A FRAMEWORK FOR MAPPING NEURAL NETWORKS ON PARALLEL ARCHITECTURES

A synopsis submitted in partial fulfillment of the requirements for the degree

of

Doctor of Philosophy

Submitted by:

SAURABH TEWARI

(2015CSZ8046)

under the guidance of

Prof. Anshul Kumar

Prof. Kolin Paul

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY DELHI

NEW DELHI

List of Included Papers

This thesis is based on the following publications:

Published:

1. **S. Tewari**, A. Kumar and K. Paul, "*SACC: Split and Combine Approach to Reduce the Off-chip Memory Accesses of LSTM Accelerators*", in DATE 2021.

2. **S. Tewari**, A. Kumar and K. Paul, "*Minimizing Off-Chip Memory Access for CNN Accelerators*", in IEEE Consumer Electronics Magazine 2021.

3. **S. Tewari**, A. Kumar and K. Paul, "*Bus Width Aware Off-Chip Memory Access Minimization for CNN Accelerators*", in ISVLSI 2020.

4. D. Stathis, Y. Yang, **S. Tewari**, A. Hemani, K. Paul, M. Grabherr and R. Ahmad, "*Approximate Computing Applied to Bacterial Genome Identification using Self-Organizing Maps*", in ISVLSI 2019.

Contents

1 Introduction

1.1 Why Off-Chip Memory Access Optimization

1.1.1 Impact on performance

DNN accelerators have large number of processing elements (PEs) to exploit the parallelism of DNNs. While these PEs can perform several operations per cycle, their performance is limited by off-chip memory bandwidth. In the near forseeable future, off-chip memory accesses limits the system performance [20]. Fig. 1 shows, two kernels with different compute intensity (FLOPS/byte), possibly due to different data reuse techniques. Kernel 2 can perform more computations per byte, compared to Kernel 1. Performance of Kernel 2 is limited by computational roof and it utilizes the compute resources fully (compute bound), while Kernel 1's performance is limited by memory bandwidth (memory bound). Memory bound kernels results

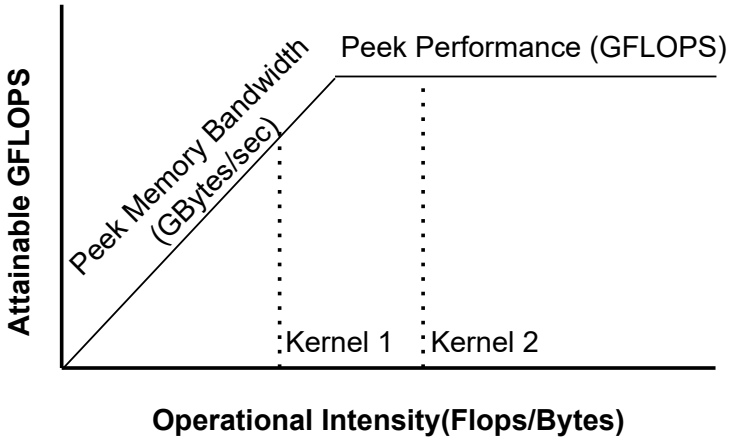


Figure 1: roofline model

in under-utilization of compute resources, which results in degraded performance and energy consumption. To improve the DNN accelerators' performance, maximizing the data reuse from on-chip memories is essential.

1.1.2 Impact on Energy Consumption

The energy of DNN accelerators is the sum of computations and data accesses energy.

