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.

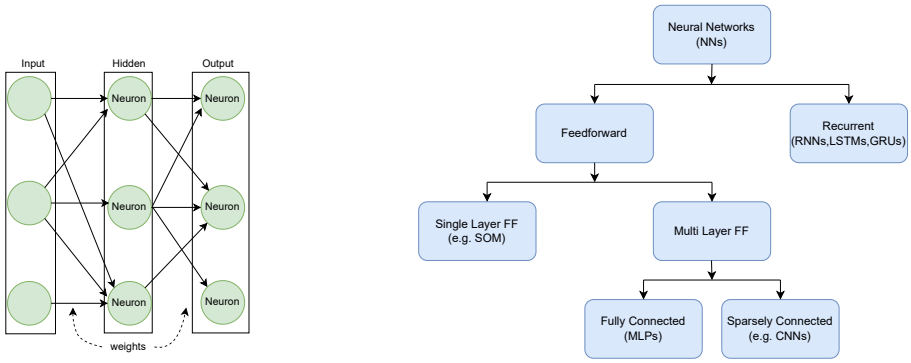
4. D. Stathis, Y. Yang, **S. Tewari**, A. Hemani, K. Paul, M. Grabherr and R. Ahmad, "*Approximate Computing Applied to Bacterial Genome Identification using Self-Organizing Maps*", in ISVLSI 2019.

Contents

1 Introduction

The past few years have seen a rapid growth in Neural Network (NN) based applications. These are widely used in healthcare, agriculture, road safety, surveillance, defense, and others. Modern computing systems capable of storing and processing large volumes of data, and the availability of big data sets, have enabled NNs to achieve human-like performance, which was not possible a few decades ago.

Neural Networks are machine learning algorithms inspired by processing mechanisms in the human brain. These consist of several neurons connected and organized as layers. Figure 1a shows a simple NN example. Each layer has weights and biases, learned using the training process. Once trained, these weights and biases are used in the applications for inferencing. There are several classes of NNs that differ from each other in the number of neurons, connections between them, and method of training.



(a) (b)

Figure 1: (a) Example of simple Neural Network. (b) Broad categories of NNs.

Figure 1b shows two broad classes of NNs, feedforward neural networks (FFNNs) and recurrent neural networks (RNNs). In FFNNs, the data flows from the input layer to the output layer via intermediate layers, and there are no loops. These can be represented as directed acyclic graphs. FFNNs do not have any internal state, and output depends on the current input and the parameters. Based on the number of layers, FFNNs can be further classified as single-layer and multi-layer FFNNs. Single-layer Feedforward NNs consist of just two layers - an input and an output layer. Only the output layer performs computation. Self-Organizing Maps (SOMs) are examples of single-layer NNs and are used in dimensionality reduction and clustering applications.

Multilayer FFNNs represent the most important class of machine learning algorithms. They consist of one or more intermediate layers (hidden layers) between the input and output layers. Depending on whether the neurons in the layer are connected to all or a few of the neurons in the previous layers, these can be further classified as fully connected FFNNs (Figure 2a) or sparsely connected FFNNs (Figure 2b), respectively. One of the most popular class of FFNNs used for image classification and recognition from images is Convolutional Neural Networks (CNNs).

Another popular class of NNs is recurrent neural networks (RNNs), which have layers with loops (Figure 2c). These networks extract information from the inputs and store it in an internal state. In this class of NNs, the output depends not only on the current input but also on the internal state. These networks are widely used in speech recognition, natural language processing (NLP), and other sequential data processing applications. Long short-term memory networks (LSTMs) are among the popular variants of RNNs.

Multilayer NNs with more than three layers are referred to as deep neural networks (DNNs), capable of learning complex functions. Recently, DNN models with several hundreds of layers have been reported [18]. These DNNs have millions of parameters (weights and biases) and perform compute-intensive and memory-intensive operations. For learning, these DNNs require large training data sets, and need to go through multiple iterations to achieve the desired accuracy, requiring significant computational resources and time. Training of these large models is mainly performed on high-end servers and consume high energy. Once these models are trained, they are deployed in applications for inferencing, where the deployment platforms may range from cloud to embedded devices.

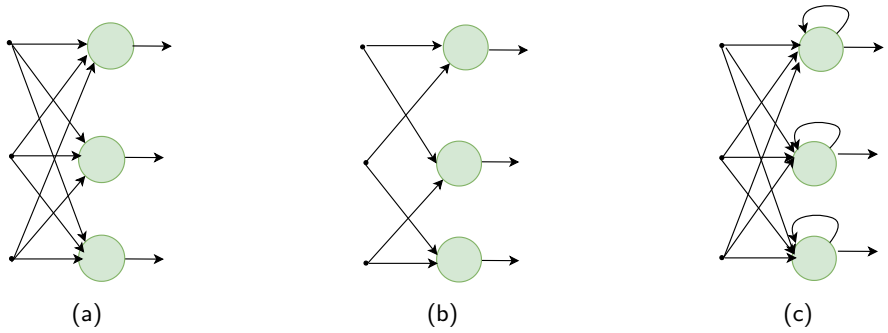


Figure 2: Types of NN layers (a) Fully Connected FF. (b) Sparsely Connected FF. (c) Recurrent.

Manufacturers are shifting the processing of NNs from cloud to edge devices like smartphones and tablets to improve the user experience, eliminate network bandwidth issues, and improve privacy and security. These battery-operated edge devices have limited resources and a tight energy, budget which poses significant challenges. Efficient processing of DNNs inferencing on edge devices is critical for their widespread usage. In this work, we focused on the efficient processing of NNs inference on edge devices.

