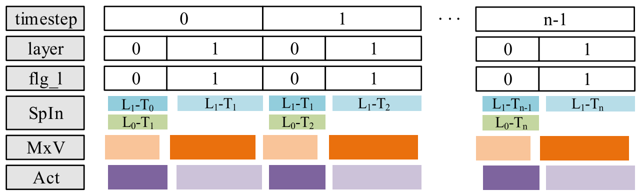

FIGURE 8. Working state of three modules along with timestep.

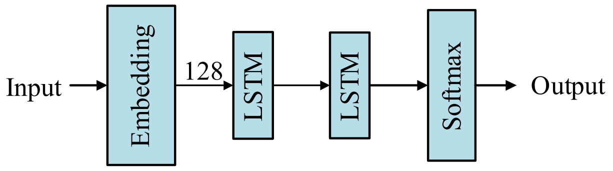

FIGURE 9. Language model.

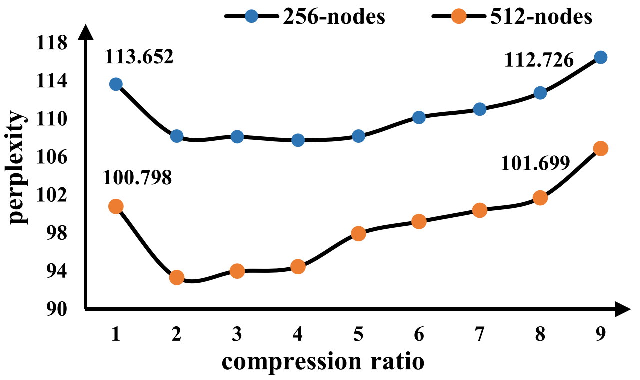

FIGURE 10. The influence of compression ratio to the performance of language model, where compression ratio = number of all weights/ number of none zero weights.

stages to produce the final activation. So the LSTM formulation is divided into 3 steps corresponding to each pipeline stage. Because c_t, h_t are in relation to the computation of next timestep, two RAM blocks are utilized to store c_t, h_t respectively.

B. LAYER PIPELINE STRATEGY

Due to the dependence of RNNs, the layers in LSTM are computed in series for the current timestep, and the computation of the next timestep relies on the results of current timestep. In order to improve the computing efficiency, the computation of different layers along with timestep needs to be well pipelined. The goal is to reduce the latency between different layers and adjacent timesteps.

Since the computing pattern is almost the same among different layers except different input, the three modules, SpIn, MxV and Act, can support the calculation of different layers. FIFOs are adopted to transfer data among these three modules. By this way, the computation of SpIn and Act can be overlapped with MxV. Due to MxV undertakes the most computation of LSTM, it is important to improve the computing efficiency of MxV to promote the overall system's performance. Using the FIFO strategy to transfer weights of different layers, the computations of several layers are put together. The MxV computation of next layer can start simultaneously as the input procedure is completed via SpIn module. A control signal *flg_l*is introduced to denote the computing schedule of different layers. According to the switch state of *flg_l*, our designed LSTM architecture can automatically practice the processing of corresponding layer. Fig. 8 shows the working state of three modules along with timestep. Following this fluent data flow, the computation between adjacent layers is pipelined.

In general, we propose an efficient LSTM architecture. By the optimization as high parallelism and effective pipeline, it can achieve high performance in hardware.

V. ANALYSIS AND EVALUATION

The performance analysis and the evaluating of our proposed LSTM methodology are presented in this section.

A. MODEL PERFORMANCE ANALYSIS

The proposed structured pruning algorithm is applied to compress the language LSTM model, as shown in Fig. 9. The language model has two LSTM layers with word embedding as input and Softmax layer as output. In this work two different size language models with 256 nodes or 512 nodes are tested for the evaluation of the proposed structured pruning method. The proposed compression method is evaluated under Penn Treebank (PTB) Corpus [23], which contains 5017k characters with only 50 different values. We split the 90% of data for training set and 5% for validation and test set, respectively.

