OPTIMIZING NEURAL NETWORKS PERFORMANCE 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:

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.

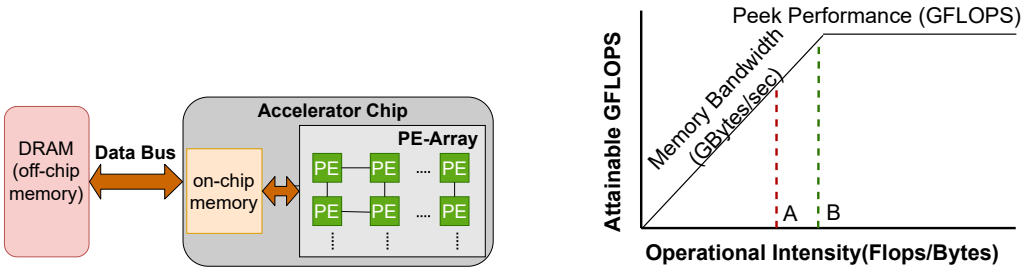
4. D. Stathis, Y. Yang, **S. Tewari**, A. Hemani, K. Paul, M. Grabherr and R. Ahmad, "*Approximate Computing Applied to Bacterial Genome Identification using Self-Organizing Maps*", in ISVLSI 2019.

Contents

1 Introduction

Last few years have seen rapid growth in Deep Neural Network (DNN) based applications due to their high accuracy. To achieve high accuracy modern DNNs perform compute and memory intensive operations. To improve the user experience and eliminate network bandwidth issues manufacturers are shifting the processing of DNNs from cloud to edge devices like smart-phones, tablets etc. These battery operated edge devices have limited resources and tight energy budget which poses additional challenges in DNNs processing on edge devices.

Energy efficiency and throughput are the two most important metrices for edge devices. High throughput is desired for better user-response time, energy-efficiency is of paramount importance for battery operated devices. To meet these energy and throughput demands, edge devices mostly use customized DNN accelerators. Several FPGA [2, 15, 17, 36, 38, 40], GPU [9] and ASIC [6–8, 11] accelerators are proposed to meet the performance and energy targets. Figure 1a shows a typical DNN accelerator architecture, which



(a) (b)

Figure 1: (a) Typical DNN accelerator architecture. (b) Roofline model

consists of an off-chip memory and an accelerator chip. Accelerator chip mainly consists of an on-chip memory of few KBs and an array of Processing Elements (PEs). The accelerator system has multiple memory levels: off-chip memory, on-chip memory, and the registers inside the PE. Each memory level has different access latency and energy cost. The memory access energy from off-chip memory is upto two orders of magnitude higher than a PE computation operation [8]. It has been observed that more than 80% of the overall energy consumption of these accelerators is due to off-chip memory accesses [6].

The PE-array (Figure 1a) has large number of processing elements, capable of performing several operations per cycle. However, throughput is often limited by off-chip memory bandwidth [37]. Fig. 1b shows, two kernels (A and B) with different compute intensity (FLOPS/byte). Performance of kernels are limited by the memory bandwidth, which is typically the case for most of the DNN accelerators. Kernel B, applies data-reuse techniques to improve the computational intensity compared to kernel A and achieves better througput. Better data-reuse techniques are required to improve the performance on bandwidth-limited parallel architectures.

Therefore, reducing the off-chip memory is the key to improve the throuhput and energy efficiency of DNN accelerators. There are different types of neural networks. Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) are the most popular ones. Recurrent Neural Networks (RNNs) are widely used in sequential data processing applications such as speech recognition and natural language processing, while Convolutional Neural Networks (CNNs) are used in the area of computer vision e.g., image recognition. Each type of NN has its own challenges and require specific techniques to improve the performance. Towards this end we propose approaches to improve the performance of DNN accelerators for three popular NNs. Our main contributions in this thesis dessertations are

1. We developed an offline-tool that computes the off-chip memory accesses of CNN layers, while considering the accelerator's architectural parameters. The offline-tool explores the large search space of tile dimensions and helpful in analysing the off-chip memory accesses for different data reuse schemes.

2. We propose an approach that determines the optimal partitoning and scheduling scheme of the convolution (CL) and fully connected (FC) layers to mimimize the off-chip memory access and improve the energy-efficiency of CNN accelerators. We analytically express the off-chip memory accesses of CNNs as a function of layer shapes, tiling parameters and the number of trips from the off-chip memory and express it as a constraint optimization problem. Using the above offline-tool we explore

the design space to find the optimal solution. We implemented CL and FC layers on FPGA to demonstrate the efficacy of our approach compared to state of the art approaches.

3. We propose a novel data reuse approach to improve the throughput and energy efficiency of recurrent neural networks (RNNs). The proposed approach partitions the data and schedules the operations in a way that reduces the off-chip memory accesses of large matrices significantly. We implemented Long-Short Term Memory Network (LSTMs) accelerators for different approaches on FPGA to measure the design power and memory accesses and show the energy and througput improvements achieved by our approach.

4. We analyse the effect of using different bit resolutions on the accuracy of a NN, as well as the benefits of using low bit width data resolution for self organizing maps (SOM) for designing a battery operated system where the area, power and performance are of critical importance. We implemented a SOM network on FPGA which can be configured for different bit resolution to compare the performace of different bit resolutions.

