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

Liu Ke, Xin He, Xuan Zhang
Washington University in St. Louis

ABSTRACT

Deep neural network (DNN) has achieved spectacular success in recent years. In response to DNN's enormous computation demand and 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 hard to arrive at an "one-size-fits-all" hardware design. In this paper, 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/micro-architectural 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).

CCS CONCEPTS

• **Hardware** → *Modeling and parameter extraction*;

KEYWORDS

Deep neural networks, Design space exploration, Accelerators

1 INTRODUCTION

Since the groundbreaking performance of AlexNet [1] in 2012 ImageNet competition, a class of machine learning methods, deep neural networks (DNNs), has achieved spectacular success, like applications in 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. Extensive toolchain and framework have been built on general-purpose graphics processing units (GPGPU) to leverage its superior parallel processing capability for DNN tasks [1]; field-programmable gate array (FPGA) based systems [5] have attracted much attentions thanks to their programmable fabrics that can accommodate flexible parallel processing primitives and adapt to rapidly evolving algorithms; finally custom DNN 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 with cloud-centric platforms, edge devices have much more stringent power budget and sensitive cost consideration, making application-specific integrated circuits (ASIC) based solution a more appealing choice [8]. Numerous NN inference accelerators have been proposed [13, 14], but the diverse nature of DNNs, both at the network level and the micro-architetcure (μArch) level, makes it hard 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 μArch designs; circuit designers and device engineers can get an early glimpse of how device/circuit level innovation can affect the overall system; algorithm developers can more deeply understand the potential advantages/penalties of their model parameters and algorithmic techniques for custom ASIC accelerators without being limited by the specific implementation.

In this paper, we present NNest[1]—an early-stage tool that is designed to facilitate systematic design space exploration for ASIC-based DNN inference accelerators. Our main contributions include:

- We propose a spatial accelerator architecture template that can be generalized to cover a variety of DNN accelerator implementations, capturing design tradeoffs before RTL.
- We develop parameterized data movement strategies and multi-layer fitting schemes that can efficiently express the inherent parallelism in DNN models.
- 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 and binarized quantization demonstrate such capability.

[1] https://github.com/xz-group/NNest-1.0

ISLPED '18, July 23–25, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5704-3/18/07…$15.00
https://doi.org/10.1145/3218603.3218647

2 BACKGROUND AND RELATED WORK

2.1 Preliminary on DNN

DNN models typically consists of cascading of different layers, including convolution, activation function, normalization, pooling, and fully-connected. Since Conv and FC layers tend to dominate computation and memory access, NNest focuses on exploring those two kinds of layers. DNN models usually employ distinctive layer shapes and sizes, defined by NN structure parameters (nnsPMs) in NNest, leading it hard 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 of FC layer employs three nnsPMs 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 (Fig. 1). 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 for several input vectors to calculate different output. 3) *Partial Sum Reuse*: the old partial sum (PSum) is reused to get the updated PSum by accumulating with the new element-wise products.

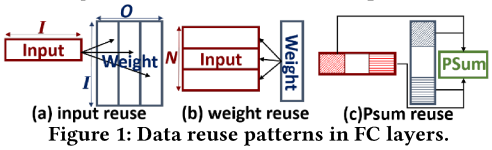

Figure 1: Data reuse patterns in FC layers.

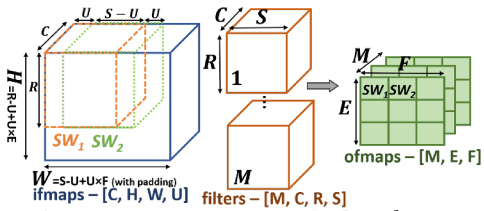

Figure 2: Data reuse patterns in Conv layers.

2.1.2 Convolutional (Conv) layer. To extract features from input feature maps (ifmaps), each output neuron in output feature maps (ofmaps) is only connected to a local region of ifmaps, known as the local connectivity property of convolution, and the weight matrix has the same size as the local ifmap region is called a filter. The size of ifmaps, ofmaps and filters is defined by Conv's nnsPMs. Conv layer's computational pattern can be regarded as a $C \times R \times S$ sliding window (SW) (Fig. 2) 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 shifting 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 $(S-U) \times R \times C$. U is stride. 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 ofmaps.

