Spring 5-2018

Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators

Liu Ke
Washington University in St. Louis

Washington University in St. Louis

School of Engineering and Applied Science

Department of Electrical and System Engineering

Thesis Examination Committee:
Xuan Zhang, Chair
Roger Chamberlain
Shantanu Chakrabartty

Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators

by

Liu Ke

A thesis presented to the School of Engineering and Applied Science
of Washington University in partial fulfillment of the
requirements for the degree of

Master of Science

May 2018
Saint Louis, Missouri

Contents

List of Tables

List of Figures

Acknowledgments

I would first like to thank my thesis advisor Prof.Zhang. Whenever I ran into a trouble spot or had a question about my research or writing, she always leads me in the right the direction.

I would also like to thank the Prof.Chamberlain and Prof.Charkrabartty who were willing to be my committee member.

I would also like to acknowledge my teammates in the lab, and I am gratefully indebted to them for their very valuable comments and technical help on this thesis.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Liu Ke

Washington University in Saint Louis
May 2018

Dedicated to my parents.

ABSTRACT OF THE THESIS

Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators

by

Liu Ke

Master of Science in Electrical Engineering

Washington University in St. Louis, May 2018

Research Advisor: Prof. Xuan Zhang

Deep neural networks (DNNs) have achieved spectacular success in recent years. In response to DNN's enormous computation demand and extensive memory footprint, numerous inference accelerators have been proposed. However, the diverse nature of DNNs, both at the algorithm level and the parallelization level, makes it difficult to arrive at an "one-size-fits-all" hardware implementation. In this dissertation, we develop NNest, an early-stage design space exploration tool that can speedily and accurately estimate the area/performance/energy of DNN inference accelerators based on high-level network topology and architecture traits, without the need for low-level RTL codes. Equipped with a generalized spatial architecture framework, NNest is able to perform fast high-dimensional design space exploration across a wide spectrum of architectural/microarchitectural parameters. Our proposed novel date movement strategies and multi-layer fitting schemes allow NNest to more effectively exploit parallelism inherent in DNN. Results generated by NNest demonstrate: 1) previously-undiscovered accelerator design points that can outperform state-of-the-art implementation

by 39.3% in energy efficiency; 2) Pareto frontier curves that comprehensively and quantitatively reveal the multi-objective tradeoffs in custom DNN accelerators; 3) holistic design exploration of different level of quantization techniques including recently-proposed binary neural network (BNN).

Chapter 1

Introduction

Since the groundbreaking performance of AlexNet in 2012 ImageNet competition [1], a class of machine learning methods known as deep neural network (DNNs) have achieved spectacular success, especially in applications such as computer vision, natural language processing, and machine translation. These hierarchical network models typically employ tens or even hundreds of connected neural network layers and incur enormous computational demands and memory footprints, rendering their execution on conventional CPU-based computing platforms quite inefficient. Despite the complexity, DNN exhibit vast amount of inherent parallelism in its computational model, which can be exploited to accelerate DNN computation. For example, extensive toolchain and framework have been built on general-purpose graphics processing units (GPGPU) to leverage its superior parallel processing capability for deep learning tasks [1]; field-programmable gate array (FPGA) based systems [4] have attracted much attentions thanks to their programmable fabrics that can accommodate flexible parallel processing primitives and adapt to rapidly evolving algorithms; finally custom machine learning accelerators such as Google's tensor processing unit (TPU) have also been making great strides, pushing the envelop of theoretical computation throughput to the range of tens of tera floating point operation per second (TFLOPs).

However, computational speed/throughput is not the only metric that matters. Power and cost are among the top concerns for DNN hardware accelerators, especially when designed for neural network inference tasks to be deployed in mobile or embedded edge devices [2]. Compared to previous cloud-centric platforms using GPUs, FPGAs, and TPUs, edge devices have much more stringent power budget and sensitive cost consideration, which makes application-specific integrated circuits (ASIC) based solution a more appealing choice [13]

for targeted implementation of specific pre-trained networks. To address this need, numerous NN inference accelerators have been proposed [18, 22], but the diverse nature of DNNs, both at the network level and the parallelization strategy level, make it difficult to arrive at an "one-size-fits-all" implementation. The complex and high-dimensional design space of DNN accelerators calls for an early-stage exploration tool that can speedily and accurately traverse the available design points and estimate their performance. Such a tool could benefit a multitude of use cases. For example, architects of DNN accelerator can be better informed of the tradeoffs between different performance metrics under distinctive parallelization strategies; circuit designers and device engineers can get an early glimpse of how device/circuit level innovation can affect the overall system performance; algorithm developers can more deeply understand the potential advantages/penalties of their model parameters and algorithmic techniques based on custom ASIC accelerators, without being limited by the specific implementation of commodity hardware.

In this paper, we present NNest—an early-stage tool that is designed to facilitate systematic design space exploration for ASIC-based DNN inference accelerators. More specifically, our main innovation and contributions include:

- We propose a spatial accelerator architecture template that can be generalized to cover a variety of DNN accelerator implementations, sufficiently capturing important design tradeoffs without the need for detailed RTL codes.
- We develop parameterized data movement strategies and multi-layer fitting schemes that can efficiently express the inherent parallelism in DNN algorithm for both the convolutional and the fully-connected layers.
- Results generated by NNest not only enable quantitative investigation of impact from memory hierarchy, data reuse, and energy/area breakdown, but also reveal previously-unknown design point with 39.3% higher energy efficiency.
- NNest facilitate holistic evaluation and comparison of DNN models and algorithmic techniques with software/hardware codesign consideration. Examples on AlexNet vs VGG models and binarized quantization are given to demonstrate such capability.

Chapter 2

Background and Related Work

2.1 Preliminary on DNN

Neural network based deep learning models typically consists of cascading of different layers, including convolution, normalization, pooling, and fully-connected layers. Since the Conv and FC layers tend to dominate computation and memory access, we focuses on exploring the design space of these two kinds of layers in NNest. While sharing composition of similar types of layers, different DNN models usually employ distinctive layer shapes and sizes, making it difficult to find a fixed hardware configuration that is optimized for all layers and NN models.

2.1.1 Fully-connected (FC) layer

The computational pattern (Fig. 2.2) of FC layer has three dimensions: input batch size (N), the number of input neurons (I), and the number of output neurons (O). Each output neuron has connections to all the neurons in the input layer. It takes multiple input vectors ($N \times I$) and multiplies them with a weight matrix ($I \times O$) to get the output vectors ($N \times O$). Computation of each output involves I element-wise multiplication and accumulation of all the products to reduce to one activation of the output vector.

There exist three types of data reuse opportunities in a FC layer as shown in Fig. 2.2. 1) *Input Reuse*: the same input vector is reused for different columns in the weight matrix to calculate one output vector. 2) *Weight Reuse*: the same column of weight matrix is reused

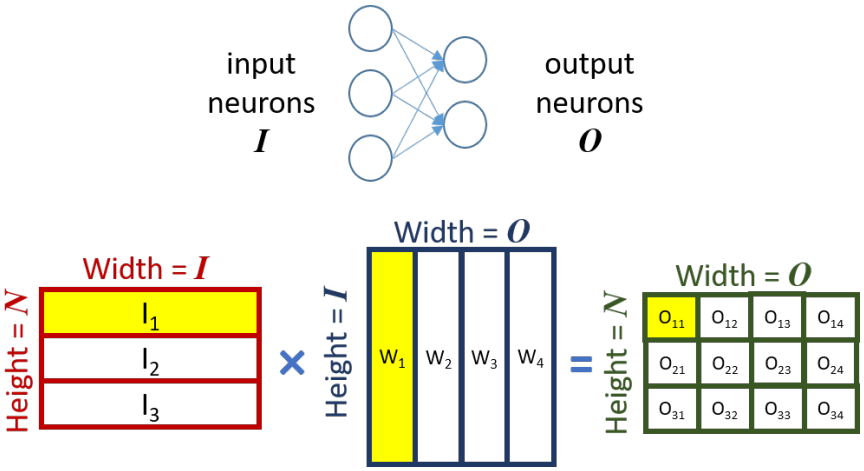

Figure 2.1: Computation pattern of FC layers.

for several input vectors to calculate different output vectors. 3) *Partial Sum Reuse*: the old partial sum (PSum) is reused to get the updated PSum by accumulating with the new element-wise products.

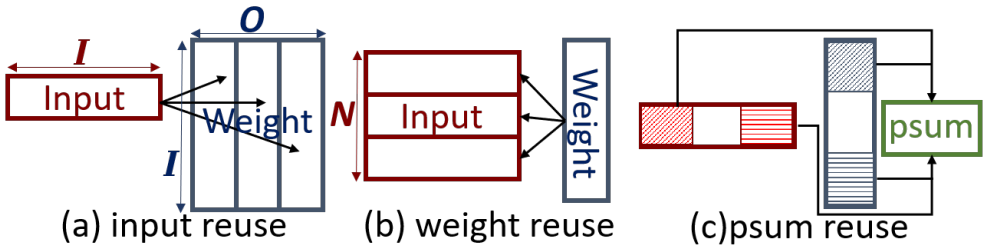

Figure 2.2: Data reuse patterns in FC layers.

2.1.2 Convolutional (Conv) layer

In a Conv layer, to extract features from input feature maps (ifmaps), each output neuron is only connected to a local region of ifmaps, known as the local connectivity property of convolution, and the weight matrix to represent the local connectivity is called a filter and has the same size as the local ifmap region. Conv layer's computational pattern can be regarded as a $C \times R \times S$ sliding window (SW) (Fig. 2.7) shifting on ifmaps. The ifmap data in one SW is element-wise multiplied with a $C \times R \times S$ filter before accumulation. Each SW shift on

ifmaps (from SW_1 to SW_2) generates the next output value. As SW moves, data overlapped between the consecutive windows (e.g. SW_1 and SW_2) is of the shape $(S-U)\times R\times C$. Therefore, only $U\times R\times C$ new input data are needed to get a new output. In total, one SW shifts F steps vertically and E steps horizontally to get $E\times F$ output values. The number of 3D filters (M) corresponds to the number of channels in output feature maps (ofmaps).