2 Related Work

To address the computational and energy efficiency of DNNs, several ASIC [4, 10, 35] and FPGA based accelerators [5, 13, 16, 19, 24] are proposed. As off-chip memory accesses are the bottleneck in improving the throughtput and energy-efficiency of DNN accelerators, most of the recent works focused on reducing off-chip memory accesses. Some approaches [13, 24, 31] used on-chip memory to store all the weights. Sizes of weights in recent NN models can be several MB's, and using large on-chip memory is expensive. These approaches are not scalable and effective only for small NN models.

There are two broad categories of work that aim to reduce the off-chip memory accesses of DNN accelerators as shown in Figure 2. One category of work exploits error-tolerence and redundancy in NNs

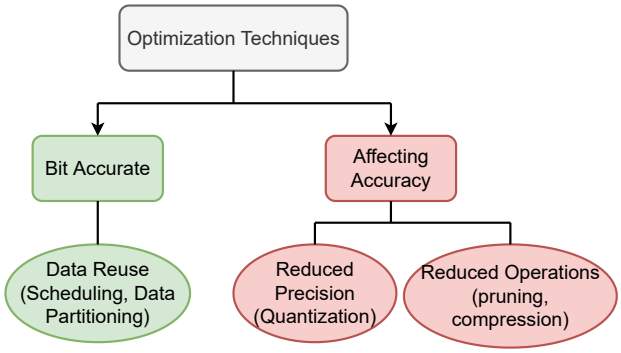


Figure 2: Broad Classification of previous works for improving the performance of DNN accelerators.

using quantization and pruning techniques to compress the models' size. Approaches [13, 34] used 18-bit, Chang et al. [5] used 16-bit, Han et al. [19] used 12-bits precision for storing the inputs and weights, Lee et al. [24] used 8-bit inputs and 6-bits for weights to reduce the model size. Han et al. [19] 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. [34] used block-circulant matrices representations to compress the LSTM/RNN model and to eliminate irregularities resulted from compression. Some approaches ([19, 27, 28]) used load balance aware pruning techniques to overcome the unbalanced load distribution problem. However, quantization and pruning approaches impacts the accuracy of the networks and may not be suitable where accuracy can not be compromised.

The other category of works applies the data-reuse techniques to reduce the memory accesses without effecting the accuracy of the network. Zhang et al. [40] and Li et al. [25] used loop tiling and scheduling techniques to optimize the off-chip memory accesses of CNNs. Que et al. [30] proposed a blocking-batching scheme to reuse the LSTM weights fetched from external memory. Park et al. [29] proposed a time step interleaved weight reuse scheme (TSI-WR).

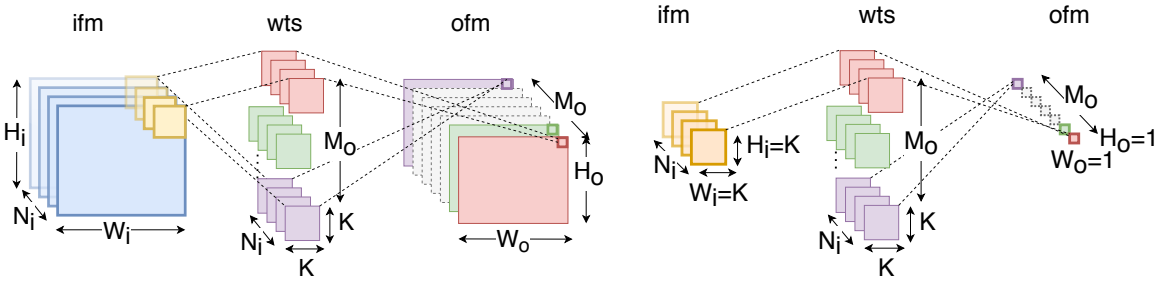
We proposed data reuse approaches for state of the art CNNs and RNNs that are bit-accurate. These data-reuse approaches are orthogonal to the quantization techniques and can be integrated with different quantization techniques to reduce the memory accesses further. We have also analysed the impact of different bit-resolution on the accuracy of Self Organizing Maps (SOM) and to observe the improvements in the power, performance and area.

3 Optimizing the Performance of CNN Accelerators

3.1 Introduction

Modern CNNs are very deep and have millions of parameters. To achieve high accuracy CNNs perform compute and memory intensive operations. CNN accelerators exploits the parallelism to speed up the computations, however limited on-chip memory restricts the data reuse from on-chip memory. This results in large number of off-chip memory accesses, which degrades the performance and energy efficiency of CNN acclerators.

CNNs have a sequence of mainly three types of layers: convolution layer (CL), pooling layer, and fully connected layer (FCL). There are several CLs, and a pooling layer usually follows each CL. The last few layers of the CNNs are FCLs. The computations of a CL and an FCL are illustrated in Figure 3a and 3b, respectively. Each CL and FCL layer takes 3D input frames (*ifm*) and applies filter weights (*wts*) to compute output frames (*ofm*).



