E-LSTM: Efficient Inference of Sparse LSTM on Embedded Heterogeneous System

Runbin Shi, Junjie Liu, Hayden K.-H. So
Dept. of Electrical and Electronic Engineering,
The University of Hong Kong
{rbshi,jjliu,hso}@eee.hku.hk

Shuo Wang, Yun Liang*
Center for Energy-efficient Computing and Applications,
School of EECS, Peking University
{shvowang,ericlyun}@pku.edu.cn

ABSTRACT

Various models with Long Short-Term Memory (LSTM) network have demonstrated prior art performances in sequential information processing. Previous LSTM-specific architectures set large on-chip memory for weight storage to alleviate the memory-bound issue and facilitate the LSTM inference in cloud computing. In this paper, E-LSTM is proposed for embedded scenarios with the consideration of the chip-area and limited data-access bandwidth. The heterogeneous hardware in E-LSTM tightly couples an LSTM co-processor with an embedded RISC-V CPU. The eSELL format is developed to represent the sparse weight matrix. With the proposed cell fusion optimization based on the inherent sparsity in computation, E-LSTM achieves up to 2.2× speedup of processing throughput.

1 INTRODUCTION

Long Short-Term Memory (LSTM) networks have demonstrated prior art accuracy in learning sequential information, such as speech recognition [4], machine comprehension [8] and translation [3]. However, running LSTM on a general-purpose CPU suffers poor real-time performance for its computation- and memory-intensive workload [1]. To speed up the LSTM inference, specific hardware accelerators have been proposed that target the cloud computing scenario [6, 7, 14]. These designs adopt the large-batch parallelism that processes multiple input sequences concurrently. In addition, large on-chip buffers are employed to store the entire networks model to meet the ultra-high bandwidth requirement of weight access. However, a majority of embedded scenarios need to conduct inference locally for real-time processing. The LSTM-specific hardware for embedded scenario has apparent differences with that for cloud scenario. First, batch-parallelism introduces extra latency since the system starts computation after receive all input sequences of a batch. Second, buffering the entire model on-chip is infeasible in the chip-area sensitive scenario. Importantly, the accelerator should be coupled with a CPU to facilitate control-flexibility and performance.

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317813

The main contributions of this work are listed below,

- E-LSTM employs the tightly-coupled heterogeneous architecture that provides an ultra-low latency on the communication between the CPU and accelerator.
- Considering the model sparsity in LSTM, eSELL sparse-matrix format was developed that reduces the area cost of the on-chip buffer by 63%.
- We observed the inherent sparsity of the hidden state in LSTM computation. Based on this, we developed the cell fusion scheme that achieves up to 2.2× performance speedup to the implementation without optimization.
- The source code of E-LSTM is publicly available at `https://github.com/rbshi/elstm`.

2 BACKGROUND

2.1 Long Short-Term Memory

A typical schematic of LSTM networks is presented in Fig. 1. LSTM cell is the fundamental component that accepts an input sequence $(x_1, x_2, ..., x_{ts})$, where each x_t is a vector representing the information of time point t and ts is the fixed length (time step) of a sample sequence. For the given input sequence, the computation of a LSTM layer consists of ts *cell iterations* and each iteration generates an output vector h_t. Equ. 1-5 listed the arithmetic of a single cell iteration. Each vector in the left-hand-side of equations represents a specific *gate* in LSTM cell, readers are referred to [5] for further understanding the gates. The weight matrix and bias vector are denoted as W, U and b respectively. For the input vector $x_t \in R^m$, if the length of h_t (hidden size) is n, we have $W \in R^{n\times m}$ and $U \in R^{n\times n}$. The arithmetic of Wx_t and Uh_t is matrix-vector multiplication that accounts for the main portion of computational workload; "$+, \cdot$" is the element-wise operation of vectors and "σ, tanh" is the activation function. In particular, as the unrolled cell iterations in Fig. 1, a *context link* connects the neighboring cell iterations via the output vector h_t and the cell-state vector c_t. The context link leads to the successive iteration depends on the results from the previous one.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{1}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{4}$$

$$h_t = o_t \cdot tanh(c_t) \tag{5}$$

Prior work presented that the regular weight matrix of LSTM networks contains redundancy, and a portion of matrix elements can be pruned to zero with an acceptable accuracy loss [15]. The sparse weight matrix significantly benefits the computational throughput

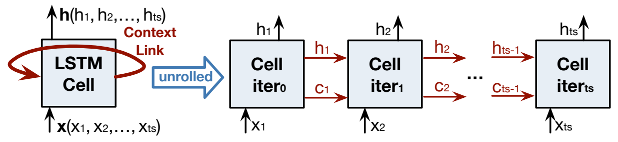

Figure 1: The LSTM cell and its unrolled schematic.