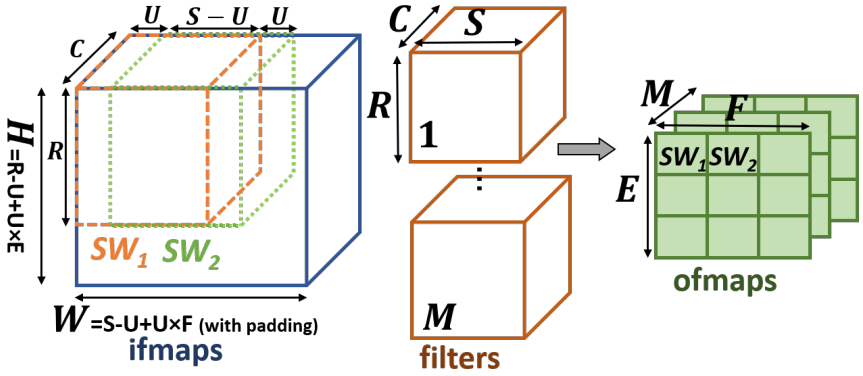

Figure 2.3: Data reuse patterns in Conv layers.

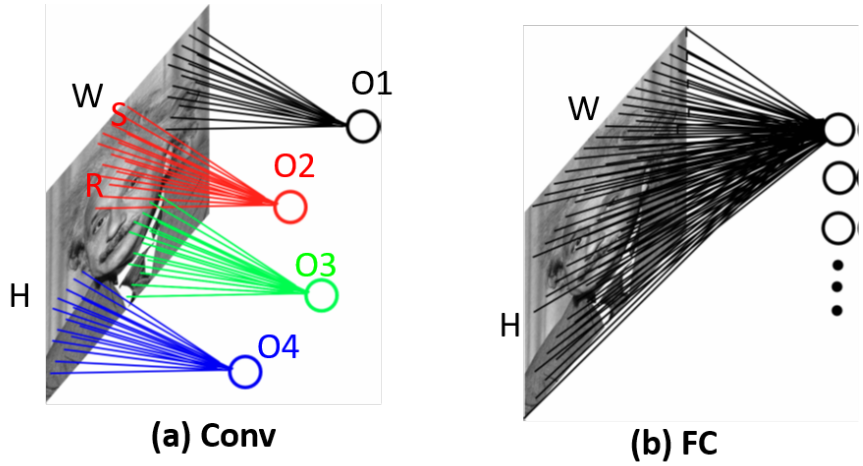

Figure 2.4: Properties of Conv layers.

Contrasted with FC layers, Conv is weight-shared due to local connectivity (Fig. 2.4), because the all-to-all connectivity ($C\times H\times W$) is reduced to a local region ($C\times R\times S$). One channel of output activations ($E\times F$) also share the same 3D filter. Hence, there is additional reuse opportunity in Conv layer, which is referred to as *sliding window reuse* in this paper. It

takes two forms: 1) the same 3D filter is reused over $E \times F$ SWs; 2) the overlap between two SWs can be reused, only the new stride ($U \times R \times C$) needs loading.

Recently, many DNN models have been proposed such as LeNet[30], AlexNet[1], VGG[21], ResNet[10] and GoogLeNet[3]. We specifically list the network structure of Alexnet (Fig. 2.5, Table. 2.1) and VGG (Fig. 2.6, Table. 2.2)

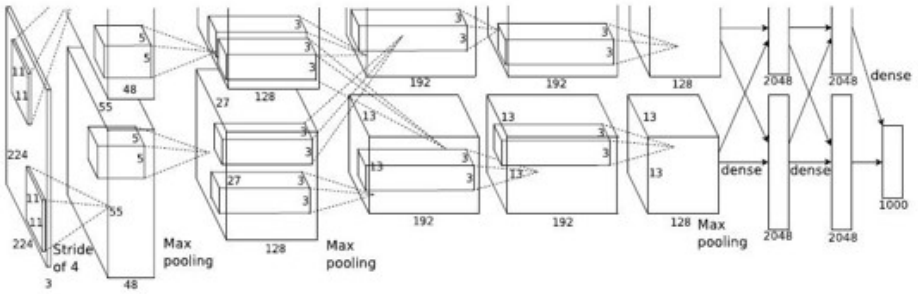

Figure 2.5: Alexnet structure [1]

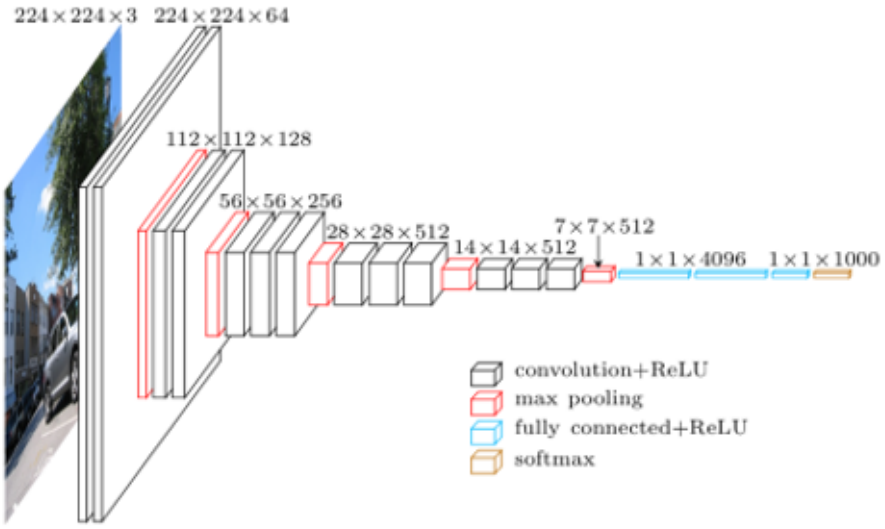

Figure 2.6: VGG structure [21]

Table 2.1: Alexnet layers' shape parameters [1]

Layer	Conv-1	Conv-2	Conv-3	Conv-4	Conv-5
H, W	227×227	31×31	15×15	15×15	15×15
C	3	48	256	192	192
M	96	256	384	384	256
R, S	11×11	5×5	3×3	3×3	3×3
U	4	1	1	1	1
E, F	55×55	27×27	13×13	13×13	13×13
Input size	$151K$	$45K$	$56.25K$	$42.2K$	$42.2K$
Weight size	$34K$	$300K$	$864K$	$648K$	$432K$
Output size	$283.6K$	$182.25K$	$63.4K$	$63.4K$	$42.25K$
# of MAC	$100.53M$	$213.57M$	$142.59M$	$106.95M$	$71.3M$

Layer	FC-1	FC-2	FC-3
I	43264	4096	4096
O	4096	4096	1000
Input size	$42.25K$	$4K$	$4K$
Weight size	$169M$	$16M$	$4M$
Output size	$4K$	$4K$	1000
# of MAC	$169M$	$16M$	$4M$

Table 2.2: VGG layers' shape parameters [21]

Layer	Conv1-1	Conv1-2	Conv2-1	Conv2-2	Conv3-1	Conv3-2	Conv3-3
H, W	224×224	224×224	112×112	112×112	56×56	56×56	56×56
C	3	64	64	128	128	256	256
M	64	64	128	128	256	256	256
R, S	3×3	3×3	3×3	3×3	3×3	3×3	3×3
U	1	1	1	1	1	1	1
E, F	224×224	224×224	112×112	112×112	56×56	56×56	56×56
Input size	$151K$	$3211.3K$	$802.8K$	$1605.6K$	$401.4K$	$802.8K$	$802.8K$
Weight size	$1.728K$	$36.9K$	$73.7K$	$147.5K$	$294.9K$	$589.8K$	$589.8K$
Output size	$3211.3K$	$3211.3K$	$1605.6K$	$1605.6K$	$802.8K$	$802.8K$	$802.8K$
# of MAC	$86.7M$	$1849.7M$	$924.8M$	$1849.7M$	$924.8M$	$1849.7M$	$1849.7M$

Layer	Conv4-1	Conv4-2	Conv4-3	Conv5-1	Conv5-2	Conv5-3
H, W	28×28	28×28	28×28	14×14	14×14	14×14
C	256	512	512	512	512	512
M	512	512	512	512	512	512
R, S	3×3	3×3	3×3	3×3	3×3	3×3
U	1	1	1	1	1	1
E, F	28×28	28×28	28×28	14×14	14×14	14×14
Input size	$200.7K$	$4014K$	$401.4K$	$100.35K$	$100.35K$	$100.35K$
Weight size	$1179.6K$	$2359.3K$	$2359.3K$	$2359.3K$	$2359.3K$	$2359.3K$
Output size	$401.4K$	$401.4K$	$401.4K$	$100.35K$	$100.35K$	$100.35K$
# of MAC	$924.84M$	$1849.7M$	$1849.7M$	$462.42M$	$462.42M$	$462.42M$

Layer	FC-1	FC-2	FC-3
I	25088	4096	4096
O	4096	4096	1000
Input size	$24.5K$	$4K$	$4K$
Weight size	$98M$	$16M$	$4M$
Output size	$4K$	$4K$	1000
# of MAC	$98M$	$16M$	$4M$

2.2 Related Work

2.2.1 Existing NN Accelerators

A recent tutorial have extensively surveyed prior work on NN inference accelerators and categorized them based on dataflow [27].

1) *No Local Reuse (NLR)* represents designs which do not allocate local storage in the form of register files or registers to each MAC unit [4, 23, 28]. Hence, all MACs share a global buffer (GB) to load inputs and weights and store intermediate PSums.

