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**
1.1 Impact of off-chip memory access 2

2 Related Work **4**

3 Mapping Feed Forward Neural Networks **4**
3.1 Introduction 4
3.2 Related Work 5
3.3 Impact of architectural parameters on memory access 6
3.4 Off-Chip Memory Accesses of CNN Layers 6
3.5 Optimizing the off-chip memory access 7
3.6 Results 7

4 Mapping Recurrent Neural Networks **8**
4.1 Introduction 8
4.2 Related Work 9
4.3 Proposed data reuse approach 9
4.4 Results 10

5 Optimizing SOMs using low bit-width **11**
5.1 Introduction 11
5.2 Related Work 11
5.3 Low bit-width FPGA Design of SOM 11
5.4 Results 12

6 Conclusion **12**

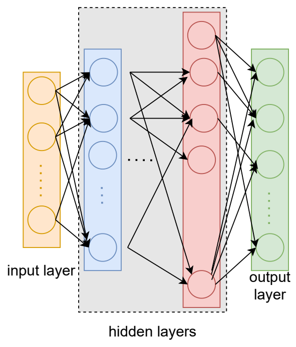
1 Introduction

Deep neural networks (DNNs) are machine learning algorithms used in wide range of applications due to their high accuracy. To improve the accuracy, recent DNNs use large number of parameters and perform memory and compute intensive operations. Many latest consumer electronics devices like smartphones and home appliances use DNNs to provide new features and improve user experiences. The manufacturers are shifting the processing of these algorithms to edge devices to improve response time and eliminate network bandwidth issues. These edge devices, usually battery operated, commonly use DNN accelerators to speed-up processing and reduce energy consumption.

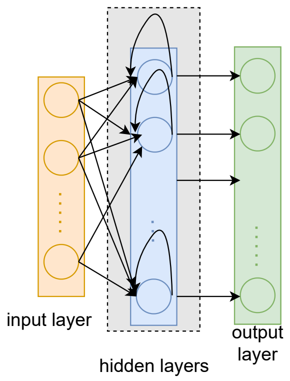
Deep neural networks consists of an input layer, an output layer and several intermediate hidden layers. Recent DNN can have upto thousands of hidden layers. Figure 1 shows two broad class of neural networks. Most of the recent DNNs can be classified as following,

1. Feed Forward NNs: This class of NNs use the outputs from a layer as input to the subsequent layers. For a given input, output is independent of previous seen inputs. Convolution Neural Networks (CNN) belongs to this category and commonly used for image and video processing applications.

2. Recurrent NNs: This class of NNs maintain the state information to store the contextual information and state is updated with every input. The output depends on the input as well as the state information. For the same input, the output may vary depending on the current state. This class of networks is used for processing sequential data e.g. speech recognition and language processing tasks. Long Short Term Memory networks (LSTMs) are one of the most popular variants of RNN.



(a) Feed Forward NN



(b) Recurrent NN

Figure 1: Feedforward and Recurrent Neural Networks.

DNNs have training and inference phase. They are first trained (training phase) to determine the learning parameters (weights and biases) and then these learned parameters are used in applications to predict the output (inference phase). The training of these deep networks require large number of inputs and several iterations to update the weights and baises. This may take several hours to days and require significant computations. The training is mostly performed offline on high end machines and servers. The parameters learned during training phase are then used in applications for inference. In this work, we focus on the inference phase of DNNs, which is performed energy constraint devices.

To improve the performance FPGA [2, 15, 17, 42, 44, 47], GPU [9] and ASIC [6–8, 11] accelerators with large number of parallel compute resources are proposed. While these accelerators can match the performance to some extent, their energy efficiency is still a concern. Energy efficiency of DNN accelerators is of paramount importance for their ubiquitous usage in energy constrained devices like smart-phones, tablets etc. More than 80% of the overall energy consumption of these accelerators is due to off-chip memory accesses [6].

