SACC: Split and Combine Approach to Reduce the Off-chip Memory Accesses of LSTM Accelerators

Saurabh Tewari
Dept. of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi, India
saurabh.tewari@cse.iitd.ac.in

Anshul Kumar
Dept. of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi, India
anshul@cse.iitd.ac.in

Kolin Paul
Dept. of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi, India
kolin@cse.iitd.ac.in

***Abstract*—Long Short-Term Memory (LSTM) networks are widely used in speech recognition and natural language processing. Recently, a large number of LSTM accelerators have been proposed for the efficient processing of LSTM networks. The high energy consumption of these accelerators limits their usage in energy-constrained systems. LSTM accelerators repeatedly access large weight matrices from off-chip memory, significantly contributing to energy consumption. Reducing off-chip memory access is the key to improving the energy efficiency of these accelerators. We propose a data reuse approach that splits and combines the LSTM cell computations in a way that reduces the off-chip memory accesses of LSTM hidden state matrices by 50%. In addition, the data reuse efficiency of our approach is independent of on-chip memory size, making it more suitable for small on-chip memory LSTM accelerators. Experimental results show that our approach reduces off-chip memory access by 28% and 32%, and energy consumption by 13% and 16%, respectively, compared to conventional approaches for character level Language Modelling and Speech Recognition LSTM models.**

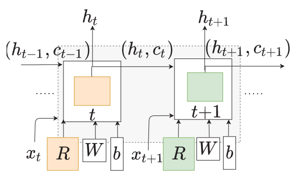
***Index Terms*—LSTM, off-chip memory access, data-reuse**

I. Introduction

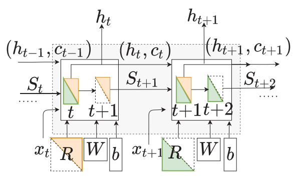
Recurrent neural networks (RNN) are deep learning algorithms specialized in handling sequential data problems. Long Short-Term Memory networks (LSTM) [1] are variants of RNNs designed to handle long-range dependencies. Several customized accelerators are proposed [2]–[4] to speed up the inference of LSTM. These accelerators have limited on-chip memory, specifically the accelerators targeted for embedded devices. LSTM computations involve matrix-vector multiplications, and the size of these matrices can be significant (in several megabytes) and often exceed the size of the accelerator's on-chip memory. These matrices are accessed repeatedly from the off-chip memory, which results in a large volume of off-chip memory accesses and energy consumption. Reducing off-chip memory access is the key to improving the energy efficiency of LSTM accelerators ([5]–[7]). As the LSTM cell state depends on the previous time-step cell state, reusing the weights among the different steps to reduce the off-chip memory access is challenging in LSTM.

In this paper, we propose a novel data reuse scheme that reduces the off-chip memory accesses of LSTM accelerators. Fig. 1 shows the LSTM cell computations for two consecutive time steps using conventional and the proposed approach. Conventional approaches access the R matrix at each step to compute the hidden state vector h_t, as shown in Fig. 1a, which results in a large volume of off-chip memory accesses. The proposed approach splits the computations into partial sums and computes the partial sum of two consecutive time steps (S_t and S_{t+1}) to reuse the weights of R, as shown in Fig. 1b. Our main contributions in this paper are as follows,

- We present a weight reuse approach that reduces the off-chip memory accesses of hidden state weight matrices by approximately half.
- We analyzed the results to show that our approach is independent of on-chip buffer sizes and effective for accelerators with small on-chip buffer sizes.



(a) Conventional approach



(b) Proposed approach

Fig. 1. LSTM cell computations for consecutive time-steps a) R accessed at each step. b) half of R accessed at each step.