Different with NLR, many works insert one smaller sized memory, which can be faster and more efficiently accessed by computational blocks, called local buffer. So this multi-layer memeory architecture is called memory hierarchy (MH) design. It keeps data stationary and reuse it for the next computational cycle. Based on the stationary data in the LB, MH can be divided to several types. 2) *Weight Stationary (WS)* refers to designs that employ local storage for weight reuse to minimize the energy consumption of frequent weight fetching for different ifmaps [13, 26, 17, 25, 20]. The input and PSums remain stored in GB. 3) *Output Stationary (OS)* stores PSums locally [18, 33, 12], rather than weights. Similarly, the goal is to minimize the energy for fetching old psum and saving back the updated one. 4) *Row Stationary (RS)* is proposed to locally store weight, input and PSum to increase data reuse opportunities [29]. Our framework encompasses all these previously-proposed architectures.

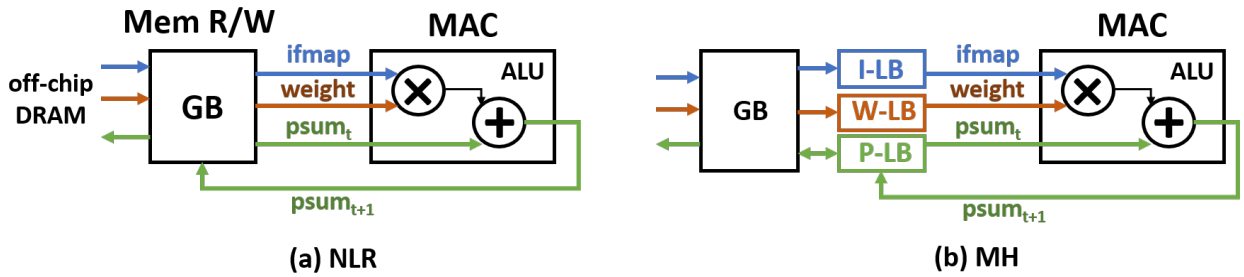

Figure 2.7: NLR, MH architecture

2.2.2 NN Design Automation

A number of design automation tools have been introduced [6, 5, 32] that focus on NN acceleration on FPGAs. These tools takes specific DNN models as input and automatically generate implementation for compatible FPGAs. Since they are designed for a fixed hardware platform, the optimization emphasizes maximum utility of on-chip resources and highest throughput for a given FPGA platform. Prior work has proposed design space exploration algorithm to study FPGA-based deep convolutional NN [15]. However, it does not explicitly address FC layers and multi-level memory hierarchy, and provides no exploration results for die area and power consumption, thus unsuitable for ASICs.

In the custom ASIC space, tools have been developed to explore acceleration for general computational kernels [31] that can be applied to limited NN design space [2]. Method to roughly estimate DNN energy consumption has been proposed to guide architecture selection [24] without providing comprehensive area/power/performance tradeoffs or a generalized architecture framework to systematically evaluate different parallelization/reuse strategies. Finally, a design tool has been introduced specifically for binary neural network (BNN) [14] to perform area/performance/energy estimation and analysis, but it does not readily apply to investigate the much broader design space of general NN accelerator with different quantization schemes.

Chapter 3

Methodology

3.1 Generalized Architecture Framework

In order to conduct effective and thorough design space exploration, we first propose a spatial NN accelerator architecture framework that can be generalized to produce numerous design points. Our generalized architecture framework is illustrated in Fig. 3.1. It has drawn inspirations from many existing accelerator prototypes and is able to encompass all previously-explored architectures with different dataflow schemes including NLR, WS, OS, and RS.

In our accelerator framework, the on-chip components include various partitioned global buffer (GB), local buffer (LB), communication network in the form of 1-to-n and 1-to-1 broadcast buses (BBus), and a 2D array of arithmetic units (ALU). A large-sized on-chip memory modeled as SRAM, GB stores input and weight data fetched from external DRAM, holds the intermediate psums during computation, and writes the final output activations back to DRAM. LB represents the smaller-sized memory located in between GB and the ALU array that can be accessed by the ALU faster and more efficiently. Depending on the data movement strategies (described in Section 3.2), LB stores certain stationary data locally for later reuse without accessing GB. More specifically, I-LB and W-LB broadcast input (I) and weight (W) data to the ALU array to reuse inputs and weights. Each ALU unit receives the broadcast I/W data and performs element-wise multiplies and one accumulation with the $PSum_t$ loaded from P-LB to get the updated $PSum_{t+1}$ and store back to P-LB. In this way, PSums in P-LB can be reused without being stored back to GB until all the

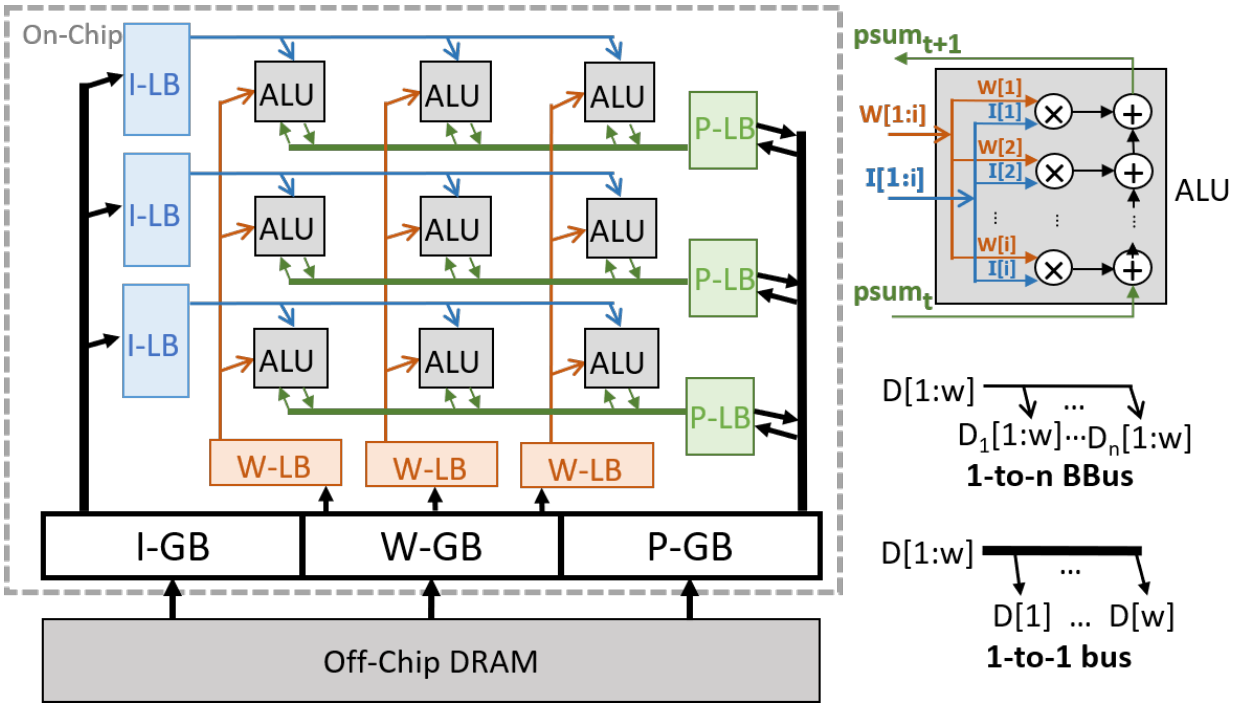

Figure 3.1: The proposed generalized accelerator framework based on spatial architecture.

computation for the same activation is complete. Our architecture allows the LBs to be set to zero (in case of NLR) or partially bypassed (in case of WS and OS).

Intuitively, it is ideal to hold all data in GB to avoid multiple energy-expensive external DRAM access for the same data, if there is no area constraint. However, the amount of I/W/P data can be quite large even for a single layer, and varies over a wide range across different NN layers. Considering the cost of large on-chip memory, the size of GB is limited in practical systems by holding only a *tile* of data at a time. It is clear that GB size also effects data reuse efficiencies in the LB and the ALU array. Therefore, we must determine optimal strategies to move and replace data between and across the partitioned GB and LB blocks in order to minimize DRAM access and achieve highest degree of data reuse, which is discussed next.

3.2 Data Movement Strategy (DMS)

It is increasingly the case in advanced technology node that data movement incurs considerably higher cost in latency and energy as compared to computation [22]. Since DNN algorithms are memory intensive, in order to achieve higher performance and energy efficiency, it is imperative to carefully formulate the data movement strategies (DMS) to optimally exploit data reuse opportunities existed in the parallel processing of NNs. Before getting into detailed discussions on dataflow in the FC and Conv layers, please note that in both cases maximal data reuse is highly desirable and is achieved by 1) broadcasting input/weight data to multiple ALUs for parallel processing and 2) keeping data stationary in on-chip storage. In this section, we derive the microarchitecture (μarch) parameters used in NNest to express different data movement strategies. In this way, we are able to effectively traverse the broad NN accelerator design space by sweeping these μarch parameters (μaPMs) values listed in Table 3.1.

Table 3.1: List of NNest μarch Parameters

	Parameter Name	
	FC Layer	Conv Layer
comp μaPM	*tti*, *tto*	*ttm*, *tte*, *ttc*, *ttr*, *tts*
mem μaPM	T_i, T_o	T_h, T_m, t_e, t_m, t_c

3.2.1 Dataflow in FC layers

Due to the large amount of weights ($I \times O$) in FC layers, hold them all in GB is impractical. Instead, we assume only a tile ($T_i \times T_o$) of weight is stored on-chip, which determines the GB size. The batch size N is used to represent the number of inputs being processed concurrently, allowing for weight reuse in a broadcasting manner. It in turn determines the size of the ALU array, since the number of ALUs, as well as the number of updated PSums per cycle, equals to $N \times tto$. There are *tti* parallel MAC operations in each ALU, resulting in *tti* multipliers and an adder tree to sum up the *tti* products. Since tto, tti used to size the ALU array, we refer to them as as *comp* μaPMs, and others that determine the size of GB and LB as *mem* μaPMs. For FC layers, we study two strategies that leverage distinctive data

reuse. For each strategy, data movement at both the GB and the LB levels are described in details.