(a) Convolution Layer (b) Fully Connected Layer

Figure 3: Convolution and fully connected layers

CNN accelerators apply loop tiling to partition the layer data into small tiles that fits into on-chip memory. 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. 4 shows a layer's data stored in off-chip memory and its tiles in the accelerator's on-chip buffer. The order

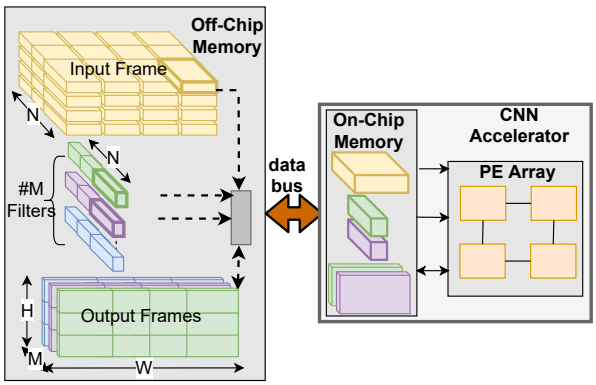


Figure 4: CNN layer tiles in off-chip and on-chip memory

of loops determines the scheduling of the tiles. The scheduling scheme, which minimizes the trips of the *ifm* tiles between the accelerator and off-chip memory, is referred to as the input-reuse-oriented scheme

(IRO). Similarly, the other two schemes, which minimize the trips of *ofm* and *wts* are referred to as output-reuse-oriented (ORO) and weight-reuse-oriented (WRO) scheme, respectively. The off-chip memory access depends on the data reuse scheme and the tile dimensions. The tile dimensions significantly impacts the off-chip memory accesses [25, 40].

CNN's layers have varying shapes. First few layers have large volume of *ifm* and ofm and last few CLs and FCLs have large volume of *wts*. The data reuse scheme optimal for one layer may be suboptimal for other layers due to varying layer shapes. In addition, memory accesses depend on the architectural parameters of the accelerator like bus width and data alignment. Our approach considers architectural parameters to compute the off-chip memory accesses and determines the optimal solution for each CL and FC layer.

3.2 Related Work

Zhang et al. [40] and Li et al. [25] used loop tiling to optimize the off-chip memory accesses. They expressed the off-chip memory access as a function of tile dimensions and layer shape. Zhang et al. [40] determined optimal tile dimensions by enumerating all the legal tile dimensions. To reduce the hardware design complexity, they determined a global optimal tile dimension and used a common data reuse scheme for all the layers. Li et al. [25] proposed a layer-wise adaptive data partitioning and scheduling scheme. However, their approach ignored the architectural parameters and address alignment, and assumed that all tiles of the same dimensions have the same off-chip memory accesses. With these assumptions, the tile dimensions determined by previous apporaches lead to a suboptimal solution.

3.3 Off-Chip Memory Accesses of CNN Layers

CNN accelerator access *ifm*, *ofm* and *wts* of the layers from off-chip memory using loop tiling. Tiles are accessed multiple times from off-chip memory. The trips count depend on the data reuse scheme, layer shape, and tile dimensions. Trip counts of IRO, ORO, and WRO schemes can be expressed as the rows of the matrix (1), where columns represent ifm, ofm, and weights.

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_{iro} \\ \mathbf{r}_{oro} \\ \mathbf{r}_{wro} \end{bmatrix} = \begin{bmatrix} 1 & (2\lceil \frac{N}{T_n} \rceil - 1) & \lceil \frac{H}{T_r} \rceil \lceil \frac{W}{T_c} \rceil \\ \lceil \frac{M}{T_m} \rceil & 1 & \lceil \frac{H}{T_r} \rceil \lceil \frac{W}{T_c} \rceil \\ \lceil \frac{M}{T_m} \rceil & (2\lceil \frac{N}{T_n} \rceil - 1) & 1 \end{bmatrix} \tag{1}$$

where $\langle T_{c_i}, T_{r_i}, T_{n_i} \rangle$, $\langle T_{c_o}, T_{r_o}, T_{m_o} \rangle$ and $\langle K, K, T_{n_i} \rangle$ are the tile dimensions, and $\langle W_i, H_i, N_i \rangle$, $\langle W_o, H_o, M_o \rangle$ and $\langle K, K, N_i \rangle$ are the data shape of *ifm*, *ofm* and *wts*, respectively. The number of bytes accessed from off-chip memory ($\mathbb{B}$) for a layer can be expressed as the following sum

$$\mathbb{B} = r_i \cdot \mathbb{B}_i + r_o \cdot \mathbb{B}_o + r_w \cdot \mathbb{B}_w \tag{2}$$

where $\mathbb{B}_i$, $\mathbb{B}_o$ and $\mathbb{B}_w$ are the number of bytes accessed in one trip and r_i, r_o and r_w are the trips count of *ifm, ofm* and *wts* for the data reuse scheme, respectively. We compute the $\mathbb{B}_i$, $\mathbb{B}_o$ and $\mathbb{B}_w$ for each layer considering the architectural parameters using equation 5.