II. Background

LSTM has recurrent connections to capture the long and short-term dependencies. Typically the computations of LSTM cell is described by the following equations

$$\begin{aligned}
i&=\sigma(W^i \cdot x_t + R^i \cdot h_{t-1} + b^i)\\
f&=\sigma(W^f \cdot x_t + R^f \cdot h_{t-1} + b^f)\\
g&=\tanh(W^g \cdot x_t + R^g \cdot h_{t-1} + b^g)\\
o&=\sigma(W^o \cdot x_t + R^o \cdot h_{t-1} + b^o)\\
c_t&=f \odot c_{t-1} + i \odot g\\
h_t&=o \odot \tanh(c_t)
\end{aligned} \tag{1}$$

where x_t is the input, h_t is the hidden state, and c_t is the cell state at time t. i, f, g, o are the computed gate values. $\odot$ denotes the element-wise multiplications. W^j and R^j are the input and hidden state weight matrices, and b^j is the bias vector, learned during the training process, where $j \in \{i, f, g, o\}$. The dimension of h_t is referred to as the number of hidden states of the LSTM (N). At every time step, x_t is taken as

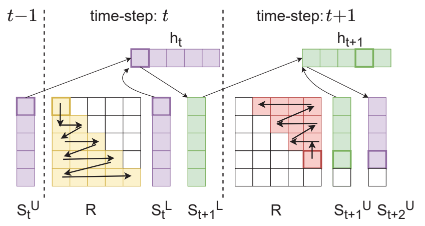


Fig. 2. Splitting the hidden state vector computations into partial sums

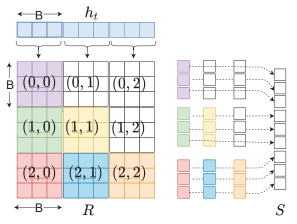


Fig. 3. Partitions of R in $B\times B$ blocks and partial sum computations.

input, and cell state (c_t) and hidden state (h_t) are computed using (1). LSTM accelerators have small on-chip memory. The large weight matrices R and W are stored in off-chip memory. The dependency of h_t and c_t on the previous time step computations makes the reuse of weight matrix R a challenge. Thus the weights are accessed from the off-chip memory at every step, resulting in sizeable off-chip memory accesses and high energy consumption of these accelerators. This work focuses on reducing the off-chip memory accesses of R during the LSTM inference phase.

III. Split And Combine Computations Approach

In this section, we first describe the basic idea of the Split And Combine Computations (**SACC**) approach and then extend the basic approach to block-wise reuse of data.

A. Basic Approach

The computation of the h_t can be expressed as shown below

$$h_t[k] = F(S_t[k] + q_t[k]) \tag{2}$$

where F is a non-linear function. q_t is computed as $W\cdot x_t+b$ and its computations are independent of previous step cell states. $S_t[k]$ is the sum of N product terms as shown below,

$$S_t[k] = \sum_{n=0}^{N-1} R[k][n]\cdot h_{t-1}[n] \tag{3}$$

$S_t[k]$ can be computed as a sum of the following two partial sums $S_t^L[k]$ and $S_t^U[k]$

$$S_t^L[k] = \sum_{n=0}^{k} R[k][n]\cdot h_{t-1}[n] \tag{4}$$

$$S_t^U[k] = \sum_{n=k+1}^{N-1} R[k][n]\cdot h_{t-1}[n] \tag{5}$$

Equation (4) uses the lower-diagonal and diagonal elements of R (R^L), and (5) uses the upper diagonal elements of R (R^U). As shown in Fig. 2, R^L and R^U are accessed in consecutive time steps and reused in the partial sum computations of two steps. At time step t, S_t^U and h_{t-1} are the inputs from the previous time step, and R^L is reused to compute the partial sums S_t^L and S_{t+1}^L. Input S_t^U is added to S_t^L to compute h_t, and S_{t+1}^L is passed to $(t+1)^{th}$ step computations. In the same way, at time step $t+1$, R^U is reused to compute S_{t+1}^U and S_{t+2}^U. Elements of R^L are accessed from top to bottom, left to right, while elements of R^U are accessed in the reverse order to satisfy the dependencies. As shown in Fig. 2, the proposed approach accesses the weight matrix R once, to compute h_t and h_{t+1}.