1.1 Impact of off-chip memory access

1.1.1 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 often limited by off-chip memory bandwidth. Fig. 2 shows, two kernels with different compute intensity (FLOPS/byte), due to different data reuse techniques. Compared to Kernel 1, Kernel 2 performs more computations per byte. 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

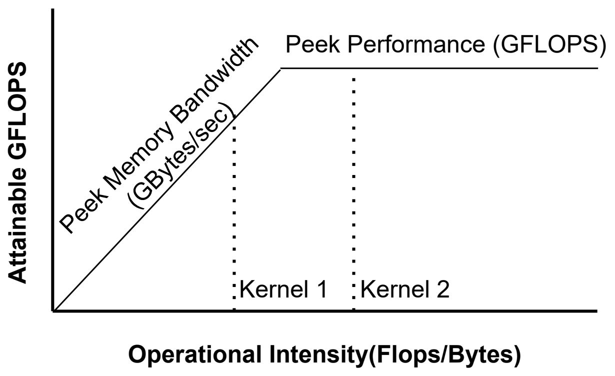


Figure 2: roofline model

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 Energy efficiency

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

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

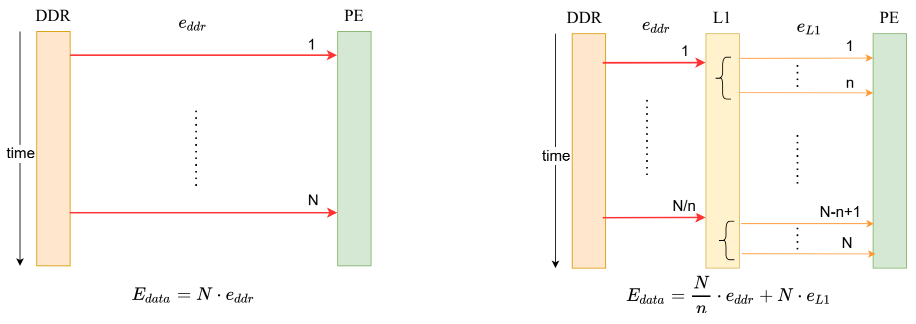
where E_{comp} and E_{data} are the computation and data movement energy, respectively. E_{comp} for a DNN layer depends on the number of operations in the layer, which is fixed as it depends on layer shape e.g. number of activations, filters, and filter sizes. However E_{data} can be optimized using data reuse techniques. Typically DNN accelerator system has multiple levels of memory hierarchy. Memories at different levels of hierarchy have different sizes, access-energies and access-time. The off-chip memory (DDR), at the top of the hierarchy, has the largest capacity, and highest access-energy and access-time compared to memories at lower level of hierarchy. The off-chip memory access energy is upto two orders higher compared to on-chip memory access energy.

Figure 3 illustrates the impact of data reuse on the energy efficiency of a system with 2 levels of memory (DDR and L1). The energy efficiency can be expressed 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, 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).

Reducing the off-chip memory accesses is the key to reduce the energy consumption of DNN accelerators. State of the art DNN accelerators aim to maximize the data reuse from on-chip memory to reduce the off-chip memory accesses. Our work aims 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. Towards


DDR
e_{ddr}
PE
1
time
N
$E_{data} = N \cdot e_{ddr}$
DDR
e_{ddr}
L1
e_{L1}
PE
1
1
n
time
N/n
N-n+1
N
$E_{data} = \frac{N}{n} \cdot e_{ddr} + N \cdot e_{L1}$

(a) no data reuse (b) with reuse ($\times n$) from L1

Figure 3: Data access energy E_{data}.

this end we propose a mapping framework 4 to partition and schedule the operations of NNs to improve the data-reuse from on-chip memory. Our main contributions in this thesis dessertations are