$$E = E_{comp} + E_{data} \tag{1}$$

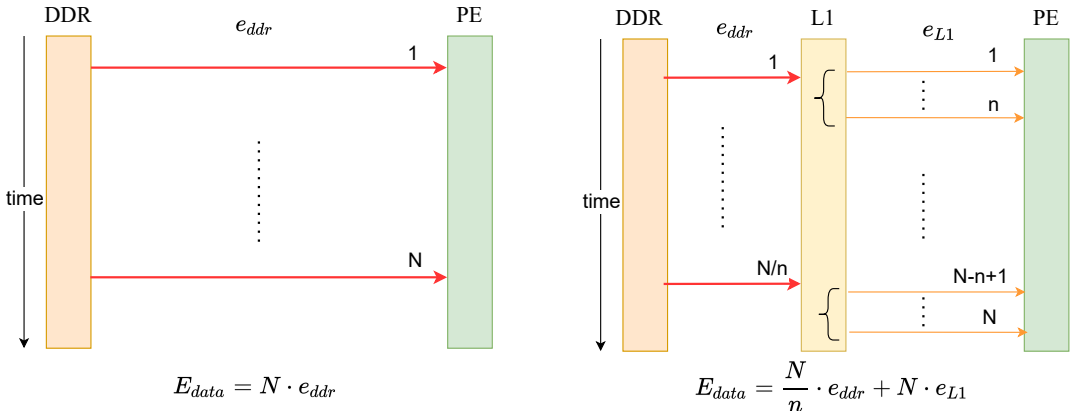
E_{comp} for a DNN layer depends on the number of operations in the layer, which is fixed as it depends on layer shape e.g. number of activations, filters, and filter sizes. However E_{data} can be optimized using data reuse techniques. Operations of CNN involve three kinds of data and E_{data} is the sum of energies for accessing all three data types.

$$E_{data} = E_{ifm} + E_{ofm} + E_{wts} \tag{2}$$

$$E_i = V_i * R_i * DW \tag{3}$$

where $i \in \{ifm, ofm, wts\}$ and DW is the data width. A typical DNN accelerator system has multiple levels of memory hierarchy. Each level of memory has different size, energy and access time. The off-chip memory (DDR) has the largest size, and highest energy and access cost compared to other levels of memories in hierarchy.

The off-chip memory access energy is upto two orders higher compared to on-chip memory access energy. Figure 2 illustrates the impact of data reuse from on-chip memories. The energy efficiency of an accelerator



(a) no reuse

(b) reuse ($\times n$) from on-chip memory L1

Figure 2: data access energy E_{data} with and without reuse.

system with 2 levels of memory (DDR and L1) can be computed as below,

$$\begin{aligned} \text{energy-efficiency} &= (1 - \frac{E^2_{data}}{E^1_{data}}) \times 100 \\ &= (1 - (\frac{1}{n} + \frac{e_{L1}}{e_{ddr}})) \times 100 \end{aligned}$$

As, $e_{L1} << e_{DDR}$, the ratio $\frac{e_{L1}}{e_{ddr}}$ is very small and the energy efficiency depends on number of reuses (n) from on-chip memory $L1$. It improves considerably with increasing the number of data reuse (n).

In this work, we focus on the key issue to improve the performance and energy efficiency of DNN accelerators by reducing the off-chip memory accesses and maximizing the data reuse from on-chip memories.

2 About the Framework

It is arduous to create a generic framework for large variety of DNNs and accelerator architectures. However, most of the modern DNNs and accelerator's architectures share similar architectural and algorithmic features. Therefore, we propose a mapping framework that addresses the most challenging problems faced by modern DNN accelerators and that are applicable to most of these DNNs accelerators.

1. Inferencing or usage of DNNs are frequently performed on edge devices. These edge devices are resource constraints and usually battery operated. They have tight energy budget which motivates to optimize the energy consumption of these DNN on edge devices. Secondly, the short response time is also expected to improve the user experience.