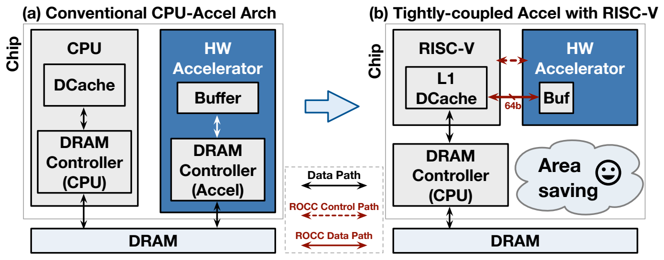

Figure 2: Two kinds of heterogeneous architecture.

by decreasing the workload on arithmetic operations. Nevertheless, sparse LSTM achieves unsatisfying performance on the general-purpose CPU due to the control-flow overhead. Thus, the specific hardware should be designed to fully utilize the advantage of sparse computation.

2.2 Tightly-coupled Accelerator with RISC-V

Although LSTM has promising performance in various machine learning applications, the heavy workload impedes the deployment of LSTM models in embedded scenarios. Heterogeneous architecture is a solution to this problem. As Fig. 2(a) shows, the typical heterogeneous system contains a general-purpose CPU and a hardware accelerator tailored for the specific application. Due to the closed developing environment of CPU hardware, the data exchange between CPU and accelerator needs to be conducted via DRAM, which introduces a long latency (≈20 ns for each word) in the collaboration. The open-source RISC-V eco-system brings a new communication mechanism to the heterogeneous system design that alleviates the communication-latency problem. As Fig. 2(b) shows, RISC-V specification provides a *ROCC interface* that allows the accelerator to directly access the data in the L1-DCache of CPU. Besides reducing the communication latency, another advantage of cache sharing is that the DRAM controller in the accelerator side is no longer needed. Meanwhile, the size of the on-chip buffer can be reduced. Consequently, the tightly-coupled accelerator in Fig. 2(b) significantly reduces the chip-area cost and power consumption compared to the conventional heterogeneous architecture in Fig. 2(a).

However, the bandwidth of the ROCC interface is limited (64-bits per clock cycle). To use the RISC-V based heterogeneous system in LSTM inference, the *data format* of sparse weight matrix should be carefully designed to efficiently utilize the interface bandwidth.

2.3 Related Works on LSTM Acceleration

The techniques used by previous LSTM accelerators are summarized in Table 1 to highlight the design challenge and research gap of the embedded LSTM in this work. In most previous works, the weight matrix is stored in a large on-chip buffer to avoid re-accessing the off-chip DRAM. These works use the sparse representation of weight matrix or structured-dense matrix with a periodic repeat

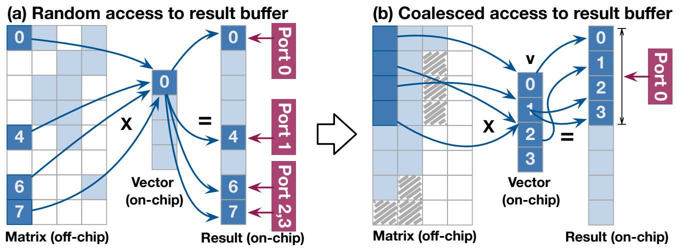

Figure 3: Parallel MACC operations in SpMV.

pattern to reduce the on-chip storage requirement. Nevertheless, the size of compressed weight matrix exceeds 10 MB which is infeasible in the chip-area sensitive scenario. In terms of workload parallelization, three schemes were adopted in the previous works. First, *intra-cell* parallelism concurrently executes multiple operations in one cell iteration. The scale of parallelism is proportional to the memory throughput of weight matrix. Second, *batch* parallelism is adopted in the cloud scenario that processes multiple input sequences in parallel. The parallelism scale increases via duplicating the hardware module and broadcasting the weight value to the parallel modules. Third, *inter-cell* parallelism is proposed in [16]. With the run-time preprocessing on the weight and input sequence, some context links can be cut off, thus the unfolded cell link (Fig. 1) can be partitioned to parallel slices.

In the latency-sensitive embedded scenario of this work, *sequence-batch* mechanism introduces long latency that the system starts execution after collecting all batch sequences. Besides, the bandwidth limitation of ROCC interface impedes the scale-up of the *intra-cell* parallelism. The *inter-cell* mechanism proposed in [16] brings extra computation overhead, that does not fit the scenario of E-LSTM. With the platform limitation, we investigated a new parallel scheme that achieves lower latency without overhead.