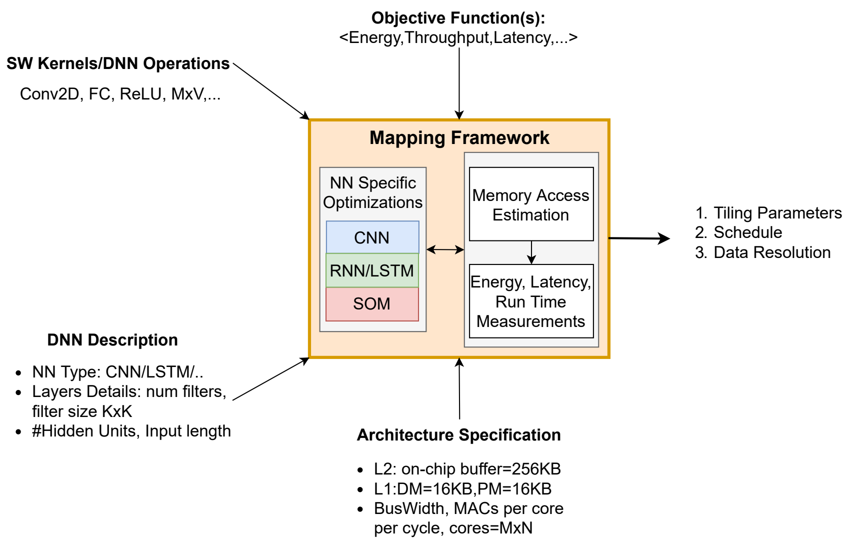

Objective Function(s):
<Energy,Throughput,Latency,...>
SW Kernels/DNN Operations
Conv2D, FC, ReLU, MxV,...
Mapping Framework
NN Specific Optimizations
CNN
RNN/LSTM
SOM
Memory Access Estimation
Energy, Latency, Run Time Measurements
1. Tiling Parameters
2. Schedule
3. Data Resolution
DNN Description
• NN Type: CNN/LSTM/..
• Layers Details: num filters, filter size KxK
• #Hidden Units, Input length
Architecture Specification
• L2: on-chip buffer=256KB
• L1:DM=16KB,PM=16KB
• BusWidth, MACs per core per cycle, cores=MxN

Figure 4: Mapping DNN on coarse grain parallel architectures

1. We developed a tool that precisely computes the off-chip memory accesses of DNN layers partitioned into tiles, while taking into account the accelerator's architectural parameters.

2. 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.

3. We propose a novel data reuse approach to improve the performance and energy efficiency of recurrent neural networks (RNN). The proposed approach partitions the data and schedules the operations in a way that reduces the off-chip memory accesses of large matrices by half.

4. We analyse the effect of using different bit resolutions on the accuracy on a NN, as well as the benefits of using it for designing a battery operated system where the area, power and performance are of critical importance.s

2 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 [4, 10, 41] and FPGA based accelerators [5, 13, 16, 20, 27] 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 5 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.

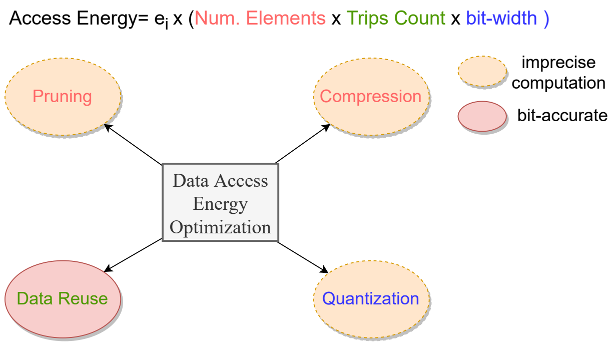

Access Energy= e_i x (Num. Elements x Trips Count x bit-width)
Pruning
Compression
imprecise computation
bit-accurate
Data Access Energy Optimization
Data Reuse
Quantization

Figure 5: Energy efficiency optimization approaches

Some approaches [13, 27, 36] 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 [13, 40] used 18-bit, Chang et al. [5] used 16-bit, Han et al. [20] used 12-bits precision for storing the inputs and weights, Lee et al. [27] 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. [20] 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. [40] used block-circulant matrices representations to compress the LSTM/RNN model and to eliminate irregularities resulted from compression. Some approaches ([20, 31, 32]) 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.

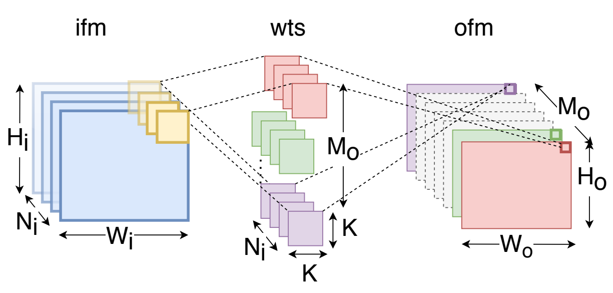
The other line of works reduced the memory accesses without compromising the accuracy of the network by applying the data-reuse techniques. Que et al. [35] proposed a blocking-batching scheme to reuse the LSTM weights fetched from external memory. Park et al.([33]) proposed a time step interleaved weight reuse scheme (TSI-WR).