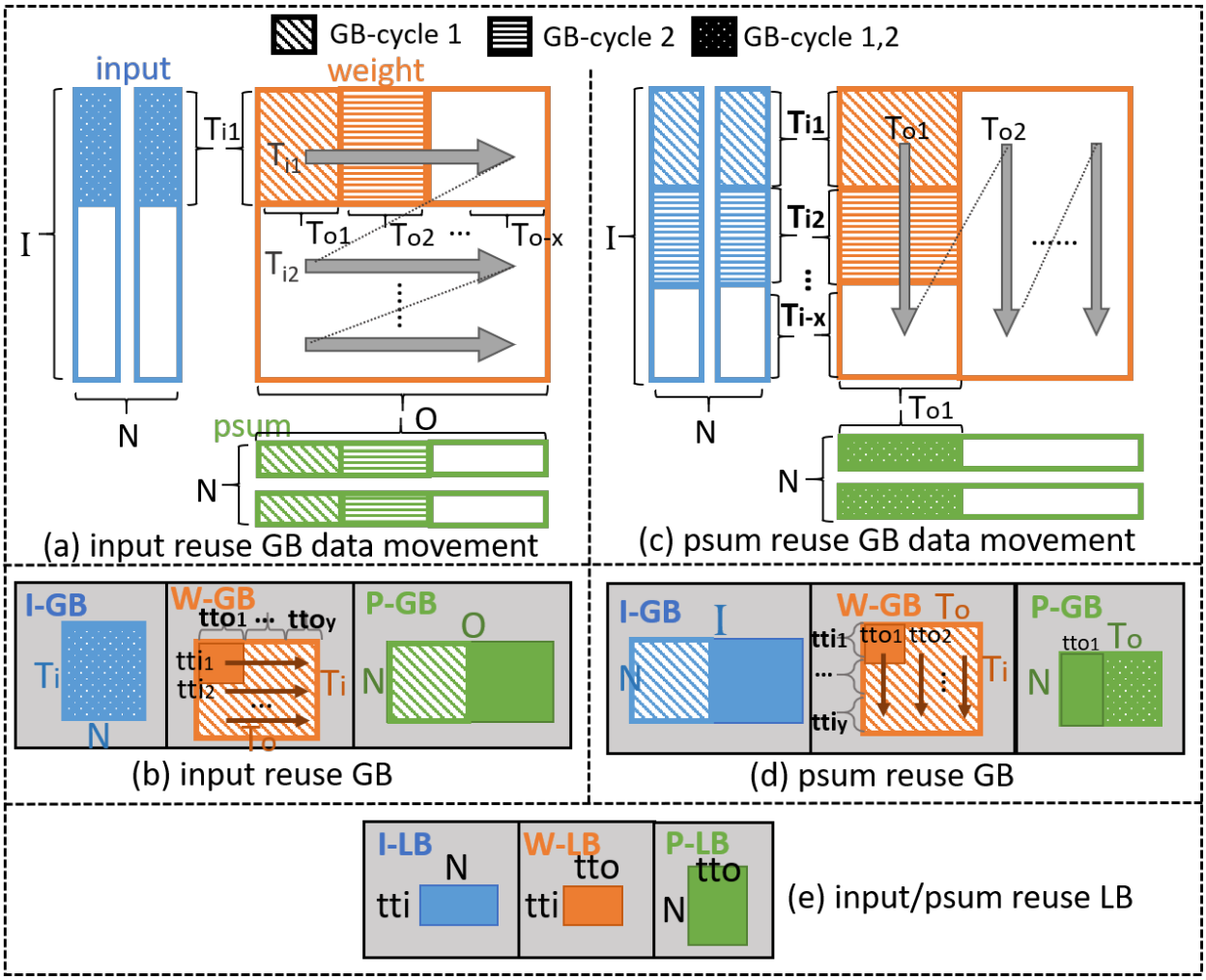

Figure 3.2: Illustration of different DMS in FC layers

Input Reuse (IR): As illustrate in Fig. 3.2(a), at the GB level, $N \times T_{i1}$ inputs are kept stationary in I-GB. The sub-index denotes the range of fetched data, $T_{ix} = [T_i \times (x-1)+1 : T_i \times x]$, and the same convention applies to other notations in the paper. In IR, Weights are replaced in O-dimension, $T_{i1} \times (T_{o1} \rightarrow T_{o2})$, to reuse the $N \times T_{i1}$ input data. After fetching all O columns of weight matrix with T_{i1} rows, $N \times O$ PSums are computed. Both inputs and weights are replaced in I-dimension, $N \times (T_{i1} \rightarrow T_{i2})$ (input) , $T_{i1} \times T_{ox} \rightarrow T_{i2} \times T_{o1}$ (weight, $x = O/T_o$), to update previous $N \times O$ PSums. To avoid frequent DRAM access, P-GB is sized to $N \times O$ as in Fig. 3.2(b) and I-GB and W-GB are sized according to mem μaPMs

(T_i, T_o). At LB level, to satisfy the processing speed of the ALU array, the size of stationary data in LB is defined by the *comp* μaPMs (tti, tto) and sized accordingly. As shown in Fig. 3.2(b)(e), to reuse inputs, $N \times tti_1$ input data are kept stationary in I-LB, the weights are replaced in T_o-dimension $tti_1 \times (tto_1 \rightarrow tto_2)$. After T_o/tto cycles, all the T_o columns with tti_1 rows of weight data in W-GB are processed. Then inputs and weights in LB are replaced in T_i-dimension to update Psums.

PSum Reuse (PR): The derivation of dataflow in PR can be done in a similar manner, and is illustrated in Fig. 3.2(c)-(e). Here, at the GB level, PSums are kept stationary in P-GB, and inputs and weights are replaced in I-dimension, $N \times (T_{i1} \rightarrow T_{i2})$ (input), $T_{o1} \times (T_{i1} \rightarrow T_{i2})$ (weight). After processing I rows of input data and weight matrix with T_{o1} columns, $N \times T_{o1}$ PSums in P-GB are finalized. Then weights are replaced in O-dimension, $T_{ix} \times T_{o1} \rightarrow T_{i1} \times T_{o2}$ (weight, $x = I/T_i$) to compute the next $N \times T_{o2}$ output, and all the input data are fetched again from $N \times (T_{i1} \rightarrow T_{ix})$. At the LB level,PSums are stationary in P-LB, and weights and inputs are replaced in T_i-dimension $N \times (tti_1 \rightarrow tti_2)$ (input), $tto_1 \times (tti_1 \rightarrow tti_2)$ (weight). After T_i/tti cycles, all the T_i rows of N input vectors and tto_1 weight columns are processed and $N \times tto_1$ PSums are written back to GB.

In addition to GB and LB sizes, DMS also determines the BBus configuration based on the broadcasting requirements between GB and LB (defined by the *mem* μaPMs) and between LB and ALU array (defined by the *comp* μaPMs). For example, $tto \times (tti)$ weights in W-LB and $N \times (tti)$ inputs in I-LB broadcast to N-by-tto ALU array, therefore, BBus's configuration is 1-to-N from W-LB to ALU,and 1-to-tto from I-LB to ALU.

3.2.2 Dataflow in Conv layers

As discussed earlier, the same IR and PR opportunities naturally exist in Conv layers, with additional possibilities from *sliding window reuse*. We briefly describe the two Conv strategies below:

PSum Reuse (PR) All data in C-dimension of inputs ($T_h \times C \times W$) and weights ($T_m \times C \times R \times S$) are stored in GB. To reuse the SW overlap, SW first shifts on inputs along W-dimension for F steps, then moves to the next t_c channels and shifts along the W-dimension