B. Block-wise reuse

The proposed approach partitions R into square blocks of size $B\times B$, that fits in the accelerator's on-chip memory, as shown in Fig.3. Each block can be indexed as (r,m), where $0\leq r,m\leq(\lceil\frac{N}{B}\rceil-1)$. The proposed approach computes h_t in $\lceil\frac{N}{B}\rceil$ steps and at each step computes a slice of length B. The k^{th} element of r^{th} slice of h_t can be computed as following

$$h_t[B*r+k]=F(\sum_{m=0}^{\lceil\frac{N}{B}\rceil-1} S_{(r,m)}[k] + q_t[B*r+k]) \tag{6}$$

$$S_{(r,m)}[k]=\sum_{j=0}^{B-1} R[B*r+k][B*m+j]\cdot h_{t-1}[B*m+j] \tag{7}$$

,where $0\leq k\leq B-1$. The summation $\sum_{m=0}^{\lceil\frac{N}{B}\rceil-1} S_{(r,m)}[k]$ in (6) can be expressed as a sum of the following partial sums

$$\sum_{m=0}^{\lceil\frac{N}{B}\rceil-1} S_{(r,m)}[k]=\sum_{m=0}^{r} S_t^L[k]+\sum_{m=r+1}^{\lceil\frac{N}{B}\rceil-1} S_t^U[k] \tag{8}$$

$S_t^L[k]$ uses the lower-diagonal and diagonal blocks of R (R^L), and $S_t^U[k]$ uses the upper diagonal blocks of R(R^U). The SACC approach reuses blocks of R to compute the partial sums of two consecutive time steps, similar to the approach described in III-A.

Algorithm 1 describes the computations of the SACC approach. The dimensions of W, R, b, and x_t are $4N\times L$, $4N\times N$, $4N\times 1$, and $L\times 1$, respectively. The algorithm stores the vectors h_t, c_t, and the partial sum vectors (s_{t+1}) in the on-chip memory and accesses the weights from the off-chip memory. It first computes the vector q_t as $W\cdot x+b$, at line 2 and then invokes the procedures UpDiagReuse at line 4 or LowDiagReuse at line 7 at alternate time steps. LowDiagReuse accesses blocks of R^L, and UpDiagReuse accesses blocks of R^U. The procedures have two nested loops. LowDiagReuse traverses the blocks from top to bottom (at line 13), left to the right (at line 15), while the UpDiagReuse traverses the blocks in the opposite order. The inner loop accesses the $(r,m)^{th}$ block of R from the off-chip memory and reuses it to compute the partial sums s_{t+1}^B and s_{t+2}^B. The outer loop iterations compute r^{th} slices of h_{t+1}, c_{t+1}, and s_{t+2}. When all the blocks of the r^{th} row are processed, s_{t+1}^B has the total sum, which is then used to compute the r^{th} slice of the vectors h_{t+1} and c_{t+1} using LSTMEquations at line 21.

