E-LSTM: An Efficient Hardware Architecture for Long Short-Term Memory

Meiqi Wang, Zhisheng Wang, Jinming Lu, Jun Lin, *Senior Member, IEEE*, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*— Long Short-Term Memory (LSTM) and its variants have been widely adopted in many sequential learning tasks, such as speech recognition and machine translation. Significant accuracy improvements can be achieved using complex LSTM model with a large memory requirement and high computational complexity, which is time-consuming and energy demanding. The low-latency and energy-efficiency requirements of the real-world applications make model compression and hardware acceleration for LSTM an urgent need. In this paper, several hardware-efficient network compression schemes are introduced first, including structured top-*k* pruning, clipped gating, and multiplication-free quantization, to reduce the model size and the number of matrix operations by 32× and 21.6×, respectively, with negligible accuracy loss. Furthermore, efficient hardware architectures for accelerating the compressed LSTM are proposed, which support the inference of multi-layer and multiple time steps. The computation process is judiciously reorganized and the memory access pattern is well optimized, which alleviate the limited memory bandwidth bottleneck and enable higher throughput. Moreover, the parallel processing strategy is carefully designed to make full use of the sparsity introduced by pruning and clipped gating with high hardware utilization efficiency. Implemented on Intel Arria10 SX660 FPGA running at 200MHz, the proposed design is able to achieve 1.4−2.2× energy efficiency and requires significantly less hardware resources compared with the state-of-the-art LSTM implementations.**

***Index Terms*— Hardware acceleration, long short-term memory (LSTM), model compression, recurrent neural network (RNN), deep learning, FPGA.**

I. Introduction

Recurrent Neural Networks have been widely used in sequence-to-sequence applications, such as machine translation [1], speech recognition [2], [3] and video analysis [4]. By introducing a memory cell and a gating scheme, some of the RNN variants, such as Long Short-Term Memory (LSTM) [5] and Gated Recurrent Unit (GRU) [6], are able to capture longer-range dependencies [7]. Thus, they can achieve higher accuracy and are more widely used.

Manuscript received November 4, 2018; revised March 2, 2019; accepted April 1, 2019. Date of publication April 17, 2019; date of current version June 11, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61774082 and Grant 61604068 and in part by the Fundamental Research Funds for the Central Universities under Grant 021014380065 and Grant 021014380087. This paper was recommended by Guest Editor C.-Y. Chen. *(Meiqi Wang and Zhisheng Wang contributed equally to this work.) (Corresponding authors: Jun Lin; Zhongfeng Wang.)*

The authors are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: mqwang@smail.nju.edu.cn; zswang@smail.nju.edu.cn; jmlu@smail.nju.edu.cn; jlin@nju.edu.cn; zfwang@nju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2019.2911739

The improved accuracy comes at the cost of a larger model size. Computations in a typical LSTM model involve large amounts of data (hundreds of Mb's) which cannot all be stored in the limited static random-access memory (SRAM) on chip [8]. The low-cost and high-capacity external memory, dynamic random-access memory (DRAM), is required in the hardware architecture for the inference of LSTM. However, accesses to DRAM consume more than two orders of magnitude of energy and involve much longer latency than those to SRAM [9]. Frequent data exchanges between on-chip SRAM and off-chip DRAM will lead to large time and power consumption. Efficient hardware architecture involving both the network compression and the memory access pattern optimization for LSTM network has become one of the hot topics in both academy and industry societies.

Most of prior works concentrate on compressing the model size to speed up the network, including compression scheme using quantized parameters [10], network pruning [11], circulant matrix [12], adaptive computing [13], and so on. For instance, a load-balance-aware pruning method is proposed in [11] and block-circulant matrices are utilized in [8], [14] to improve inference speed. Although some improvements in system performance are achieved in these designs, the further increase of the system speed is bounded by the external memory interface bandwidth. Several efforts have been made to tackle this problem. Emerging nonvolatile memory (NVM) technologies, such as RRAM, have been studied as the potential replacements for the external memory DRAM [15], [16]. The in-memory computing scheme introduced by RRAM is able to eliminate the energy-consuming data exchanges between the on-chip and off-chip memory [17]. However, RRAM suffers from poor reliability which cannot be acceptable in many practical applications [18]. The DRAM bandwidth bottleneck of the architecture design for LSTM cannot be properly alleviated with RRAM. A delicately designed memory access pattern, together with multiple model compression techniques, is needed to speed up the LSTM network.

This paper focuses on implementing an efficient LSTM network involving both the network compression and the memory access pattern optimization, called E-LSTM. Its main contributions are as follows.

- A hardware-oriented compression algorithm, which includes structured top-*k* pruning, clipped gating, and multiplication-free quantization, is introduced to largely reduce the model size and the number of matrix operations for both the speech recognition and neural

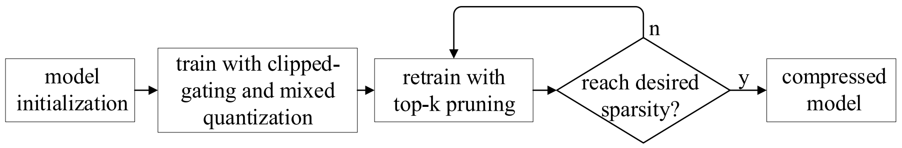

model initialization
train with clipped-gating and mixed quantization
retrain with top-k pruning
reach desired sparsity?
n
y
compressed model

Fig. 1. Overview of the hardware-oriented compression algorithm.

machine translation tasks, with negligible accuracy loss. Compared with ESE in [11], the resulting compressed model requires significantly less memory storage and computational complexity.

- After the compression of the network, an efficient memory access scheme is developed, in which the computation dependencies in LSTM are decoupled to alleviate the memory bandwidth bottleneck. With this method, the time budget for data fetching is increased by T times, where T is the length of the input sequence that is fetched on chip one time. Therefore, the amount of data fetched on chip can be improved by T times due to the increase in available fetching time under a certain external memory bandwidth, which makes it possible to increase the system throughput by better utilizing the inner parallelism of the network. With this scheme, an efficient architecture with a higher potential throughput compared with those introduced in [11], [12] is developed. Moreover, the memory-bound problem in RNN is largely alleviated without the need of replacing the DRAM with the unstable RRAM introduced in [19].
- Efficient hardware architectures for accelerating inference of the compressed LSTM are proposed. In the proposed architecture, the parallel processing strategy is carefully designed to make better use of the introduced sparsity. Moreover, the computation process is judiciously reorganized to utilize the well optimized memory access pattern. Implementation results show that an energy efficiency of 21052 FPS/W is achieved. Compared with the state-of-the-art LSTM implementations, the proposed design is able to achieve $1.4 - 2.2\times$ energy efficiency with significantly less hardware resources.

The rest of the paper is organized as follows. Section II gives a brief review of the LSTM. In Section III, the hardware-oriented compression algorithm and its evaluation results are presented. Section IV introduces efficient hardware architectures for both single-sample and batch LSTMs, which are further evaluated and compared with some related works in Section V. Section VI concludes this paper.

II. Background

LSTM is one of the most widely used RNN variants. In LSTM, memory cells are introduced to store the previous state of the input sequence. The typical structure of LSTM without peephole connections is described as follows:

$$i_t = \sigma(W_x^i x_t + W_h^i h_{t-1} + b^i), \quad \text{input gate} \tag{1}$$

$$f_t = \sigma(W_x^f x_t + W_h^f h_{t-1} + b^f), \quad \text{forget gate} \tag{2}$$

$$o_t = \sigma(W_x^o x_t + W_h^o h_{t-1} + b^o), \quad \text{output gate} \tag{3}$$

$$\tilde{c}_t = \tanh(W_x^c x_t + W_h^c h_{t-1} + b^c), \quad \text{candidate memory} \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad \text{memory cell} \tag{5}$$

$$h_t = o_t \odot \tanh(c_t). \quad \text{hidden state} \tag{6}$$

Here x_t denotes the t-th element of the input sequence. The hidden state h_{t-1} denotes the output of the last neuron. In a multi-layer LSTM model, x_t of the higher layer takes the output h_t of the last layer as input. A gating scheme realized by element-wise multiplication is employed to control the information flow. The input gate controls the information flows from input sequence to the memory cell. The output gate controls the flows from memory cell to output. The forget gate determines how much information will be passed between the memory cells of consecutive time steps. W_x^j and W_h^j $(j = i, f, o, c)$ are weight matrices and b^j $(j = i, f, o, c)$ denote the biases. Considering an LSTM layer with input dimension n_i and output dimension n_o, we have $x_t \in \mathbb{R}^{n_i \times 1}$, h_t, b^j, and $c_t \in \mathbb{R}^{n_o \times 1}$, $W_x^j \in \mathbb{R}^{n_o \times n_i}$, $W_h^j \in \mathbb{R}^{n_o \times n_o}$ $(j = i,f,o,c)$. Unless specifically stated, it is assumed that $n_i = n_o = n$ in the following discussions.

