SoC-DNN Design Space Exploration and Optimization

Zishen Wan, Zheyu Wu
Harvard University
Cambridge, MA

ABSTRACT

The recent emergence of deep learning has reshaped many aspects of daily life and enabled numerous computer vision and natural language processing applications. However, these applications post a high computational demand on traditional processing units, including CPUs and GPUs. To mitigate this increasingly critical bottleneck, people have proposed many dedicated hardware accelerators to speed up the training and inference phase of deep neural networks. Designing these hardware accelerators involves effective tools to provide principled insights on both design trade-offs and hardware/software co-optimization.

A few simulators (e.g. SCALE-Sim[26]) do exist to enable parametric search in various microarchitecture features and neural network configurations. However, two of the most critical SoC design metrics, area and power, are not included. In addition, limited experiments were performed to exhaustive design space exploration (DSE). These challenges make it hard for this tool to be deployed in real machine learning accelerator design.

In this paper, we design AutoDSE framework, which integrates power and area estimations into the current framework provided by SCALE-Sim. Using AutoDSE, we conduct experiments to further study the effect of SRAM, dataflow and array size to runtime, area and power estimates across popular convolutional network topologies. We also analyze layer-by-layer utilization and proposed a per-layer frequency scaling mathematical model. The experiments demonstrate the benefits of layer-wise frequency scaling and validate the effectiveness of our proposed model. We believe these insights will be highly valuable to assist computer architects in designing state-of-the-art hardware accelerators.

Index Terms – Co-design, Design Space Exploration, Deep Neural Network, Frequency Scaling

1 INTRODUCTION

Since deep learning era started in 2012, a great number of deep neural networks (DNN) topologies have been proposed to advance technologies [16] that have not been imagined before: from self-driving cars to smart digital voice assistants. Among them, convolutional neural network (CNN) [12] enables real-time object detection but also posts a huge computational demand for traditional processing units (CPUs and GPUs). As Moore's law for CPUs comes to an end [27] and high power budget is required by GPUs [7], there is an imperative need to design specialized hardware accelerators that outperform CPUs and GPUs, yet are still more power-efficient to be deployed in embedded applications.

Systolic arrays [13] are one of the most popular compute substrates within deep learning accelerators today. The design of these neural network-based accelerators is a time- and resource-consuming process that involves requirements refinement, architecture-level design, microarchitecture design, implementation, verification and productization [2]. To speed up this process, architects rely on simulators to provide principled insights to guide the design process. It is not hard to predict that the need for such simulators will surge in the near future as more innovative CNN-based topologies are proposed, which require many demanding computational resources. Therefore, it is important for CNN-based accelerator simulators to allow a faster prototyping and design validation for all critical aspects of circuit design a priori.

Simulators, such as SCALE-Sim [26], do exist to help modeling systolic arrays and speed up the design cycle. It is able to calculate the cycle time for various systolic size and dataflow. However, SCALE-Sim does not incorporate power consumption and area estimation into the model. As mobile computing is ubiquitous today [21], longer battery life and smaller form factor are two of the most important criteria for customers [3]. It is thus useful to consider power consumption and area estimation in the simulation stage, which would eliminate further time and resource cost for iterative design.

On the other hand, the revolution of DNNs is enabling dramatically better autonomy in autonomous driving. However, it is not straightforward to simultaneously achieve both timing predictability (i.e., meeting job latency requirements) and energy efficiency that is essential for any DNN-based autonomous driving system, as they represent two (often) conflicting goals.

To this end, we introduce AutoDSE, an extended simulation framework based on SCALE-Sim. It allows the architect to quickly prototype and simulates systolic-array based CNN accelerators by providing a detailed breakdown of hardware performance, including but not limited to performance, power and area (PPA) estimations, layer-wise hardware utilization and memory read/write performance. Based on that framework, we conduct a comprehensive design space exploration. Furthermore, we propose a layer-wise frequency scaling mathematical model that taking timing predictability and energy efficiency into account at the same time.