Energy efficiency and throughput are the two most important metrics for edge devices. While energy efficiency is of paramount importance for longer battery time, high throughput is desired for better user-response time. Edge devices mostly use customized accelerators to meet energy and throughput demands. Several FPGA [2, 13, 14, 31, 33, 35], GPU [8] and ASIC [5–7, 9] accelerators have been proposed to meet the performance and energy targets.

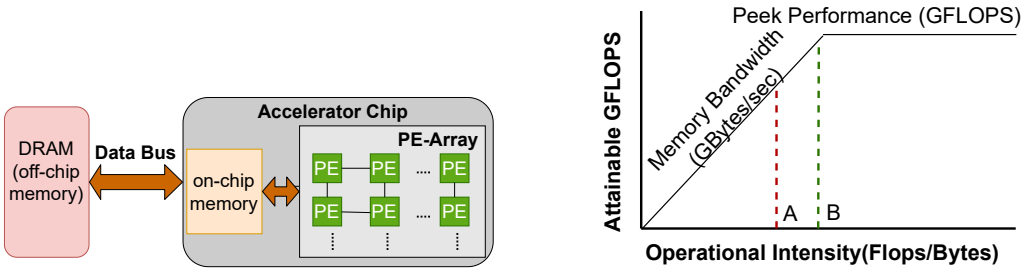


Figure 3: (a) Typical DNN accelerator architecture. (b) Roofline model

Figure 3a shows a typical DNN accelerator architecture, which consists of an off-chip memory and an accelerator chip. An accelerator chip mainly consists of an on-chip memory of a few hundred 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 PEs. Each memory level has different access latency and energy costs. The memory access energy from off-chip memory is up to two orders of magnitude higher than a PE computation operation [7]. It has been observed that more than 80% of the overall energy consumption of these accelerators is due to off-chip memory accesses [5].

The PE-array (Figure 3a) has a large number of processing elements, capable of performing several operations per cycle. However, throughput is often limited by off-chip memory bandwidth [32]. Fig. 3b shows attainable throughput in such a scenario as a function of the operational intensity of an application. The figure shows two application kernels (A and B) with different operational intensities (FLOPS/byte), with performance limited by the memory bandwidth in both cases, which is typically the case for most DNN accelerators. Kernel B, however, has better operational intensity than kernel A and achieves better throughput. Effective data-reuse techniques are required to improve the performance of bandwidth-limited parallel architectures. Therefore, reducing the off-chip memory is the key to improving the throughput and energy efficiency of DNN accelerators. Much recent research has focused on reducing off-chip memory accesses.

Recent works that aim to reduce the off-chip memory accesses of NN accelerators can be classified into

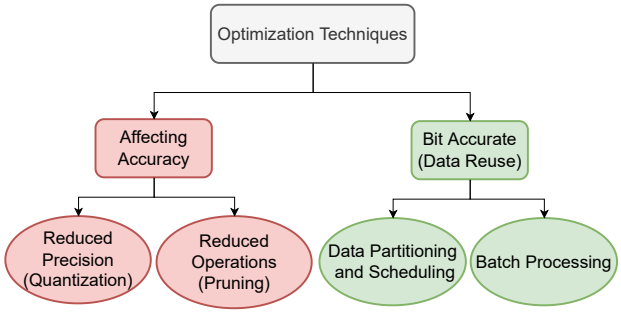


Figure 4: Broad Classification of previous works for improving the performance of DNN accelerators.

two broad categories, as shown in Figure 4. One category of work exploits error-tolerance and redundancy in NNs using quantization, compression, and pruning techniques to reduce the precision, the number of operations and models' size [4, 11, 16, 21, 30]. With reduced precision, the storage requirement and memory accesses reduce proportionally and improve energy efficiency [29]. In shallow NNs, the number of parameters is smaller compared to DNNs. Quantization and pruning techniques result in reduced NN model sizes. The reduced model for shallow NNs may fit into the on-chip memory and thus eliminate the off-chip memory bandwidth bottleneck. However, quantization and pruning approaches impact the accuracy of the networks and may not be suitable where accuracy can not be compromised. Also, the number of parameters in modern DNNs is significantly large. For these DNNs, besides quantization and pruning, additional techniques are required to further reduce the off-chip memory accesses.

The other approach, which does not affect the accuracy of the network, is to reduce repeated off-chip accesses to the same NN coefficients when the entire set of NN coefficients does not fit in the on-chip memory. This is quite effective for many modern DNNs (e.g., CNNs, RNNs) with a significantly large number of parameters [22,25,26,35]. The weights are repeatedly used for all the inputs during the inference phase. Inputs can be grouped as a batch and processed to reuse the weights. This technique is referred to as batch processing. Increasing the batch size increases the weights reuse but also increases the latency. Another prominent data reuse technique is data partitioning and scheduling. In this technique, the data is partitioned into tiles, and operations on tiles are scheduled in such a way that data can be accessed from the on-chip memory, as far as possible. For a given DNN, there are numerous ways of doing data partitioning and scheduling, offering different extents of data reuse. Choosing an optimal way here is non-trivial.

The data-reuse approaches are orthogonal to the quantization techniques and can be combined to reduce the off-chip memory accesses further. In this work, we have explored both types of approaches and contributed some new ideas.

Our main contributions to this thesis dissertations are as follows.