III. Hardware-Oriented Compression Algorithm

The Hardware-Oriented Compression Algorithm (HOCA) [20] is able to convert the computation-intensive dense matrix computations in LSTM into structured sparse operations. Additionally, most of the multiplications can be replaced by hardware-efficient shift operations [12], [21]. This section briefly explains the idea of the HOCA, more details can be found in [20].

A. A Brief Introduction of HOCA

The HOCA involves several compression schemes, which are combined in a two-step training process. As shown in Fig. 1, network parameters are first randomly initialized and trained with clipped gating and mixed quantization schemes. Then, an iterative retrain process with top-k pruning is employed to obtain structured sparse weights until the desired sparsity is achieved.

1) Clipped Gating: The clipped gating method is presented to generate sparse activations. Specifically, a clipping function $\sigma_{clip}(x)$ is used to replace the σ function in Eq (3), where

$$\sigma_{clip}(x) = \begin{cases} \sigma(x) & \text{if } \sigma(x) > T_\sigma, \ T_\sigma \in [0, 1), \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

For o_t, those elements that are smaller than the given threshold T_σ will be clipped and set to zero, resulting in zero outputs of h_t. A regularization term is also introduced to control the desired activation sparsity [20].

2) Top-k Pruning: The top-k pruning scheme is proposed to generate structured sparse weights. Here we introduce a notation (c, k) to illustrate a pruning process: the parameters in weight matrices of LSTM are first judiciously grouped, with each group containing c elements. Then, the weights in each group are pruned based on their magnitudes, leaving at most k non-zero elements in a group. More specifically, all the weight matrices in Eqs. (1)−(4) are first stacked to form a big weight matrix W, where

$$W = \begin{pmatrix} W_x^i & W_h^i \\ W_x^f & W_h^f \\ W_x^o & W_h^o \\ W_x^c & W_h^c \end{pmatrix} \in \mathbb{R}^{4n\times 2n}.$$

Then, each column of W, which is represented as $W_{:,e}$ ($e \in \{1, 2, \cdots, 2n\}$), will be grouped into $\lceil 4n/c \rceil$ groups $\{G_1^e, G_2^e, \cdots, G_{\lceil 4n/c\rceil}^e\}$, with each group G_l^e ($l \in \{1, 2, \cdots, \lceil 4n/c\rceil\}$) containing c elements of $W_{:,e}$.

$$G_l^e = \{W_{l,e}, W_{l+\lceil 4n/c\rceil,e}, \cdots, W_{l+\lceil 4n/c\rceil\cdot(c-1),e}\},$$

where $W_{d,e}$ ($d \in \{1, 2, \cdots, 4n\}$) is the element of W in its d-th row and e-th column. Each group will then be pruned with at most k non-zero elements left. When $4n \mod c \neq 0$, extra zeros will be appended in some groups to make sure that each group contains c elements.

3) Mixed Quantization Schemes: Two quantization schemes are introduced in the HOCA. One is the fixed-point quantization ($Q_{m,f}$) which can be explained as the following:

$$Q_{m,f}(W) = \lfloor 2^f \cdot W_{clip} + \frac{1}{2} \rfloor \cdot 2^{-f}, \tag{8}$$

where $W_{clip} = \min(|W|, 2^m - \frac{1}{2^f}) \cdot sign(W)$. Values quantized with $Q_{m,f}$ need m integer bits and f fractional bits. With one extra sign bit, the total number of bits needed is $m + f + 1$.

The other scheme is log-domain quantization (Log$Q_{m,-f}$) where values are quantized to an integer power of two [12], [21]:

$$\begin{aligned} \text{Log}Q_{m,-f}(W) &= 2^{\min(\max(W_{log},-f),m)} \cdot sign(W) \\ &\in \{\pm 2^m, \cdots, \pm 2^0, \pm 2^{-1}, \cdots, \pm 2^{-f}, 0\}, \end{aligned} \tag{9}$$

where $W_{log} = \lfloor log_2 |W| + \frac{1}{2} \rfloor$, and $m, f > 0$. For Log$Q_{m,-f}$, the total number of bits needed is $\lceil \log_2(2 \cdot (m + f + 1) + 1) \rceil$.

In the case of quantizing LSTM, Log$Q_{m,-f}$ is adapted to quantize the weight matrices which account for the major part of all parameters. The remaining parameters and intermediate results are quantized using $Q_{m,f}$.

B. A Vanilla Case Study

To clearly illustrate the effectiveness of HOCA for complexity reduction and load-balance computation, we provide a vanilla case study in this section. As shown in Fig. 2-(a), initially a 4×4 dense matrix is multiplied by a dense vector with dimension 4 (left). The HOCA is then applied: elements in the matrix and the vector are quantized with Log$Q_{1,-2}$ and $Q_{1,2}$, respectively; the matrix is pruned with $P(2, 1)$ and grouped into eight groups G_i^j ($i \in \{0, 1\}, j \in \{0, 1, 2, 3\}$). Each column of the matrix is divided into two groups with each group containing two elements (middle). When clipped gating is further adopted, the computations will be changed into sparse-matrix sparse -vector multiplications (right). In this case only 4 computations are needed compared to the original 16 multiplications if all zero computations can be skipped. With top-k pruning, we can store the pruned weight matrices by orderly storing all the groups. For each group, only values of the non-zero elements and corresponding indices are needed to be stored. Back to this case, we can orderly store the resulting eight groups as shown in Fig. 2(b). Since the elements are quantized with Log$Q_{m,-f}$, a special data format is introduced to represent the quantized non-zero elements. Specifically, each number v_{log} is represented with a sign-bit (s) along with a shift bit ($shift$):

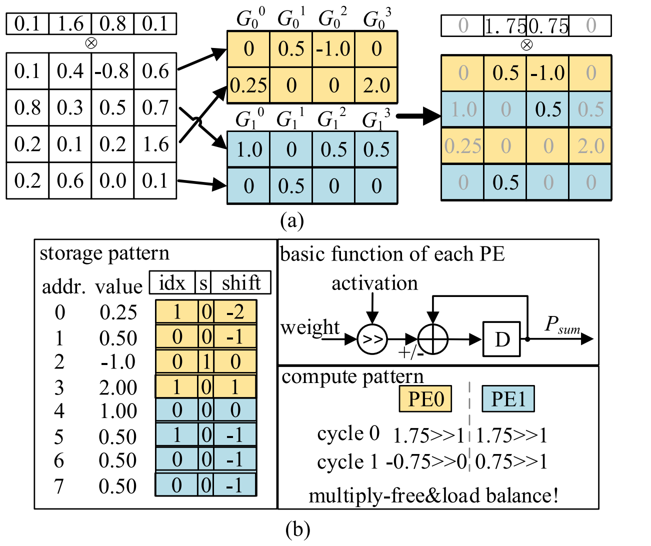


Fig. 2. A vanilla case of HOCA. The "idx" in the figure indicates the relative position of the weight in a group.

$$v_{log} = (2 \cdot s - 1) \cdot 2^{shift}. \tag{10}$$

A number v (quantized with $Q_{m,f}$) multiplied by v_{log} can be rewritten as hardware-efficient shift operations:

$$v \cdot v_{log} = (2 \cdot s - 1) \cdot (v \ll shift). \tag{11}$$

Common pruning methods usually result in a unstructured sparsity pattern in the pruned matrices. During parallel processing, the unstructured sparsity pattern may cause unbalanced workload across multiple processing units (PE), which is known as the load-imbalance problem [22]. With (c, k) pruning, each group will contain the same number of non-zero elements. If we assign related computations regarding the same number of groups to one PE, each PE will exactly execute the same number of non-zero operations. Thus, the load-imbalance problems can be easily eliminated. For example, if we allocate two PEs for the above mentioned matrix-vector multiplications and further assign related computations regarding G_0^j for PE0 and G_1^j ($j \in \{0, 1, 2, 3\}$) for PE1, each PE will exactly execute two non-zero operations when all zero operations can be skipped. Details are shown in Fig. 2(b).