To summarize, we make the following contributions:

- We propose AutoDSE, which is a simulation framework based on SCALE-Sim with extended functionality such as PPA estimations.
- Using AutoDSE, we conduct exhaustive DSE by sweeping input parameters and draw useful insights for future designs.
- We propose a concept of Determine Factor and build a layer-wise frequency scaling mathematical model. Results validate the effectiveness of our proposed model.
- We demonstrate the benefits of per-layer frequency scaling and explore the effect of layer granularity and penalty of the scheme.

The rest of paper is organized as follows. Section 2 provides the background on different DNN simulators and dynamic voltage

and frequency scaling schemes. Section 3 introduces our simulator framework, the workload and the neural network template we use. Section 4 presents the results of comprehensive design space exploration. Section 5 presents our mathematical model for layer-wise frequency scaling. Section 6 demonstrates the results of frequency scaling experiments and validate our mathematical model. Section 7 concludes the paper and goes over the future directions for this work.

2 RELATED WORK

DNN accelerator simulator

There are many public and open-source accelerator simulators. Aladdin [28] is a tool for simulating power, performance and silicon area of arbitrary accelerators. The methodology follows an HLS-inspired approach that starts with C-code, then parsed to an LLVM graph and scheduled into a hardware pipeline. It's ideal for rapid exploration of the hardware cost of a range of algorithms, but somewhat limited in the sophistication of hardware structures.

Gem5-Aladdin [29] embeds the Aladdin inside the Gem5 system simulator environment to allow for system trade-offs to be explored. However, it lacks accurate account for DNN-specific accelerators, and therefore are not readily available for studying system-level behavior of ML-based applications.

More recent work includes SCALE-Sim[26]. SCALE-Sim provides a customized interface for architects to adjust network topology and hardware configurations. Its Python-based cycle-accurate model is much faster compared to tradition RTL simulation, and it allows easy expansion to its functionalities. However, it does not provide power or area estimates for further optimization.

Dynamic Voltage and Frequency Scaling (DVFS)

At a high level, DVFS [15] is the mechanism to dynamically adjust voltage and frequency settings on computer processors. On one hand, longer battery life is expected by end users and DVFS is able to lower operation voltage and clock frequency. On the other hand, high-performance applications demand more processing capability, and DVFS is able to scale up the voltage and frequency temporarily. There are ongoing researches on different DVFS algorithms, such as [4], [6], and [19]. In this paper, we would like to use AutoDSE to explore DVFS for hardware accelerators of CNNs.

3 METHODS

3.1 Simulator Framework

We use SCALE-Sim as our simulator framework [26]. SCALE-Sim is a configurable systolic array-based cycle-accurate DNN accelerator simulator. It exposes various micro-architectural parameters such as array size (number of MAC units); array aspect ratio (array height vs. width); scratchpad memory size for Ifmap, filter, and Ofmap; dataflow mapping strategy (output/weight/input stationary); as well as system integration parameters, e.g., memory bandwidth. Taking these architectural parameters, filter dimensions of each DNN layer and the input image size as input, SCALE-Sim generates the latency, array utilization, SRAM accesses, DRAM accesses, and DRAM bandwidth requirement. Figure 1 shows the schematic representation of SCALE-Sim.

While SCALE-Sim only generates performance metrics, we augmented it with power models. The SRAM power is estimated using CACTI [18], the DRAM power is estimated using Micron's DDR4 specification and power calculator [1]. Figure 2 shows the framework of design space exploration framework.

To be specific, we model the SRAM sizes in CACTI to obtain the read/write energy per access numbers. We use this energy per access to multiply the number of read/write access estimated from SCALE-Sim to get the total SRAM energy for each model. For DRAM, we specify read ratio and write ratio generated from SCALE-Sim in Micron DRAM model. The parameters we set for DRAM is: System VDD=1.2V, System VPP=2.5V, System clock frequency=800MHz, DRAM Density=4Gb, DRAM Width=16, Speed Grade=-093, DBI Model is off. For PE, obtained from an in-house place-and-route design [17], each PE consumes 0.3pJ per cycle at 1GHz and occupies 525um2.

We also assume that there are two low power MCU class cores in the SOC. For the MCU class cores, we use E2 cores based on RISC-V ISA architecture [31]. E2 cores consume about 0.53 mW at 28 nm process and are clocked at 725 MHz[1]. We account for the power of the ultra-low-power core into our final power numbers.