1. Previous research has shown that for data partitioning and scheduling, making a choice independently for each layer of a DNN is better than making a common choice for the entire NN because of the different shapes of various layers. We observe that the choice of optimal data partitioning and scheduling not only depends on the shape of a layer but also on some architectural parameters. We present an analytical framework that quantifies the off-chip memory accesses and compute cycles for DNN layers of varying shapes, also taking into account the architectural constraints. It is useful for comparing different data partitioning and scheduling schemes to explore the large design space in order to find the optimal solution for improving the energy and throughput.

2. Based on the above analytical framework, we propose a data reuse approach that takes into account the architectural parameters and determines the optimal partitioning and scheduling scheme to minimize the off-chip memory access of DNN layers. We demonstrate the efficacy of our partitioning and adaptive scheduling approach on the compute and memory-intensive CNN layers.

3. We propose a novel data reuse approach to improve the throughput and energy efficiency of state-of-the-art recurrent neural networks (RNNs). The proposed approach splits the computations and combines them in a way that reduces the off-chip memory accesses of large matrices significantly. We measure the design power and memory accesses on FPGA implementation of Long-Short Term

Memory Network (LSTM) accelerators and show the energy and throughput improvements achieved by our approach.

4. We analyze 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 (SOMs) for designing energy-constrained systems where the area, power, and performance are of critical importance. Using an efficient implementation of SOM design on FPGA, which can be configured for different bit resolutions, we show performance comparison for different data precisions.

2 Analytical Framework for Memory Access Estimation

NN accelerators use on-chip memory and reuse the data from it to minimize off-chip memory accesses. Figure 5 illustrates the impact of on-chip memory data reuse on off-chip memory accesses, and data access energy. Figure 5a shows the memory accesses and energy estimates when N bytes are directly accessed from the off-chip memory and Figure 5b shows the reduction in off-chip memory accesses when re-using the data from on-chip memory. If the energy per bytes access from the off-chip and on-chip memories are e_1 and e_2, respectively, the energy efficiency can be expressed as,

$$E_{efficiency} = 1 - \frac{E_2}{E_1} = 1 - (\frac{1}{n} + \frac{e_2}{e_1}) \tag{1}$$

For a given architecture, e_1 and e_2 are fixed and e_1 is significantly high compared to e_2. Energy efficiency mainly depends on data reuse and improves significantly with it. DNN accelerators use multiple levels of on-chip memories and techniques to maximize the data reuse from the lower memories to improve energy efficiency.

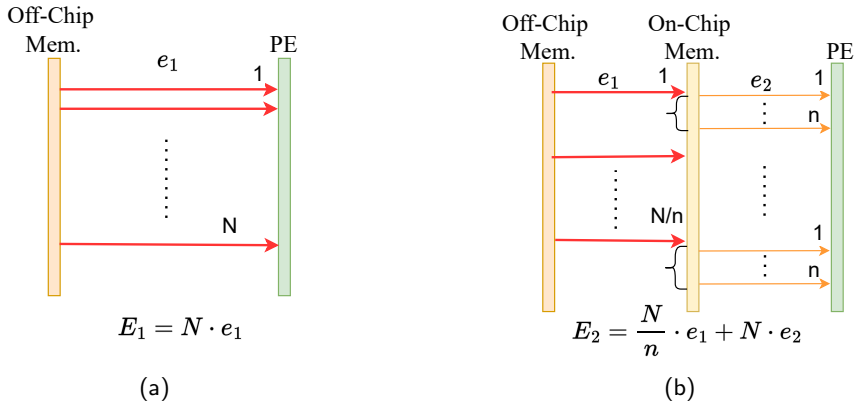


Figure 5: Memory accesses and energy estimates for accessing data (a) always from the off-chip memory. (b) from two-level of memory hierarchy while reusing the data from on-chip memory.

The NN accelerator reads the input data (or input activations), filter weights, and partial computations from the off-chip memory and stores them temporarily in the on-chip memory to perform the layer computations. The outputs of the computations are then finally stored in the off-chip memory, as shown in Figure 6a. The layer data is stored as multi-dimensional arrays in the off-chip memory, which is generally too large to fit in the local on-chip memory. 3D layer data (Figure 6b) is partitioned into small tiles (Figure 6c), and these tiles are fetched from the off-chip memory repeatedly to compute the final output sum. The tile dimensions and the order in which these tiles are processed significantly impact the volume of data reuse and, thus, the overall energy consumption and throughput of DNN accelerators.

Determining the optimal tile dimensions requires comparing the off-chip memory accesses for different tile dimensions and data reuse approaches. Modern DNNs have a variety of layers (convolution, fully connected, recurrent, pooling), each exhibiting different types of data access patterns. Even the layers of the same type differ in shape and size. Due to varying layer shapes and sizes, optimal partitioning and scheduling vary among layers. Finding the optimal partitioning and scheduling scheme by performing the measurements on the hardware is time-consuming, and vast search space makes it practically impossible.