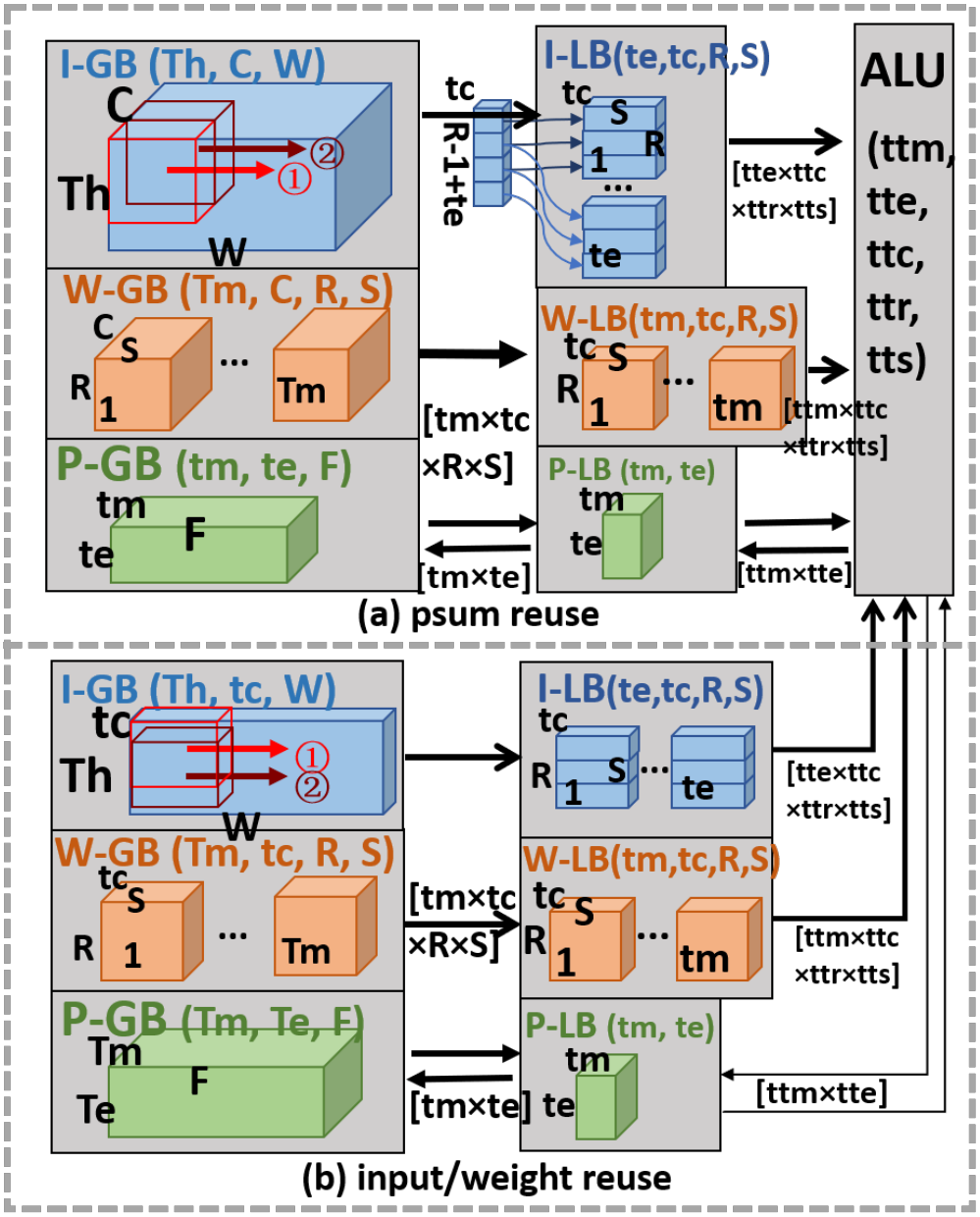

Figure 3.3: Illustration of different DMS in Conv

again to update Psums computed in the last F steps. Input SW size is defined by t_e and t_c, and determines I-LB size as $t_e \times t_c \times R \times S$. The corresponding filter window is defined as $t_m \times t_c \times R \times S$. In the F steps of SW shifting, t_m 3D filters are kept stationary in W-LB and reused to get $tm \times te \times F$ PSums. After shifted through all C channels, $t_e \times t_m \times F$ PSums are finalized, setting P-GB size as $t_e \times t_m \times F$. This process is graphically illustrated in Fig. 3.3(a). In this DMS, P-GB size can be small to store a limited number of intermediate PSums, but I-GB and W-GB should have sufficient space to store C channels of inputs and weights.

Input/Weight Reuse (IWR): To increase input and weight reuse, the parameters that define the number of SWs should be large. Due to limited GB size, only t_c input channels are held in GB, and t_c is usually small to accommodate large T_h, T_m. As shown in Fig. 3.3(b), SW first shifts along the W-dimension and then moves to the next t_e input rows and shifts along W-dimension. After shifting through all T_h input rows in I-GB, $t_m \times T_e \times F$ PSums are generated with the t_m 3D filters kept stationary in W-LB. The next step replaces W-LB with the next t_m weights and SW is shifted along the entire T_h rows of input data. After processing all T_m 3D filters in W-GB, $T_m \times T_e \times F$ PSums are generated, then both inputs and filters are replaced to the next t_c channels. In this DMS, all of the intermediate $T_m \times T_e \times F$ results are Psums to be held in P-GB. Therefore P-GB accounts for the majority of GB size. The IWR strategy described here is similar to the row stationary method introduced in Eyeriss, where $80\% \sim 90\%$ GB are used for PSums. The ALU array size is again defined by Conv's *comp* μaPMs ($ttm \times tte$), and the parallel MAC operations in each ALU is $ttc \times ttr \times tts$.

3.3 Multi-Layer Fitting

Methods detailed in Section 3.2 can be used to design custom accelerators tailored for a specific NN layer. However, most DNN algorithms consist of multiple layers with diverse shapes and thus require efficient schemes to fit them on a single hardware.

To determine the optimal architecture configuration that can fit multiple layer, we resort to finding the consistency between layer types and shapes. Recall that Conv and FC layer types (Fig. **??**) are related is we consider Conv as weight-shared FC layers. Assuming the size of Conv filter ($R \times S$) is expanded to cover the entire infmaps (i.e. $H = R, W = S$), the Conv layer is effectively converted to a FC. The number of input neurons I is equal to the size of ifmaps $I = C \times H \times W$, and the number of output neurons O is equal to the ofmaps size $O = M \times E \times F, E = F = 1$. The 3D filter's size is equal to the size of ifmaps, and the total number of the 3D filters is M. Based on this methodology, we can simply regard FC layer is a special type of Conv layer.

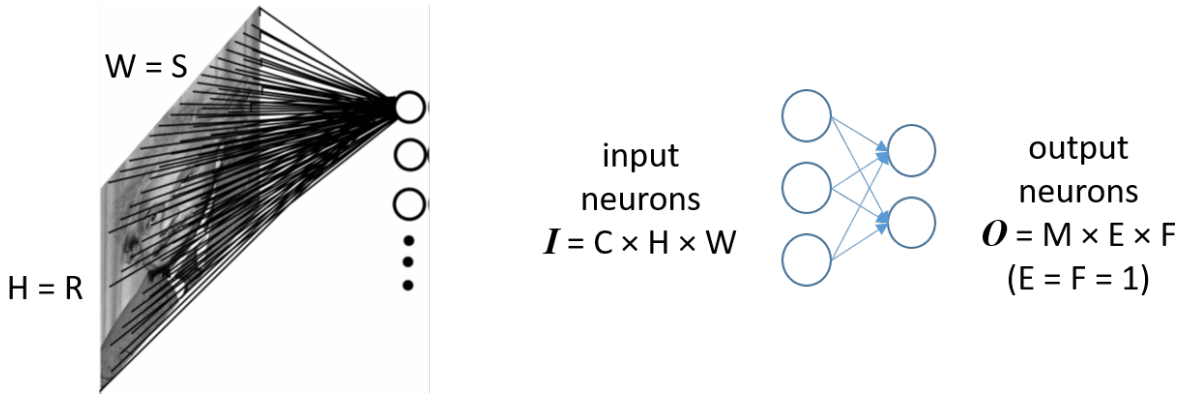

Figure 3.4: FC layer's Conv view

In the Conv architecture, I-LBs consist of $te \times R_{acc}$ shifters and each shifter stores $t_c \times S_{acc}$ data. R_{acc} and S_{acc} is fixed in the hardware accelerator, but R_{nn} and S_{nn} can take different values for each layer in a network. To fit different Conv sizes on a single accelerator, we proposed certain methods for two cases, $R_{nn} < R_{acc}$ and $R_{nn} > R_{acc}$.

To fit the $R_{nn} < R_{acc}$ Conv layers (Fig. 3.5), the LBs and ALUs cannot be fully utilized to during the execution, since the LB size is defined by fixed R_{acc}, S_{acc} values.

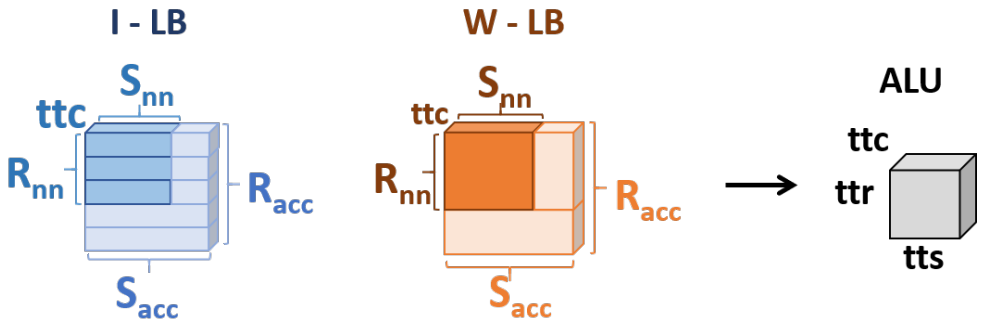

Figure 3.5: $R_{nn} < R_{acc}$

So the utilization rate among those resources can be calculated by the following equations (considering R, S dimension):

$$I - LB\ util\ rate = \left\lceil \frac{R_{nn}}{R_{acc}} \right\rceil \times \left\lceil \frac{S_{nn}}{S_{acc}} \right\rceil$$

$$W - LB\ util\ rate = \left\lceil \frac{R_{nn} \times R_{acc}}{S_{nn} \times S_{acc}} \right\rceil$$

$$ALU\ util\ rate = \left\lceil \frac{R_{nn} \times S_{nn}}{\#\ of\ cycles \times \#\ of\ MACs} \right\rceil$$

$$\#\ of\ cycles = \left\lceil \frac{R_{nn} \times ttr}{S_{nn} \times tts} \right\rceil$$

$$\#\ of\ MACs = ttr \times tts$$

To fit $R_{nn} > R_{acc}$ Conv layers (Fig. 3.6), we define a fitting parameter $n_f = (R_{acc} \times S_{acc})/(R_{nn} \times S_{nn})$. If n_f is larger than 1, the I-LB can store $\lfloor n_f \rfloor \times t_e$ input SWs in I-LB and requires a reconfigurable multiplexer (MUX) to fetch $(R - U) + U \times \lfloor n_f \rfloor \times t_e$ rows of inputs under different shape configurations. W-LBs should be similarly implemented by $R_{acc} \times S_{acc}$ SRAMs and each SRAM feeds $t_m \times t_c$ data blocks to W-LB. This fitting scheme allows for consistent dataflow for different layers. At the GB level, the I-GB, W-GB, and P-GB SRAMs are sized to store the largest data structure among the layers. If the n_f is smaller than 1, meaning the I-LB and W-LB are too small to store the entire SW, fitting is achieved