Table 1: Design methods of LSTM acceleration.

Design	Weight	Sparse/Dense	Parallelization Technique	Scenario
ESE [7]	On-chip	Structured Sparse	Intra-cell & Batch	Cloud
C-LSTM [14]	On-chip	Structured Dense	Intra-cell & Batch	Cloud
Park et al. [12]	Off-chip	Sparse	Intra-cell & Batch	-
Zhang et al. [16]	On-chip	Dense	Inter- & Intra-Cell (with run-time overhead)	Mobile-GPU
E-LSTM (this work)	Off-chip	Sparse	Inter- & Intra-Cell (without run-time overhead)	Embedded, IoT

3 METHOD

In this section, we first describe the eSELL sparse format for weight matrix, that not only decreases data-transfer but also reduces the area-cost of on-chip buffer. Subsequently, we prepose the inherent sparsity of hidden state in LSTM computation and the inter-cell parallelization scheme based on it.

3.1 eSELL Format of Sparse Model

3.1.1 Access Coalescing of On-chip Buffer. Compressed Sparse Column/ROW (CSC/R) and Encoded Column (EC) are used to represent the sparse weight matrix in the previous LSTM accelerators [7, 12]. Fig. 3(a) depicts the data access of both methods in Sparse Matrix-vector multiplication (SpMV). Although the non-zero elements of each weight column are arranged to successive memory addresses,

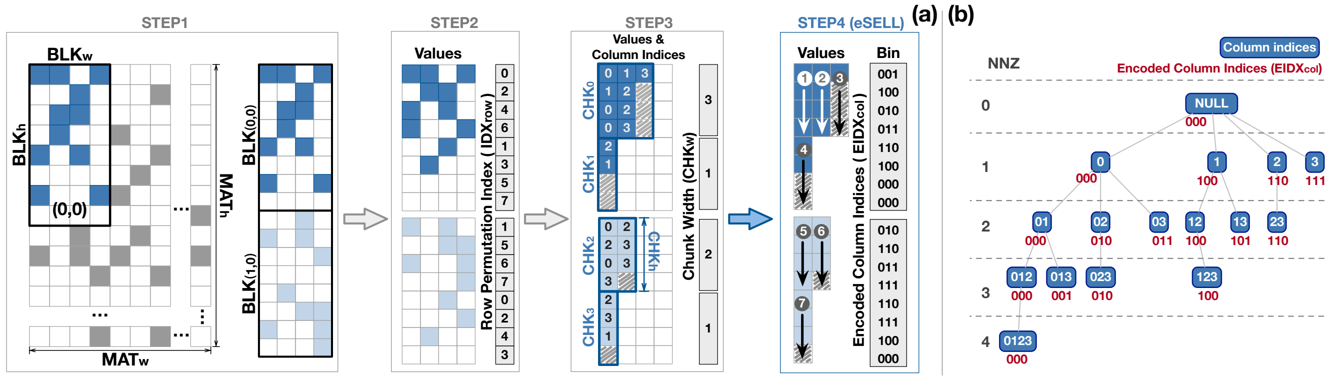

Figure 4: Construction steps of eSELL format for the sparse matrix.

the results show random accesses to the on-chip buffer since the address is determined by the indices of the compressed column. While parallel multiply-accumulate (MACC) operations are conducted in each cycle, a multi-ports SRAM must be used to construct the result buffer for *random* and *parallel* access (e.g., 4-ports SRAM is used in Fig. 3(a)). Note that the port number ($\#port$) greatly affects the chip-area cost and timing performance. According to the SRAM evaluation model in [2], the SRAM area is proportional to $(\#port)^{0.7}$, and the access latency is proportional to $(\#port)^{1.3}$. Thus, we investigate an advanced sparse format that coalesces the results access of the parallel MACC computation to continuous data in the on-chip buffer. By doing so, a single-port SRAM with wider port width can fulfill the requirements, and the SRAM reshaping does not increase the area cost. For example, in Fig. 3(b), the non-zero elements of each row are shifted to left-hand-side during compression. Thus the parallel MACC computation with the newly constructed weight column have continuous access to the result buffer.