Contrasted with FC layers, Conv is weight-shared due to local connectivity, 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, referred 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.

2.2 Existing NN Accelerators

A recent tutorial have extensively surveyed prior work on NN inference accelerators and categorized them based on dataflow [19]: 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 [3, 15, 20]. Hence, all MACs share a global buffer (GB) to load inputs and weights and store intermediate PSums. 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 [8, 12, 17, 18]. The input and PSums remain stored in GB. 3) *Output Stationary (OS)* stores PSums locally [13, 24], rather than weights. Similarly, the goal is to minimize the energy for fetching old PSums and saving back the updated one. 4) *Row Stationary (RS)* is proposed to locally store weight, input and PSum to increase data reuse opportunities [21]. Our framework encompasses all these previously-proposed architectures.

2.3 NN Design Automation

A number of design automation tools have been introduced [4, 6, 23] 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 to study FPGA-based DNN accelerator [10]. However, it does not explicitly address FC layers and multi-level memory hierarchy, and provides no exploration results for die area and power consumption.

In the custom ASIC space, tools have been developed to explore acceleration for general computational kernels [22] that can be applied to limited NN design space [2]. Method to roughly estimate DNN energy consumption has been proposed to guide architecture selection [16] 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 binarized neural network (BNN) [9] to perform estimation and analysis, but it does not readily apply to investigate the much broader design space of general NN accelerator with different quantization schemes.

3 METHODOLOGY

3.1 Generalized Architecture Template

In order to conduct effective and thorough design space exploration, we first propose a spatial NN accelerator architecture template (Fig. 3) that can be generalized to produce numerous design points. It has drawn inspirations from many existing accelerator prototypes and is able to encompass 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 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).

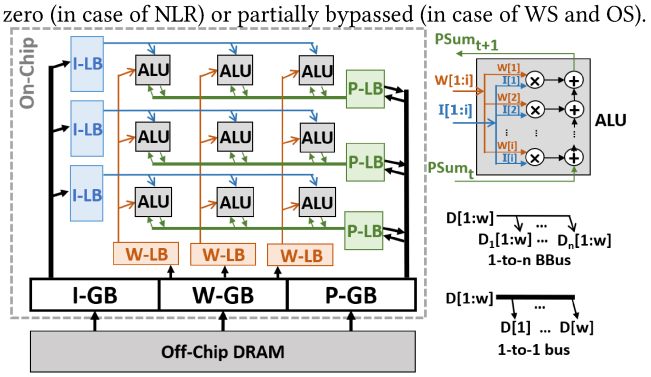

Figure 3: generalized accelerator template

Intuitively, it is ideal to hold all data in GBs to avoid multiple energy-expensive 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 LBs and ALU array. Therefore, we must determine optimal strategies to move and replace data between and across the partitioned GBs and LBs to minimize external/on-chip memory access and achieve highest degree of data reuse, 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 [14]. Since DNN models are memory intensive, 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 DNN accelerators. 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 without energy-expensive memory access for next computation cycle. In this section, we derive the μArch parameters (μAPMs) used in NNest to define memory and computation engines and 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 values listed in Table 1. *comp* μAPMs define the size of ALU array representing the computational speed of the NN accelerator. *mem* μAPMs and nnsPMs define the storage capacity of GBs and LBs.

Table 1: List of NNest μArch Parameters

	Parameter Name	
	FC Layer	Conv Layer
comp μAPMs	tti, tto	ttm, tte, ttc, ttr, tts
mem μAPMs	T_n, T_i, T_o	T_m, T_e, t_m, t_e, 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, $T_i \times T_o$ (*mem* μAPMs in Table. 1), of weight is stored on-chip, determining W-GB size. The tiled batch size T_n is used to represent the number of inputs being processed concurrently, allowing for weight reuse in a broadcasting manner. For FC DMS, we regard a SW shifting without overlap on the weight matrix to fetch data. However, the different SW shift orders effect GBs implementation a lot. We study two effective strategies (Fig. 4(a)(c)) among the sea of shift order combination, which consume reasonable memory access and employ high data reuse rate. For each strategy, DMS at both GBs and LBs are described in details.