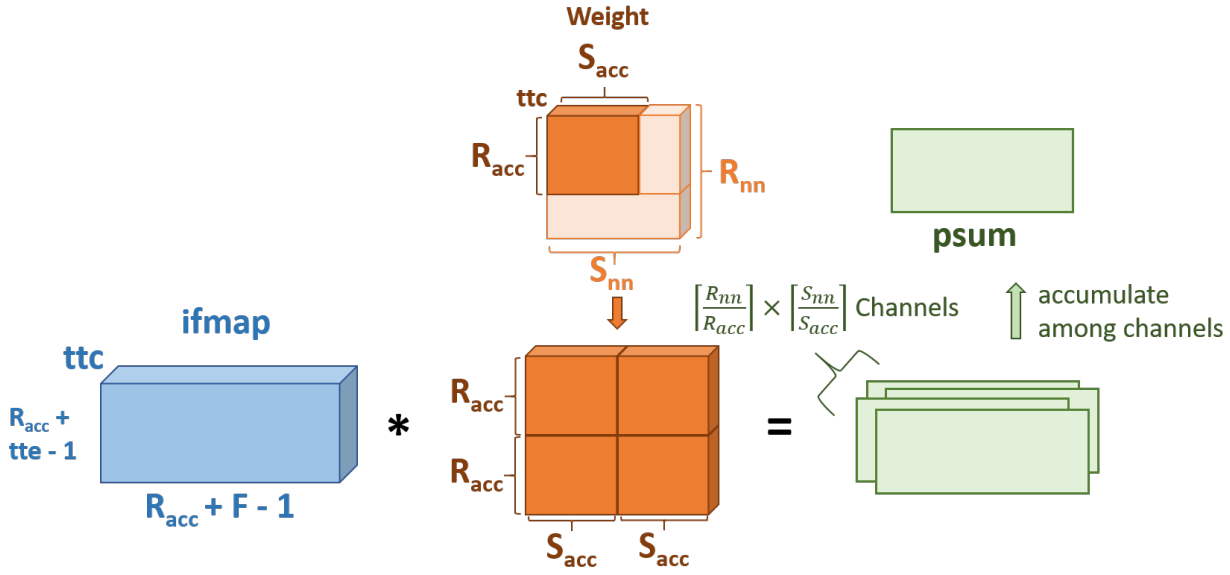

Figure 3.6: $R_{nn} > R_{acc}$

by decomposing the larger convolution ($R \times S$) into n^2 smaller filters ($\lceil R/n \rceil$-by-$\lceil S/n \rceil$) and accumulate the n^2 channels to get the final outputs.

So the utilization rate among those resources can be calculated by the following equations (considering R, S dimension):

$$I - LB\ util\ rate = \frac{R_{nn} \times S_{nn}}{(\lceil \frac{R_{nn}}{R_{acc}} \rceil \times \lceil \frac{S_{nn}}{S_{acc}} \rceil) \times (R_{acc} \times S_{acc})}$$

$$W - LB\ util\ rate = \frac{R_{nn} \times S_{nn}}{(\lceil \frac{R_{nn}}{R_{acc}} \rceil \times \lceil \frac{S_{nn}}{S_{acc}} \rceil) \times (R_{acc} \times S_{acc})}$$

$$ALU\ util\ rate = \left\lceil \frac{R_{nn} \times S_{nn}}{\#\ of\ cycles \times \#\ of\ MACs} \right\rceil$$

$$\#\ of\ cycles = (\left\lceil \frac{R_{nn}}{R_{acc}} \right\rceil \times \left\lceil \frac{S_{nn}}{S_{acc}} \right\rceil) \times (\left\lceil \frac{R_{acc}}{ttr} \right\rceil \times \left\lceil \frac{S_{acc}}{tts} \right\rceil)$$

$$\#\ of\ MACs = ttr \times tts$$

3.4 Area/Performance/Energy Modeling

To accurately model the design tradeoffs of NN accelerators, we need actual area/performance/energy data for each on-chip components (GB, LB, BBus, and ALU) characterized in a parameterized manner. As illustrated in Fig. 3.7, these characterization data is obtained through NNest interfaces with other simulator tools, such as Cacti [16] for SRAM modeling, memory compiler for register file modeling, Synopsys Design Compiler (DC) for arithmetic block modeling. To account for the area/latency/power of arithmetic blocks in the ALU array, we synthesize the basic multipliers and adder trees in the datapath using DC with a 40nm standard cell library, similar to the method used in Aladdin [31]. Although 40nm is used in our experiment due to limited access to process technology, the same block-level characterization can be easily ported and implemented in a new technology. Apart from block-level characterization data, NNest can take NN structure parameter directly from DNN software tools such as TensorFlow and it also accounts for user-specified area/timing/energy constraints.

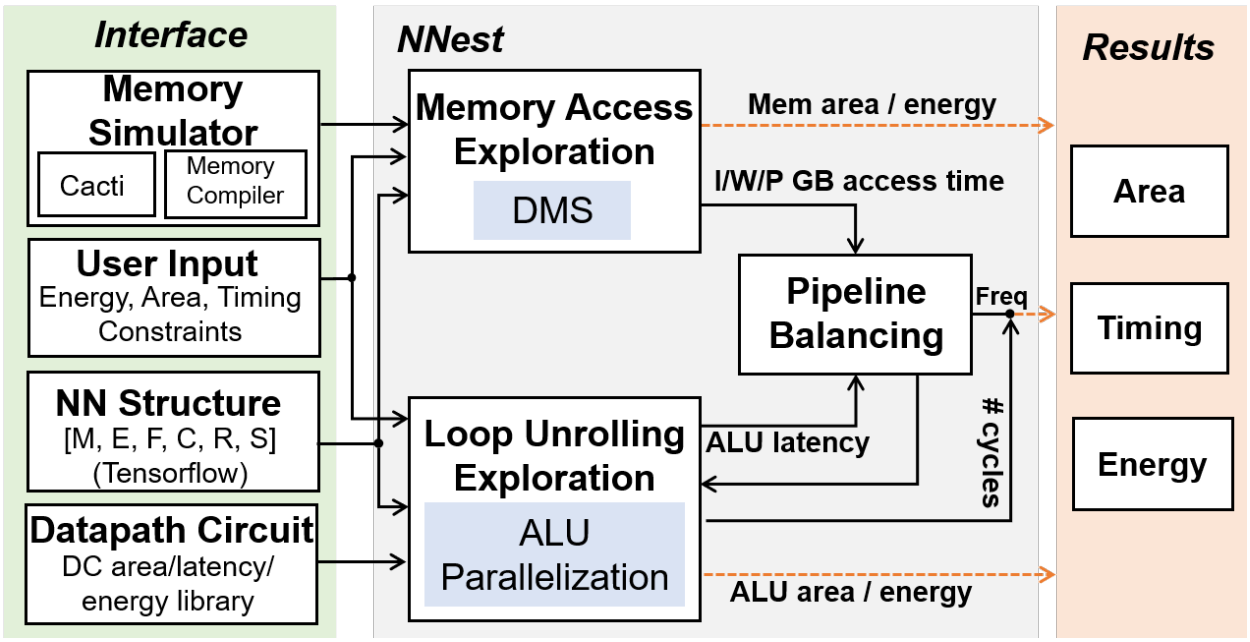

Figure 3.7: NNest's area/performance/energy modeling framework

The data movement strategies and fitting schemes introduced earlier allows NNest to efficiently explore the design space defined by the μarch parameters in Table 3.1. This

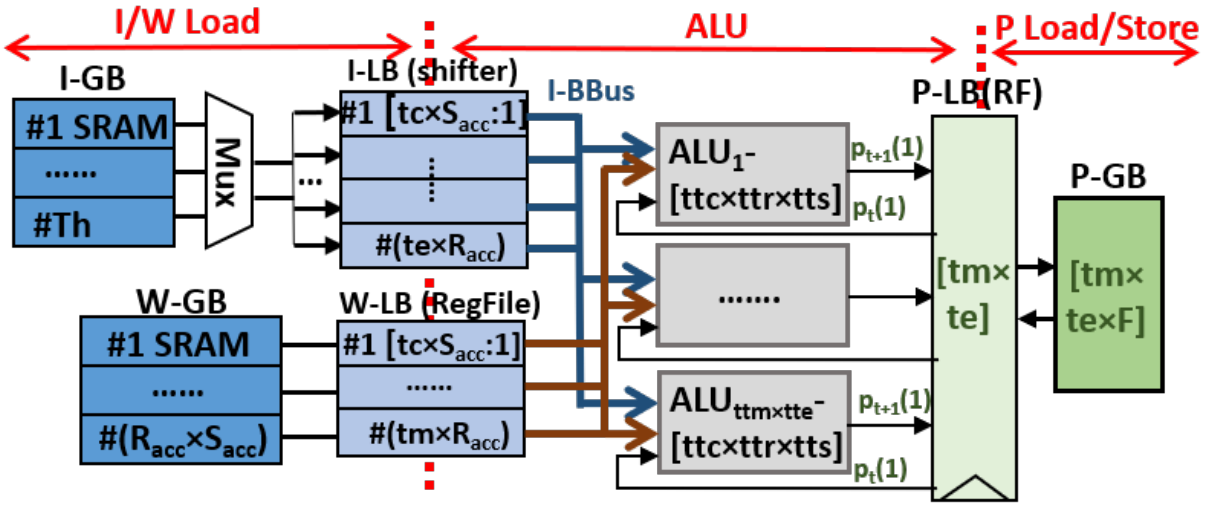

Figure 3.8: NNest pipeline stage

architectural-level exploration has already taken into consideration of different parallelization strategies such as MAC loop unrolling and hierarchical memory optimization.