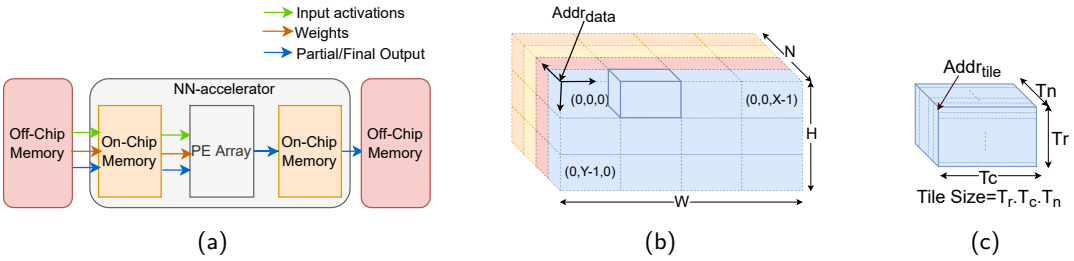


Figure 6: (a) Read/Write of inputs, weights and partial/final outputs from different level of memories. (b) 3D data partitioned into tiles. (c) A 3D tile.

To address this, we have developed an analytical framework that integrates models of NN layers to compute a layer's off-chip memory accesses and data access energy and the number of compute cycles for mapping a layer on a given PE array. Figure 7 shows the block diagram of the analytical framework. The framework is used as a design space exploration engine to find the optimal partitioning and scheduling scheme for a given layer shape to optimize NN accelerators' energy efficiency and throughput. The data

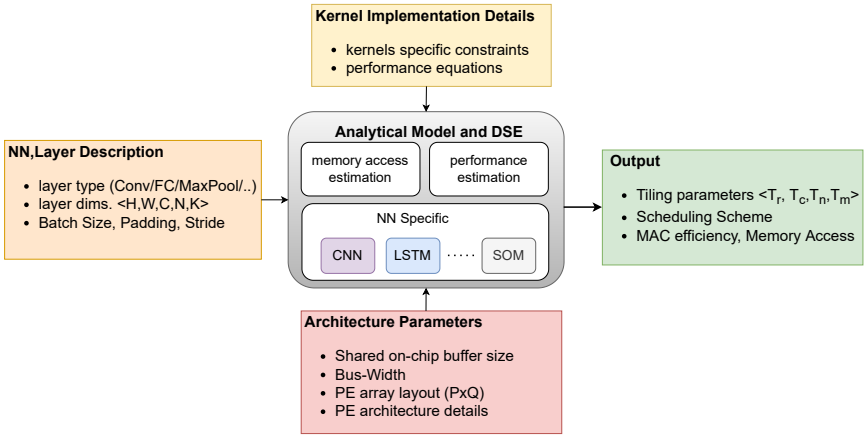


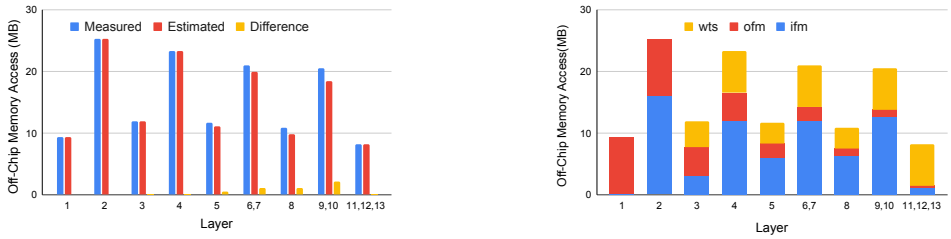
Figure 7: Analytical framework to estimate the performance, off-chip memory accesses and energy of DNNs.

shape and tile dimensions are $\langle W, H, N\rangle$ and $\langle T_c, T_r, T_n\rangle$, as shown in Figure 6b and Figure 6c, respectively. For a given data-reuse approach, all the tiles of a given 3D data type make the same number of trips from off-chip memory. The off-chip memory access of a partitioned 3D data can be computed as follows

$$\mathbb{B}_{3D} = \sum_{t=1}^{N_{tiles}} (\mathbb{B}_t \times r) = r \times \sum_{t=1}^{N_{tiles}} \mathbb{B}_t \tag{2}$$

where N_{tiles} is the number of tiles. r and $\mathbb{B}_t$ are the trips count and the number of bytes accessed from off-chip memory for the t^{th} tile, respectively. The analytical framework computes the off-chip memory accesses of a given 3D data using the above methodology while considering the data resolution and architectural constraint. For example, the limited size of the on-chip memory constrains the tile dimensions. In addition, the framework considers the bus width and data alignment to precisely compute the off-chip memory accesses. Analytical framework implements models for different layers and data reuse schemes to precisely compute the memory accesses and data access energy.

Figure 8a shows the comparison between the off-chip memory accesses estimated by the analytical framework and measurements performed on Xilinx FPGA for different layers of varying shapes and sizes of a popular CNN, VGG16. The experimental results on popular CNNs, AlexNet, and VGG16, show that the difference between estimated and measured off-chip memory accesses is less than 4%. The framework is also helpful in analyzing the layer-wise distribution and breakdown of memory accesses, as shown in Figure 8b. The proposed framework is a valuable tool for quickly analyzing the memory accesses and data access energy for different tile dimensions, and data reuse schemes to search for the optimal solution.