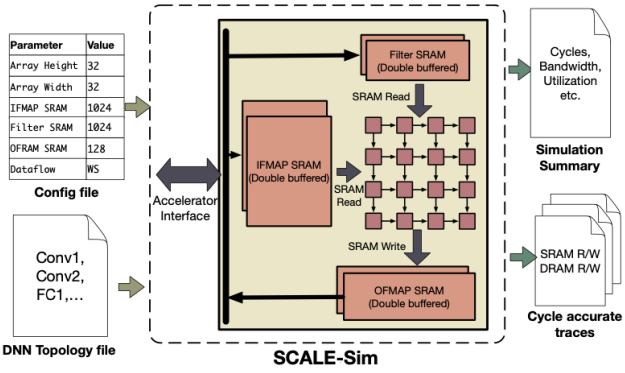


Figure 1: Schematic depicting SCALE-Sim, with inputs and outputs. The tool takes in architecture parameters and the workload hyper-parameters; and generates cycle accurate traffic traces and simulation summary

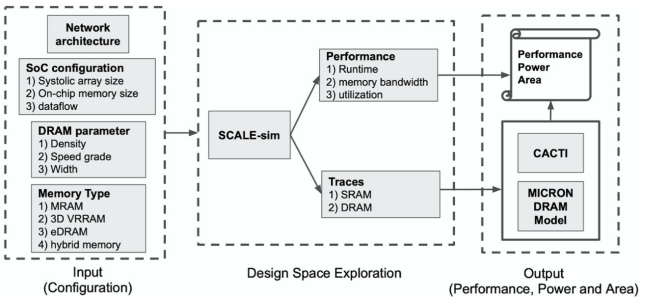


Figure 2: The overview of whole Design Space Exploration framework

[1] https://www.sifive.com/cores/e20

3.2 Workloads

The workloads in this design refer to convolutional neural network topologies for which we use not only as a way to validate the effectiveness of our framework, but also serve a starting point that allows further investigations regarding the relationship between power, area and cycle time with respect to network characteristics. The default workloads in this design include MobileNet, DroNet and YOLO-tiny, as they present a wide spectrum of popular object detection algorithms that are fast and possible to be deployed on the limited computational infrastructure that we have access. However, as will be shown in the next section, it is worth noticing that our framework allows easy integration with customized network topologies.

MobileNet [10] is an architecture that is suitable for mobile and embedded vision applications. It features a streamlined architecture that uses depth-wise separable convolutions to build light weight deep convolutional networks with comparable performance with VGG and Inception V2 when trained on the COCO dataset [20].

DroNet is an efficient convolutional network that enables real-time UAV applications. Deployed in a resource-constrained environment, DroNet is able to perform object detection of 5 - 18 FPS with an overall accuracy of 95% [14].

YOLO-tiny represents the minimalistic version of a popular real-time object detection algorithm YOLO (You Only Look Once) [25]. Similar to the above two networks, YOLO-tiny is intended for the resource-constrained environment. It is able to perform object detection with a mean average precision of 23.7 and 244 FPS [24].

3.3 Network Template

Our DSE framework also supports network architectures proposed in Air Learning, a robot simulator to train End-to-End models for aerial robot navigation. The authors of that work [11] proposed that having a multi-modal architecture is essential and each input modality contributes to the success rate for a particular task. The basic template of the architecture used in that work is shown in figure 3. Therefore, our DSE framework can not model standard networks used in computer vision tasks such as image recognition and object detection, but also can model End-to-End learning for aerial robots.

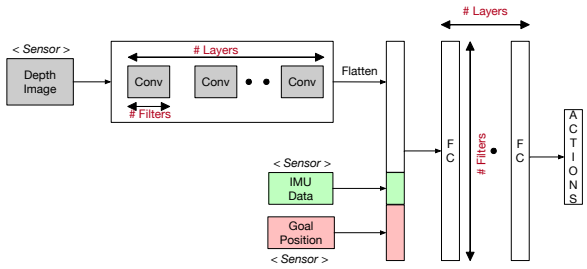


Figure 3: For evaluating different E2E models, we use the neural network architecture proposed in Air Learning as a template. Using that template we tune two parameters namely the `# Layers` and `# Filters`.