Algorithm 1 SACC algorithm

```
procedure CompLSTMCell($W$,$R$,$b$,$x_{t+1}$)
    $q_{t+1} \leftarrow$ MxV($W, x_{t+1}, 4N, L$)+$b$
    if ($stage$ is even) then
        ($h_t, c_t, s_{t+1}$)←UpDiaReuse($R, q_t, h_t, c_t, s_{t+1}$)
        $stage \leftarrow odd$
    else
        ($h_t, c_t, s_{t+1}$)←LowDiaReuse($R, q_t, h_t, c_t, s_{t+1}$)
        $stage \leftarrow even$
    end if
    return ($h_t$)
end procedure
procedure LowDiagReuse($R, q_t, h_t, c_t, s_{t+1}$)
    for $r \leftarrow 0$ to ($\lceil \frac{N}{B} \rceil - 1$) do
        $(i_s, i_e) \leftarrow (r \cdot B, (r+1) \cdot B - 1), s_{t+2}^B \leftarrow \vec{0}$
        for $m \leftarrow 0$ to $r$ do
            $\mathbf{R}^B$←getDDRBlks($R, r, m, B$)
            $(h_t^B, c_t^B, q_t^B) \leftarrow$ GetSlice($h_t, c_t, q_t, m$)
            $s_{t+1}^B \leftarrow$ GetSlices($s_{t+1}, m$)
            $s_{t+1}^B \leftarrow s_{t+1}^B$+MxV($\mathbf{R}^B, h_t^B, 4B, B$)
            if $m = r$ then
                $(h_{t+1}^B, c_{t+1}^B)$←lstmEqns($v_x^B, s_{t+1}^B, c_t^B$)
                $h_{t+1}[i_s\colon i_e] \leftarrow h_{t+1}^B, c_{t+1}[i_s\colon i_e] \leftarrow c_{t+1}^B$
            end if
            $h_{t+1}^B \leftarrow h_{t+1}[m \times B\colon (m+1) \times B - 1]$
            $s_{t+2}^B \leftarrow s_{t+2}^B$+MxV($\mathbf{R}^B, h_{t+1}^B, 4B, B$)
        end for
        $s_{t+2} \leftarrow$ updateVect($s_{t+2}, s_{t+2}^B, r$)
    end for
    return ($s_{t+2}, h_{t+1}, c_{t+1}$)
end procedure
procedure UpDiagReuse($R, q_t, h_t, c_t, s_{t+1}$)
    for $r \leftarrow (\lceil \frac{N}{B} \rceil - 1)$ downto $0$ do
        $(i_s, i_e) \leftarrow (r \cdot B, (r+1) \cdot B - 1), s_{t+2}^B \leftarrow \vec{0}$
        for $m \leftarrow (\lceil \frac{N}{B} \rceil) - 1$ downto $r+1$ do
            $\mathbf{R}^B$←getDDRBlks($R, r, m, B$)
            $(h_t^B, c_t^B, q_t^B) \leftarrow$ GetSlice($h_t, c_t, q_t, m$)
            $s_{t+1}^B \leftarrow$ GetSlices($s_{t+1}, m$)
            $s_{t+1}^B \leftarrow s_{t+1}^B$+MxV($\mathbf{R}^B, h_t^B, 4B, B$)
            $h_{t+1}^B \leftarrow h_{t+1}[i_s\colon i_e]$
            $s_{t+2}^B \leftarrow s_{t+2}^B$+MxV($\mathbf{R}^B, h_{t+1}^B, 4B, B$);
        end for
        $(h_{t+1}^B, c_{t+1}^B)$←lstmEqns($v_x^B, s_{t+1}^B, c_t^B$)
        $h_{t+1}[i_s\colon i_e] \leftarrow h_{t+1}^B, c_{t+1}[i_s\colon i_e] \leftarrow c_{t+1}^B$
        $s_{t+2} \leftarrow$ updateVect($s_{t+2}, s_{t+2}^B, r$)
    end for
    return ($s_{t+2}, h_{t+1}, c_{t+1}$)
end procedure
```

Both the procedures reuse the blocks of R to reduce the off-chip memory accesses. Algorithm 2 implements the LSTM equations using the partial sum vector.

IV. Experimental Setup And Results

We have implemented LSTM layers using conventional and proposed approaches and synthesized the design using the

Algorithm 2 LSTM Equations

```
procedure lstmEqns($q_{t+1}, s_{t+1}, c_t$)
    $(v_i, v_f, v_g, v_o) \leftarrow ExtractVecs(q_t, B)$
    $(s_i, s_f, s_g, s_o) \leftarrow ExtractVecs(s_{t+1}, B)$
    for $n \leftarrow 0$ to $B-1$ do
        $i[n] \leftarrow$ sigmoid($s_i[n] + v_i[n]$)
        $f[n] \leftarrow$ sigmoid($s_f[n] + v_f[n]$)
        $g[n] \leftarrow$ tanh($s_g[n] + v_g[n]$)
        $o[n] \leftarrow$ sigmoid($s_o[n] + v_o[n]$)
        $c_{t+1}[n] \leftarrow f[n] \otimes c_t[n] + i[n] \otimes g[n]$
        $h_{t+1}[n] \leftarrow o[n] \otimes$ tanh($c$)[$n$]
        return($h_{t+1}, c_{t+1}$)
    end for
end procedure
```