3.1.2 eSELL Sparse Matrix Representation. We propose Encoded-SELL (eSELL), an improved version of SELL sparse-matrix format [9] that coalesces the data access and reduces the port number of result buffer. The construction of eSELL format consists of four steps. A demonstration is presented in Fig. 4. In STEP1, the original sparse matrix is partitioned to *blocks* and each block is encoded by the subsequent steps independently. STEP2 permutates the rows that sort the rows with descending order by the number of non-zero (NNZ) elements in each row. Meanwhile, the original row indexes before permutation (denoted as IDX_{row}) are recorded. In STEP3, the non-zero values are shifted and gathered to the left-hand-side. Also, the original column indices (denoted as IDX_{col}) are recorded and labeled on Fig. 4(a)-STEP3. Subsequently, one block is partitioned to *chunks* and each chunk is regarded as a dense matrix during computation. Due to the row permutation in STEP2, the first row contains the most non-zero elements in the chunk. Thus, the value of NNZ in the first row of each chunk is recorded as the *chunk width* (CHK_w). The zero elements in the rest rows of the chunk are kept (shade squares in Fig. 4(a)-STEP3) that construct a regular dense chunk. In Fig. 4(a) example, parameters BLK_h, BLK_w and CHK_h are set to 8, 4 and 4 respectively. As a result, the width of CHK0 and CHK1 are 3 and 1. The SpMV computation with eSELL adopts the chunk-column-major order. As indicated by arrows on Fig. 4(a)-STEP4, CHK_w columns of a chunk are sequentially fetched in computation followed by the the columns in the chunk below. With this scheme, the parallel accesses to the result buffer are coalesced to 8 (BLK_h) continuous elements that can be realized with a single-port SRAM.

To further compress the sparse representation, we present STEP4 that encodes the column indices of each row to a 3-bit value. The encoding scheme is presented as a tree in Fig. 4(b). Each node is labeled with a sequence of column indices from a chunk row, and the depth of each node is equal to the NNZ of this row. All possible sequences of column indices with $BLK_w = 4$ are listed in the tree, and the corresponding coding is labeled under each node. The node shares the same coding with its leaf nodes in the left-hand-side, and the uniqueness of coding is kept among nodes with the same depth. For example, in the chunk with $CHK_w = 3$, the column indices sequence $(1, 2, 3)$ and $(1, 2)$ have the same coding. These two rows perform the same computation in SpMV due to the zero element is remained in STEP3, and each row contains 3 (CHK_w) elements. In Fig. 4(a)-STEP4, the Encoded-IDX_{col} ($EIDX_{col}$) is listed corresponding to the column indices labeled on elements of STEP3.

3.1.3 Alignment with ROCC Interface. In E-LSTM, we set BLK_w, BLK_h to 4, 8 respectively and each block is partitioned to 2 chunks as Fig. 4. The value of matrix element is represented by 16-bit half-precision floating-point format. The eSELL representation is packed as 64-bit words and sent to the accelerator sequentially in the chunk-column-major order. As Fig. 5 shows, the packed storage of a block is composed of two chunk heads and element values. The chunk head contains the compression information IDX_{row}, $EIDX_{col}$ and CHK_w that have 27 bits in total. The bit-width of each component is labeled on each segment of the ROCC word in Fig. 5. The element values of a chunk column are packed to a word and stored following the block head.

3.2 Optimization for Inter-cell Parallelization

3.2.1 Accelerator Architecture and Throughput Bottleneck. A generic accelerator architecture used in E-LSTM is depicted in Fig. 6. The

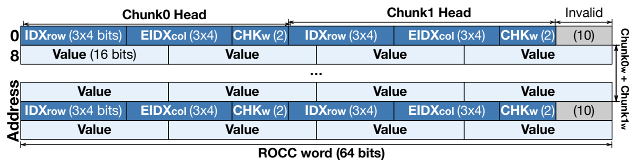

Figure 5: Bit-wise packing to construct ROCC words.

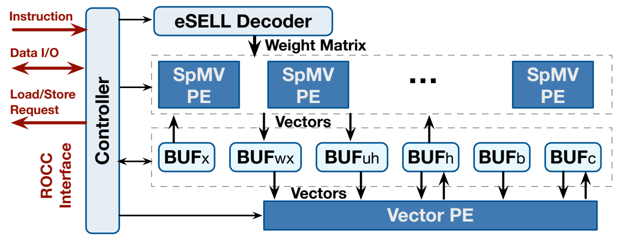

Figure 6: Generic accelerator architecture in E-LSTM.

CPU core sends the customized instructions via ROCC interface to the `Controller` in accelerator, which generates control signals of submodules. After loading the parameters of LSTM networks layer to the control registers, the `Controller` sends *load request* of input and bias vector (x, b) on ROCC interface, and then stores the response data to the buffer (BUF_x, BUF_b) on-chip. Subsequently, `Controller` loads the eSELL representation of weight matrix (W, U) to `eSELL Decoder` and starts the computation on SpMV processing element (`SpMV PE`). The result vector of SpMV is stored in BUF_{wx} and BUF_{uh} for the following vector operation as Equ. 1-5. `Vector PE` performs the vector-wise computation of each LSTM gate, and it stores the hidden state (h_t) to BUF_h. After one cell iteration, the output vector h_t are sent back to CPU via ROCC.