(a) (b)

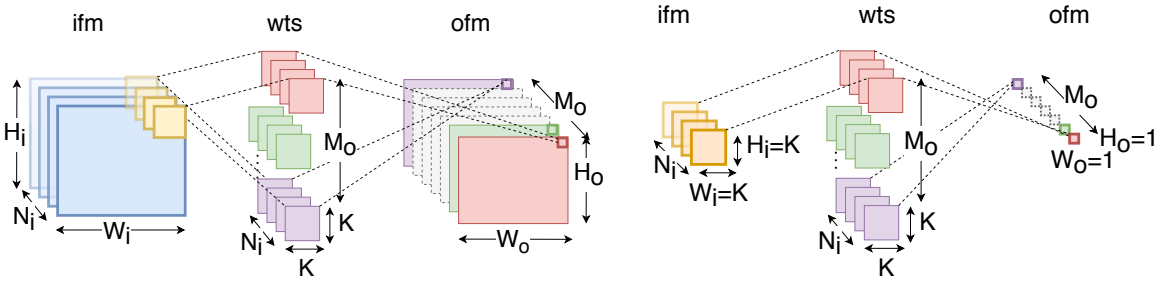
Figure 8: Off-chip memory accesss of VGG16 layers (a) Comparison between the analytical framework and measured on hardware. (b) Breakdown by data type using analytical framework.

3 Optimizing the Performance of CNN Accelerators

3.1 Introduction

CNNs are the state of the art machine learning algorithms. Modern CNNs can achieve human-like accuracy in computer vision-related tasks. In order to achieve high accuracy, CNNs perform compute-intensive and memory-intensive operations. CNN accelerators use many processing elements to exploit parallelism to speed up the computations. However, limited off-chip memory bandwidth limits their performance. In addition, considerable data transfer volume from the off-chip memory also results in high energy consumption.

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 9a and 9b, 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 9: Convolution and fully connected layers

To improve energy efficiency and throughput, it is critical to maximizing the data reuse from the on-chip memory. Due to limited on-chip memory size, CNN accelerators apply loop tiling to partition the layer data into small tiles that fit 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. 10 shows a layer's data stored in off-chip memory and its tiles in the accelerator's on-chip buffer. The 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) schemes, respectively. The scheduling scheme and the tile dimensions significantly impact the off-chip memory accesses [22, 35].

CNN's layers have varying shapes. First few layers have large volume of *ifm* and ofm and last few CLs

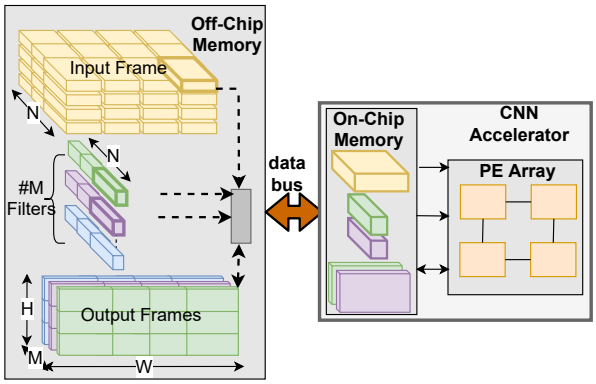


Figure 10: CNN layer tiles in off-chip and on-chip memory

and FCLs have large volume of *wts*. The scheduling scheme optimal for one layer may be suboptimal for other layers. In addition, memory accesses depend on the architectural parameters of the accelerator, like bus width and data alignment.

3.2 Related Work

Zhang et al. [35] 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 and 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. Due to varying layer shapes, the optimal tile dimension and data-reuse scheme for different layers vary. Li et al. [22] proposed a layer-wise adaptive data partitioning and scheduling scheme to overcome this. 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 this assumption, the tile dimensions determined by their approaches are suboptimal.

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 [5,7]. 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 11 illustrates memory accesses on a 64-bit data bus. Fig. 11a 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 11b, 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. 11. 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

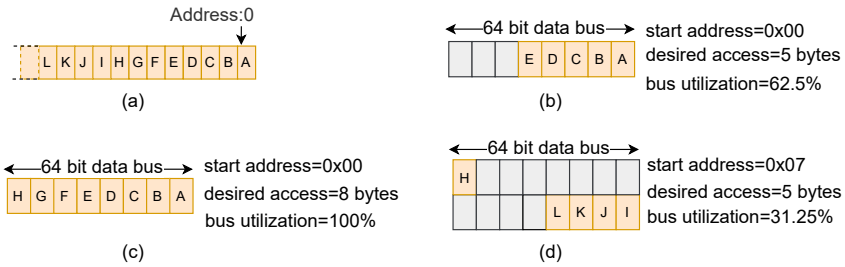


Figure 11: Off-chip memory accesses on 64-bit wide data bus

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 with 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

$$\mathbb{B}(Addr, l, BW) = (\lceil \frac{Addr + l}{BW} \rceil - \lfloor \frac{Addr}{BW} \rfloor) \cdot BW \tag{3}$$

CNN accelerators access layer data (ifm, ofm, and weights) partitioned into tiles. The tile dimensions should be chosen carefully to minimize the transfer of extra bytes to reduce off-chip memory access. 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 and determines the optimal tile dimensions.

3.4 Off-Chip Memory Accesses of CNN Layers

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. Limited on-chip memory size results in accessing the tiles multiple times from the off-chip memory. The trip count depends 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 (5), where columns represent ifm, ofm, and weights.

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_{iro} \\ \mathbf{r}_{oro} \\ \mathbf{r}_{wro} \end{bmatrix} = \begin{bmatrix} 1 & (2\lceil \frac{N}{T_n} \rceil - 1) & \lceil \frac{H}{T_r} \rceil \lceil \frac{W}{T_c} \rceil \\ \lceil \frac{M}{T_m} \rceil & 1 & \lceil \frac{H}{T_r} \rceil \lceil \frac{W}{T_c} \rceil \\ \lceil \frac{M}{T_m} \rceil & (2\lceil \frac{N}{T_n} \rceil - 1) & 1 \end{bmatrix} \tag{5}$$

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.