3 Mapping Feed Forward Neural Networks

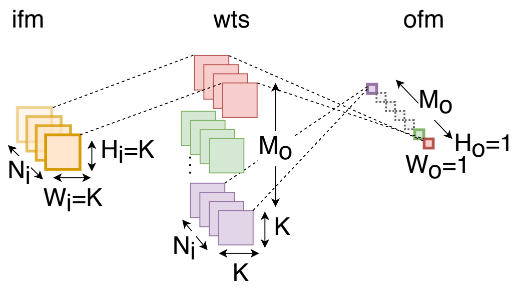
3.1 Introduction

The most popular feed forward neural networks are convolution neural networks (CNN). They are widely used in image and video processing applications and able to achieve high accuracy.

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 6a and 6b, respectively. Each CL and FCL layer takes 3D input frames (*ifm*) and applies filter weights (*wts*) to compute output frames (*ofm*). Modern CNNs are very deep and have millions of parameters. To achieve



(a) Convolution Layer



(b) Fully Connected Layer

Figure 6: Convolution and fully connected layers

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.

To fit the layer data into on-chip memory, CNN accelerators apply loop tiling to partition the layer data into small tiles. 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. 7 shows a layer's data stored in off-chip memory and its tiles in the accelerator's on-chip buffer. The

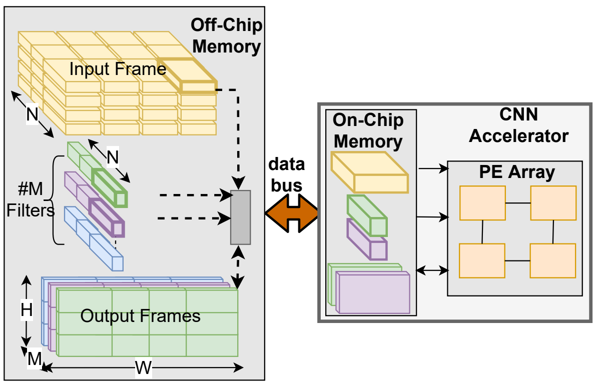


Figure 7: CNN layer tiles in off-chip and on-chip memory

order 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 trips count and the off-chip memory access in a data reuse scheme depend on the layer shape and the tile dimensions. The tile dimensions significantly impacts the off-chip memory accesses [28,47]. Memory accesses also 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 accurately and determines the optimal tile dimensions and data reuse scheme for CNN layers.

3.2 Related Work

To meet the computation and energy demands of CNNs, researchers have proposed efficient accelerators [2, 8, 15, 44]. Zhang et al. [47] and Li et al. [28] 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. [47] 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. 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. Li et al. [28] proposed a layer-wise adaptive data partitioning and scheduling scheme. However, their approach ignored the architectural parameters and address alignment, and assumed 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 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 8 illustrates memory accesses on a 64-bit data bus. Fig. 8a 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 8b, 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. 8d. 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

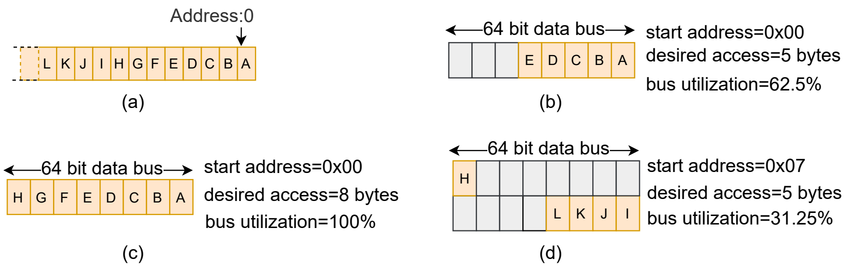


Figure 8: Off-chip memory accesses on 64-bit wide data bus

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

$$\mathbb{B}(Addr, l, BW) = (\lceil \frac{Addr + l}{BW} \rceil - \lfloor \frac{Addr}{BW} \rfloor) \cdot BW \tag{2}$$

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

3.4 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 (3), where columns represent ifm, ofm, and weights.