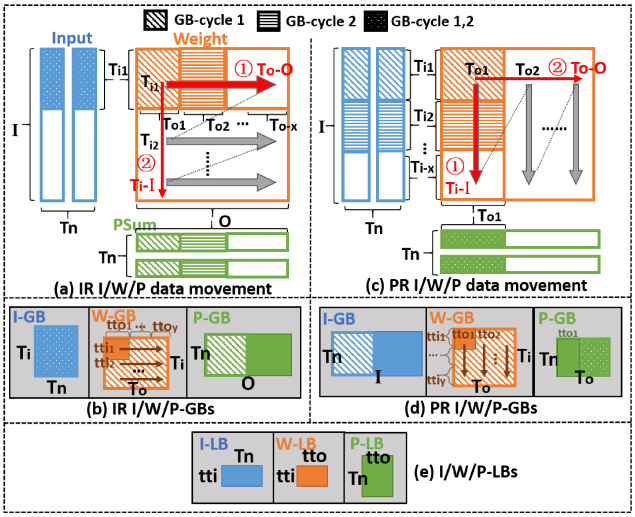

Figure 4: Illustration of different DMS in FC layers

Input Reuse (IR): At DRAM-to-GB level, SW is shifted in order $\{O, I\}$. SW is shifted in O-dimension first on weight matrix (Fig. 4(a)), fetching $T_i \times T_o$ weight data to W-GB. While $T_n \times T_{i1}$ inputs are kept stationary in I-GB to reuse it. The sub-index denotes the range of fetched data, $T_{ix} = [T_i \times (x-1) + 1 : T_i \times x]$, same rule applied in the following content. After fetching all O columns of weight matrix with T_{i1} rows, $T_n \times O$ PSums are generated. Then the second step is that both inputs and weights are shifted in I-dimension, $T_n \times (T_{i1} \rightarrow T_{i2})$ (input) , $T_{i1} \times T_{ox} \rightarrow T_{i2} \times T_{o1}$ (weight, $x = O/T_o$), to update PSums. To avoid frequent DRAM access, P-GB is sized to $T_n \times O$ (Fig. 4(b)) to hold all intermediate PSums and I/W-GB are sized to $T_i \times T_n$, $T_i \times T_o$. At GB-to-LB level, SW is shifted in order $\{To, Ti\}$. GB-to-LB DMS is same as DRAM-to-GB DMS patterns by replacing corresponding μAPMs. Shown in Fig. 4(b)(e), to satisfy the processing speed of ALU array, I/W/P-LBs are defined by *comp* μAPMs (tti, tto). W-LB is replaced in T_o-dimension first. $T_n \times tti_1$ input data are kept stationary in I-LB. After T_o/tto cycles, I/W-LBs are then replaced in T_i-dimension to update PSums.

PSum Reuse (PR): The derivation of dataflow in PR can be done in a similar manner as illustrated in Fig. 4(c)-(e), with reverse SW shifting order compared with IR, $\{I, O\}$ at DRAM-to-GB level and $\{Ti, To\}$ at GB-to-LB level. Here, at DRAM-to-GB level, PSums are

kept stationary in P-GB, and both inputs and weights are replaced in I-dimension, $T_n \times (T_{i1} \rightarrow T_{i2})$ (input), $T_{o1} \times (T_{i1} \rightarrow T_{i2})$ (weight). After processing I rows of input data and weight matrix with T_{o1} columns, $T_n \times T_{o1}$ PSums in P-GB are finalized and send to external DRAM. 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 $T_n \times T_{o2}$ output, and all the input data are fetched again from $T_n \times (T_{i1} \rightarrow T_{ix})$. Similar DMS patterns are employed at GB-to-LB level.