We compute the $\mathbb{B}_i$, $\mathbb{B}_o$ and $\mathbb{B}_w$ for each layer considering the architectural parameters using equation 3.

3.4.1 Constraints

To perform the computations, the tiles of *ifm*, *ofm* and *wts* are stored in the accelerator's on-chip memory. The on-chip memory size constraints the tile dimensions as shown in (6) below

$$\begin{aligned} &(V_i + V_o + V_w) \leq buffSize \\ &0 < T_{c_o} \leq W_o, \quad 0 < T_{r_o} \leq H_o \\ &0 < T_{n_i} \leq N_i, \quad 0 < T_{m_o} \leq M_o \end{aligned} \tag{6}$$

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

$$\begin{aligned} V_i &= T_{c_i} \cdot T_{r_i} \cdot T_{n_i} \cdot DW \\ V_o &= T_{c_o} \cdot T_{r_o} \cdot T_{m_o} \cdot DW \\ V_w &= K^2 \cdot T_{n_i} \cdot T_{m_o} \cdot DW \end{aligned} \tag{7}$$

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 (6) 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.

3.5 Design Space Exploration

Finding optimal tile dimensions requires analyzing off-chip memory accesses for different tiling parameters and data reuse schemes. Performing measurements on hardware are time-consuming, and the large design space of tile dimensions makes it practically impossible.

For a given CNN, the optimal tile dimensions and data reuse scheme need to be determined once. We have developed an architecture-aware off-chip memory access estimation model that computes the off-chip memory access of CLs and FCLs of a CNN layer while considering the architectural parameters (section 3.3). The model is integrated with the analytical framework, described in section 2 to explore the design space for searching for the optimal solution. The model takes layers description and architecture

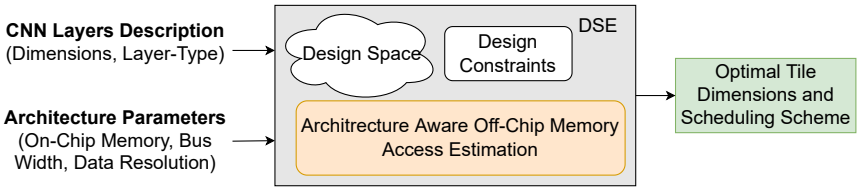
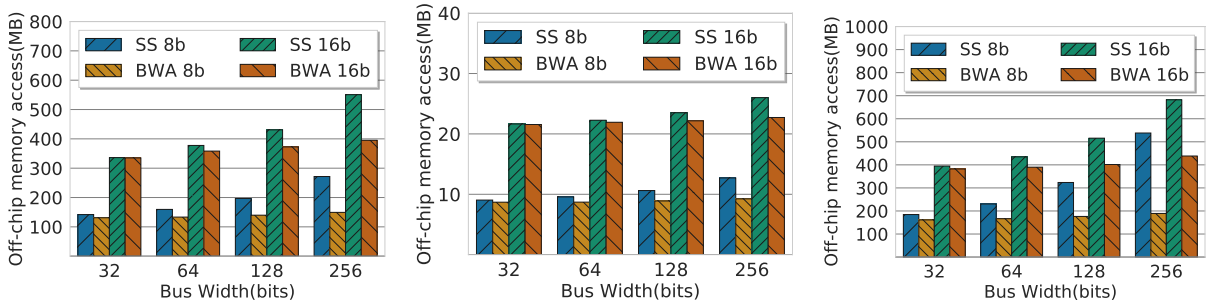


Figure 12: Architecture Aware design space exploration

parameters as input and analyzes the off-chip memory accesses ($\mathbb{B}$) for all feasible solutions that satisfy the constraints (Figure 12). It can be configured for different on-chip buffer sizes, data reuse schemes, bus width, and data resolution and determines each layer's optimal data reuse scheme and tile dimensions. The optimal solution determined by the framework is then used on the FPGA implementation of CNN to measure the energy and run-time.

3.6 Results

We experimented with three popular CNN networks, AlexNet [20], VGG16 [27], and ResNet [18] 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. We have compared our approach with the state-of-the-art Smart Shuttle (SS) approach [22]. Fig. 13 shows the number of bytes accessed from off-chip memory ($\mathbb{B}$) of CLs of the CNNs



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

Figure 13: Off-chip memory access of convolution layers for 8 and 16 bits data width. BWA: Bus Width Aware, SS: SmartShuttle

for different bus widths. The proposed BWA 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 a suboptimal solution. As shown in Fig. 13, our approach reduces $\mathbb{B}$ compared to SS for all three CNNs. For ResNet:50 it reduces $\mathbb{B}_{ResNet}$ by 13%, 28%, and 46% 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.

Figure 14a and Figure 14b 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

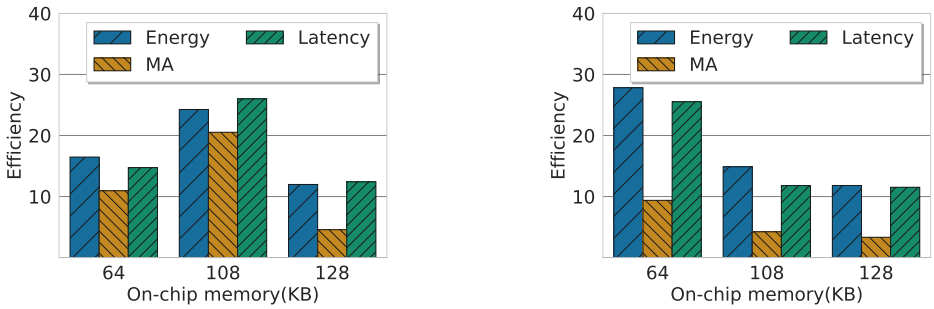


Figure 14: Energy and latency efficiency of BWA compared to SS. (a)VGG16, (b)AlexNet

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, and 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 [19] are variants of RNNs designed to handle long-range dependencies by storing useful information about previous inputs for a long duration.