At the circuit level, we also consider different pipelining strategies. The execution of the NN accelerator based on spatial architecture can be broken down into 3 stages—loading data, computing for one cycle, and storing PSum results, as illustrated in Fig. 3.8. We start with an initial 3-stage pipeline implementation and simulate the critical paths of each stage. Due to the different memory and ALU array size, the initial 3-stage pipeline may not be balanced, and NNest searches through different arithmetic block designs in the characterization database to balance the pipeline. If failed, NNest would attempt to break the ALU stage into multiple pipeline stages and iterate through the balancing step, until a reasonable pipeline solution is arrived. Finally, the last resort is to reduce the parallelism of ALU to sequentially process the computation in several cycles.

Chapter 4

Experiment

All the experiments are performed in 40nm technology.

4.1 Layer-Level Exploration

First, we use NNest to generate designs tailored to specific NN layers and explore single-layer design spaces. AlexNet's Conv-3 and FC-1 layers are used as examples. Results are shown in Fig 4.1, where the all the design points are scatter-plotted and the Pareto frontiers consisting of the optimal design points are identified. In AlexNet Conv-3 design space, we observe the general trend that NLR dataflow consumes power than PR and IWR where multi-level memory hierarchy is employed for local data reuse. The Pareto frontiers of IWR and PR lie closely together. We choose one design point on IWR and PR frontier each with similar performance and analyze their energy and area breakdown. The PR's total energy is about 93.4% of IWR due to energy saving from GB access, which comes with extra area overhead because PR needs to store the entire C channels of input and weight rows. In IWR, inputs and weights consume less memory space, but PSums need to be fetched from a large P-GB every cycle and this frequent PSum load/store increases GB access energy. We also mark the design points based on the dataflow strategies introduced in previous work (Eyeriss [29] and ASP-DAC [15]). Both designs are located away from the frontier identified by NNest, where more energy-efficient design points can be found (improvement of 28.5% compared to ASP-DAC [15] and 39.3% compared to Eyeriss [29]).

In AlexNet FC-1 design space, IR represent more optimal designs than PR with less area overhead. The reason PR performs poorly is due to its GB. PR holds all input data $N \times I$ in I-GB, whereas IR holds all PSums $N \times O$ in P-GB. In AlexNet FC-1, $I = 43264$, $O = 4096$, $I \gg O$, making the PR's GB area and energy cost grow. So, in FC layer, if I is much larger than O in FC layer, IR dataflow outperforms PR significantly. Comparing NLR and MH frontiers reveal that data reuse in LB can save energy consumption around 9% with only 0.3% extra area. FC layer consumes much more energy on DRAM+GB memory access than Conv layer (more that 98% of total energy for both PR and IR frontier in FC vs 70% $\sim$ 80% in Conv). This stems from Conv layer's local connectivity property.

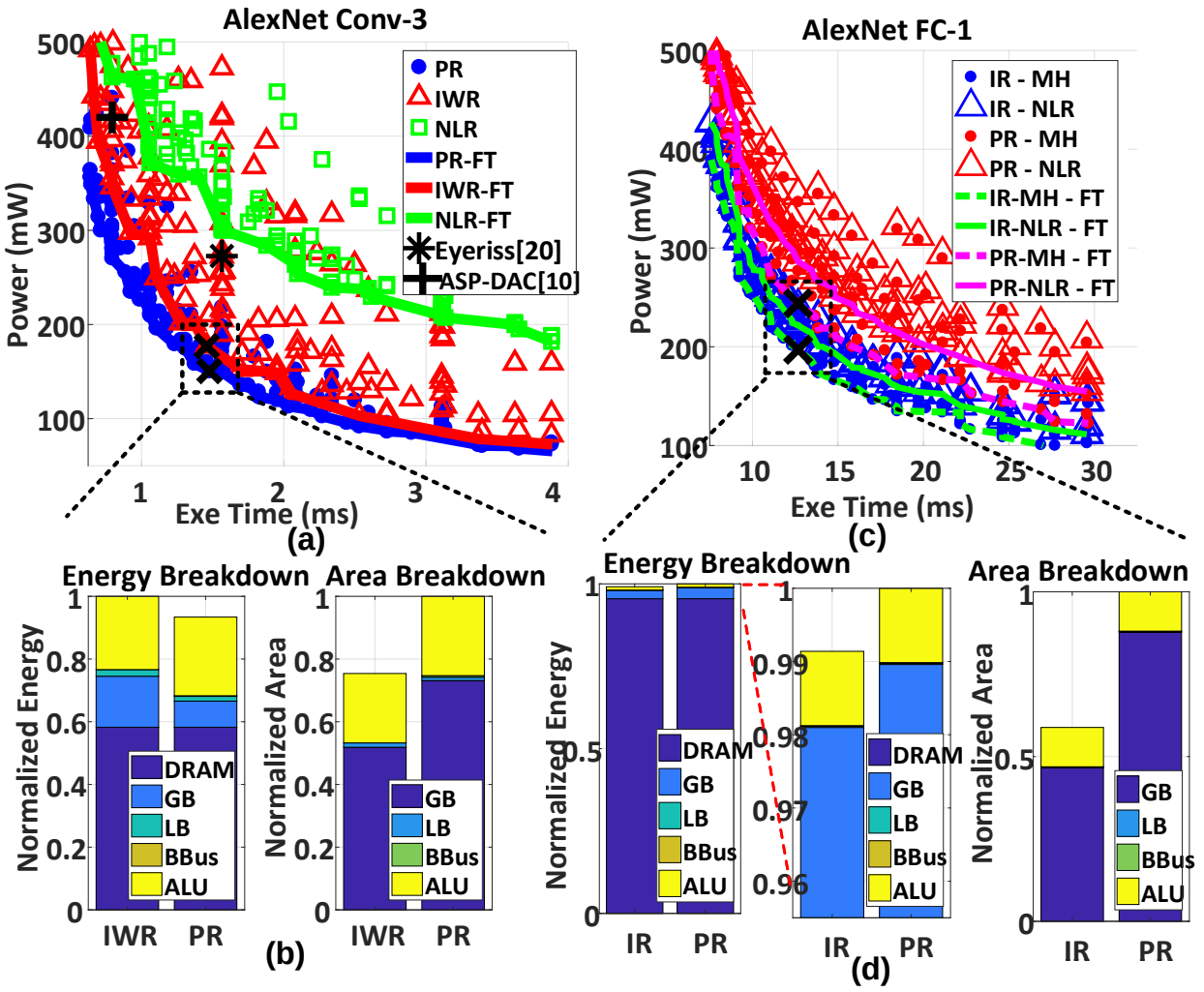

Figure 4.1: (a) Conv-3 design space, (b) Conv-3 energy/area breakdown, (c) FC-1 design space, (d) FC-1 energy/area breakdown

4.2 Network-Level Exploration

Next, we apply the multi-layer fitting scheme described in Section 3.3 to explore the design space for a complete neural network. We use AlexNet [1] and VGG [21] as examples. The design space results are presented in Fig. 4.2. Unsurprisingly, VGG consumes higher power than AlexNet, as it uses deeper network and larger weight parameters. We identify the minimum total energy design points for both networks, which indicates across the entire design space, VGG is at least 4× more energy hungry than AlexNet. Interestingly, if we plot the design points that minimize the energy per MAC operation for both AlexNet and VGG, we observe that for each MAC operation, accelerator tailored to VGG consumes only 17% energy as compared to per MAC operation in AlexNet-optimized accelerators. The reason is that filter sizes ($R, S = 11, 5, 3$) in different AlexNet layers vary greatly, which causes a large hardware overhead when fitted across multiple layers. VGG has much more uniform layer pattern ($R, S = 3$), allowing it to better utilize the hardware resources. NNest can conveniently generate design points based on other specifications. For example, to achieve real-time image processing of $33 frame/s$, we pick the two design points on Fig 4.2 with the lowest power consumption (87 mW for AlexNet and 306 mW for VGG). The die area of each design can also be obtained of that AlexNet and VGG accelerator design are $4.2mm^2$ and $8.02mm^2$

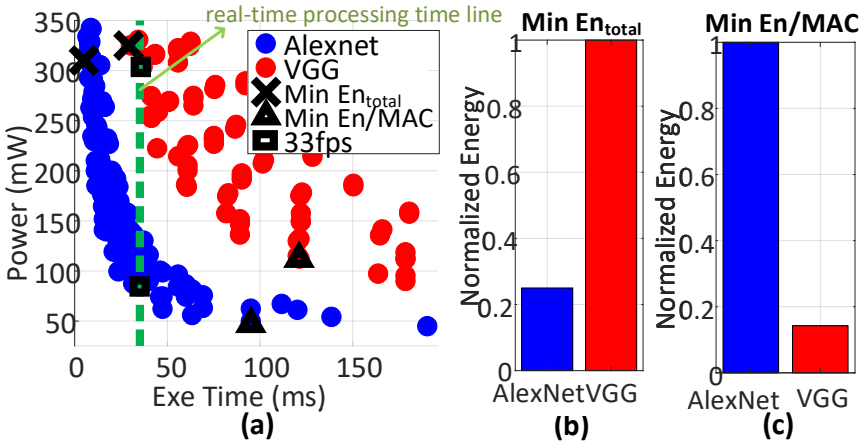

Figure 4.2: Alexnet, VGG design space comparison

4.3 Quantization Technique Exploration

Finally, we use NNest to holistically evaluate different level of quantization in DNNs. It is important to note that certain quantization techniques require larger network to maintain acceptable accuracy. For example, BNN has to employ higher number of convolutional filters to compensate for its accuracy loss due to extreme quantization. Therefore the naive approach of simply binarize the original network does not accurately capture the actual design tradeoff for BNN and would significantly underestimate the power and area cost. Instead, NNest allows the user to specify both NN structure parameters and circuit-level parameters such as bitwidth and therefore is able to quickly generate the correct design space and Pareto frontiers for early-stage evaluation of different quantization techniques. In conclusion, an early-stage design exploration tool for NN accelerator such as NNest can prove useful in many design scenarios as demonstrated in these experiment examples.