SDSoC framework, SDx v2018.3. The design can be configured for different input vector lengths, on-chip buffer sizes, and the number of hidden units. We carried out the experiments on Zedboard, and the target frequency is 100MHz. The off-chip memory is DDR3 connected using a 64-bit AXI bus. We have integrated the Xilinx AXI Performance Monitor (APM) IP to log the number of bytes transferred and memory access latencies for DRAM accesses.

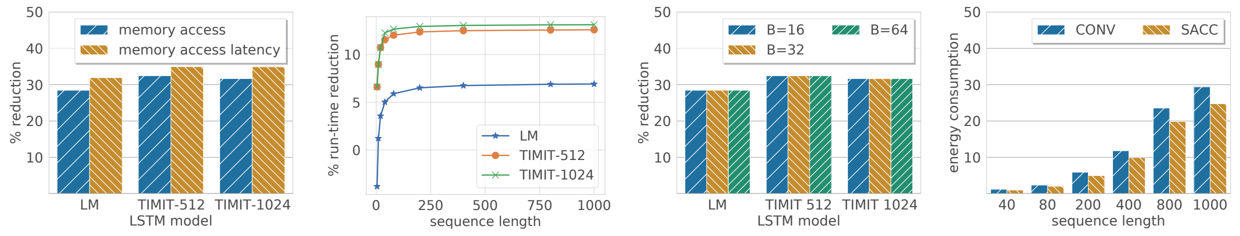
We have compared our approach with conventional approaches that access the hidden state weight matrices R at each step from the off-chip memory. We have used the same on-chip buffer size to store the weight matrices ($4 \times B^2$) to perform a fair comparison. The proposed approach requires additional on-chip memory ($4N+4B$) to store the four partial sum and temporary vectors. We have also compared the off-chip memory accesses with the TSI-WR approach [8]. We have experimented with LSTM models used in speech recognition (for TIMIT [9]) and character level Language Modelling (LM) [10]. The LSTM models are adopted from [6], [7], [11]. Each LSTM model has 2 LSTM layers with 128, 512 and 1024 hidden units in each layer for LM, TIMIT-512, and TIMIT-1024 models, respectively.

A. Results

1) Memory Accesses: Fig. 4a shows the proposed approach's reduction in off-chip memory accesses compared to conventional approaches. The proposed approach reduces the memory accesses of the R matrices by half. Total memory accesses, including R, W, and b, for LM and TIMIT models reduce by 28% and 32%, and memory access latencies reduce by 29% and 31%, respectively.

2) Run-time improvement: Fig. 4b shows the reduction in run-time achieved by the proposed approach while using comparable computing resources. We observe small execution overhead for a short input sequence of length 4 for LM. The proposed approach reduces the run-time by 12.8% and 16% for LM and TIMIT models.

3) On-chip memory size: Fig. 4c shows the reduction in off-chip memory access for different on-chip buffer sizes. The on-chip buffer sizes used for storing the weights are $4 \times B^2 \times d_w$, where d_w is the data bit width. The proposed approach reduces



(a) (b) (c) (d)

Fig. 4. (a) off-chip memory access and latency (b) run-time (c) memory access for different on-chip buffer sizes (d) normalized energy. CONV: conventional

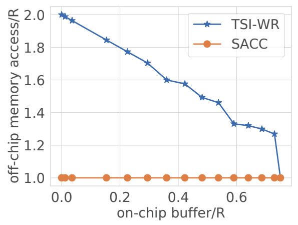


Fig. 5. Off-chip memory access comparison between TSI-WR and SACC approach for two consecutive time steps.

the off-chip memory accesses by 28% and 32% for LM and TIMIT models for different on-chip buffer sizes.