C. Complexity Reduction Analysis

Here we consider only the parameters and computations related to the matrix-vector multiplications (MVMul) in LSTM since they dominate the overall model size and the computations.

1) Model Size: The original weight matrices in LSTM have $8n^2$ parameters, with each parameter stored as a 32-bit floating-point value. After (c, k) top-k pruning, only the resulting non-zero parameters are needed to be stored. The total number of parameters can be reduced to $8n^2 \cdot s_{wei}$, where $s_{wei} = k/c$ is the ratio of non-zero weights. Hence, the compression ratio brought by top-k pruning is:

$$r_{s,tkp} = \frac{8n^2}{8n^2 \cdot s_{wei}} = \frac{c}{k}. \tag{12}$$

For each non-zero parameter, we need to store its value and corresponding index inside the group. Since the indices range from 0 to $c-1$, $\lceil \log_2 c \rceil$ bits are required. When $\text{Log}Q_{m,-f}$ is further applied, we can store the non-zero values with $\lceil \log_2(2 \cdot (m + f) + 1) \rceil$ bits. Compared to 32-bit floating-point representation, the quantization process can further lead to a compression ratio of

$$r_{s,mq} = \frac{32}{\lceil \log_2(2 \cdot (m + f + 1) + 1) \rceil + \lceil \log_2(c) \rceil}. \tag{13}$$

Taking Eqs. (12)-(13) into consideration, the total reduction in model size r_s can be computed as follows:

$$\begin{aligned} r_s &= r_{s,tkp} \cdot r_{s,mq} \\ &= \frac{32c}{k \cdot (\lceil \log_2(2 \cdot (m+f+1)+1) \rceil + \lceil \log_2 c \rceil)}. \end{aligned} \tag{14}$$

2) Computational Complexity: With clipped gating, $s_{act} \times 100\%$ of MVMul operations can be skipped, where s_{act} is the ratio of non-zero activations. Hence, the ratio of computational complexity reduction brought by clipped gating is $r_{op,cg} = 1/s_{act}$. Similarly, top-k pruning can bring a reduction ratio of $r_{op,tpk} = 1/s_{wei} = c/k$. Taking sparse weights and activations into consideration, the total ratio of computational complexity reduction r_{op} can be calculated as follows:

$$r_{op} = r_{op,cg} \cdot r_{op,tkp} = \frac{c}{k \cdot s_{act}}. \tag{15}$$

r_{op} is deterministic given $s_{wei} = k/c$ and s_{act} due to the regularity of the pruned weight matrices.

D. HOCA Evaluation

1) Speech Recognition: The HOCA is first evaluated on the phoneme speech recognition task under TIMIT Dataset [23]. Phone Error Rate (PER) is introduced as the evaluation metric. Lower PER indicates higher performance. The audio file is first preprocessed to obtain a 50-channel Mel-frequency cepstral coefficients (MFCCs) and one energy channel, together with their first and second temporal derivatives, leading to 153 input channels in total. A neural network model with two LSTM layers, a fully-connected layer followed by softmax and connectionist temporal classification (CTC) loss [24] is adopted for evaluation. While the LSTM variant used in E-RNN [14], C-LSTM [8] and ESE [11] is slightly different from ours, the total number of model parameters is kept close to each other so that the comparison is as fair as possible. Table I summarizes the model parameters.

TABLE I
SUMMARY OF THE MODEL STRUCTURE AND PARAMETERS

Layer	$(n_i \times n_o)$	#Parameters
LSTM1	(153 × 768)	2.83M
LSTM2	(768 × 768)	4.72M
FC	(768 × 40)	0.31M
total		7.58M

HOCA is adopted to compress the network model. For two LSTM layers, we use $\text{Log}Q_{1,-5}$ (4 bits) and $Q_{0,7}$ (8 bits) to quantize weight matrices and activations, respectively. $Q_{m,f}$ with 16 bits is adopted for the rest parameters and intermediate results. For clipped gating, we choose $T_\sigma = S_e = 0.5$. During predictions, beam search decoding is adopted with beam width value set to 10. Following the compression process presented in Fig. 1, the network model is first trained with clipped gating and mixed quantization schemes. Then, the trained model is iteratively pruned with top-k pruning for 3 times with a gradually increased weight sparsity: $((16, 4) \rightarrow (16, 3) \rightarrow (16, 2))$.

Table II summarizes the training and fine-tuning results. With the proposed HOCA, we are able to achieve 32× and 22.61× reduction in model storage and computational complexity, respectively, with no performance loss. Specifically, mixed quantization leads to 4× reduction in model size, and clipped gating brings about 2.53× computational complexity reduction. Compared to E-RNN, HOCA achieves comparable r_s but significantly larger r_{op}. Note that the baseline model used in ESE is able to achieve better PER due to their more complex speech recognition pipeline: a hybrid HMM-DNN system, which contains a language model to improve the decoding performance. While in our experiments, an end-to-end DNN-CTC ASR system without extra language model is adopted. The simpler speech recognition pipeline accounts for the lower baseline performance used in this paper. Despite this, higher compression ratio can still be achieved with no PER degradation, which shows the effectiveness of the proposed HOCA.

2) Neural Machine Translation: The HOCA is further evaluated on the IWSLT 2015 Vietnamese-English machine translation (NMT) tasks. The translation quality is measured in terms of BLEU scores [25]. The training code is modified from Google's NMT (GNMT) implementation.[1] A neural network model with 512 embedding dimension, one bidirectional LSTM layer for the encoder, two LSTM layers with attention [26] for the decoder, is used for evaluation. All LSTM layers have 512 hidden nodes. Default hyper-parameter setting are used to obtain a strong baseline model, which reaches 23.8 BLEU on the test set.

To obtain a compressed model, for all LSTM layers, $\text{Log}Q_{1,-5}$ and $Q_{0,7}$ are adopted to quantize weight matrices and activations, respectively. $Q_{m,f}$ with 16 bits is adopted for the rest parameters (including the embedding layers) and

[1]available at https://github.com/tensorflow/nmt

TABLE II
TRAINING AND COMPRESSION RESULTS FOR THE ASR TASK. MQ:MIXED QUANTIZATION, CG: CLIPPED GATING, TKP: TOP-k PRUNING

Model	Compression Ratio		#parameters††	Model Size	PER (Degradation)
	r_s	r_{op}			
baseline	1×	1×	7.55M	28.82MB	23.20%
MQ+CG, 4-bit quantized	4×	2.53×		7.21MB	22.95%
E-LSTM(MQ+CG+TKP)	32×	21.62×		933.38KB	23.18% (−0.02%)
C-LSTM, baseline	1×	1×	8.01M	32.04MB	24.15%
C-LSTM, block size=16,16-bit quantized†	29.12×	3.70×	0.55M	1.1MB	25.48% (1.33%)
E-RNN, baseline	1×	1×	8.01M	32.04MB	20.01%
E-RNN, block size=16,12-bit quantized	38.84×	3.70×	0.55M	805.66KB	20.32% (0.31%)
ESE, baseline	1×	1×	8.01M	32.04MB	20.4%
ESE, compressed	20×	10×		-	20.7% (0.30%)

† For C-LSTM, the model size is 0.55M·16bit=1.1MB. r_s is calculated with 32.01MB/1.1MB≈29.12.
†† Only consider the weight matrices of LSTM layers.