Inter-cell parallelism in LSTM is not an embarrassing parallel case as the processing throughput cannot be easily scaled up by increasing the number of PE (N_{pe}). We give a temporal analysis of the bottleneck in the matrix-vector multiplication, that accounts for the majority of computation workload and ROCC communication. Because the workload of vector operation (on `Vector PE`) is much less and can be simultaneously executed with SpMV, its time consumption is omitted in the following content. Fig. 7(a) shows the working status of accelerator with single `SpMV PE`, and the PE is capable of calculating 4 MACC operation with the weight values in a ROCC word. The PE calculates Wx_t and Uh_t in interleave, and the ROCC is fully utilized. When PE number is increasing, as Fig. 7(b), Wx_t of different cell iterations can be calculated in parallel with weight sharing, while the Uh_t can only be processed in sequential due to the data dependency between successive iterations. Moreover, it is infeasible to compute Wx_t and Uh_t concurrently because of the conflict on ROCC interface in loading the W and U. Thus, the parallel PEs are low utilized during the run-time.

3.2.2 *Computation with Inherent Sparsity of Hidden State.*

With the timing diagram in Fig. 7(b), an intuitive optimization target is to shorten the computational period of Uh_t which is the bottleneck of overall throughput. The inherent sparsity of hidden state (h_t) contributes to this. LSTM uses the Sigmoid (σ) and hyperbolic tangent (tanh) as the activation functions. Note that the σ function outputs a value close to zero in a significant probability. For instance, assuming the value of input x uniformly distributes in $[-6, 6]$, the probability of $P(\sigma(x) < 0.1) \approx 0.32$. Thus, a considerable portion of h_t values are close to zero, as it is obtained from multiplying $\sigma(\cdot)$ by $\tanh(c_t)$ and $\tanh(c_t)$ ranges in $(-1, 1)$. Furthermore, as Equ. 3 , two $\sigma(\cdot)$ items exist in the multiplication factors of c_t, thus $\tanh(c_t)$ vector also contains considerable elements close to zero. As the values in h_t that close to zero have a tiny impact on both the final results and the subsequent computation, we propose to set these values to zero during run-time, and we name the sparsity of h_t as

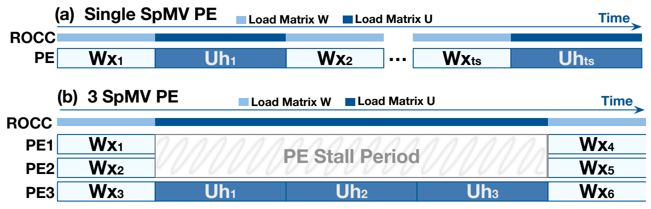

Figure 7: Working status of SpMV-PEs and ROCC interface.

inherent sparsity because the LSTM algorithm inherently results to it. The sparsity of h_t (denoted as Sp_h) is evaluated in Section 4 with the real-world LSTM models.

Based on the inherent sparsity of h_t, the computation of Uh_t becomes Sparse Matrix-Sparse Vector multiplication (SpMSpV). Because E-LSTM adopts the column-major order in matrix-vector multiplication, the accelerator loads and computes only a portion of weight columns corresponding to non-zero values in h_t. Thus, the workloads of both the PE and ROCC interface decrease and this reduces the time consumption of Uh_t.

3.2.3 *Computation with Cell Fusion.*

Although the time consumed on Uh_t is decreased, the PE stall period on Fig. 7(b) still exists and significantly degrades the performance. We propose the *cell fusion* scheme to alleviate this limitation via increasing the arithmetic intensity of weight matrix. As Fig. 8 example, the accelerator has N_{pe} PEs (N_{pe}=3), ($N_{pe} - 1$) of them calculates Wx_t in parallel and the rest one processes Uh_t. In particular, operations of Wx_t from different cell iterations (t) are fused and computed by one PE in interleave. Specifically, once the PE obtains a weight value from the ROCC interface, it multiplies the weight to several x_t vectors in interleave. The number of fused cell iteration is defined as the fusion factor and denoted as N_{fuse}. In the Fig. 8 example, N_{fuse} is 3 and each PE computes Wx_t of 3 cell iterations concurrently (e.g., $Wx_{1,2,3}$). Because each value of W is reused for N_{fuse} times on one PE, the cell fusion increases the arithmetic intensity of W and decreases the ROCC load request to a frequency of once per N_{fuse} cycles. Therefore, $(N_{fuse}-1)/N_{fuse}$ of the ROCC period is available, and the free period can be used to load matrix U which drives the Uh_t computation in the rest one PE. Compared to Fig. 7(b), cell fusion with a proper N_{fuse} effectively decreases the stall period. The cost of cell fusion is N_{fuse} times on-chip memory usage for the temporary results vector buffer BUF_{wx}.