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_{iro} \\ \mathbf{r}_{oro} \\ \mathbf{r}_{wro} \end{bmatrix} = \begin{bmatrix} 1 & (2\lceil \frac{N}{T_n} \rceil - 1) & \lceil \frac{H}{T_r} \rceil \lceil \frac{W}{T_c} \rceil \\ \lceil \frac{M}{T_m} \rceil & 1 & \lceil \frac{H}{T_r} \rceil \lceil \frac{W}{T_c} \rceil \\ \lceil \frac{M}{T_m} \rceil & (2\lceil \frac{N}{T_n} \rceil - 1) & 1 \end{bmatrix} \tag{3}$$

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

$$\mathbb{B} = r_i \cdot \mathbb{B}_i + r_o \cdot \mathbb{B}_o + r_w \cdot \mathbb{B}_w \tag{4}$$

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 2.

3.5 Optimizing the off-chip memory access

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 (5) below

$$\begin{aligned} &(V_i+V_o+V_w) \leq buffSize \\ &0 < T_{c_o} \leq W_o, \quad 0 < T_{r_o} \leq H_o \\ &0 < T_{n_i} \leq N_i, \quad 0 < T_{m_o} \leq M_o \end{aligned} \tag{5}$$

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

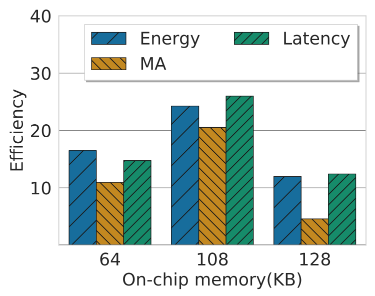
$$\begin{aligned} V_i &= T_{c_i} \cdot T_{r_i} \cdot T_{n_i} \cdot DW \\ V_o &= T_{c_o} \cdot T_{r_o} \cdot T_{m_o} \cdot DW \\ V_w &= K^2 \cdot T_{n_i} \cdot T_{m_o} \cdot DW \end{aligned} \tag{6}$$

where DW is the data width in bytes. Determining the optimal tile dimensions that minimizes $\mathbb{B}$ (4), is a constraint optimization problem. Equations (4) and (5) 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.

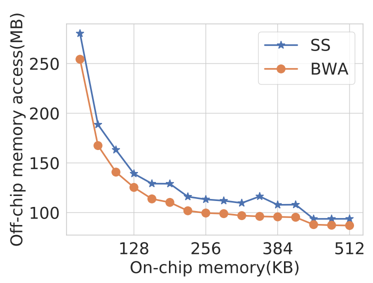
For a given CNN, the optimal tile dimensions and data reuse scheme need to be determined once. To determine the optimal tile dimensions, we have developed an offline tool that computes the off-chip memory access of CLs and FCLs of CNN. The tool takes layers shape, bus width, data bit width, and on-chip memory size as input and analyzes off-chip memory accesses ($\mathbb{B}$) for all tile dimensions that satisfy the constraints. It analyzes the off-chip memory access for different data reuse schemes and determines each layer's optimal data reuse scheme and tile dimensions.

3.6 Results

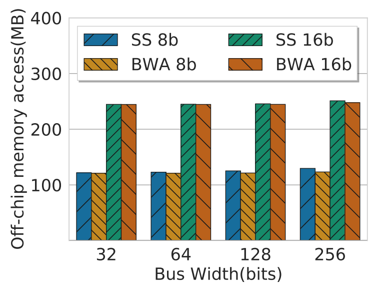
We experimented with three popular CNN networks, AlexNet [26], VGG16 [37], and ResNet [22] having 8, 16, and 50 layers, respectively, with varying layer shapes and using filters of dimensions 1×1, 3×3, 5×5, 7×7, and 11×11. To compare the results with other approaches, we have used the on-chip buffer size of 108 KB, batch size of 3 for VGG16, and 4 for ResNet and AlexNet. Figure 9 shows the comparison of the proposed approach with SmartShuttle (SS) [28] for VGG16. The SS approach determines the tile dimensions and data reuse scheme adaptively for each layer to minimize the data size (volume) accessed from off-chip memory.