TABLE III
TRAINING AND COMPRESSION RESULTS FOR THE NMT TASK

Model	Compression Ratio		Model Size†	BLEU
	r_s	r_{op}		
baseline	1×	1×	36MB	23.8
MQ+CG, 4-bit quantized	4×	2.38×	9MB	23.5
HOCA, (16,2) top-k	32×	19.04×	1.12MB	22.7

† Only consider the weight matrices of LSTM layers.

intermediate results except for the attention layer. For clipped gating, we choose $T_\sigma = 0.5$ without extra regularizer. After three times of iterative top-k pruning, the compressed model reaches 87.5% $(16, 2)$ weight sparsity. The results are shown in Table III. On average, 58% activation sparsity can be achieved. We are able to achieve 32× and 19.04× reduction in model storage and computational complexity, respectively, with 1.1 BLEU degradation. The results on both speech recognition and machine translation demonstrate the scalability and adaptability of the introduced HOCA.

IV. EFFICIENT ARCHITECTURES FOR ACCELERATING LSTM

In this section, firstly, motivations for the proposed design are presented in detail. Then the system overview, memory access pattern and parallel processing scheme for an efficient batch LSTM architecture are introduced. After that, an improved version of this architecture is proposed to better support single-sample applications. Finally, an optimized memory access scheme is introduced and its efficiency is analysed in detail.

A. Motivation

Dedicated hardware architectures for accelerating LSTM is explored based on the following motivations:

1) LSTM-related applications usually require large amounts of data which are quite memory consuming. A standard LSTM network may require a memory space of more than one hundred Mb, but local buffers on chip in embedded systems are usually tens of Mb or even less [8]. The limited storage space makes it hard to store the whole network on chip and external memory must be used to store the parameters and input data, which will bring some problems. First, the external memory interface has its bandwidth limitation, due to which the parallelism of the LSTM network cannot be fully exploited to increase the system throughput. If the degree of parallelism is too high, the data exchanging speed will not keep up with the data processing speed, thus limiting the improvements in throughput. The second problem is that frequent data exchanges between the external and on-chip memory will result in considerable energy consumption, most of which can be saved with a properly scheduled data flow. Therefore, we need to carefully design the computation schedule of LSTM to tackle the bandwidth limitation problem and minimize the number of unnecessary data transfers.
2) The weight and activation compression scheme introduced in Section III-A greatly reduces the computational complexity and saves the required memory. For this structured model compression, a well-designed architecture is required to make better use of the introduced sparsity and quantization schemes.

B. System Overview

The proposed architecture is capable of efficient batch processing. It consists of multiple channels, each of which has the same inner structure and independently processes a sequence in the input batch. The whole system is shown in Fig 3(a).

The main component of each channel is the LSTM accelerator. It includes the Computing Unit (CU), the Element-Wise Unit (EWU) and various memory buffers. Two categories of operations required by LSTM, matrix-vector multiplications (MVMuls) and element-wise multiplications (EleMuls), are executed by CU and EWU, respectively. CU takes charge of the most complicated and resource consuming computation, MVMul, which can be summarized as: $psum = W_x \cdot x_t + W_h \cdot h_{t-1} + bias$. W_x is the big matrix stacked by the four weight matrices related to input sequence x_t, as introduced in Eqs (1)–(5), and W_h represents the one stacked by matrices related to the hidden state h_{t-1}. $psum$ is the partial sum of MVMul's intermediate results. Multiple processing elements (PEs) are allocated for the MVMuls to perform the multiplications concurrently. Weights, biases and activations (including input sequence x_t $(t = 1, 2, ..., T)$ and hidden state h_0) are sent to on-chip buffers via the data bus and

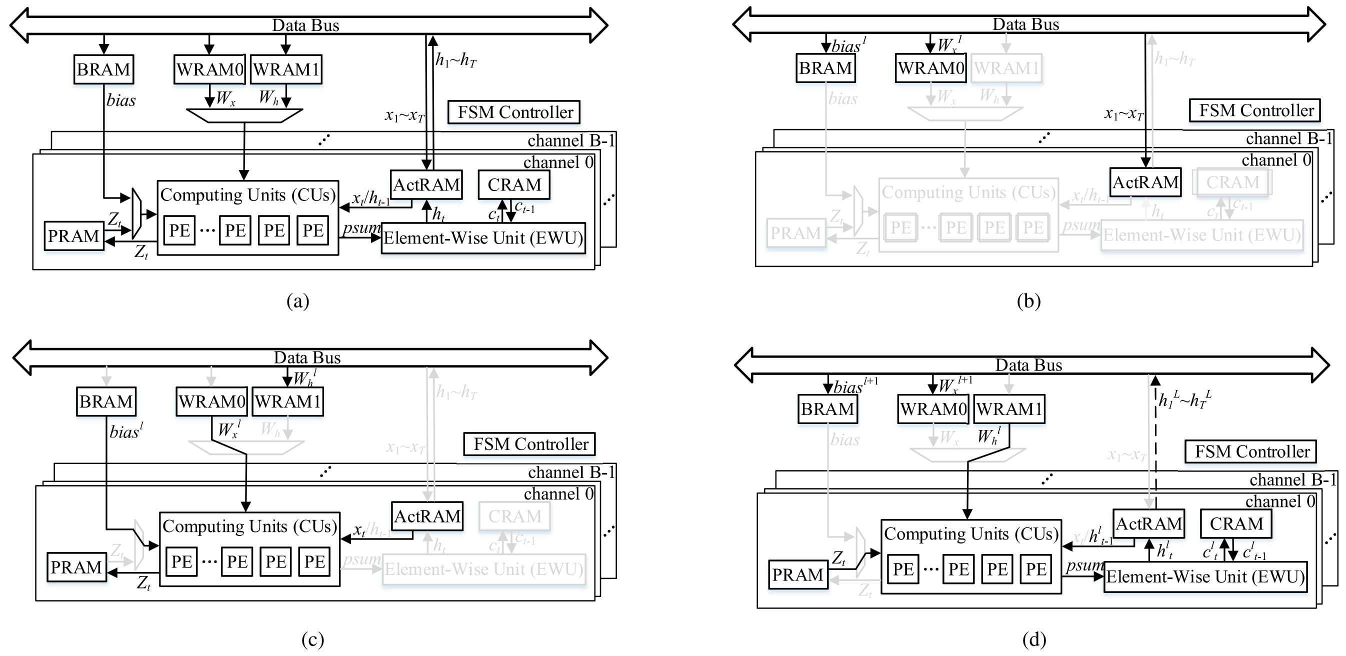


Fig. 3. The system for batch applications and its whole computation process. Fig 3(b) is the initial procedure in which data are sent into the on-chip buffers via the data bus. Fig 3(c) and Fig 3(d) describe the two phases of computations introduced in IV-C. (a) The whole system for batch LSTM applications. (b) Initial state of the whole computation process. (c) Phase1 of the computation process. (d) Phase2 of the computation process.

used by CUs and EWUs. Weights and biases are stored in WRAM and BRAM, respectively, and broadcast to different channels. After MVMuls, various EleMuls will be executed in EWU, including the non-linear functions and other simple EleMuls as shown in Eqs (1)–(6). EWU receives *psum* from CU and applies the non-linear functions (sigmoid or tanh). The generated gates i_t, f_t, o_t and candidate memory $\tilde{c}_t$ will be consumed together with c_{t-1} from CRAM. After some combinational logic operations and EleMuls, h_t and c_t values will be obtained and stored in ActRAM and CRAM, respectively. It should be noted that all RAMs mentioned above and in Fig 3(a) are on-chip memory.

C. Rearrangement of the Computation Process

In this section, an efficient computation process is proposed to tackle the bandwidth limitation and data transfer issues introduced in Section IV-A.