3.3.1 Constraints

The CNN accelerator stores the tiles of *ifm*, *ofm* and *wts* in on-chip memory to perform the computations. Limited on-chip memory and the layer shape constraints the tile dimensions. Constraints are shown in (3) below

$$\begin{aligned} &(V_i + V_o + V_w) \leq buffSize \\ &0 < T_{c_o} \leq W_o, \quad 0 < T_{r_o} \leq H_o \\ &0 < T_{n_i} \leq N_i, \quad 0 < T_{m_o} \leq M_o \end{aligned} \tag{3}$$

where $buffSize$ is the on-chip memory size and V_i, V_o and V_w are the *ifm*, *ofm* and *wts* tile sizes computed as following

$$\begin{aligned} V_i &= T_{c_i} \cdot T_{r_i} \cdot T_{n_i} \cdot DW \\ V_o &= T_{c_o} \cdot T_{r_o} \cdot T_{m_o} \cdot DW \\ V_w &= K^2 \cdot T_{n_i} \cdot T_{m_o} \cdot DW \end{aligned} \tag{4}$$

where DW is the data width in bytes. Determining the optimal tile dimensions that minimizes $\mathbb{B}$ (2), is a constraint optimization problem. Equations (2) and (3) are non linear and involve four variables $T_{c_o}, T_{r_o}, T_{n_i}, T_{m_o}$. Thus it is not trivial to find the optimal solution.

3.4 Impact of architectural parameters on memory access

The CNN accelerators use a wide data bus to access off-chip memory to meet the high memory bandwidth requirement [6, 8]. 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 illustrates 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. The

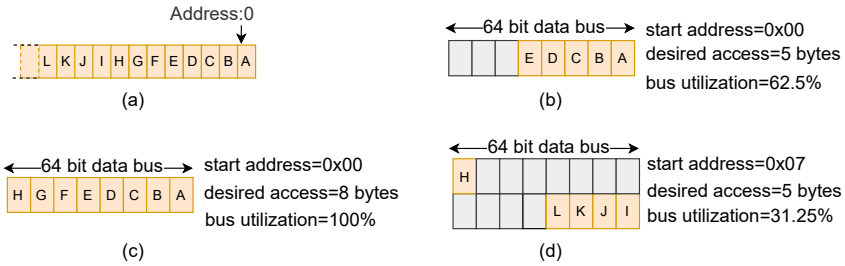


Figure 5: Off-chip memory accesses on 64-bit wide data bus

off-chip memory bus protocol supports burst based transactions where multiple data transfers of *transfer size* happen from a starting address [3]. Most transfers in a transaction are aligned to the *transfer size*. However first transfer may be unaligned to the word boundary. The number of bytes accessed from off-chip memory ($\mathbb{B}$) for accessing l bytes from address $Addr$ on BW bytes of bus width can be expressed as

$$\mathbb{B}(Addr, l, BW) = (\lceil \frac{Addr + l}{BW} \rceil - \lfloor \frac{Addr}{BW} \rfloor) \cdot BW \tag{5}$$

CNN accelerators access 3D data (ifm, ofm and weights) partitioned into tiles. We proposed a bus width aware approach (BWA) that factors in the architectural parameters to precisely compute the off-chip memory access of CNN layers.

3.5 Offline-Tool For Design Space Exploration

Exploring design space of tile dimensions on hardware is time consuming and practically not possible. For a given CNN, the optimal tile dimensions and data reuse scheme need to be determined once. To determine the optimal tile dimensions, we have developed an offline tool that computes the off-chip memory access of CLs and FCLs of CNN which considering the architectural parameters as described in section 3.4 The tool takes layers description and architecture parameters as input and analyzes the off-chip memory accesses ($\mathbb{B}$) for all feasible solutions that satisfy the constraints (Figure 6). The offline-tool can be configured for different on-chip buffer sizes, data reuse schemes, bus-width, data-resolution and determines each layer's optimal data reuse scheme and tile dimensions. The optimal solution determined by the offline-tool is then used in the CNN layers implementation on FPGA implementation to measure the energy and run-time.

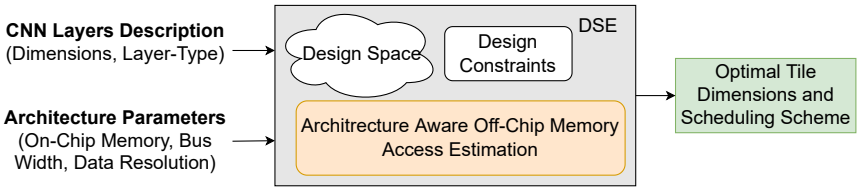
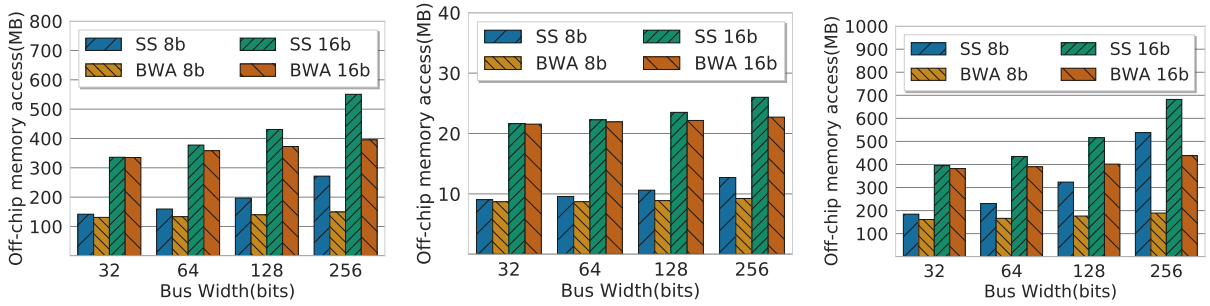