DMS also determines the BBus configuration based on the broadcasting requirements of GB-to-LB (*mem* μAPMs) and LB-to-ALU (*comp* μAPMs). For example, $tto \times (tti)$ weights in W-LB and $T_n \times (tti)$ inputs in I-LB broadcast to T_n-by-tto ALU array. BBus is 1-to-T_n from W-LB to ALU, and 1-to-tto from I-LB to ALU.

3.2.2 Dataflow in Conv layers. Similar with FC DMS, SW shifting order influences Conv GBs design. Comparing various shifting orders, the following two DMS, IWR and PR, achieve energy efficient and high data reuse, which are conceptual similar with the IR and PR in FC layer with additional reuse possibilities from *sliding window reuse*. GB-to-LB level DMS in both are briefly described below. Similar pattern exists in other memory levels.

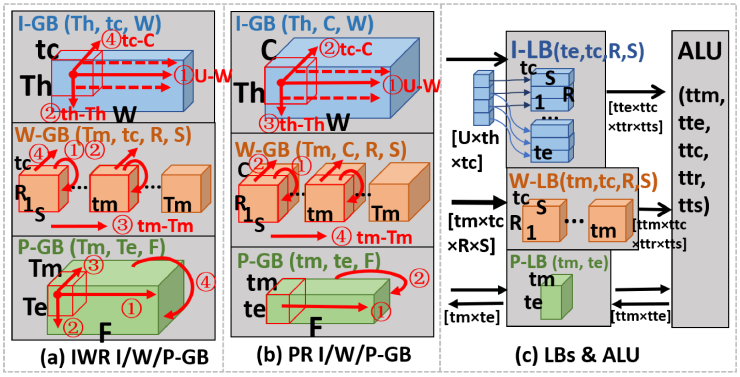

Figure 5: Illustration of different DMS in Conv

Input/Weight Reuse (IWR): At GB-to-LB level, SW is shifted in order $\{W, T_h, T_m, C\}$, and IWR aims at maximizing input and weight reuse. In the first-order (W) (Fig. 5(a)) SW first shifts along the W-dimension to reuse the SW overlap data. And then in the second order (T_h), SW moves to the next t_h input rows and shifts along W-dimension. After processing 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 third-order (T_m) step replaces W-LB with the next t_m weights and SW is shifted along the entire T_h rows of input data again. After processing all T_m 3D filters in W-GB, $T_m \times T_e \times F$ PSums are generated. Then in the last order (C), both inputs and filters are replaced to the next t_c channels, and the intermediate PSums would be updated to their finalized values. To increase the I/W reuse rate, the parameters (T_h, T_m) that define the number of SWs should be large. Due to limited GB size, only t_c I/W channels are held in GB, and t_c is usually small to accommodate large T_h, T_m. Therefore P-GB holds large amount of PSum data ($T_m \times T_e \times F$) and accounts for the majority of GBs size. IWR described here is similar to the row stationary introduced in Eyeriss, where 80% ~ 90% GB are used for PSums. 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 is defined as $t_m \times t_c \times R \times S$. ALU array size (Fig. 5(c)) is defined by Conv's *comp* μAPMs ($ttm \times tte$), and the parallel MAC operations in each ALU is $ttc \times ttr \times tts$.

PSum Reuse (PR): At GB-to-LB level, SW is shifted in order $\{W, C, T_h, T_m\}$, and PR aims at maximizing PSum reuse. Similar with IWR, in the first-order (W), SW (Fig. 5(b)) first shifts along the W-dimension, and $t_m \times t_e \times F$ PSums are generated. Then in the second-order (C), SW moves to the next t_c I/W channels and shifts along W-dimension to update the intermediate PSums computed in the last F shifting steps until those PSums are finalized and then stored back to DRAM, setting P-GB size as $t_e \times t_m \times F$. To keep the same DRAM access with IWR, all data in C-dimension of inputs ($T_h \times C \times W$) and weights ($T_m \times C \times R \times S$) are held in GBs. The third-order (T_h) step shifts to the next t_h rows of input data. The final order (T_m) processes all T_m 3D filters in W-GB. So, in PR, P-GB size could be small to store a limited number of PSums, but I/W-GB should have sufficient space to store C channels of inputs and weights. And PR's LBs and ALU is same defined as IWR.