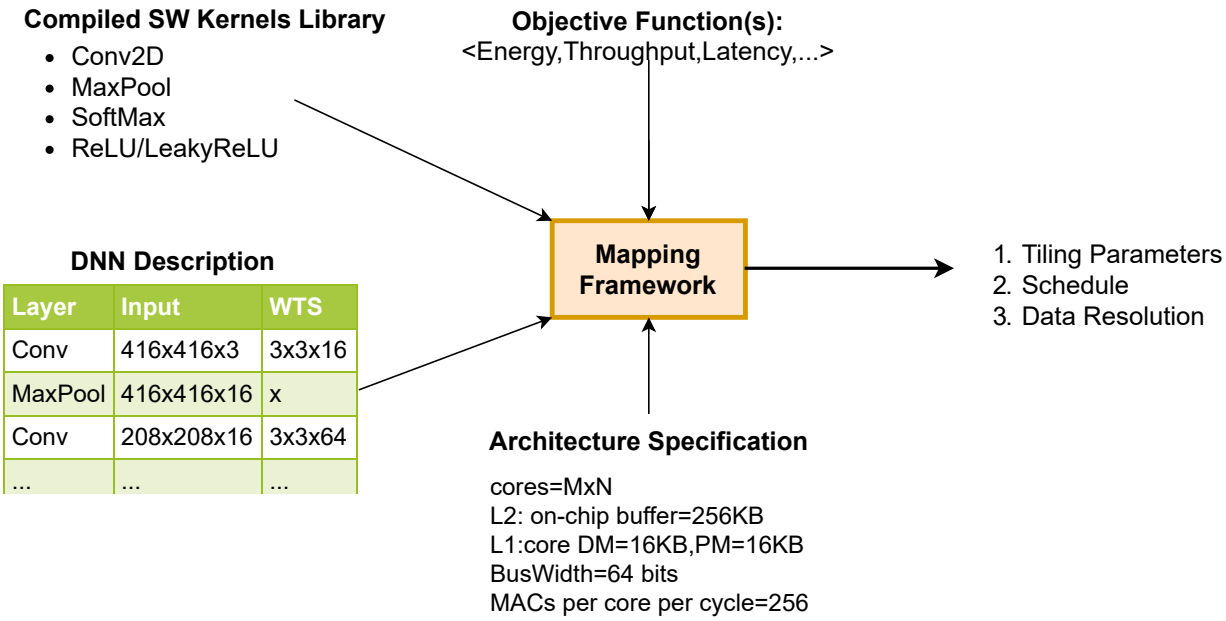
2. To optimize the performance of DNN based applications, DNN accelerators are commonly used. These accelerators have several compute units to perform large number of computations in parallel to improve the performance and user response time. However energy efficiency of these accelerators is far from desired, which restricts their usage in edge devices.

3. DNNs have large amount of parallelism, which can be exploited to speed up the processing, however limited memory bandwidth is the bottleneck.

4. It is observed, that more than 80% of overall energy consumption is due to off-chip memory accesses. Hence optimizing the off-chip memory accesses is the key to improve the energy efficiency of these DNN accelerators.

Our main contributions in this thesis dessertations are

1. We identify the key issue to be addressed for DNN accelerator's wide usage in energy constraint devices is their off-chip memory accesses. We propose a mapping framework which takes input DNNs



Use Cases
Optimizing memory accesses of CNN to improve energy efficiency on CNN Accelerators
Optimizing throughput and energy efficiency of LSTM/RNN
Data Resolution for mapping SOM on DRRA/FPGA
Mapping Homography CNN on CGRA/DRRA

Figure 3: Mapping DNN on coarse grain parallel architectures

shape and sizes and analyzes the performance and energy consumption of DNN accelerators while considering accelerator's architectural parameters.

2. We propose a model that precisely estimates the off-chip memory accesses and the energy consumption of the DNN's layers, while taking into account the accelerator's architectural parameters.

3. We propose an approach that determines the optimal partitoning of the DNN layers for reducing the off-chip memory accesses and energy consumptions. We analytically express the off-chip memory accesses of DNNs using the layer shapes and tiling parameters and express it as a constraint optimization problem.

4. We propose a novel data reuse approach to improve the run time, energy efficiency of recurrent neural networks (RNN). The proposed approach schedules the computations in a way that reduces the memory accesses of large matrices by half and optimizes the performance and energy efficiency of RNN accelerators.

5. We analyzed the impact of data bit width on the performance, energy consumption and the accuracy of neural networks. We experimented with different data bit widths to analyze the trade-off between accuracy and energy efficiency of the NNs on reconfigurable parallel architectures.

2.1 Related Work

We compare our approaches and show that our work significantly improves the performance and energy efficiency of DNN accelerators compared to the state of the art. To address the computational and energy

efficiency of DNNs, several ASIC [2, 6, 19] and FPGA based accelerators [3, 7–9, 11] are proposed. The energy efficiency of DNN accelerators is critical for their widespread usage, and off-chip memory access is the key to improving energy consumption. Most of these works focused on improving energy efficiency by reducing off-chip memory accesses. Figure 4 shows broad categories of state of the art approaches that aim to improve energy efficiency of DNN accelerators by reducing the off-chip memory accesses. Some