LSTM computations involve several large matrix-vector multiplications, and these matrix-vector multiplications are performed for a large number of time steps. The inputs to the network are a time sequence of vectors, and these large matrices hold weights, which are learned during the training process. The size of these matrices can be significant in several MBs 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 computations of the LSTM cell are described by the following equations

$$\begin{aligned} i&=\sigma(W^i\cdot x_t+R^i\cdot h_{t-1}+b^i)\\ f&=\sigma(W^f\cdot x_t+R^f\cdot h_{t-1}+b^f)\\ g&=\tanh(W^g\cdot x_t+R^g\cdot h_{t-1}+b^g)\\ o&=\sigma(W^o\cdot x_t+R^o\cdot h_{t-1}+b^o)\\ c_t&=f\odot c_{t-1}+i\odot g\\ h_t&=o\odot\tanh(c_t) \end{aligned} \tag{8}$$

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 (8). 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.

To address the dependency of current time step computations on previous time step computations, we have proposed a Split And Combine Computations (SACC) approach. The proposed approach splits the LSTM cell computations in a way that reduces the off-chip memory accesses of LSTM hidden state matrices by 50%. In addition, the data reuse efficiency of the proposed approach is independent of on-chip memory size, making it more suitable for small on-chip memory LSTM accelerators.

4.2 Related Work

The matrix-vector multiplication $W^j \cdot x$ in Equation (8), where $j \in \{i, f, g, o\}$, is independent of previous state computation. Que et al. [26] proposed a blocking-batching scheme that reuses the weights of W^j matrix by processing a 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 the required number of input vectors. As the LSTM cell states (h_t and c_t) computations depend on previous time-step cell states, the 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.([25]) 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, and a fraction of it is reused to compute the partial sum of next time step state h_{t+1}. However, their approach does not fully exploit the data reuse, and several weights are accessed repeatedly from the off-chip memory. In addition, the data reuse in the 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

$$h_t[k] = F(S_t[k] + q_t[k]) \tag{9}$$

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,

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

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

$$S_t^L[k] = \sum_{n=0}^{k} R[k][n] \cdot h_{t-1}[n] \tag{11}$$

$$S_t^U[k] = \sum_{n=k+1}^{N-1} R[k][n] \cdot h_{t-1}[n] \tag{12}$$

Equation (11) uses the lower-diagonal and diagonal elements of R (R^L), and (12) uses the upper diagonal elements of R (R^U). As shown in Figure 15, R^L and R^U are accessed in consecutive time steps and reused

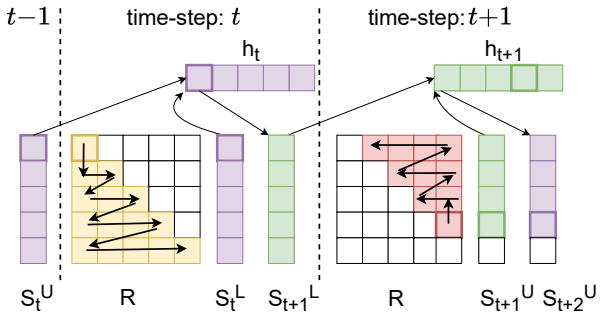
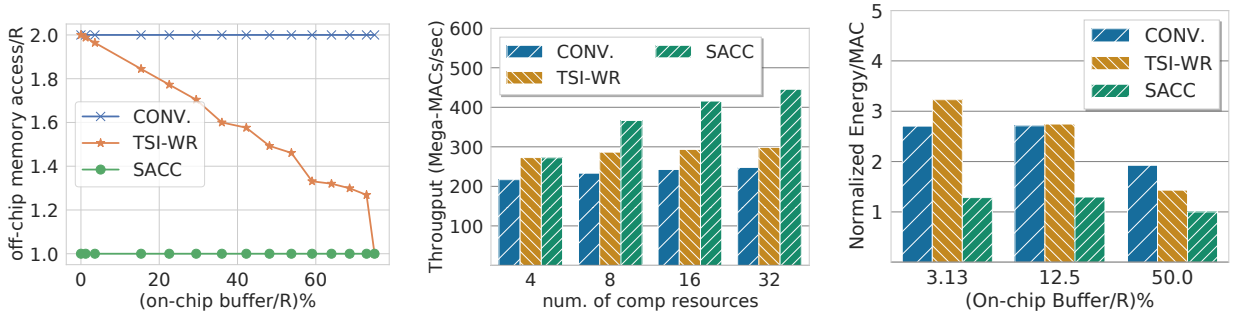


Figure 15: Splitting the hidden state vector computations into partial sums

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 15, 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 proposed SACC approach with conventional approaches and state-of-the-art TSI-WR approach [25]. 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 [12]) and character level Language Modelling (LM) [28].