3.2.4 *Fusion Factor Selection.*

The fusion factor value N_{fuse} is the key to the efficiency of E-LSTM accelerator. In the case without cell fusion (N_{fuse}=1), ($N_{pe} - 1$) PEs stalls during Uh_t as Fig. 7(b). In the case that all cell iterations of the input sequence are fused (N_{fuse}=ts), the PEs stall at the beginning and end of computation (as Fig. 8) because Uh_t should start after the entire Wx_t workload. Therefore, we construct a model to estimate the time consumption on processing a sequence and help to select the proper N_{fuse}. As labeled on Fig. 8, the entire period is separated into three segments in which PEs perform different behaviors. In *prolog* segment, the accelerator only calculates the Wx_t as the following h computation depends on it. The *main* segment computes Wx_t and Uh_t simultaneously, and the ROCC loads W, U in interleave. In the *epilog* segment, one PE finishes the rest Uh_t that corresponds to the last group of fused cell iterations. The time consumption on one sample

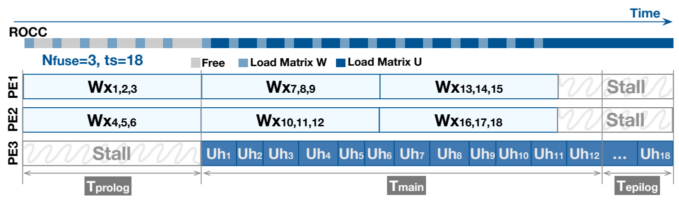

Figure 8: Working status with cell-fusion optimization.

sequence (T) is the sum of the periods of three segments. Equ. 6 gives the computation model,

$$\begin{aligned}
T_{prolog} &= T_{wx} \times N_{fuse} \\
T_{main} &= (\lceil \frac{ts}{(N_{pe}-1)\times N_{fuse}})\rceil - 1) \times T_{iter} \\
T_{epilog} &= ((ts-1)\%((N_{pe}-1)\times N_{fuse}) + 1) \times T_{uh} \\
T_{iter} &= T_{prolog} + \max(0, T_{uh} \times N_{fuse} \times (N_{pe}-1) - T_{prolog} \times \frac{N_{fuse-1}}{N_{fuse}}) \\
T_{wx} &= len(x) \times len(h) \times (1 - Sp_w) \\
T_{uh} &= len(h) \times len(h) \times (1 - Sp_u) \times (1 - Sp_h)
\end{aligned} \tag{6}$$

where ts is the time step in each sample sequence, $len(x, h)$ is the vector length of x_t and h_t that determines the weight matrix size. $Sp_{w,u,h}$ represent the sparsity of matrix W, U and hidden state h_t respectively. The objective of performance fine-tuning is to find a proper N_{fuse} that minimizes the T and can be formatted as follows,

$$\begin{aligned}
\underset{N_{fuse}}{\text{minimize}} \quad & T(N_{fuse}) = T_{prolog} + T_{main} + T_{epilog} \\
\text{subject to} \quad & N_{fuse} \times N_{pe} \times len(x) \le MaxSize(\mathsf{BUF}_{\mathrm{x}}), \\
& N_{fuse} \times N_{pe} \times 4 \times len(h) \le MaxSize(\mathsf{BUF}_{\mathrm{wx}}), \\
& N_{fuse} \times N_{pe} \le ts.
\end{aligned} \tag{7}$$

As mentioned above, cell fusion increases the storage requirements on the vector x_t and the intermediate result vector of Wx_t. Thus, N_{fuse} should guarantee the storage requirement is less than the corresponding buffer size inside the accelerator. In practice, the optimal N_{fuse} is obtained by linear searching the feasible field and selecting the one with minimum T.

4 EXPERIMENTS AND EVALUATION

4.1 Experimental Setup

The entire E-LSTM framework has been implemented and embedded with the RISC-V toolchains. In terms of compilation software, we wrote Python scripts to translate the weight data to the eSELL representation and then generate the ROCC words in binary format; Meanwhile, scripts were implemented to find the proper cell fusion factor (N_{fuse}) with the input of the LSTM model parameters and N_{pe} in the accelerator. On the hardware system, with the convenience of Spike (the cycle-level simulator in RISC-V eco-system), we built the behavioral model of the E-LSTM accelerator and embedded it to the CPU simulator to evaluate the performance of the entire heterogeneous system.