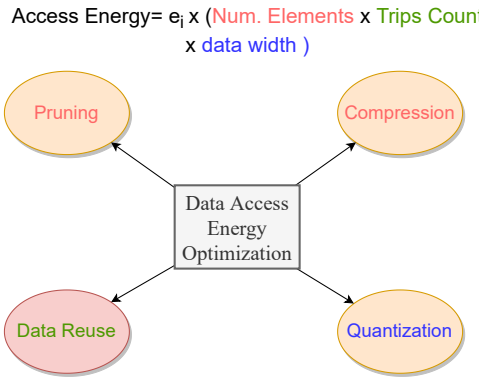


Figure 4: Energy efficiency optimization approaches

approaches [7, 11, 17] used on-chip memory to store all the weights. Sizes of weights in recent multi-layer LSTM models can be several MB's, and using large on-chip memory is expensive. These approaches are not scalable and effective only for small LSTM models. The proposed approach is independent of model size and effective for large LSTM models.

Several approaches used the fact that neural networks are error-tolerant and have lots of redundancy. They used the quantization and pruning techniques to compress the models' size. Approaches [7, 18] used 18-bit, Chang et al. [3] used 16-bit, Han et al. [9] used 12-bits precision for storing the inputs and weights, Lee et al. [11] used 8-bit inputs and 6-bits for weights to reduce the model size. The proposed approach is orthogonal to the quantization techniques and can be integrated with different quantization techniques to reduce the memory accesses further.

Han et al. [9] used pruning to compress the model. However, pruning results in irregular network structure, and the sparse matrix require additional computational and storage resources and causes unbalanced load distribution. To overcome this Wang et al. [18] used block-circulant matrices representations to compress the LSTM/RNN model and to eliminate irregularities resulted from compression. Some approaches ([9,13,14]) used load balance aware pruning techniques to overcome the unbalanced load distribution problem. Quantization and pruning approaches impacts the accuracy of the networks and may not be suitable where accuracy is at priority.

The other line of works reduced the memory accesses without compromising the accuracy of the network by applying the data-reuse techniques. Que et al. [16] proposed a blocking-batching scheme to reuse the LSTM weights fetched from external memory. Their approach reuses the weights of W matrix on a group of input vectors, by processing those input vectors as a batch. The input vectors in the same batch share the same weight matrices (W). However, it is difficult to collect required number of input vectors. As the LSTM cell states computations depend on cell states of the previous time-step, benefit of their batching schemes is limited to W matrix. Reusing weights of R across different time-steps has not been successful because of the state dependency.

Park et al.([15]) proposed a time step interleaved weight reuse scheme (TSI-WR) which reuses the weights of R matrix between two adjacent time steps by performing computations in a time-interleaved manner. Their approach logically partitions the R matrix into blocks. A block is accessed from off-chip memory to compute the two consecutive hidden state vectors partially. However, their approach do not fully exploit the data reuse, and several weights are accessed repeatedly from the off-chip memory. In addition, the data reuse in TSI-WR approach depends on the on-chip storage size which limits the benefits of their approach.

3 The Mapping Framework

The mapping framework optimizes the performance of feed-forward and recurrent neural networks. Each of these networks has specific challenges that are addressed differently.

3.1 Off-chip Memory Access Estimation Model

DNNs have large amount of parallel operations. However due to large volume of data required for performing the parallel operations, the DNN acclerators' performance are limited by memory bandwidth. In addition, the DNN accelerators have limited on-chip memory, which constrains the data size that can be stored in on-chip memory. The off-chip memory accesses of a layer depends significantly on the tile dimensions [12, 21]. Determining optimal tile dimensions requires analyzing off-chip memory accesses for different tile dimensions. Memory accesses also depend on the data bit width and architectural parameters of the accelerator like bus width.

The CNN accelerators use a wide data bus to access off-chip memory to meet the high memory bandwidth requirement [4,5]. If the number of bytes accessed from an off-chip memory address is not a multiple of bus width or the address is not aligned to the word boundary, it results in unused bytes lanes of the data bus. Figure 5 shows examples of memory accesses on a 64-bit data bus. Fig. 5a shows a read transaction of 8 bytes from an aligned address and uses the full bus width. However, if only 5 bytes are read from an aligned address, as shown in Figure 5b, 8 bytes are still accessed. If 5 bytes are read from an unaligned address, it results in 16 bytes of data access, as shown in Fig. 5d. The unused byte lanes do not carry any useful data, but they contribute to overall energy consumption. The length of the data read should be chosen such that bus utilization is high, and off-chip memory accesses and energy consumption are minimized. To estimate the performance and energy efficiency, we have developed an offline model that estimates the