In a multi-layer LSTM, the arrangement of the data flow mainly faces two challenges. First, the parameters of one layer must be read into the on-chip memory as fast as possible at the beginning or when the computations of the previous layer are finished. This puts a heavy burden on the external memory data bus. Due to the bandwidth limitation, this process usually takes much time and causes a long system latency. The other challenge is that the MVMul computations at different time steps have dependencies on each other, thus they need to be executed sequentially. Moreover, MVMuls at different time steps use the same set of parameters, which leads to repeated memory accesses to the same set of data. If the parameters are stored in the external memory, the repeated transfers will significantly increase the power consumption and system latency. If they are fetched into the on-chip buffers at first, how to utilize the limited storage resources and efficiently organize the data access flow is also worth investigating. To address the above mentioned challenges, a judicious bandwidth friendly computation schedule is introduced, which both allows sufficient time for data transfer and reduces the total amount of data accesses to largely save the power consumption.

In LSTM, MVMuls involve the most complicated computations and contribute most to the challenges discussed above, so the proposed schedule is mainly for these operations. The calculations of MVMul can be described with the formula $psum = W_x \cdot x_t + W_h \cdot h_{t-1} + bias$. Note that there are two weight matrices, both of which require many computational and storage resources. Different from the conventional design in which W_x and W_h are allocated and calculated together, we separate the calculations of the two matrices and organize their computations and transfers in a ping-pong form. First, W_x is read into one on-chip buffer. Then, calculations of $W_x \cdot x_t$ for all input vectors x_t $(t = 1, 2, ..., T)$ are performed. During this process, the other matrix W_h is fetched from the external memory into another on-chip buffer. Because the calculations of $W_x \cdot x_t$ will repeat T times, there is enough time for W_h to be read from off-chip memory. Hence, the data fetching time of W_h can be overlapped with data computation time, thus reducing the total latency. After the calculations of $W_x \cdot x_t + bias$ for all T steps are finished, $W_h \cdot h_{t-1}$ are repeated for another T cycles, during which W_x for the next layer has sufficient time to be read in from off-chip memory.

Based on the computation rearrangement scheme discussed above, the calculation of the psum is split into two phases, which are shown in Algorithm 1. The active calculations of the initial procedure and different phases are illustrated from Fig 3(b) to Fig 3(d).

As we can see in Algorithm 1, Equations (*) and (**) are identical in form, so they can reuse the same set of modules,

Algorithm 1 Pseudocode of processing a cascade of L LSTM layers. Each layer $l \in 1, 2, ..., L$ has parameters W_x^l, W_h^l, $bias^l$ and initial intermediate layer states h_0^l, c_0^l.

```
Input: $x_1, x_2, ..., x_T$
Output: $h_1^L, h_2^L, ..., h_T^L$
1: for $l = 1$ to $L$ do
2:     for $t = 1$ to $T$ do (Phase 1)
3:         if $l = 1$ then
4:             $Z_t = W_x^1 \cdot x_t + bias^1$   (*)
5:         else
6:             $Z_t = W_x^l \cdot h_t^{l-1} + bias^l$
7:         end if
8:     end for
9:     for $t = 1$ to $T$ do (Phase 2)
10:        $psum = W_h^l \cdot h_{t-1}^l + Z_t$   (**)
11:        $i_t, f_t, o_t, \tilde{c_t} = func(psum)$
12:        $c_t^l = i_t \cdot \tilde{c_t} + f_t \cdot c_{t-1}^l$
13:        $h_t^l = o_t \cdot tanh(c_t^l)$
14:    end for
15: end for
```

but with different inputs in two phases. In the first phase, the inputs of CU are W_x, x_t (or h_t^{l-1}, x_t for the first layer and h_t^{l-1} for another layers) and $bias$. In the second phase, CU's inputs are W_h, h_{t-1}^l and Z_t. After the output $psum$ is obtained, it will be sent to EWU to perform the following EleMuls. The two phases are described in Fig 3(c) and Fig 3(d).

Compared to the conventional implementation in [11], the architecture proposed significantly increases the time budget for data transferring. Therefore, the burden of high data transfer requirement can be reduced and the system parallelism can be increased accordingly. In this way, the DRAM bandwidth bottleneck that limits the improvements in throughput can be alleviated and inner parallelism of LSTM can be better exploited.

D. Parallel Processing Scheme

In this section, we illustrate the parallel processing scheme for the most expensive computation in LSTM, MVMul, and how this computation can be overlapped with subsequent EleMul. Benefiting from the structured sparse pattern of weight parameters, we can utilize the sparsity while achieving load balance among different PEs. This will significantly increase the hardware utilization efficiency and reduce system latency compared with traditional works.

The scheme includes parallelism in both row and column directions of weight matrices. For row parallelism, several weight groups are multiplied simultaneously to increase the system throughput and reduce the times of repeated memory access of activations. For column parallelism, there are a number of PEs performing multiplications concurrently. Taking the MVMul operation $W_h \cdot h_{t-1}$ as an example, the operations in PEs are described in Fig. 4.

The definition of group notation G_l^e in Fig. 4 is introduced in section III-A. The difference is that in the proposed architecture with rearranged computation process, the two weight matrices, W_x and W_h, are divided into multiple groups

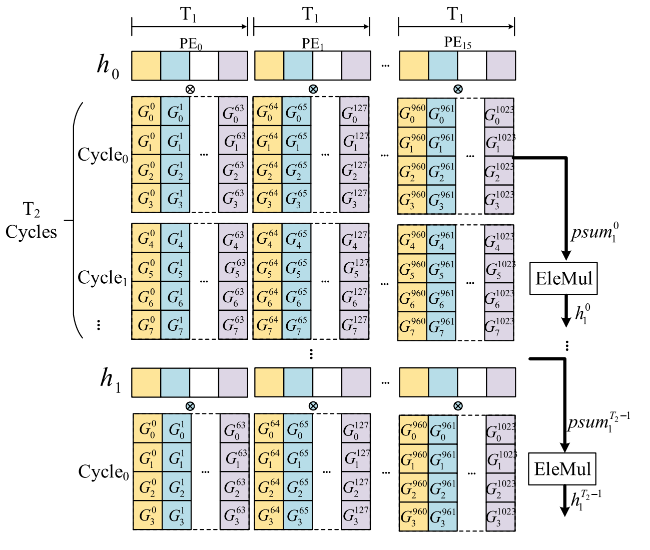


Fig. 4. The parallel processing scheme of the proposed architecture of E-LSTM. The figure is drawn based on the real experiment settings introduced in V-A.

separately. As is shown in Fig. 4, it takes T_2 cycles to calculate all T_2 parts of $psum_1$ and h_1. In each cycle, weight groups and activations with the same color are multiplied concurrently and it takes T_1 steps to finish the multiplications between all elements of h_0 and the corresponding W_h groups for one PE. Operations in the vertical direction have to be executed sequentially, while the MVMuls and EleMuls in the horizontal direction should be executed concurrently. After h_1 is obtained, the above processes are executed repeatedly to obtain all h_t $(t = 2, 3, ..., T)$. It should be noted that the MVMul between W_h and h_1 starts before the final part of h_1, $h_1^{T_2-1}$, is calculated. This is because the partition method of activations and weights guarantees that the first few steps of $W_h \cdot h_1$ don't involve any calculations with elements of $h_1^{T_2-1}$.

Assume that P_{col} and P_{row} are the degree of parallelism of column and row, respectively, and the size of each weight matrix is $n * n$, we have:

$$T_1 = \frac{n}{P_{col}}, \tag{16}$$

$$T_2 = \frac{4n}{c \cdot P_{row}/k}. \tag{17}$$

In the proposed architecture, we have $n = 1024$, $P_{col} = 16$ and $P_{row} = 8$. In each PE, the number of multiplications for each $psum_i$ is exactly the same, all equals to T_1. So there's no load imbalance issue as in the conventional implementation.

The calculation of psum is divided into T_2 steps so that the part of $psum$ calculated at the previous step can be used for subsequent EleMuls when we calculate the new part of $psum$. In this way, EleMuls and MVMuls can be overlapped with each other and operated in a pipelined manner, thus saving time and reducing latency. The execution time of EleMuls, T_{elem}, should be no more than that of MVMuls, T_{mvm}, to introduce no extra latency. T_{mvm} in the proposed design is large enough to enable the EleMul operations being executed sequentially, which greatly reduces the hardware resources and still satisfies the constraint $T_{elem} < T_{mvm}$.