Figure 6: Architecture Aware Offline-Tool for design space exploration

3.6 Results

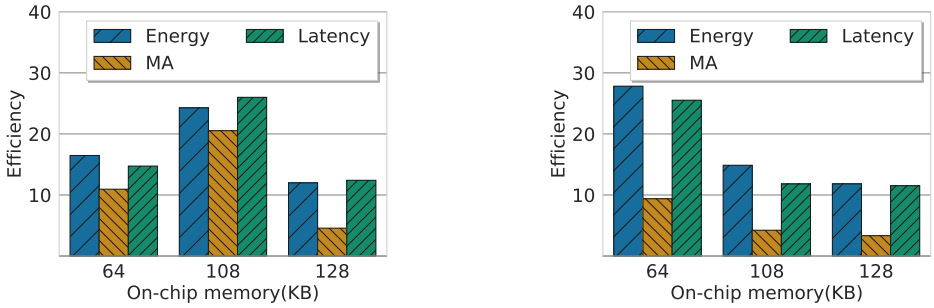
We experimented with three popular CNN networks, AlexNet [23], VGG16 [32], and ResNet [21] having 8, 16, and 50 layers, respectively, with varying layer shapes and using filters of dimensions 1×1, 3×3, 5×5, 7×7, and 11×11. To compare the results with other approaches, we have used the on-chip buffer size of 108 KB, batch size of 3 for VGG16, and 4 for ResNet and AlexNet. Fig. 7 shows the number of



(a) $\mathbb{B}_{VGG16}$ (b) $\mathbb{B}_{AlexNet}$ (c) $\mathbb{B}_{ResNet}$

Figure 7: Off-chip memory access of convolution layers for 8 and 16 bits data width. BWA: Bus Width Aware, SS: SmartShuttle

bytes accessed from off-chip memory ($\mathbb{B}$) of CLs of the CNNs for different bus widths. $\mathbb{B}$ is more for wider data buses because the overhead of unaligned access is more on a wider data bus. These unused bytes increase overall off-chip memory accesses. The proposed approach considers the bus width and addresses alignments to reduce the unaligned accesses. SS approach ignores architectural parameters and uses the same tile dimensions regardless of bus width, which results in increased off-chip memory accesses. As shown in Fig. 7, our approach reduces $\mathbb{B}$ compared to SS for the three CNNs. For ResNet:50 it reduces $\mathbb{B}_{ResNet}$ by 13%, 28%, 46%, and 65% for 8 bits data width and by 10%, 22% and 36% for 16 bits data width on 64, 128, and 256 bits wide data bus, respectively, compared to SS.



(a) (b)

Figure 8: Energy and latency efficiency of BWA compared to SS. (a)VGG16, (b)AlexNet

Figure 8a and Figure 8b show the energy, off-chip memory accesses, and latency efficiency achieved using the BWA compared to the SS approach for VGG16 and AlexNet, respectively, for 8 bits data width and

64 bits bus width. We observed that the changes in energy and latency are proportional to the changes in memory access. This observation confirms that off-chip memory access dominates the energy consumption of the CNN accelerators.

4 Optimizing the Performance of RNN/LSTM Accelerators

4.1 Introduction

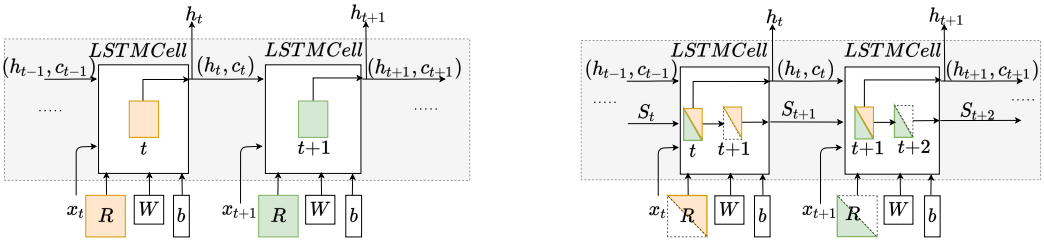
Many applications involve sequential data processing and time-series predictions, e.g., natural language processing, speech recognition, video activity recognition. Processing sequential data requires remembering the contextual information from previous data. Recurrent neural networks (RNNs) are specialized in handling such problems by maintaining an internal state based on previously seen data. LSTMs [22] are variants of RNNs designed to handle long-range dependencies by storing useful information about previous inputs for a long duration.

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 at time t. $\odot$ 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 c_t and h_t are computed using Equation (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.

LSTM computations involve multiple matrix-vector multiplications, and these matrix-vector multiplications are performed for large number of time-steps. The size of 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. Figure 9 shows the LSTM cell computations for two



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

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

Figure 9: LSTM cell computations for consecutive time-steps showing the weight accesses.

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. 9a. Accessing the weights at each time step results in large volume of off-chip memory accesses. We propose an approach that reduces off-chip memory accesses by splitting the computation in two, as shown in Figure 9b. 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, which reduces the R matrix accesses by approximately half. The data reuse in our approach is independent of on-chip buffer sizes which makes it suitable for small on-chip memory accelerators.

4.2 Related Work

The matrix-vector multiplication $W^j \cdot x$ in Equation (6), where $j \in \{i, f, g, o\}$, is independent of previous state computation. Que et al. [30] proposed a blocking-batching scheme which reuses the weights of W^j matrix by processing group of input vectors as a batch. The input vectors in the same batch share the same weight matrices (W^j). However, it is difficult to collect required number of input vectors. As the LSTM cell states (h_t and c_t) computations depend on previous time-step cell states, benefit of their batching schemes is limited to $W^j \cdot x$. Reusing weights of R across different time-steps has not been successful because of the dependency on previous time-step states.