(a) (b) (c)

Figure 16: (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 16a compares the off-chip memory accesses of the proposed (SACC), conventional, and 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%, the TSI-WR approach reduced 50% off-chip memory accesses compared to the conventional approach. However, for smaller on-chip buffer sizes, the reduction is less. The proposed approach reduces memory access of the 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 of the conventional approaches, as shown in Figure 16b. The TSI-WR approach improves the throughput by increasing the number of computing resources. However, throughput improvement in the TSI-WR approach is observed only for the large on-chip buffer to matrix size ratio. Figure 16b 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 by increasing the number of parallel resources.

4.4.3 Energy Efficiency

Figure 16c 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 three approaches. For smaller on-chip buffer to R matrix size ratios, the 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 the conventional and TSI-WR approaches, respectively.

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

5.1 Introduction

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 [37]. 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 [34]. The tolerance of imprecise computation in approximate computing to acquire substantial performance gains is the basis for a wide range of architectural innovations [10]. It has been demonstrated that high-precision computations are often unnecessary in the presence of statistical algorithms [24, 36]. Zhang et.al. [36] report less than 5% of quality loss obtained by simulation of the real hardware implemented in a 45nm CMOS technology.

Representing data with reduced bit-widths by trading off accuracy is one of the popular low-power strategy. The benefit of using reduced bit-width is improved energy performance. This is because there is a reduction in the energy cost consumption for data transfers, which usually dominates the total energy consumption for such systems.

Gupta et al. present results where they train deep networks with 16 bits fixed-point number representations and stochastic rounding [15]. 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 [23]. 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 representations is feasible with negligible degradation in performance [17].

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

5.2 Low bit-width FPGA Design of SOM

We have implemented SOM on FPGA, mapped on a Xilinx Virtex7 485t chip, for the 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 17a 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 nucleotides A, C, G, or T. Thus, a 16-bit word contains eight symbols. The Neural Network weights are stored in BRAMs. Each neuron has eight 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 17b.

5.3 Results

The SOM has been implemented with a range of fixed-point formats. With fewer bits, one naturally expects the SOM network 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 ten different bacteria DNA sequences. Each SOM network has 100 neurons inside, and each neuron has 20 weights.

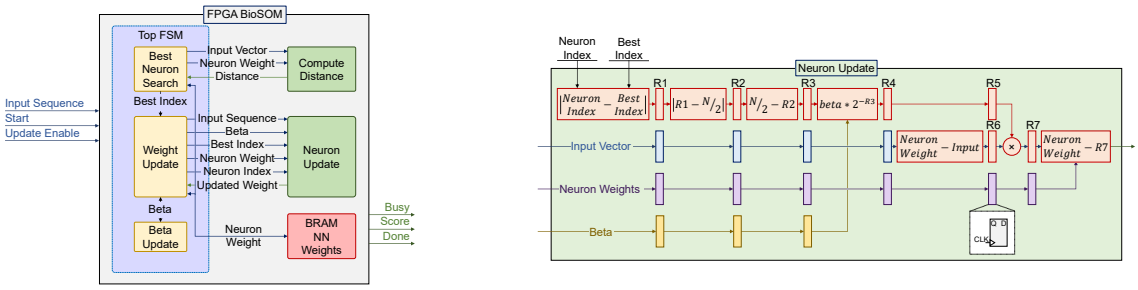


Figure 17: (a) Hardware Module for BioSOM. (b) Neuron Update Module.

We trained the networks with two independent training processes running in parallel. One is implemented using a double precision floating point, and the other is implemented with fixed-point weights. After training, we used the trained networks to identify the unknown 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 the 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 18a and 18b. Both the amount of utilized LUTs and total energy in Joule are 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 to improve the performance and energy efficiency of DNN accelerators for their ubiquitous usage in energy-constrained

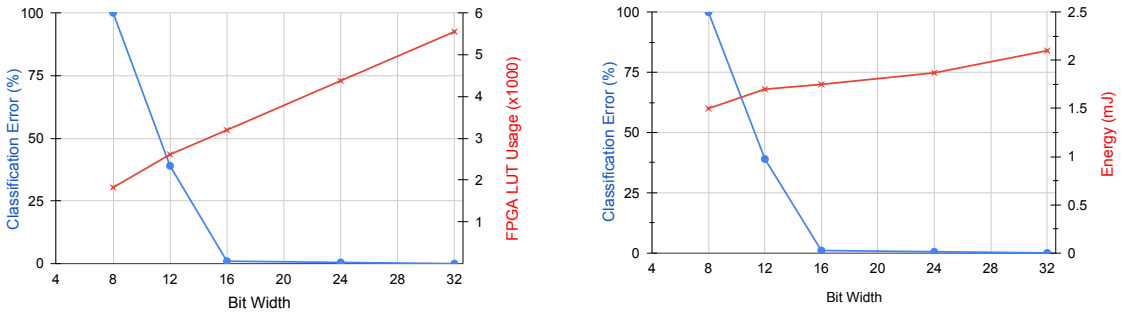


Figure 18: comparison between different fixed-point fromat (a) FPGA LUT utilization (b) energy.

devices. The performance of DNN accelerators is limited by the memory bandwidth, and off-chip memory accesses dominate the energy consumption. The key to improving the energy efficiency of these NNs 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 improve the data reuse from on-chip memories for CNNs and LSTMs by partitioning the data and scheduling the operations. We have also analyzed the impact of low-bit resolution on the accuracy and energy area performance of NNs.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] 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.

[23] D. D. Lin and S. S. Talathi. Overcoming challenges in fixed point training of deep convolutional networks. *arXiv preprint arXiv:1607.02241*, 2016.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[30] 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.

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

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.