4.2 LSTM Benchmarks

As listed in Table 2, three real-world tasks were used as the LSTM benchmarks. These tasks belong to the fields of Optical Character

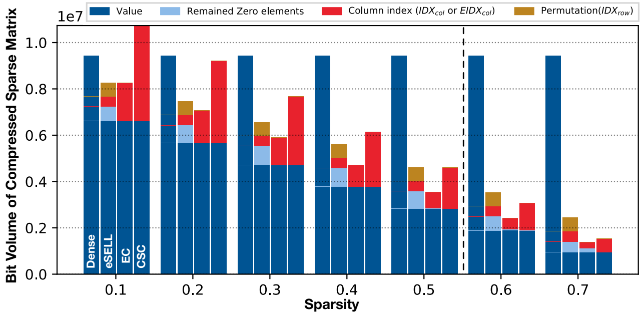

Figure 9: Storage requirements of different sparse formats.

Recognition (OCR) and Language Model (LM), which are widely adopted in the machine comprehension and translation on embedded systems. In the OCR case, the 2-layer LSTM networks were trained with the handwritten-digits dataset MNIST [10], and recognition accuracy is the evaluation metric. The LM benchmarks were used to predict the subsequent texts with the input word sequence. We built two LM networks with PTB [13] and Wikitext [11] dataset respectively. Perplexity (PPL) was employed as the metric, and a lower PPL score represents a higher accuracy of prediction. Each of the three tasks contains two LSTM layers and one fully connected layer. The models are trained and verified within PyTorch framework. Note that the values in the model are cast to *half-precision floating-point* format (16-bit) in the inference.

To obtain the model with sparse weight matrix, we adjusted the sparsity of W and U (Sp_w, Sp_u) layer by layer and selected the proper values considering the trade-off between accuracy and model sparsity. The sparsity of each weight matrix is listed in Table 2.

Table 2: Benchmark LSTM layers.

Name	Layer	len(x)	len(h)	ts	Sp_w	Sp_u	Sp_h	Score
OCR	LSTM1	28	128	28	0.3	0.5	0.22	98.68/98.61/98.11
(MNIST)	LSTM2	128	128	28	0.2	0.4	0.29	the higher score, the better
LM	LSTM1	800	800	35	0.2	0.5	0.56	81.33/81.67/88.52
(PTB)	LSTM2	800	800	35	0.2	0.6	0.41	the lower score, the better
LM	LSTM1	1500	1500	35	0.4	0.5	0.37	101.63/102.15/106.5
(Wikitext)	LSTM2	1500	1500	35	0.3	0.4	0.39	the lower score, the better

4.3 Effectiveness of the eSELL Sparse Format

The performance E-LSTM is limited by the weight loading throughput on ROCC interface, and it is proportional to the compressed matrix size. The eSELL format is evaluated and compared to other matrix representations (dense, CSC, Encoded Column (EC)). Fig. 9 gives a data breakdown of the dense/compressed matrix in size of 768×768. The eSELL format has an apparent advantage compared with the dense and CSC format when the matrix sparsity is less than 60%. Note that the values of weight sparsity in all benchmark layers are in this range. Compared with the original SELL format, the column-indices encoding scheme in eSELL reduces the size of column indices by 10%-50%. The competitive advantage of eSELL is the lower buffer-area cost for its access-coalescing property. Although the EC matrix representation has slightly less data volume than eSELL, the corresponding area cost on the result-vector buffer is 2.7× the consumption in eSELL-based design.

[1]The scores are measured with dense-weight model / sparse-weight model / sparse-weight model & sparse hidden state respectively.

4.4 Inherent Sparsity of Hidden State

The inherent sparsity reduces the computation time of Uh_t, and it is an important factor in the cell fusion optimization. Different from the weight sparsity selection that directly adjusts the proportion of zero elements in the matrix, the sparsity of h_t is determined by a threshold parameter (th) for each LSTM layer. The h_t elements are compared to th during run-time, and the ones with smaller absolute value are set to zero (pruned). Note that the granularity of pruning in the E-LSTM is four elements in vector h_t instead of one. It is because four weight columns compose the chunk column in eSELL, and each chunk column is loaded to the accelerator as a whole. Thus, one chunk column is skipped during computation if the absolute value accumulation of the corresponding four h_t elements is less than the th. We considered the trade-off between accuracy and sparsity of hidden state, then selected the proper threshold (th) for each layer. The average h_t sparsity over the entire dataset is given in Table 2, that was used in the cell fusion factor selection.