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 models 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 are related is we consider Conv as local-connected and 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. Hence, there exists a relationship between FC and Conv nnsPMs as $I = C \times R \times S = C \times H \times W, O = M$.

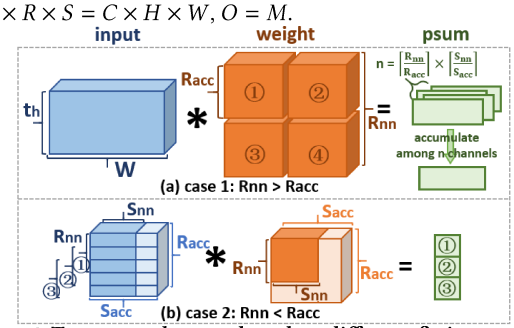

Figure 6: Two example cases based on different fitting methods

To fit various Conv layers on a single accelerator, the fitting method focuses on GBs and LBs partially defined by nnsPMs. While ALU is fully defined by μAPMs, and fits for different nnsPMs values.

At GB level, GBs storage capacity and bandwidth are fixed, for example, PR's I-GB is fixed, then $T_{m1} \times C_1 \times W_1 = T_{m2} \times C_2 \times W_2$. Different nnsPMs leads to different value of *mem* μAPMs. As introduced in sec. 3.2, higher *mem* μAPMs value could apply higher data reuse rate in GBs. At LB level, I-LBs consist of $te \times R_{acc}$ shifters and each shifter stores $t_c \times S_{acc}$ data. R_{acc}, S_{acc} is fixed and defines the accelerator, but R_{nn} and S_{nn} can take different values for each layer in a network. So, there would be two cases: $R_{nn} > R_{acc}$ and $R_{nn} < R_{acc}$. In the first case (Fig. 6(a)), based on the computation pattern of Conv, a large sized filter could be separated to several n smaller sized filters, $n = \lceil R_{acc}/R_{nn} \rceil \times \lceil S_{acc}/S_{nn} \rceil$. Those n filters are convolved with input data. Then n channels of PSums are generated. PSums are accumulated among the n channels to get

the same PSums value as the original convolution. By applying this fitting trick, in case-1, the larger R_{nn}, S_{nn} Conv could execute on the fixed accelerator. In the second case (Fig. 6(b)), the $R_{nn} \times S_{nn}$ sized SW is shifted on input data. Extra rows of input could be held in I-LB to get multiple PSums. However, due to the access bandwidth of I-GB, only $U \times S_{acc}$ sized data could be fetched each cycle. Each shifter in I-LB would not be fully utilized. In W-LB, $R_{nn} \times S_{nn}$ sized of memory is utilized to store the weight data.

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. 7, these characterization data is obtained through NNest interfaces with other simulator tools, such as Cacti [11] for SRAM, memory compiler for register file, Synopsys Design Compiler (DC) for arithmetic logic. To account for the area/latency/power of ALU array, we synthesize the basic multipliers and adder trees in datapath circuit using DC with a 40nm standard cell library, similar to the method used in Aladdin [22]. Although 40nm is used in our experiment due to limited access to process technology, the same block-level characterization can be easily ported for a new technology. NNest also accounts for nnsPMs, such as from TensorFlow, and user-specified area/timing/energy constraints. DMS and multi-layer fitting schemes allow NNest to efficiently explore the design space defined by μAPMs (Table 1). This architectural-level exploration has already taken MAC for-loop unrolling (parallelization) and hierarchical memory optimization into consideration by sweeping μAPMs values.