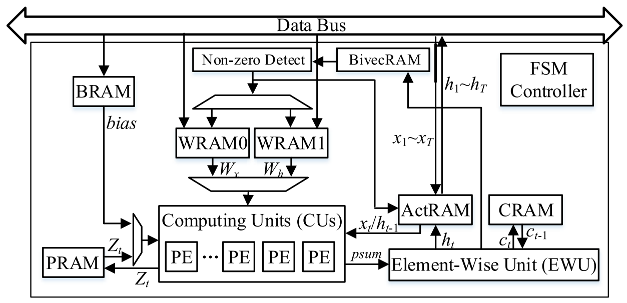


Fig. 5. The whole system for single-sample LSTM applications.

E. Architecture for Single-Sample Applications

For the single-sample inference case, some extra modules are added to utilize the activation sparsity introduced in Section III-A to further reduce the overall computational complexity and speed up the computations. The improved architecture is shown in Fig. 5. The results of non-linear functions output by the EWU will be used to generate a bitmask of the same length as the activation, which indicates whether or not elements of the corresponding position in an activation is non-zero. The generated vectors are stored in the Binary Vector buffer (BivecRAM). The Non-zero Detect module reads the binary vectors, detects the non-zero bit positions and uses it to fetch only the non-zero activation elements and the corresponding weight columns from the ActRAM and WRAM [27]. The MUX is controlled by the FSM Controller and sends non-zero bits to different weight buffers at different phases of the computation.

If we adopt the same parallel processing scheme as discussed in Section IV-D in the new architecture, sparse activations need to be evenly split and processed by multiple PEs concurrently when $P_{col} > 1$. Due to the irregular sparse pattern in the activations, PEs with less multiplications need to wait for those with the most number of multiplications for synchronism, which will cause the load imbalance problem among different PEs. Larger P_{col} will result in more imbalanced computations. When $P_{col} = 1$, load-balance computations can still be achieved. Detailed analysis of the relationships between column parallelism P_{col} and the computation reduction rate brought by activation sparsity will be introduced in Section V-D.

F. Improvement for the Memory Access Scheme

In this section, an improved version for the memory access scheme introduced in Section IV-C is proposed. The comparison between the new scheme and the original one is illustrated in Fig. 6 and Fig. 7. The superscripts of W_x and W_h denote the corresponding part of weight matrices that is used in a specific cycle. As is shown in Fig. 7, instead of fetching the whole W_x matrix of one layer at a time, we divide the fetching process into T_2 steps and transfer one piece of W_x at the beginning. The data accessing process and data computation process are executed concurrently and operated in a ping-pong manner. This reduces the system latency further and saves storage space the matrix W_x occupies with a negligible increase of hardware complexity. The analysis of the efficiency of this improved memory access scheme will be discussed in V-C.

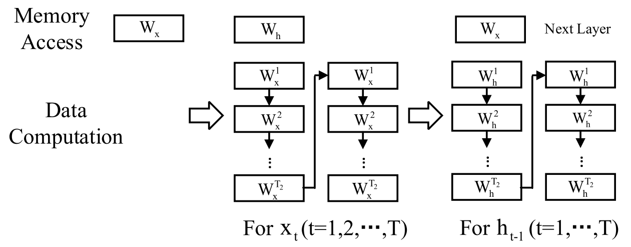


Fig. 6. The original memory access scheme.

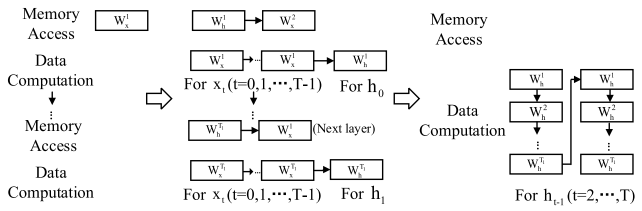


Fig. 7. The improved memory access scheme.

V. Implementation and Comparisons

A. Experiment Settings and Model Definitions

The proposed design is coded using hardware description language (HDL) and implemented on the Intel Arria10 SX660 FPGA. The detailed model definitions are as follows.

To make a fair comparison with ESE proposed by Han *et al.* [11], the total parallelism degree is the same with that of ESE, and both are 1024. Parallelism degrees of column and row are $P_{col} = 16$ and $P_{row} = 8$, respectively, and batch_size is equal to 8. The performance of the proposed hardware architecture is evaluated using an LSTM layer with 153 input node and 1024 hidden node. The total number of parameters is $(153 + 1024) \cdot 1024 \cdot 4 \approx 4.82M$. The weight sparsity scheme $(16, 2)$ is adopted, which means there are two non-zero weight elements in each group and each group has 16 weights. The weight matrices and activations are quantized with $LogQ_{1,-5}$ and $Q_{0,7}$, respectively, as defined in Section III-A. Each weight is represented by an 8-bit number with four bits indicating its position in the group and the other four bits denoting its real integer value. Activations are also represented by 8-bit numbers and other parameters are all quantized as 16-bit values. A sequence of length eight is chosen to verify the effectiveness of this architecture. Larger sequence lengths can also be supported using multiple rounds of computations.

B. Comparisons and Analysis

Running at 200MHz, the implementation results and comparisons with related works are summarized in Table IV. Since we do not have the physical platform of KU060, only the synthesis results of hardware utilizations are shown in the table. For C-LSTM and E-RNN, we compare them with the best reported results. Per sample latency and frames per second (FPS) [8], [14] are introduced for performance evaluation. The breakdown of on-chip memory usage is shown in Table V.

TABLE IV
IMPLEMENTATION RESULTS OF KEY FEATURES OF E-LSTM AND COMPARISONS WITH ESE, C-LSTM, AND E-RNN

	ESE [11]	C-LSTM FFT16 [8] (Block size: 16)		E-RNN FFT16 [14] (Block size: 16)		E-LSTM (Ours)	
LSTM Model	Google LSTM					Vanilla LSTM	
#Parameters(M)	3.25					4.82	
#Compressed Parameters(M)	0.36	0.20				0.60	
Wei.&Act. Quantization†	16-bits	16-bits		12-bits		8-bits	
Wei. Compression Ratio	8.9:1	15.9:1				8:1	
Evaluation Platform	KU060	KU060	7V3	KU060	7V3	KU060	Arria10 SX660
Clock Frequency	200MHz						
Evaluated Batch Size/#PEs per Batch	32/32	-/128				8/128	
DSP(%)	54.5	98.0	77.4	96.4	79.6	**0.9**	1.4
BRAM(%)	87.7	89.1	63.3	90.3	65.2	**39.9**	32.1
LUT(%)	88.6	72.8	55.3	76.5	59.4	91.7	87.8
FF(%)	68.3	63.4	48.1	65.1	55.3	**29.0**	15.6
Power(W)	41	-	23	-	25	-	15.9
per Sample Latency(us)††	82.7	-	9.1	-	8.3	-	23.9†††
Frames per Second (FPS)	386941	-	330275	-	382510	-	334728
PER Degradation	0.3%	1.23%		0.31%		-0.02%	
Throughput(GOP/s)	282.2	-		-		282.2	
Energy Efficiency(FPS/W)	9438	-	14359	-	15300	-	**21052**

† In ESE, each weight is represented using 16-bits with 12-bits for non-zero values and 4-bits for its non-zero indices. For E-LSTM, we use 4-bits for representing the log-quantized value and the other 4-bits for the non-zero indices.
†† Since the latency and FPS values provided by C-LSTM and E-RNN on the KU060 platform are only estimated values, we do not list them here.
††† The latency in ESE is calculated with sequence length $T = 1$. To make a reasonable comparison, the latency here is also estimated with $T = 1$. The per sample latency in C-LSTM and E-RNN is measured under the condition of sequentially processing one input sample. For processing all batch samples, the total latency required will be larger than that given in their papers.