As shown in Fig.10, the proposed pruning algorithm presents powerful ability in pruning network without sacrificing performance. The original model without compression has many redundant weights having no contribution to performance promotion can be pruned away without accuracy loss [24]–[26]. After pruning moderate amount of weights, the generalization capacity of the model has been strengthened. However, if overmuch weights are pruned away, the performance will decrease. Put it all together, the perplexity decreases firstly and then increases with the increase of compression ratio. And the language model gets worse performance when compression ratio is larger than 8. The proposed pruning algorithm can remove the ineffective weights to improve performance but it will reach the limit when the

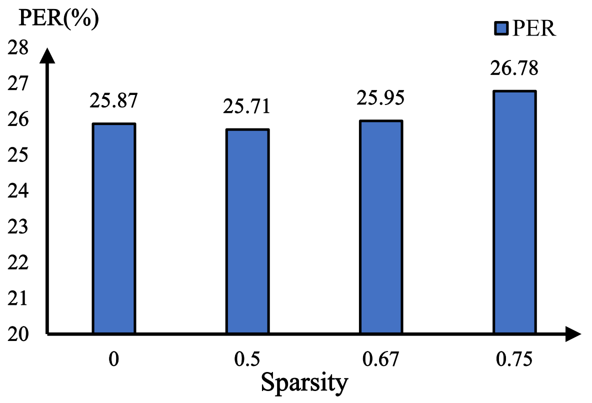

FIGURE 11. The influence of weigths sparsity to the performance of acoustic model.

compression ratio is fairly high. Our compression strategy adopted to the LSTM language model makes it attaining superior performance to standard ones, even the 7/8 of weights are removed. Our method can achieve better compression on language model than the ISS [27] which can achieve 7.48× operations reduction and about 3.03× model size reduction. Specifically, our method can attain both 8× operations and model size reduction on the language model. What's more, thanks to the identity of generating regular sparse weight matrix in the algorithm, it's favorable to hardware friendly design on FPGA.

The proposed structured pruning algorithm is also applied to an acoustic model which has two 512-nodes LSTM layers and one fc layer. The input of the acoustic model is a 40-dimensional log mel-frequency filterbank feature vector with energy and their delta and double-delta values, resulting in a 123-dimensional vector. The Fig.11 presents details of trade-offs among different sparsity evaluated on TIMIT dataset. The acoustic model compressed by our method can achieve 0.75 sparsity with only 0.91% PER decrease. Since the acoustic model is compressed 4 times in columns, it will achieve about 4 times speedup when it's implemented on FPGA.

B. FPGA IMPLEMENTATION PERFORMANCE ANALYSIS

The total number of multiplication and addition operations of the two LSTM layers with inputs length of N_{in} and N_{out} outputs of the language model or acoustic model per timestep can be approximately estimated as:

$$N_{op} = (N_{in} + 3 \times N_{out}) \times N_{out} \times 4 \times 2 \tag{15}$$

Since MVM operations dominate the total number of operations, cost of element-wise multiplications, additions and activation functions are ignored. Then the effective throughput is defined as:

$$T.eff = \frac{N_{\mathrm{op}} \times Timesteps}{\mathrm{Time}} \tag{16}$$

Each DSP block in our FPGA can be instantiated as a MAC unit to perform 16-bit fixed point multiplications or 32-bit floating point multiplications and accumulation. In this case, the potential peak throughput can be calculated in terms of the number of DSPs instantiated in the design:

TABLE 1. Resource utilization.

	ALMs	RAMS	DSPs
Available	234720	2560	256
256-nodes LM	88416(38%)	554(22%)	160(63%)
512-nodes LM	106712(45%)	822(32%)	160(63%)
512-nodes AM	120248(51%)	1238(48%)	160(63%)

$$T \cdot peak = 2 \times N_{DSP} \times f_{\max} \tag{17}$$

where N_{DSP} is the number of DSPs consumed by the whole system, and f_{max} is the max working frequency. (17) is to evaluate the baseline of potential peak throughput, considering one DSP can complete one addition and multiplication. 16bit fixed point data is employed in this work to improve the DSP efficiency. Each DSP can complete two additions and multiplications when 16bit fixed point data is used, making $T.peak$ doubled on the base of (17). Then, the computation efficiency measuring how efficiently the hardware makes use of DSP is defined as:

$$Com.Efficiency = \frac{T.eff}{T.peak} \tag{18}$$

As mentioned above, the whole system is able to perform $64N$ MAC operations every clock, where N indicates the parallelism of dot16 computation of each PE. The larger N is, the less running time is required. According to N, the total number of the DSPs utilized is the sum of three blocks' consumptions:

$$N_{DSp} = N_{SpIn} + N_{MxV} + N_{Act} = 32 \times N + 8 \times N \tag{19}$$

Considering the limited resource of FPGA and the size of LSTM unit, the design parameter N should be reasonably chosen to achieve satisfying throughput.