(a) VGG16: Energy and latency efficiency



(b) $\mathbb{B}_{VGG16}$: Varying on-chip buffer sizes



(c) VGG16: Fully connected layers

Figure 9: Latency and off-chip memory access. BWA: Bus Width Aware, SS:SmartShuttle

3.6.1 Latency And Energy Analysis

Figure 9a show the energy, off-chip memory accesses (MA), and latency efficiency achieved using the proposed approach compared to the SS approach for VGG16 for 8 bits data width and 64 bits bus width. Using our FPGA implementation, we measured the memory access latencies and execution time for CLs to estimate the energy efficiency.

3.6.2 On-chip Memory Size

The size of the on-chip memory constrains the tile dimensions. Larger on-chip memory can accommodate bigger tiles which reduces the trips and off-chip memory access transactions. Figure 9b shows off-chip memory access for VGG16. Off-chip memory access reduces with increasing on-chip memory size for both approaches. However, the proposed approach performs better than SS for all the on-chip memory sizes.

3.6.3 Impact of Bus Width on $\mathbb{B}$ of FCLs

In FCLs, the height and width of tiles and data are the same. Tiles in FCL can be accessed from off-chip memory using a single transaction which results in fewer transactions in FCLs compared to CLs. Whereas in CLs, multiple transactions are required to access different rows of the tiles. Thus the impact of address alignment and bus width is less on $\mathbb{B}$ of FCLs as compared to CLs. Figure 9c shows the off-chip memory accesses of FCLs of VGG16 for 8 bits data width. Our approach reduces $\mathbb{B}^{FC}_{VGG16}$ by 1%, 2%, 3%, and 4% on 32, 64, 128, and 256 bits wide buses, respectively.

4 Mapping Recurrent Neural Networks

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 [23] are variants of RNNs designed to handle long-range dependencies by storing useful information about previous inputs for a long duration.

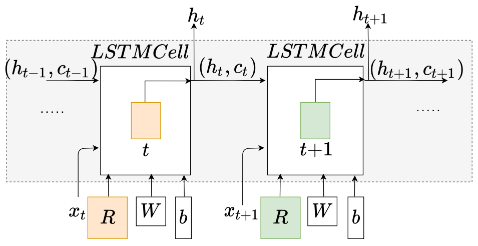
Typically the computations of LSTM cell is described by the following equations

$$\begin{aligned}
&i{=}\sigma(W^i{\cdot}x_t{+}R^i{\cdot}h_{t-1}{+}b^i)\\
&f{=}\sigma(W^f{\cdot}x_t{+}R^f{\cdot}h_{t-1}{+}b^f)\\
&g{=}\tanh(W^g{\cdot}x_t{+}R^g{\cdot}h_{t-1}{+}b^g)\\
&o{=}\sigma(W^o{\cdot}x_t{+}R^o{\cdot}h_{t-1}{+}b^o)\\
&c_t{=}f{\odot}c_{t-1}{+}i{\odot}g\\
&h_t{=}o{\odot}\tanh(c_t)
\end{aligned} \tag{7}$$

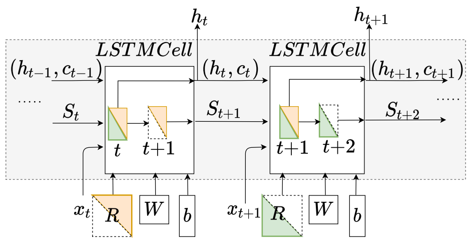
where x_t is the input, h_t is the hidden state, and c_t is the cell state at time t. i, f, g, o are the computed gate values. $\odot$ denotes the element-wise multiplications. W^j and R^j are the input and hidden state weight matrices, and b^j is the bias vector, learned during the training process, 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 (7). 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 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.

Figure 10 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. 10a. 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. 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. As shown in Figure10b this 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.



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



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

Figure 10: LSTM cell computations for consecutive time-steps showing the weight accesses.

4.2 Related Work

Que et al. [35] proposed a blocking-batching scheme to reuse 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 (h_t and c_t) 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.([33]) 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.