4.5 Performance

The benefits brought by the inherent sparsity of h_t and the cell fusion were evaluated with the benchmark LSTM layers. The LSTM accelerator was configured with 3 SpMV-PEs, and each PE performs 4 MACC operations per cycle. In terms of software, the E-LSTM framework calculated the optimal N_{fuse} with the parameters in Table 2 and then generated the corresponding instructions. With the RISC-V toolchain compilation, the executable file was loaded to the Spike-based simulator for a behavioral emulation. The program was repeatedly executed with 500 different sample sequences, and the average time consumption (measured in clock cycle) on one sequence was obtained from the simulator. To present the improvement of E-LSTM optimizations, we also report the performance in the settings that without E-LSTM optimization ($Sp_h = 0, N_{fuse} = 1$) and with only sparse h_t ($N_{fuse} = 1$).

Fig. 10 shows the system time consumption on processing one sample sequence. In each of the six benchmark layers, the experiment without E-LSTM optimization (original) is the baseline for comparison. The rest two experiments with E-LSTM optimization show apparent improvement over the baseline, and the corresponding value of speedup is labeled on each bar. Note that the optimal N_{fuse} values are labeled on top of the third bar in each layer, that varies with different network parameters and sparsities in Table 2. With the benchmark layers, E-LSTM achieves 1.4×-2.2× speedup to the original scheme with slight overhead on the vector buffer consumption. From the results, we learned the Sp_h significantly affects the improvement of E-LSTM optimization, as the PTB-LSTM1 achieves the best speedup with the maximal Sp_h. In layer MNIST-LSTM1, the improvement achieved by employing cell fusion is not significant. This is because the $len(x)$ and $len(h)$ is greatly different thus the throughput is limited by Uh_t computation. Besides the benchmark layers, the users are easy to obtain the optimized performance for their LSTM applications with the E-LSTM framework before tapping out the entire system.

5 CONCLUSION

In this work, we propose an efficient scheme for LSTM inference on the embedded heterogeneous system with the sparse weight matrix.

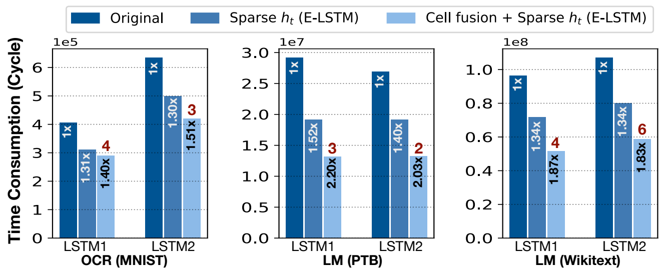

Figure 10: Performance in different optimization schemes.

The eSELL matrix representation significantly reduces the area cost of on-chip buffer. Furthermore, we observed the inherent sparsity of hidden state in LSTM inference and proposed the cell-fusion parallel scheme, that contributes to a higher pipeline efficiency and a significant speedup under the bandwidth limitation on CPU-accelerator interface.

ACKNOWLEDGMENTS

This work is supported by Beijing Natural Science Foundation (No.L172004), Municipal Science and Technology Program under Grant Z181100008918015; Croucher Foundation (Croucher Innovation Award 2013) and the Research Grants Council of Hong Kong grant number CRF C7047-16G, GRF 17245716.

REFERENCES

[1] Robert Adolf et al. 2016. Fathom: Reference workloads for modern deep learning methods. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.
[2] Banit Agrawal et al. 2007. Guiding architectural sram models. In *Computer Design, 2006. ICCD 2006. International Conference on*. IEEE, 376–382.
[3] Kyunghyun Cho et al. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).
[4] Alex Graves et al. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.
[5] Klaus Greff et al. 2017. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2017), 2222–2232.
[6] Yijin Guan et al. 2017. FPGA-based accelerator for long short-term memory recurrent neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 629–634.
[7] Song Han et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 75–84.
[8] Karl Moritz Hermann et al. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*. 1693–1701.
[9] Moritz Kreutzer et al. 2014. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. *SIAM Journal on Scientific Computing* 36, 5 (2014), C401–C423.
[10] Yann LeCun et al. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010), 18.
[11] Stephen Merity et al. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).
[12] Junki Park et al. 2018. Maximizing system performance by balancing computation loads in LSTM accelerators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 7–12.
[13] Ann Taylor et al. 2003. The Penn treebank: an overview. In *Treebanks*. Springer, 5–22.
[14] Shuo Wang et al. 2018. C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 11–20.
[15] W Wen et al. 2018. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*.
[16] Xingyao Zhang et al. 2018. Towards Memory Friendly Long-Short Term Memory Networks (LSTMs) on Mobile GPUs. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 162–174.