Park et al.([29]) 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 hidden state vector h_t, a fraction of it is reused to compute the partial sum of next time step state h_{t+1}. 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 to accelerators with larger on-chip memory.

4.3 Proposed data reuse approach

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,

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

$S_t[k]$ 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 10, R^L and R^U are accessed in consecutive time steps and reused

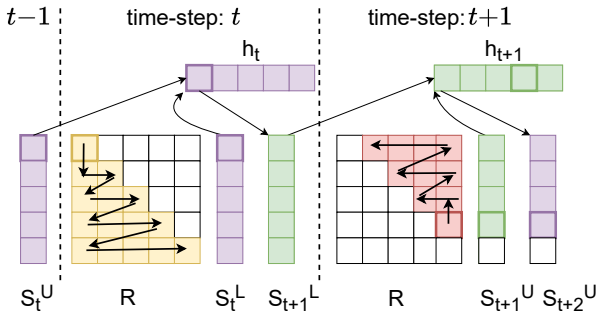
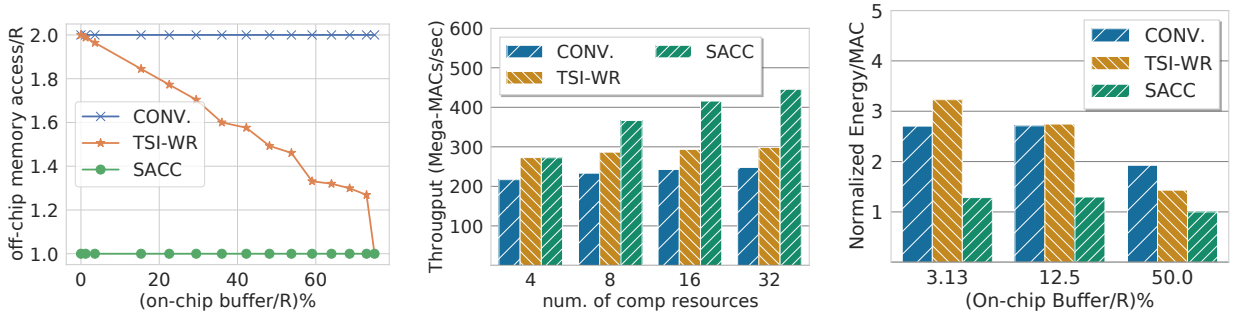


Figure 10: Splitting the hidden state vector computations into partial sums

in the partial sum computations of two steps. 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 10, the proposed approach accesses the weight matrix R once, to compute the output of two consecutive time steps h_t and h_{t+1}.

4.4 Results

We have compared our approach with conventional approaches and state of the art TSI-WR approach [29]. We have used the same on-chip buffer size to store the weight matrices to perform a fair comparison. The proposed approach requires additional $4N$ elements storage for the partial sum vectors. We have experimented with LSTM models used in speech recognition (for TIMIT [14]) and character level Language Modelling (LM) [33].



(a) (b) (c)

Figure 11: (a) Off-chip memory accesses for matrix-vector multiplication (MxV) of two consecutive time-steps with different on-chip buffer/R ratio, (b) Throughput variation of MxV for different compute resources for (on-chip buffer/matrix size)=0.5, (c) Energy improvement for different on-chip buffer size/R ratio.

4.4.1 Memory Accesses

Figure 11a compares the off-chip memory accesses of the proposed (SACC), conventional and the TSI-WR approaches. Conventional approaches access the full matrix $2\times R$ at each time step. For the TSI-WR approach data reuse depends on on-chip buffer sizes. For larger on-chip buffer sizes the data reuse is more. When the on-chip buffer size is 70%, TSI-WR approach reduced 50% off-chip memory accesses compared to conventional approach. However, for smaller on-chip buffer sizes the reduction is less. The proposed approach reduces memory access of R matrix by 50%, irrespective of the on-chip buffer size.

4.4.2 Througput Improvement

Off-chip memory bandwidth typically limits the performance of LSTM/RNN accelerators. Increasing the number of computing resources does not improve the performance for the conventional approaches, as shown in Figure 11b. TSI-WR approach improves the throughput with increasing the number of compute resources. However, throughput improvement in the TSI-WR approach is observed only for the large on-chip buffer to matrix size ratio. Figure 11b shows the results for on-chip buffer to matrix size ratio of 50%. The proposed approach, always reuses the data for two time step computations, which results in throughput improvement with increasing the number of parallel resources.

4.4.3 Energy Efficiency

Figure 11c shows the normalized energy efficiency per MAC operation for different on-chip buffer to R matrix size ratios for 128 KB on-chip buffer sizes. Increasing the on-chip buffer size to matrix size ratio improves the energy-efficiency for all the three approaches. For smaller on-chip buffer to R matrix size ratios conventional approach performs better than TSI-WR due to their simpler control logic. For large on-chip buffer sizes, TSI-WR outperforms the conventional approaches. Out of all the three approaches, the proposed approach performs better than the other two approaches for all the on-chip buffer size ratios. For 50% on-chip buffer to matrix size ratio, the SACC approach reduces 48% and 30% energy compared to conventional and TSI-WR approach, respectively.