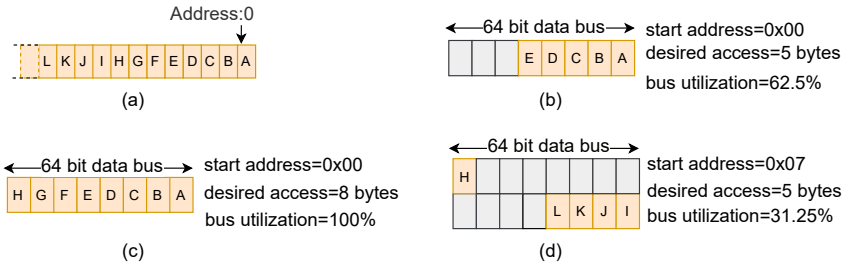


Figure 5: Off-chip memory accesses on 64-bit wide data bus

off-chip memory accesses while taking into account the data alignment and memory bus width.

3.2 Optimal data partitioning for CNNs

Due to limited on-chip memory and large layer data sizes, the CNN accelerators apply loop tiling to partition the layer data. Loop tiling is a compiler technique [1] that partitions the loop iteration space and large arrays into smaller tiles to increase the data locality and ensures that data fits into smaller memories.

Fig. 6 shows a CL data stored in off-chip memory and its tiles in the accelerator's on-chip buffer. For a given data reuse scheme, all the tiles of a 3D data make the same number of trips from off-chip memory. The off-chip memory access of 3D data can be computed as following

$$\mathbb{B}_{3D}{=}r{\times}\sum_{t=1}^{N_{tiles}}\mathbb{B}_t \tag{4}$$

where N_{tiles} is the number of tiles. r and $\mathbb{B}_t$ are the trips count and the number of bytes accessed from off-chip memory of the t^{th} tile, respectively. The trips count r depends on the data reuse scheme, and $\mathbb{B}_t$ is computed using the model described in section 3.1

Tiles of *ifm, ofm* and *wts* reside in on-chip memory.If the on-chip memory buffer size is $buffSize$, and V_{ifm}, V_{ofm}, and V_{wts} are the size of *ifm, ofm*, and *wts* tiles, respectively, then constraints on tile dimensions

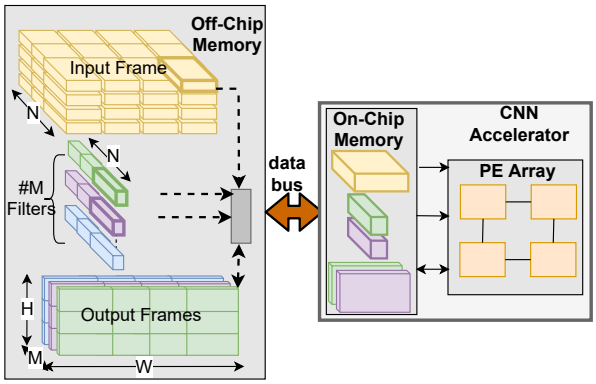


Figure 6: CNN layer tiles in off-chip and on-chip memory

are

$$(V_{ifm}+V_{wts}+V_{ofm}) \leq buffSize \tag{5}$$

Determining the tile dimensions which minimize the off-chip memory accesses, is a constraint optimization problem. The number of bytes accessed from off-chip memory and the constraints (5) are non-linear functions of four variables and thus solving it is non-trivial.

The optimal tile dimensions can be determined by computing the number of bytes accessed from off-chip memory ($\mathbb{B}$) at all the feasible points in the solution space using section 3.1. Our approach determines the optimal tile dimensions for each layer for different data reuse schemes and it can be configured for different on-chip memory sizes, bus widths, and data bit width.

4 Mapping Recurrent Neural Network

4.1 Introduction

Many applications involve sequential data processing and time-series predictions, e.g., natural language processing, speech recognition, video activity recognition, sentiment classification. Processing sequential data requires remembering the contextual information from previous data. Recurrent neural networks (RNNs) are deep learning algorithms specialized in handling such problems by maintaining an internal state based on previously seen data. LSTMs [10] are variants of RNNs designed to handle long-range dependencies by storing useful information about previous inputs for a long duration.