4.3 Proposed data reuse approach

The computation of the h_t can be expressed as shown below

$$h_t[k] = F(S_t[k] + q_t[k]) \tag{8}$$

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,

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

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

$$S_t^L[k] = \sum_{n=0}^{k} R[k][n] \cdot h_{t-1}[n] \tag{10}$$

$$S_t^U[k] = \sum_{n=k+1}^{N-1} R[k][n] \cdot h_{t-1}[n] \tag{11}$$

Equation (10) uses the lower-diagonal and diagonal elements of R (R^L), and (11) uses the upper diagonal elements of R (R^U). As shown in Figure 11, R^L and R^U are accessed in consecutive time steps and reused

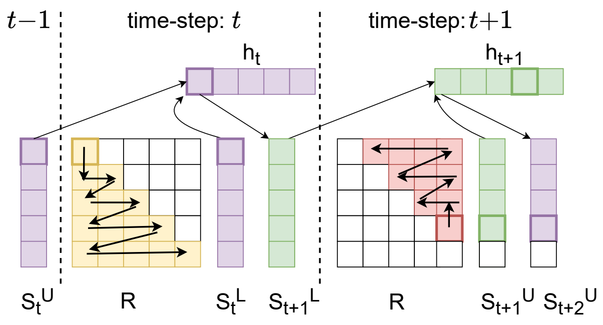


Figure 11: Splitting the hidden state vector computations into partial sums

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 11, 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 TSI-WR approach [33]. 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 experimented with LSTM models used in speech recognition (for TIMIT [14]) and character level Language Modelling (LM) [38].

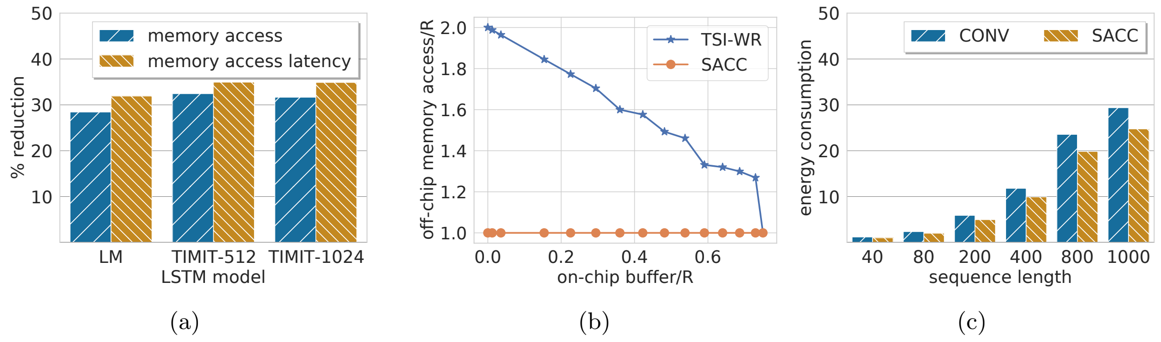


Figure 12: (a) off-chip memory access and latency (b) Off-chip memory access comparison between TSI-WR and SACC approach for two consecutive time steps. (c) normalized energy. CONV: conventional

4.4.1 Memory Accesses

Figure 12a 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. The proposed approach reduces the run-time by 12.8% and 16% for LM and TIMIT models.

4.4.2 Comparison with TSI-WR

Figure 12b 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.

4.4.3 Energy Efficiency

We computed the energy consumption using the design power reported by the Vivado synthesis tool, execution time, and off-chip memory accesses. Figure 12c 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.

5 Optimizing SOMs using low bit-width

5.1 Introduction

Neural Networks have intrinsic error resilience and one of the popular low-power stategy is performing computations at reduced bit-widths by trading off accuracy. The benefits of doing this results in improved energy performance metrics in both the data path and in the memory path. 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 explore the design space of a self-organizing map (SOM) used for rapid and accurate identification of bacterial genomes which is an important health care problem. 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

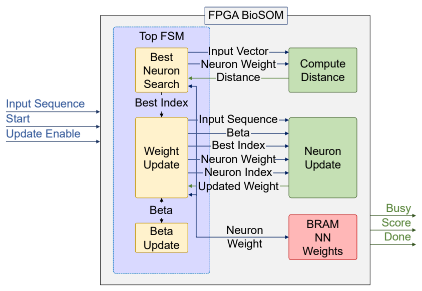
An emerging design paradigm that is able to achieve better energy efficiency by trading off the quality (e.g., accuracy) and effort (e.g., energy) of computation is approximate computing [49]. Many modern applications, such as machine learning and signal processing, are able to produce results with acceptable quality despite most of the calculations being computed imprecisely [46]. 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 [30,48]. 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 [21].