5 Performance Improvement of SOM by using Low Bit-Width Resolution

5.1 Introduction

One of the popular low-power stategy is performing computations at reduced bit-widths by trading off accuracy. The benefits of using reduced bit-width results in improved energy performance metrics. This is because there is a reduction of the energy cost for data transfers, which usually dominates the total energy consumption for such systems.

We explored the design space of a self-organizing map (SOM) to analyse the impact of different bit resolutions on the accuracy, as well as its benefits. SOM uses a type of unsupervised learning called competitive ANN learning model. We have implemented SOM on FPGA and to lower the energy consumption, we exploit the robustness of SOM by successively lowering the resolution to gain in efficiency and lower the implementation cost. We do an in depth analysis of the reduction in resolution vs. loss in accuracy. The objective of this method is to design a bacterial recognition system for battery operated clinical use where the area, power and performance are of critical importance. We demonstrate that with 39% loss in accuracy in 12 bits and 1% in 16 bit representation can yield significant savings in energy and area.

5.2 Related Work

An emerging design paradigm that is able to achieve better energy efficiency by trading off the quality (e.g., accuracy) and effort (e.g., energy) of computation is approximate computing [42]. Many modern applications, such as machine learning and signal processing, are able to produce results with acceptable quality despite most of the calculations being computed imprecisely [39]. The tolerance of imprecise computation in approximate computing to acquire substantial performance gains and is the basis for a wide range of architectural innovations [12].

Previous works has demonstrated that high-precision computations are often unnecessary in presence of statistical algorithms [26,41]. Znag et.al. report a less than 5% of quality loss obtained by simulation of the real hardware implemented in a 45nm CMOS technology. Gupta et. al. also present similar results where they train deep networks with 16 bits fixed-point number representations and stochastic rounding [18]. Talathi et. al. show that the best performance with reduced precision can be achieved with 8 bits weights and 16 bits activation, which, if reduced to 8 bits, results in a 2% drop in accuracy. Hashemi et. al. look at a broad range of numerical representations applied to ANNs in both inputs and network parameters and analyze the trade-off between accuracy and hardware implementation metrics and conclude that a wide range of approximation parameters are feasible with negligible degradation in performance [20].

5.3 Low bit-width FPGA Design of SOM

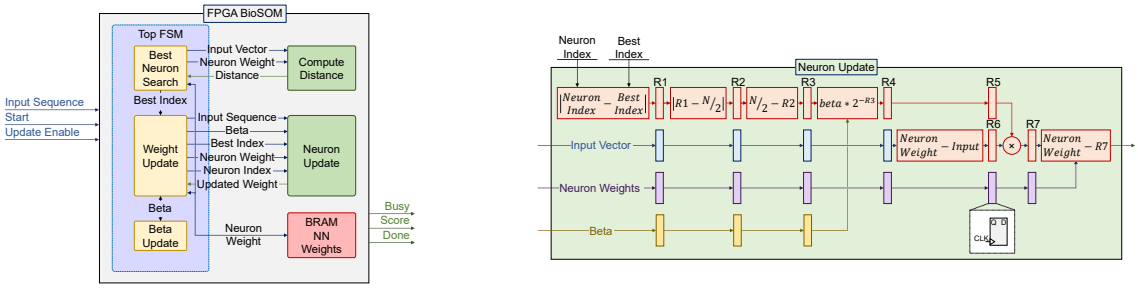
We have implemented SOM on FPGA, mapped on a Xilinx Virtex7 485t chip, for identification of bacterial genomes. A custom semi systolic array was hand crafted, for different bit width implementations, to analyze the area versus energy trade off.

Figure 12a shows a high level schematic of the FPGA implementation of BioSOM and illustrates the key components in the design. The input is a n-bit vector. Each pair of bits in the input represents one of nucleotide A,C,G or T. Thus a 16-bit word contains 8 symbols. The Neural Network weights are stored in BRAMs. Each neuron has 8 weights and each weight is stored as a fixed-point number. Bit width analysis is performed by varying the number of bits (8, 12, 16, 24 and 32) used to represent the weights.

During the training phase *Neuron Update* component is enabled by setting (*Update Enable=1*). The weights of the Neurons are updated using the distance output of the *Compute Distance* module. The *Neuron Update* component is also pipelined design with II=1. The pipeline stages are shown in Figure 12b.

5.4 Results

The SOM has been implemented with a range of fixed-point formats. With fewer bits, one naturally expect that the SOM network for bacterial identification to suffer from accuracy degradation. A MATLAB simulation model was created to analyze the accuracy loss when using fixed-point implementation. We



(a) (b)

Figure 12: (a) Hardware Module for BioSOM. (b) Neuron Update Module.

trained 10 SOMs with 10 different bacteria DNA sequences. Each SOM network has 100 neurons inside, and each neuron has 20 weights. We trained the networks by two independent training processes running in parallel. One is implemented using double precision floating point and the other is implemented with fixed-point weights. After training, we used the trained networks to identify the unknow sequence and record their scores.