Customized accelerators are proposed to speed up the computations of LSTM. These accelerators have limited on-chip memory, specifically the accelerators targeted for embedded devices. LSTM computations involve multiple matrix-vector multiplications, and these matrix-vector multiplications are performed several times. The size of these matrices can be significant in several MB's and often exceed the size of the accelerator's on-chip memory. These matrices are partitioned into blocks and accessed from off-chip memory repeatedly by the accelerator, which results in a large volume of off-chip memory accesses and energy consumption. The high energy consumption limits the usage of these accelerators on embedded devices.

Typically the computations of LSTM cell is described by the following equations

$$\begin{aligned}
i &= \sigma(W^i \cdot x_t + R^i \cdot h_{t-1} + b^i) \\
f &= \sigma(W^f \cdot x_t + R^f \cdot h_{t-1} + b^f) \\
g &= \tanh(W^g \cdot x_t + R^g \cdot h_{t-1} + b^g) \\
o &= \sigma(W^o \cdot x_t + R^o \cdot h_{t-1} + b^o) \\
c_t &= f \odot c_{t-1} + i \odot g \\
h_t &= o \odot \tanh(c_t)
\end{aligned} \tag{6}$$

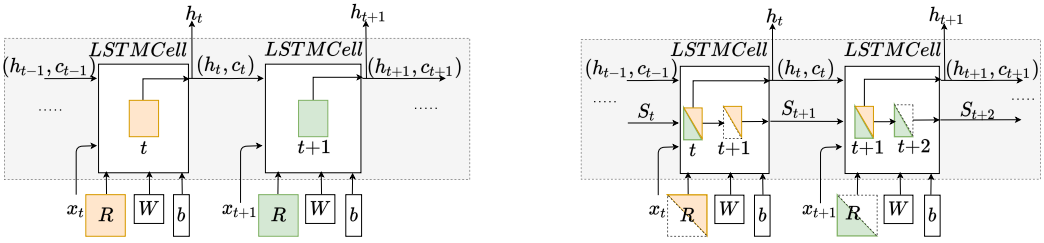
where x_t is the input, h_t is the hidden state, and c_t is the cell state at time t. i, f, g, o are the computed gate values. $\odot$ denotes the element-wise multiplications. W^j and R^j are the input and hidden state weight

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 input, and cell state (c_t) and hidden state (h_t) are computed using (6). The dependency of h_t on h_{t-1} and c_{t-1} prevents the parallel processing of multiple time steps and limits the data reuse.

Figure 7 shows the LSTM cell computations for two consecutive time steps using conventional and the proposed approach. To compute the hidden state vector h_t, conventional approaches accesses R matrix at each time step t, as shown in Fig. 7a. Accessing the weights at each time step results in a large volume of off-chip memory accesses.

Que et al. [16] proposed a blocking-batching scheme to reuse the LSTM weights fetched from external memory. Their approach reuses the weights of W matrix on a group of input vectors, by processing those input vectors as a batch. The input vectors in the same batch share the same weight matrices (W). However, it is difficult to collect required number of input vectors. As the LSTM cell states computations depend on cell states of the previous time-step, benefit of their batching schemes is limited to W matrix. Reusing weights of R across different time-steps has not been successful because of the state dependency.

Park et al.([15]) proposed a time step interleaved weight reuse scheme (TSI-WR) which reuses the weights of R matrix between two adjacent time steps by performing computations in a time-interleaved manner. Their approach logically partitions the R matrix into blocks. A block is accessed from off-chip memory to compute the two consecutive hidden state vectors partially. However, their approach do not fully exploit the data reuse, and several weights are accessed repeatedly from the off-chip memory. In addition, the data reuse in TSI-WR approach depends on the on-chip storage size which limits the benefits of their approach.



(a) Conventional approaches: R matrix is accessed at each time step.

(b) Proposed approach: R matrix accesses are reduced by half.

Figure 7: LSTM cell computations for consecutive time-steps showing the weight accesses.

4.2 Proposed Approach

We proposes an approach that reduces off-chip memory accesses by splitting the computation in two. At each time step, while the computations of a hidden state vector of one time step h_t completes, partial computation of next time step (S_{t+1}) is also performed by reusing the weights. Our approach schedules the computations in a way that reuses all the weights of R between two adjacent time steps. The data reuse in our approach is independent of on-chip buffer sizes which makes it suitable for accelerators with very small on-chip memory.