C. HARDWARE EVALUATION

Our proposed implementation of the LSTM models is evaluated on DE5-NET which employs a Stratix V GXA7 FPGA and has two 4GB DDR3 memories. The Intel®FPGA SDK for OpenCL$^{\mathrm{TM}}$is used to synthesize the C/C++ based LSTM design onto FPGA. The DE5-NET board is connected to the host via PCIe interface to build OpenCL heterogeneous environments with an Intel Core i5-6500 CPU.

The language models have two LSTM layers with 128 inputs length and 256 or 512 outputs respectively and the acoustic model has two 512-nodes LSTM layers. Set $N = 4$ for the three different models considering the FPGA's resource limitation. The total resource utilization of different LSTM models is listed in Table 1. For the three models, the model size and weights sparsity are different, which leads to different logic and RAM blocks utilization.

The influence of the sparsity of weights pruned by our proposed pruning method is explored on FPGA, as shown

TABLE 2. Comparison with relative works.

	ESE [18]	DeltaRNN [20]	BBS [21]	Ours		
				256-nodes LM	512-nodes LM	512-nodes AM
Precision	12-16 bit	16bit	16bit	16bit	16bit	16bit
Platform	KU060	XC7Z100	Arria 10 GX1150	Stratix V GXA7	Stratix V GXA7	Stratix V GXA7
Frequency/ MHz	200	125	200	245	225	220
DSP	1504	768	1518	160	160	160
Effective Throughput/GOPS	2520	1198	2432.8	652	681.6	339.7
Computation Efficiency	418.88%	623.96%	400.66%	831.63%	946.63%	482.61%

In this table LM means language model and AM means acoustic model.

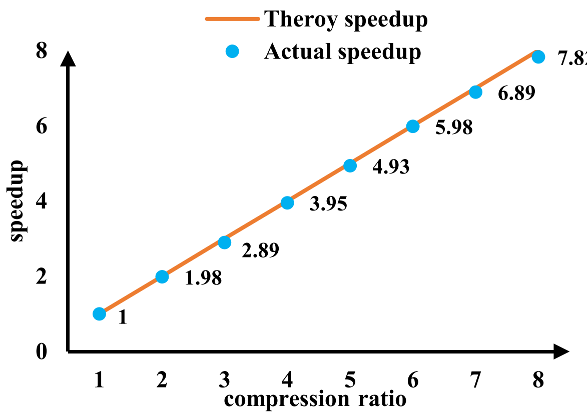

FIGURE 12. Speedup with respect to compression ratio.

in Fig. 12. Only the inference stage is implemented on FPGA, and the compression process is completed in the training procedure. The measured results fit the theory exactly, that the speedup increases with the raise of compression ratio. With 8 times compression, the sparse model can achieve 7.82 times speedup over the dense model, which is more outstanding than other works. Because the weight matrix is pruned in the column direction, the speedup has linear correlation with sparsity, which infers our method has good potential for large scale network applications where high compression ratio is important under limited resources.

To verify our design's capacity, it is compared with some state-of-art works in Table 2. ESE used deep compression method to compress the model 8.9×, and DeltaRNN created sparsity in input and activation vectors, and BBS proposed Bank-Balanced Sparsity pattern to accelerate LSTM. These models have different size and weights sparsity leading to different loop iterations in MVM calculation, thus these models implemented on FPGA using OpenCL complier have different frequency. From it one can see, our FPGA implementation of 256-nodes language model is able to attain 81.5 GOPS on the 8× sparse model being equivalent to 652 GOPS effective throughput for dense LSTM. Computation efficiency of 831.63% is achieved under the condition of 7/8 weights pruned away. And the 512-nodes language model can achieve 681.6GOPS effective throughput and 946.63% computation efficiency when the model is compressed eight times. Because the acoustic model is only 4× compressed, effective throughput of 339.7 GOPS and computation efficiency of 482.61% are achieved on FPGA.

Since the limited resource strategy is considered, our work has lower throughput comparing with others, while the resource (DSP) cost is significantly reduced. That verifies our method's high computation efficiency, as that 946.63% is achieved on language model under the condition of 7/8 weights pruned away and 482.61% is achieved on acoustic model while only 4× is compressed. Compared to ESE, our design can attain higher computation efficiency with lower compression ratio. This merit should be attributed to the proposed structured pruning method which makes the design almost the same compared to dense one, through eliminating the irregular computation and memory accesses of sparse model. The efficient hardware design methodology is another reason for the efficient implementation of the pruned LSTM on FPGA. These two aspects make our work both compact and powerful.