4) Comparison with TSI-WR: Fig. 5 compares the off-chip memory accesses of the SACC and the TSI-WR approaches using the simulation results. The performance of the TSI-WR approach depends on on-chip buffer sizes. TSI-WR reduces 50% off-chip memory accesses when the on-chip buffer size is 70% of the R matrix. The proposed approach reduces R's memory access by 50%, irrespective of the on-chip buffer size.

5) Energy Efficiency: We computed the energy consumption using the design power reported by the Vivado synthesis tool, execution time, and off-chip memory accesses. Fig. 4d shows the normalized energy consumption of the conventional and the proposed approach for TIMIT-1024. Due to the reduction in the run-time and off-chip memory accesses, the SACC approach reduces the energy consumption on average by 13% and 16% for LM and TIMIT models.

V. Previous Work

Several ASIC [11] and FPGA based accelerators [2]–[4], [6] focused on improving the energy efficiency of LSTM accelerators by reducing off-chip memory accesses. Some approaches [2], [3] used on-chip memory to store all the weights. However, these approaches are not scalable. Approaches [2], [5], [6] used the quantization and pruning techniques to compress the models' size. The proposed approach is orthogonal to the quantization techniques and can be integrated with different quantization techniques. Han et al. [6] used pruning to compress the model, which results in irregular network structure, and the sparse matrix causes unbalanced load distribution.

The other line of works used the data-reuse techniques to reduce the off-chip memory accesses. Que et al. [12] proposed a blocking-batching scheme to reuse the weights of W on a group of input vectors. However, the benefit of their scheme is limited to W matrix only. Park et al. [8] proposed an approach that reuses the R matrix's weights for two adjacent time steps computations. However, their approach partially reuses the weights of R, and reuse efficiency depends on the on-chip buffers sizes. The proposed approach reuses all the weights of R and is effective for small on-chip memory buffers.

VI. Conclusion

This paper proposes a novel data reuse approach (SACC) to reduce the off-chip memory accesses of LSTM accelerators. The data reuse in SACC does not depend on the on-chip buffer sizes, making it suitable for systems with small on-chip memory. Compared to conventional methods, the SACC approach reduces the memory access of LSTM models by 28−32% and improves the energy consumption by 13−16%.

References

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

[2] J. C. Ferreira and J. Fonseca, "An FPGA implementation of a long short-term memory neural network," in *2016 Int. Conf. on ReConFigurable Comput. and FPGAs (ReConFig)*, 2016, pp. 1–8.

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

[4] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *ASP-DAC*, 2017, pp. 629–634.

[5] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proc. of the ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 11–20.

[6] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proc. of the ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, pp. 75–84.

[7] J. Park, J. Kung, W. Yi, and J.-J. Kim, "Maximizing system performance by balancing computation loads in lstm accelerators," in *DATE*, 2018, pp. 7–12.

[8] N. Park, Y. Kim, D. Ahn, T. Kim, and J.-J. Kim, "Time-step interleaved weight reuse for LSTM neural network computing," in *Proc. of the ACM/IEEE Int. Symp. on Low Power Electron. and Des.*, 2020, pp. 13–18.

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

[10] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE/ACM Trans. on Audio, Speech, and Lang. Process.*, vol. 23, no. 3, pp. 517–529, 2015.

[11] E. Azari and S. Vrudhula, "ELSA: A Throughput-Optimized Design of an LSTM Accelerator for Energy-Constrained Devices," *ACM Trans. on Embedded Comput. Syst. (TECS)*, vol. 19, no. 1, pp. 1–21, 2020.

[12] Z. Que, T. Nugent, S. Liu, L. Tian, X. Niu, Y. Zhu, and W. Luk, "Efficient weight reuse for large LSTMs," in *ASAP*, vol. 2160, 2019, pp. 17–24.