TABLE V
BREAKDOWN OF ON-CHIP MEMORY USAGE

	BRAM	WRAM	PRAM	ActRAM	CRAM
Memory (KB)	14.5	228	116	64	8

As shown in Table IV, we are able to achieve 1.4-2.2× higher energy efficiency compared to related works. Besides, comparisons among the resource utilization of different designs under the KU060 platform show that E-LSTM occupies only a few DSPs and several times less usage of BRAMs and FFs, while achieving comparable FPS values. It is noted that the FPS value of E-LSTM is slightly less than related works. The reason is that the evaluated LSTM layer of E-LSTM contains more parameters and operations compared to that of ESE, C-LSTM, and E-RNN. Therefore, longer time is required for processing the whole layer.

In C-LSTM and E-RNN, batch samples are fetched into a pipeline processing sequentially. The per sample latency and the FPS are measured under the condition of sequentially processing one input sample. For processing all batch samples, the total latency required will be larger than that given in their papers. By contrast, in ESE and the proposed E-LSTM, batch samples are processed simultaneously. Hence, though the per sample latency value of E-LSTM seems to be several times lower than that of C-LSTM and E-RNN, E-LSTM still has comparable FPS and higher efficiency. It is noted that the original FPS of ESE provided in [8], [14] is only the FPS value for processing one sample despite the fact that 32 samples are processed simultaneously in ESE. We have modified it in Table IV to have a fairer comparison.

The reasons for the high performance and low hardware consumption can be summarized as follows:

Firstly, the structured compression scheme proposed in this work eliminates extra idle time and hardware resources due to the unbalanced work load. Secondly, with the compression technique of E-LSTM, both weights matrices, W_x and W_h of one layer, can be stored in BRAM. Thus only one time of memory fetching is needed for one weight matrix, which significantly reduces energy consumption. Moreover, the rearrangement of computation process not only reduces the latency for memory access but also provides enough time budget for EleMul computations. Therefore, EleMuls can be executed serially and thus greatly reducing the number of DSPs and FFs. The quantization scheme replaces all the multiplications in MVMul with additions, which also contributes to the big reduction on the usage of DSPs.

C. Discussion

In this section, the relationships among the memory bandwidth requirements, the sequence length T, and the parallelism of the system are discussed to show the highest throughput that can be achieved given the sequence length T. To avoid the hardware and time resource waste caused by idle cycles, the data transferring speed should be no slower than the data processing speed. The constraint is shown in Eq (18):

$$\frac{T \cdot size\_W_{cmp} \cdot comp\_ratio}{P \cdot f_{PE}} \geq \frac{size\_W_{load} \cdot comp\_ratio \cdot 8bits}{mem\_bandwidth}. \tag{18}$$

The expression to the left is the computation time for currently occupied weights W_{cmp} and that to the right is the transferring time for weights W_{load} which will be used in the following computations. In E-LSTM, there are $W_{cmp} = W_x$, $W_{load} = W_h$ or vice versa. $size\_W_{cmp}$ and $size\_W_{load}$ are the sizes of weight matrices W_{cmp} and W_{load}, respectively. P is the total degree of parallelism. $comp\_ratio$ represents the compression ratio of weight matrices. $mem\_bandwidth$ is the memory bandwidth and f_{PE} is the working frequency of PEs. Eq (18) can be simplified and transformed into the following two equations:

$$mem\_bandwidth \geq \frac{8 \cdot P \cdot f_{PE} \cdot size\_W_{load}}{T \cdot size\_W_{cmp}}, \tag{19}$$

$$P \leq \frac{T \cdot mem\_bandwidth \cdot size\_W_{cmp}}{8 \cdot f_{PE} \cdot size\_W_{load}}. \tag{20}$$

If $size\_W_{cmp} = size\_W_{load} = size\_W$ (the size of one weight matrix), Eq (19) can be further simplified to

$$mem\_bandwidth \geq \frac{8 \cdot P \cdot f_{PE}}{T}. \tag{21}$$

Without considering T, the minimal required DDR bandwidth will be $8 \cdot P \cdot f_{PE}$. According to Eqs. (19)–(20), the maximally allowed degree of parallelism is proportional to sequence length T while the minimal required DDR bandwidth is proportional to $1/T$. Hence, with larger T, either higher throughput or lower bandwidth requirements can be achieved.

For the architecture with the improved memory access scheme presented in IV-F, the original fetching process of W_x is divided into multiple steps. As we can see from Fig. 7, two pieces of weight matrices are fetched on chip during a period of $T + 1$. Hence, Eq. (18) should be rewritten as the followings:

$$\frac{(T+1) \cdot size\_W}{P \cdot f_{PE}} \geq \frac{2 \cdot size\_W \cdot 8bits}{mem\_bandwidth}, \tag{22}$$

Based on Eq. (22), the memory bandwidth requirements can be expressed as

$$mem\_bandwidth \geq 8 \cdot P \cdot f_{PE} \cdot \frac{2}{T+1}. \tag{23}$$

It can be found that $\frac{1}{T} < \frac{2}{T+1} < 1$ holds true when $T \geq 2$, from which we can conclude that although the architecture with the improved memory access scheme requires higher memory bandwidth compared to the original E-LSTM implementation, compared with conventional implementations, lower bandwidth requirements can be achieved as long as $T \geq 2$.

D. Analysis of the Single-Sample Architecture

As is introduced in Section IV-E, the single-sample architecture utilizes both weight and activation sparsity to reduce the system latency. As discussed in Section III-C, computations can theoretically be reduced to $r_{op} = 1/(s_{act} \cdot s_{wei})$ times, where activation sparsity contributes to $1/s_{act}$ times computation reduction. For activation sparsity, there will be computation workload imbalance problem since PEs with higher sparsity need to wait for that with the lowest sparsity to complete all computations. So the computation reduction brought by sparse activations under different parameter settings needs a detailed analysis.

Here we pick the second layer's output activations of a trained model with $s_{act} = 48.9\%$ to analyze the relationships

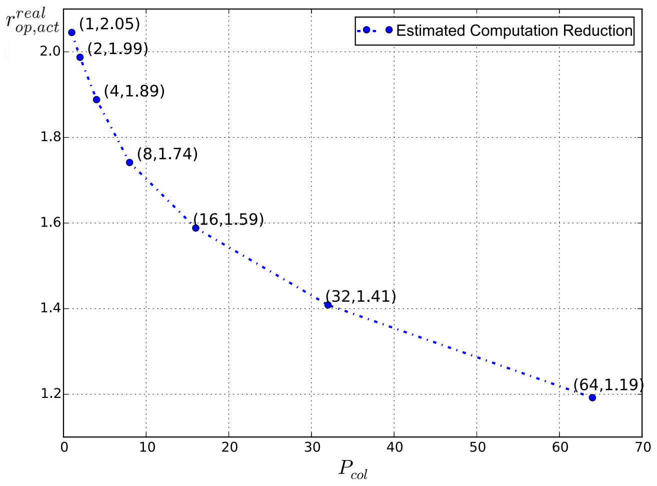


Fig. 8. The relationships between column parallelism P_{col} and the estimated computation reduction brought by sparse activations.

between column parallelism P_{col} and the estimated computation reduction rate $r^{real}_{op,act}$ brought by activation sparsity. The result is shown in Fig. 8.

The result is averaged over multiple input samples with multiple time steps. As is shown in Fig. 8, when $P_{col} = 1$, there is $r^{real}_{op,act} = 1/s_{act} = 2.05$. In this case, the calculations of all elements in the activation are completed by one PE, which means there's no extra waiting time and the sparsity is fully utilized. When P_{col} is increased, the activations will be split to more pieces and allocated to different PEs. Then the computation reduction rate will be lower due to the increased sparsity diversity among split activations. Imagine the extreme case when the value of P_{col} is equal to the length of the activation, then the average operation time of MVMuls will be the same as that without activation sparsity before, which means no computation reduction can be achieved. The load imbalance problem also leads to a decrease PE utilization. PE's utilization ratio under certain P_{col} can be calculated as corresponding $r^{real}_{op,act}$ value divided by the theoretical computation reduction (the computation reduction when $P_{col} = 1$). For example, when $P_{col} = 4$, the utilization rate of the PEs is $\frac{1.89}{2.05} \cdot 100\% \approx 92.20\%$.