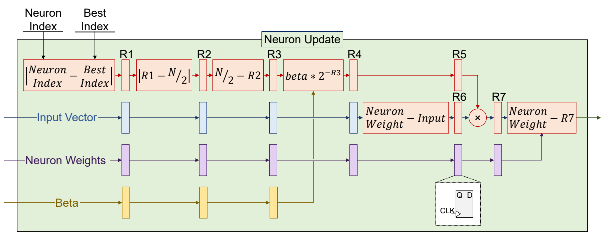
5.3 Low bit-width FPGA Design of SOM

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 13a 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.



(a)



(b)

Figure 13: (a) Hardware Module for BioSOM. (b) Neuron Update Module.

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 pipe-lined design with II=1. The pipeline stages are shown in Figure 13b.

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 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. The results are summarized in the Figure 14a and 14b. 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.

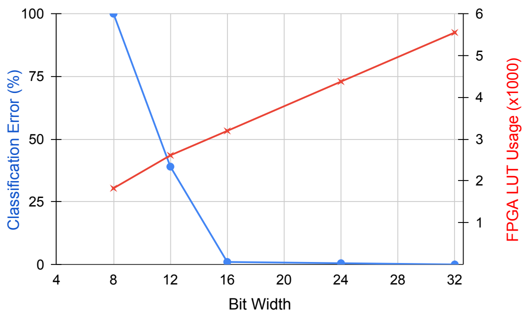
6 Conclusion

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 developed a mapping framework that

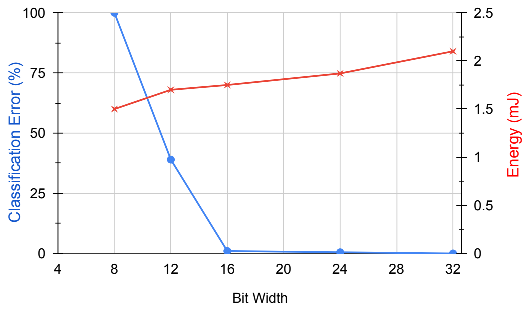
Table 1: Resource Comparison of different fixed point formats

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



(a)



(b)

Figure 14: comparison between different fixed-point fromat (a) FPGA LUT utilization (b) energy.

applies approaches to reduce the off-chip memory accesses for DNNs. The proposed approaches optimizes 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] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[20] 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.

[21] 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.

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

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

[24] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang. Accelerating low bit-width convolutional neural networks with embedded fpga. In *27th International Conference on Field Programmable Logic and Applications*, pages 1–4, Sep. 2017.

[25] T. Kohonen. Essentials of the self-organizing map. *Neural Netw.*, 37:52–65, Jan. 2013.

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

[27] 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.

[28] 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.

[29] S. McConnell, R. Sturgeon, G. Henry, A. Mayne, and R. Hurley. Scalability of self-organizing maps on a GPU cluster using OpenCL and CUDA. *Journal of Physics: Conference Series*, 341:012018, feb 2012.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] M. Porrmann, U. Witkowski, and U. Rückert. *Implementation of Self-Organizing Feature Maps in Reconfigurable Hardware*, pages 247–269. Springer US, Boston, MA, 2006.

[35] 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.

[36] 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.

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

[38] 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.

[39] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Proceedings of the 49th Annual Design Automation Conference*, pages 796–801, 2012.

[40] 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.

[41] 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.

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

[43] 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.

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

[45] Y. Yang, D. Stathis, P. Sharma, K. Paul, A. Hemani, M. Grabherr, and R. Ahmad. RiBoSOM. In *Proceedings of the 18th International Conference on Embedded Computer Systems Architectures, Modeling, and Simulation - SAMOS*, pages 105–114, New York, New York, USA, 2018. ACM Press.

[46] 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.

[47] 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.

[48] 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.

[49] 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.