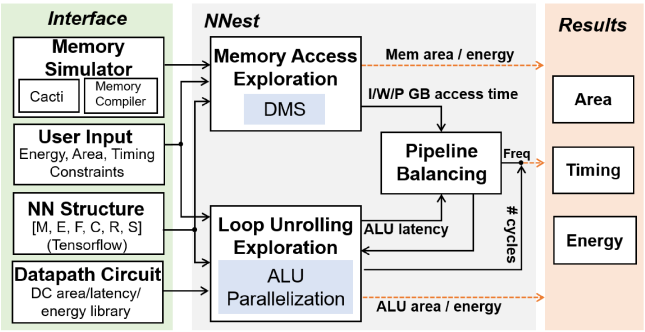

Figure 7: NNest's area/performance/energy modeling framework

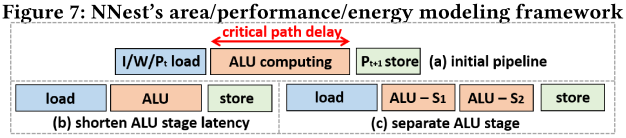

Figure 8: NNest pipeline stage

At the circuit level, the execution of NN accelerator based on spatial architecture could initially be broken down into 3 stages (Fig. 8(a)) —loading I/W/P data, computing for one cycle, and storing updated PSums. The critical paths of each stage are simulated by stacking the latency of each single blocks from memory simulator and device library. Due to the different μAPMs values, the initial 3-stage pipeline may not be balanced. Based on simulation results, we found ALU computing consumes the longest delay among the three stages. The solution is to shorten the ALU stage's latency (Fig. 8(b)) or separate one cycle computing to multiple stages (Fig. 8(c)). NNest first searches through arithmetic device library to find shorter latency multipliers or adders to reduce the computing delay, but shorter latency leads to higher area and energy consumption resulting area/energy constraints unsatisfied. If failed, NNest would break the ALU stage into multiple 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.

4 EXPERIMENT RESULTS

All the experiments are performed using 40nm CMOS technology.

4.1 Layer-Level Exploration

First, specific single-layer accelerators design spaces are explored in NNest. It costs 3.61959 seconds (3.61911 sec on Cacti) exploring one design point on Intel-i7 2.70GHz cpu with 8GB RAM platform.

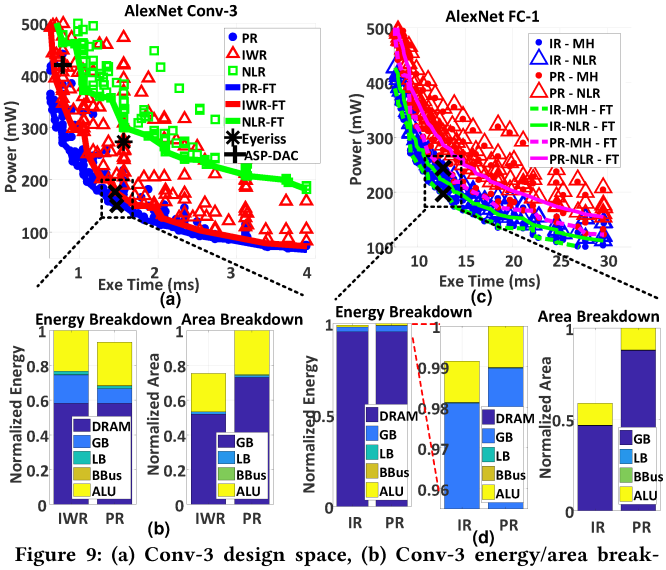

Figure 9: (a) Conv-3 design space, (b) Conv-3 energy/area breakdown, (c) FC-1 design space, (d) FC-1 energy/area breakdown

AlexNet's Conv-3 and FC-1 layers are used as examples. Results are shown in Fig 9, 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. PR's total energy is about 93.4% of IWR due to energy saving from GB access, which comes with extra area overhead. The reason is that: 1) in IWR, inputs and weights consume less memory space, but PSums need to be fetched from a large-sized P-GB every cycle and this frequent PSums load/store increases GB access energy; 2) in PR, I/W-GB is area-consuming to store the entire C channels of input and weight rows, but I/W data could be held and reused in LBs to reduce GBs access. We also mark the design points based on the dataflow strategies introduced in previous work (Eyeriss [21] and ASP-DAC [10]). 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 [10] and 39.3% compared to Eyeriss [21]).

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 outperforms PR. 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% ∼ 80% in Conv). This stems from Conv layer's local connectivity and weight sharing properties.

4.2 Network-Level Exploration

Next, we apply the multi-layer fitting scheme described in sec 3.3 to explore the design space for a complete neural network.