VI. CONCLUSIONS

In this paper, we propose a novel structured pruning method to prune the weights away. It can automatically remove the whole column of ineffective weights along with the network training procedure, therefore generating regular sparse weight matrix which guarantees the regular computation and memory access of sparse model. Implementing this algorithm with a highly parallel and effective pipelined accelerator architecture for FPGA, a computation-efficient compressed V GXA7 FPGA, it not only can achieve an effective throughput of 681.6GOPS and computation efficiency of 946.63% in an 8× compressed language model, but also can achieve 339.7GOPS effective throughput and 482.61% computation efficiency in a 4× compressed acoustic model with less than 1% accuracy loss, which gives an inspiration of the acceleration of recurrent neural networks especially under the resource-limited environments.

ACKNOWLEDGMENT

Part of calculation in this work has been done on the supercomputing system in the Supercomputing Center of Wuhan University.

REFERENCES

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

[2] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. ICASSP*, Vancouver, BC, Canada, May 2013, pp. 6645–6649.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. NIPS*, 2014, pp. 3104–3112.

[4] W. Byeon, M. Liwicki, and T. M. Breuel, "Scene analysis by mid-level attribute learning using 2D LSTM networks and an application to Web-image tagging," *Pattern Recognit. Lett.*, vol. 63, pp. 23–29, Oct. 2015.

[5] Y. Liang, H. P. Huynh, K. Rupnow, R. S. M. Goh, and D. Chen, "Efficient GPU spatial-temporal multitasking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 748–760, Mar. 2015.

[6] K. Hwang and W. Sung, "Single stream parallelization of generalized LSTM-like RNNs on a GPU," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 1047–1051.

[7] M. F. Stollenga, W. Byeon, M. Liwicki, and J. Schmidhuber, "Parallel multi-dimensional LSTM, with application to fast biomedical volumetric image segmentation," in *Proc. NIPS*, 2015, pp. 2998–3006.

[8] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, "FPGA acceleration of recurrent neural network based language model," in *Proc. FCCM*, Vancouver, BC, Canada, May 2015, pp. 111–118.

[9] A. X. M. Chang, B. Martini, and E. Culurciello. (2016). "Recurrent neural networks hardware implementation on FPGA." [Online]. Available: https://arxiv.org/abs/1511.05552

[10] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "FPGA-based low-power speech recognition with recurrent neural networks," in *Proc. IEEE Int. Conf. Signal Process. (SIPS)*, Oct. 2016, pp. 230–235.

[11] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, Aug./Sep. 2016, pp. 1–4.

[12] Z. Wang, J. Lin, and Z. Wang, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2763–2775, Oct. 2017.

[13] Z. Hanqing *et al.*, "A framework for generating high throughput CNN implementations on FPGAs," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 117–126.

[14] A. Utku, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenC deep learning accelerator on Arria 10," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 55–64.

[15] Z. Jialiang and L. Jing, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 25–34.

[16] Khronos OpenCL Working Group. *The OpenCL Specification Version 1.1*. Accessed: May 19, 2019. [Online]. Available: https://www.khronos.org/registry/OpenCL/specs/opencl-1.1.pdf

[17] P. Ouyang, S. Yin, and S. Wei, "A fast and power efficient architecture to parallelize LSTM based RNN for cognitive intelligence applications," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, p. 63.

[18] H. Song *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. FPGA*, 2017, pp. 75–84.

[19] S. Wang *et al.*, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 11–20.

[20] C. Gao *et al.*, "DeltaRNN: A power-efficient recurrent neural network accelerator," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 21–30.

[21] C. Shijie *et al.*, "Efficient and effective sparse LSTM on FPGA with Bank-Balanced Sparsity," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2019, pp. 63–72.

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

[23] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The penn treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[24] S. Han, H. Mao, and W. J. Dally. (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding." [Online]. Available: https://arxiv.org/abs/1510.00149

[25] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[26] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E. (2016). "Convolutional neural networks with low-rank regularization." [Online]. Available: https://arxiv.org/abs/1511.06067

[27] W. Wei *et al.*, "Learning intrinsic sparse structures within long short-term memory," presented at the 6th ICLR, 2018. [Online]. Available: https://openreview.net/pdf?id=rk6cfpRjZ