In conclusion, as P_{col} increases, the system throughput will be improved at a cost of higher hardware resource consumption. However, these hardware resources cannot be fully utilized due to the load imbalance problem involved in the single-sample architecture when $P_{col} > 1$, and the utilization ratio decreases when P_{col} becomes larger. A trade-off can be made according to the available hardware resources and the throughput requirements of practical applications.

E. Comparisons With Other Related Works

In [9], an energy-efficient processing unit for LSTM called E-PUR was presented. One of its main contributions is the technique of Maximizing Weight Locality (MWL), which improves the temporal locality of the memory accesses for fetching the weights as what we do in this work. However, the computation process introduced in [9] involves MVMuls for weight matrices of two different sizes, which means that the computation units cannot be reused for MVMuls of W_x and W_h. This will reduce the hardware utilization efficiency and increase the hardware complexity.

Moreover, in E-PUR with MWL, one row of weight matrices is collected one time and reused for the whole input sequence. Hence, the on-chip storage space of the weight used in E-PUR is small enough. However, power consumption is largely increased because the activations will be accessed as many times as the row dimension of one weight matrix, usually around several hundreds. On the contrary, in the E-LSTM proposed in this work, a dedicated parallel scheme is designed to achieve the balance between power consumption and available on-chip storage resources.

VI. Conclusion

This paper explores algorithm-hardware co-optimization for accelerating the computing of LSTM model. First, several network compression schemes are introduced to significantly reduce the model storage size and the number of matrix operations, at a cost of negligible accuracy loss. Then, efficient hardware architectures for the compressed LSTM are proposed. With judiciously reorganized computation process and memory access optimization, the DDR bandwidth bottleneck is alleviated, which enables higher system throughput. Besides, the parallel processing strategy is carefully designed to achieve high hardware utilization ratio. Implementation results on Intel Arria10 SX660 FPGA running at 200MHz show that the proposed design is able to achieve 21052 FPS/W energy efficiency on dense networks with significantly less hardware resources compared with the state-of-the-art LSTM implementations, which demonstrates great potential of the proposed design for many real-time applications on embedded systems.

References

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

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

[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[4] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using LSTMs," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 843–852.

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

[6] K. Cho, B. Van Merrienboer, D. Bahdanau, and Y. Bengio. (2014). "On the properties of neural machine translation: Encoder-decoder approaches." [Online]. Available: https://arxiv.org/abs/1409.1259

[7] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1310–1318.

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

[9] F. Silfa, G. Dot, J.-M. Arnau, and A. Gonzalez. (2017). "E-PUR: An energy-efficient processing unit for recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1711.07480

[10] Q. He *et al.* (2016). "Effective quantization methods for recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1611.10176

[11] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 75–84.

[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. Wang, F. Sun, J. Lin, Z. Wang, and B. Yuan. (2018). "SGAD: Soft-guided adaptively-dropped neural network." [Online]. Available: https://arxiv.org/abs/1807.01430

[14] Z. Li *et al.* (2018). "E-RNN: Design optimization for efficient recurrent neural networks in FPGAs." [Online]. Available: https://arxiv.org/abs/1812.07106

[15] A. Chen, "Emerging nonvolatile memory (NVM) technologies," in *Proc. 45th Eur. Solid State Device Res. Conf. (ESSDERC)*, Sep. 2015, pp. 109–113.

[16] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[17] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, 2016,

[18] P. Pouyan, E. Amat, and A. Rubio, "Insights to memristive memory cell from a reliability perspective," in *Proc. Int. Conf. Memristive Syst. (MEMRISYS)*, Nov. 2015, pp. 1–2.

[19] Y. Long, T. Na, and S. Mukhopadhyay, "ReRAM-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2781–2794, Dec. 2018.

[20] Z. Wang, J. Lin, and Z. Wang, "Hardware-oriented compression of long short-term memory for efficient inference," *IEEE Signal Process. Lett.*, vol. 25, no. 7, pp. 984–988, Jul. 2018.

[21] D. Miyashita, E. H. Lee, and B. Murmann. (2016). "Convolutional neural networks using logarithmic data representation." [Online]. Available: https://arxiv.org/abs/1603.01025

[22] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[23] J. S. Garofolo, "TIMIT acoustic phonetic continuous speech corpus," *Linguistic Data Consortium, 1993*, 1993.

[24] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 369–376.

[25] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguistics*. Association for Computational Linguistics, 2002, pp. 311–318.

[26] M.-T. Luong, H. Pham, and C. D. Manning. (2015). "Effective approaches to attention-based neural machine translation." [Online]. Available: https://arxiv.org/abs/1508.04025?context=cs

[27] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 1, pp. 124–128, Mar. 1994.

Meiqi Wang received the B.S. degree in electronic information science and technology from Nanjing University, Nanjing, China, in 2018, where she is currently pursuing the M.S. degree in communication and information systems.

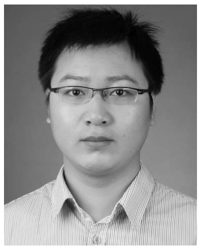
Her current research interests include very-large-scale integration (VLSI) design and deep learning, especially efficient hardware implementations of deep learning.

Zhisheng Wang received the B.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2016, where he is currently pursuing the M.S. degree in integrated circuit engineering.

His research interests include VLSI design and model compression algorithms for machine learning, especially deep learning-related applications.

Jinming Lu received the B.S. degree in microelectronics from Nankai University, Tianjin, China, in 2018. He is currently pursuing the Ph.D. degree in information and communication systems with Nanjing University, Nanjing, China.

His current research interests include automatic speech recognition and deep learning, especially its hardware acceleration and compression algorithms.



Jun Lin (S'14–SM'18) received the B.S. degree in physics and the M.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the Ph.D. degree in electrical engineering from Lehigh University, Bethlehem, PA, USA, in 2015.

From 2010 to 2011, he was an ASIC Design Engineer at AMD, Inc. In summer 2013, he was an Intern at Qualcomm Research, Bridgewater, NJ, USA. In 2015, he joined the School of Electronic Science and Engineering, Nanjing University, where he is currently an Associate Professor. His current research interests include low-power high-speed VLSI design, specifically VLSI design for digital signal processing and cryptography. He is a member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the IEEE Signal Processing Society. He was a co-recipient of the Merit Student Paper Award at the IEEE Asia Pacific Conference on Circuits and Systems in 2008. He was a recipient of the 2014 IEEE Circuits and Systems Society (CAS) Student Travel Award.

Zhongfeng Wang (M'00–SM'05–F'16) received the B.E. and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, in 2000.

He recently joined Nanjing University as a Distinguished Professor after serving Broadcom Inc., as a leading VLSI Architect for nearly nine years. From 2003 to 2007, he was an Assistant Professor at the School of Electrical Engineering & Computer Science, Oregon State University, Corvallis, OR, USA. Even earlier, he was working for the National Semiconductor Corporation. He is a world-recognized expert on VLSI for Forward Error Correction Codes. He has published over one hundred technical papers, edited one book (VLSI), and filed tens of U.S. patent applications and disclosures. During his tenure at Broadcom Inc., he has contributed significantly on 10Gbps and beyond high-speed networking products. In addition, he has made critical contributions in defining FEC coding schemes for 100Gbps and 400Gbps Ethernet standards. So far, his technical proposals have been adopted by many networking standard specs. His current research interests are in the areas of VLSI for high-speed signal processing systems.

Dr. Wang has served as a technical program committee member, session chair, track chair, and review committee member for many IEEE and ACM conferences. He was a recipient of the IEEE Circuits and Systems (CAS) Society VLSI Transactions Best Paper Award in 2007. In 2013, he has served in the Best Paper Award Selection Committee for the IEEE Circuits and System Society. In the current record (2007–present), he has had five papers ranked among top twenty most downloaded manuscripts in the IEEE TRANSACTIONS ON VLSI SYSTEMS. Since 2004, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I (TCAS-I), TCAS-II, and IEEE TRANSACTIONS ON VLSI SYSTEMS for many terms.