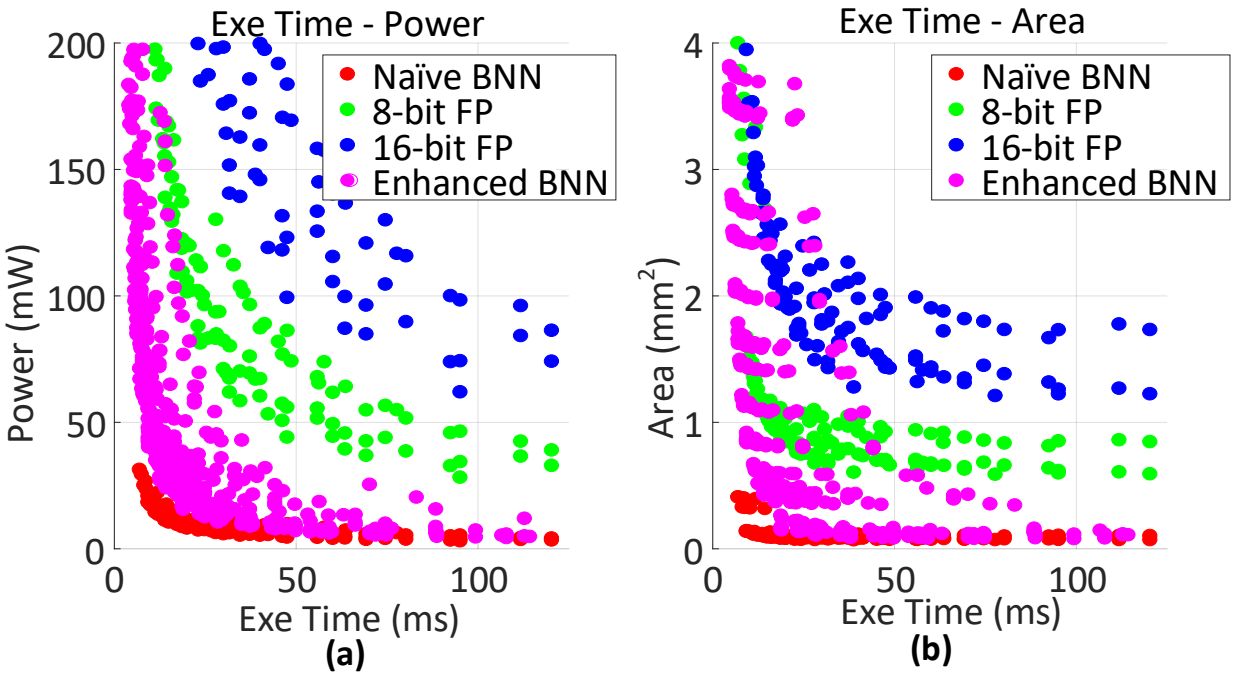

Figure 4.3: Alexnet bitwidth design space

Chapter 5

Conclusion

We have present NNest, an early-stage design space exploration tool for NN inference accelerator, to speedily and accurately estimate the area/performance/energy for the ASIC-based NN accelerators with low-level RTL code. Based on the general architecture framework, a parameterized data movement and parallel processing strategies are proposed. The high-dimensionallt broad design space is explored by sweeping μarch parameters. The several experiment results reveal that NNest can be applied in many design scenarios, like finding the optimal μarch designs, evaluating DNN models and NN optimization algorithms, etc.

As for the future work, it could have many possibilities. The hardware platform can be extended to many other popular parallelization processors, like FPGA, GPU and CGRA. In the algorithm level, our tool currently only includes FC and Conv layer. Other layers, like activation function, normalization and pooling layer can be extended to our framework. Also, other NN algorithms, like RNN/LSTM and deep reinforcement learning, can be considered as an extension. A failure rate or error model can be explored to compute the error from the hardware, like approximate multiplier and fuzzy memory.

References

[1] I. Sutskever A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, pages 1097–1105, 2012.

[2] et al. Brandon Reagen. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. *ISCA*, 2016.

[3] et al C. Szegedy. Going deeper with convolutions. *in Proc. CVPR*, pages 1–9, 2015.

[4] et al. C. Zhang. Optimizing fpga-based accelerator design for deep convolutional neural networks. *in Proc. FPGA*, pages 161–170, 2015.

[5] et al. Chen Zhang. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *ICCAD*, 2016.

[6] et al. H. Sharma. From high-level deep neural models to fpgas. *MICRO*, 2016.

[7] Adrien Prost-Boucle Frdric Ptrot Hande Alemdar, Vincent Leroy. Ternary neural networks for resource-efficient ai applications. *IJCNN*, 2017.

[8] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *NIPS*, pages 4107–4115, 2016.

[9] Daniel Soudry-Ran El-Yaniv Yoshua Bengio Itay Hubara, Matthieu Courbariaux. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv*, 2016.

[10] S. Ren K. He, X. Zhang and J. Sun. Deep residual learning for image recognition. *in Proc. CVPR*, pages 770–778, 2016.

[11] C. Mayer-S. Willi-B. Muheim L. Cavigelli, D. Gschwend and L. Benini. Origami: A convolutional network accelerator. *in Proc. GLVLSI*, pages 199–204, 2015.

[12] B. Mesman M. Peemen, A. A. A. Setio and H. Corporaal. Memory-centric accelerator design for convolutional neural networks. *in Proc. ICCD*, page 1319, 2013.

[13] et al M. Sankaradas. A massively parallel coprocessor for convolutional neural networks. *in Proc. ASAP*, pages 53–60, 2009.

[14] Luca Benini Manuele Rusci, Lukas Cavigelliy. Design automation for binarized neural networks: A quantum leap opportunity? *arXiv*, 2017.

[15] et al. Mohammad Motamedi. Design space exploration of fpga based deep convolutional neural networks. *ASP-DAC*, 2016.

[16] Vaishnav Srinivas Naveen Muralimanohar, Ali Shafiee. https://github.com/hewlettpackard/cacti.

[17] V. Jakkula S. Chakradhar, M. Sankaradas and S. Cadambi. A dynamically configurable coprocessor for convolutional neural networks. *in Proc. ISCA*, pages 247–257, 2010.

[18] K. Gopalakrishnan S. Gupta, A. Agrawal and P. Narayanan. Deep learning with limited numerical precision. *in Proc. ICML*, pages 1737–1746, 2015.

[19] K. Gopalakrishnan S. Gupta, A. Agrawal and P. Narayanan. Deep learning with limited numerical precision. *ICML*, pages 1737–1746, 2015.

[20] D. Shin J. Lee-S. Choi S. Park, K. Bong and H.-J. Yoo. A 1.93 tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications. *in IEEE ISSCC Dig. Tech. Papers*, pages 1–3, 2015.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *in Proc. ICLR*, 2015.

[22] Huizi Mao Jing Pu-Ardavan Pedram Mark Horowitz William J. Dally Song Han, Xingyu Liu. Eie: Efficient inference engine on compressed deep neural network. *ISCA*, June 2016.

[23] et al T. Chen. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *in Proc. ASPLOS*, pages 269–284, 2014.

[24] Joel Emer-Vivienne Sze Tien-Ju Yang, Yu-Hsin Chen. A method to estimate the energy consumption of deep neural networks. *Asilomar*, 2017.

[25] A. Dundar B. Martini V. Gokhale, J. Jin and E. Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. *in CVPR Workshop*, pages 682–687, 2014.

[26] K. H. Tsoi V. Sriram, D. Cox and W. Luk. Towards an embedded biologically inspired machine vision processor. *in Proc. FPT*, pages 273–278, 2010.

[27] et al. Vivienne Sze. Efficient processing of deep neural networks: A tutorial and survey. *arXiv*, 2017.

[28] et al Y. Chen. Dadiannao: A machinelearning supercomputer. *in Proc. MICRO*, pages 609–622, 2014.

[29] J. Emer Y.-H. Chen, T. Krishna and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *in IEEE ISSCC Dig. Tech. Papers*, pages 262–263, Jan. 2016.

[30] Y. Bengio Y. LeCun, L. Bottou and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278–2324, Nov. 1998.

[31] Gu-Yeon Wei-David Brooks Yakun Sophia Shao, Brandon Reagen. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. *ISCA*, 2014.

[32] et al. Yijin Guan. Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. *ICCAD*, 2016.

[33] et al Z. Du. Shidiannao: Shifting vision processing closer to the sensor. *in Proc. ISCA*, pages 92–104, 2015.

Vita

Liu Ke

Degrees

B.E. Software Engineering, Sun Yat-Sen University, June 2016
M.S. Electrical Engineering, Washington University in St.Louis, May 2018

Publications

Liu Ke, Jun Wang, Xijun Zhao, Fan Liang, Fast-Gaussian SIFT and its Hardware Architecture for Keypoint Detection, 2016 IEEE Asia Pacific Conference on Circuit and Systems (APCCAS 2016), pp.436-439

Liu Ke, Jun Wang, Zhixian Ye, Fast-Gaussian SIFT for Fast and Accurate Feature Extraction, The 2016 Pacific-Rim Conference on Multimedia (PCM 2016), pp.355-365

Eun-Young Kang, **Liu Ke**, Meng-Zhe Hua and Yu-Xuan Wang, Verifying Automotive Systems in EAST- ADL/Stateflow using UPPAAL, The Asia-Pacific Software Engineering Conference (APSEC) 2015, pp.143-150

Eun-Young Kang, Jianda Chen, **Liu Ke** and Shangyu Chen, Statistical Analysis of Energy-aware Real-Time Automotive Systems in EAST-ADL/Stateflow, The 11th IEEE Conference on Industry Electronics and Applications(ICIEA), pp.1328-1333

May 2018