The proposed data reuse approach splits and combines the LSTM cell computations in a way that reduces the off-chip memory accesses of hidden state matrices by 50%. The computation of the h_t can be expressed as shown below

$$h_t[k] = F(S_t[k] + q_t[k]) \tag{7}$$

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,

$$S_t[k] = \sum_{n=0}^{N-1} R[k][n] \cdot h_{t-1}[n] \tag{8}$$

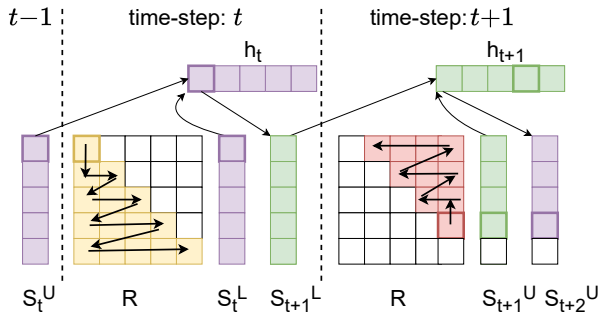


Figure 8: Splitting the hidden state vector computations into partial sums

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

$$S_t^L[k] = \sum_{n=0}^{k} R[k][n] \cdot h_{t-1}[n] \tag{9}$$

$$S_t^U[k] = \sum_{n=k+1}^{N-1} R[k][n] \cdot h_{t-1}[n] \tag{10}$$

Equation (9) uses the lower-diagonal and diagonal elements of R (R^L), and (10) uses the upper diagonal elements of R (R^U). As shown in Figure 8, 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 Figure 8, the proposed approach accesses the weight matrix R once, to compute h_t and h_{t+1}.

References

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. Compilers: Principles, techniques, and tools, 2006.

[2] E. Azari and S. Vrudhula. Elsa: A throughput-optimized design of an lstm accelerator for energy-constrained devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(1):1–21, 2020.

[3] A. X. M. Chang, B. Martini, and E. Culurciello. Recurrent neural networks hardware implementation on fpga. *arXiv preprint arXiv:1511.05552*, 2015.

[4] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ASPLOS*, 2014.

[5] Y.-H. Chen, J. S. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM/IEEE ISCA*, 2016.

[6] F. Conti, L. Cavigelli, G. Paulin, I. Susmelj, and L. Benini. Chipmunk: A systolically scalable 0.9 mm 2, 3.08 gop/s/mw@ 1.2 mw accelerator for near-sensor recurrent neural network inference. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2018.

[7] J. C. Ferreira and J. Fonseca. An fpga implementation of a long short-term memory neural network. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8. IEEE, 2016.

[8] Y. Guan, Z. Yuan, G. Sun, and J. Cong. Fpga-based accelerator for long short-term memory recurrent neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 629–634. IEEE, 2017.

[9] 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 *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84, 2017.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung. Fpga-based low-power speech recognition with recurrent neural networks. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 230–235. IEEE, 2016.

[12] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. *DATE*, 2018.

[13] J. Park, J. Kung, W. Yi, and J.-J. Kim. Maximizing system performance by balancing computation loads in lstm accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 7–12. IEEE, 2018.

[14] J. Park, W. Yi, D. Ahn, J. Kung, and J.-J. Kim. Balancing computation loads and optimizing input vector loading in lstm accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(9):1889–1901, 2019.

[15] N. Park, Y. Kim, D. Ahn, T. Kim, and J.-J. Kim. Time-step interleaved weight reuse for lstm neural network computing. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 13–18, 2020.

[16] Z. Que, T. Nugent, S. Liu, L. Tian, X. Niu, Y. Zhu, and W. Luk. Efficient weight reuse for large lstms. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, volume 2160, pages 17–24. IEEE, 2019.

[17] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott. Finn-l: Library extensions and design trade-off analysis for variable precision lstm networks on fpgas. In *2018 28th international conference on field programmable logic and applications (FPL)*, pages 89–897. IEEE, 2018.

[18] 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 *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 11–20, 2018.

[19] Z. Wang, J. Lin, and Z. Wang. Accelerating recurrent neural networks: A memory-efficient approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2763–2775, 2017.

[20] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *FPGA*, 2015.