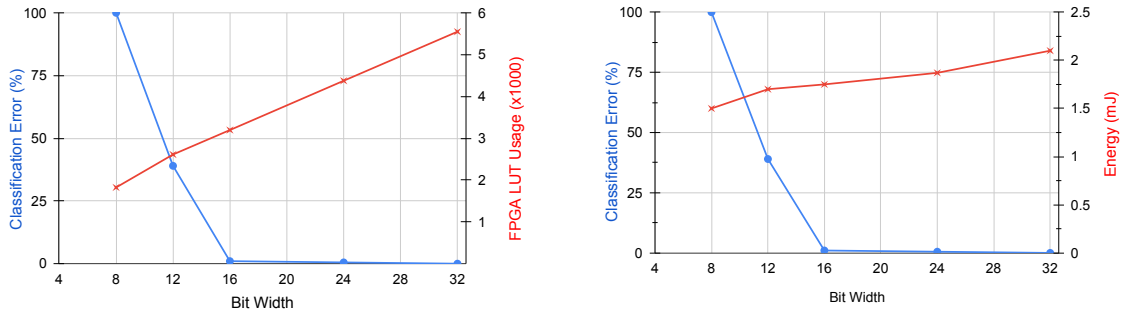
The FPGA design is implemented with Vivado v.2016.4 used for synthesis and analysis of the HDL Designs. Our design is implemented in VHDL and validated using the Vivado simulator. Experimentation is done for different fixed point representations of weights by modifying parameters in VHDL code.

The area and power numbers for different weight resolutions are extracted from the reports generated by Vivado tool post placement and routing with a working frequency of 100 MHz. Table 1 compares the resources and area for 8, 12, 16, 24, and 32 bits fixed point formats, for a SOM network with 512 neurons. The second part of the table compares the average power in the different fixed point formats, for the same SOM.

Table 1: Resource Comparison of different fixed point formats

Resource	8b	12b	16b	24b	32b
LUTs	1823	2611	3196	4375	5549
Registers	3481	4679	5871	8255	10639
Slice	854	1158	1369	1809	2395
LUT FF Pairs	1007	1369	1750	2372	3043
B-RAM	4	6	8	11	15
DSP48E1	17	17	17	17	33
Bonded IOB	57	61	65	73	81
Power(W)	8b	12b	16b	24b	32b
Total Power	0.295	0.314	0.332	0.356	0.392
Dynamic	0.052	0.071	0.089	0.113	0.148
Device Static	0.243	0.243	0.243	0.244	0.244

The results are summarized in the Figure 13a and 13b. Both the amount of utilized LUTs and total energy in Joule is presented against the classification error. From the figures, we can easily conclude that, we can substantially reduce the resources used and the energy by using a 16-bit fixed-point representation, without losing accuracy. We can reduce the resources even further by moving to the 12-bit representation, by sacrificing 39% of the SOM accuracy.



(a) (b)

Figure 13: comparison between different fixed-point fromat (a) FPGA LUT utilization (b) energy.

6 Conclusion

With the sudden surge in Neural Network based applications, there is a pressing need for improving the performance and energy efficiency of DNN accelerators for their ubiquitous usage in energy constrained devices. The performance of DNN accelerator's is limited by the memory bandwidth and off-chip memory accessed dominates the energy consumption. The key to improving the energy efficiency of these NN is reducing the expensive off-chip memory accesses. Towards this we proposed approaches to reduce the off-chip memory accesses for DNNs. The proposed approaches optimizes the improves the data reuse from on-chip memories for CNNs and LSTMs by partioning the data and scheduling the operations, without compromising the accuracy. We have also analyzed the impact of low bit resolution on the accuracy and energy, area performance.

References

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

[2] M. Alwani, H. Chen, M. Ferdman, and P. Milder. Fused-layer CNN accelerators. In *MICRO*, 2016.

[3] ARM. *AMBA AXI Protocol Specification [Online]. Available at https://developer.arm.com/docs*, 2010. Rev. 2.0.

[4] 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.

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

[6] 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.

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al. DaDianNao: A machine-learning supercomputer. In *MICRO*, 2014.

[8] 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.

[9] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN:efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[10] 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.

[11] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. ShiDianNao: Shifting vision processing closer to the sensor. In *ISCA*, 2015.

[12] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460, 2012.

[13] 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.

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

[15] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. A 240 G-ops/s mobile coprocessor for deep neural networks. In *CVPR Workshops*, 2014.

[16] 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.

[17] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *ICML*, 2015.

[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, pages 1737–1746, 2015.

[19] 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.

[20] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1478–1483, 2017.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

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

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[24] 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.

[25] 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.

[26] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 246–247, 2017.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] M. Sundermeyer, H. Ney, and R. Schlüter. From feedforward to recurrent lstm neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):517–529, 2015.

[34] 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.

[35] 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.

[36] X. Wei, Y. Liang, and J. Cong. Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management. In *DAC*, 2019.

[37] 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.

[38] L. Xie, X. Fan, W. Cao, and L. Wang. High throughput CNN accelerator design based on FPGA. In *FPT*, 2018.

[39] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–54, 2013.

[40] 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.

[41] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. Approxann: An approximate computing framework for artificial neural network. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 701–706, San Jose, CA, USA, 2015. EDA Consortium.

[42] Q. Zhang, F. Yuan, R. Ye, and Q. Xu. Approxit: An approximate computing framework for iterative methods. In *Proceedings of the 51st Annual Design Automation Conference*, pages 97:1–97:6, 2014.