4 DESIGN SPACE EXPLORATION

Systolic array is one of the most efficient ways to implement matrix multiplication in hardware. At a high level, systolic array works by passing operands and performing computation one after another from neighbors. There is neither global communication nor address generation required, making it a natural fit for high-throughput and high-efficiency applications.

The design choice for 2D is typically not straight-forward due to a high number of adjustable parameters and various workloads involved. We used AutoDSE to sweep a series of critical parameters with the hope to get valuable design insights for 2D systolic array.

In the next few sections, we closely examined the power, cycle time, area and utilization with respect to different hardware configurations using AutoDSE with the target towards efficient accelerator design. The experiments here are similar to SCALE-Sim, with the extension of power, area and utilization estimates.

4.1 Effect of Dataflow

The term dataflow in the context of systolic array refers to the mapping strategy of computation in the systolic array. As described in Eyeriss [5], there are three main dataflows, i.e. Output Stationary (OS), Input Stationary (IS) and Weight Stationary (WS). We first provide a description of these dataflows and then examine their relationship with power and cycle time.

Output Stationary (OS)

The stationary term refers to the fact that the matrix of interest is "pinned" to a given processing element (PE). Output stationary refers to a mapping where each output pixel is pinned to a given PE. With limited resource, each PE is time-shared, meaning that once a pixel is generated by a PE, its result is saved and the PE is reassigned to a new output pixel. Input pixels are fed from the left edge of the array while the filter or weight matrices are fed from the top edge.

Weight Stationary (WS)

Weight Stationary refers to the mapping where each weight is pinned to a given PE. WS is a two-step process. The first step is assigning each column of the systolic array to a given filter. The elements of the filter matrix are fed to the column from the top edge. Once this mapping is completed, the second step is feeding input feature map from the left edge.

Input Stationary (IS)

Similar to WS, Input Stationary (IS) refers to the mapping where each input pixel is uniquely mapped to each PE. It is also a two-step process. The first step is assigning each column of the systolic array to a convolutional window, where each window is defined as the set of all pixels in the input feature map to generate one pixel in the output feature map. This is step is done by steaming elements from the convolutional window from the top edge. This step is followed by a second step, where the weight matrices are streamed from the left edge.

As shown in Figure 4 and Figure 5, different dataflows behave differently on runtime and power. For runtime, Output Stationary (OS) outperforms the other two dataflows regardless of the workload. From the power consumption perspective, the difference between dataflows shrinks. This difference is magnified when the array size changes from 8x8 to 128x128. When the array size is 128x128, OS

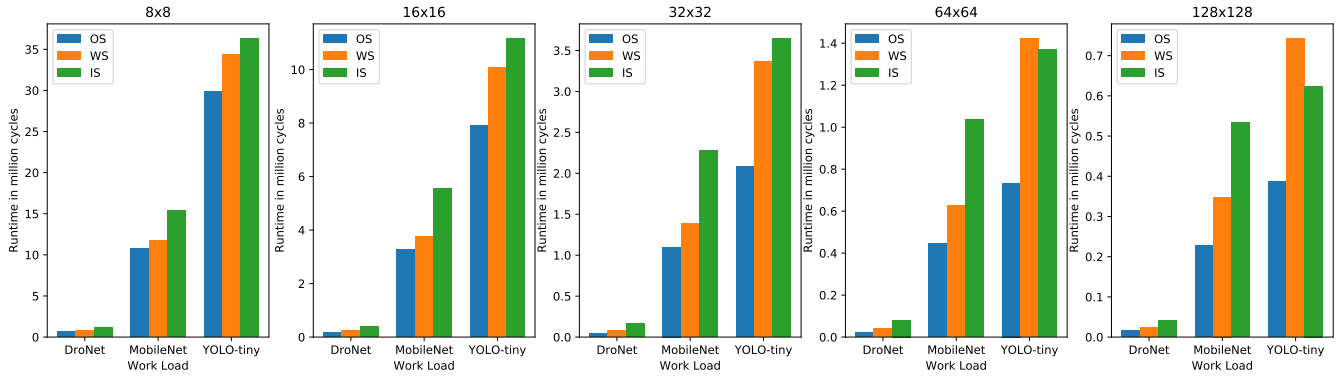


Figure 4: Cycle time with respect to dataflow type

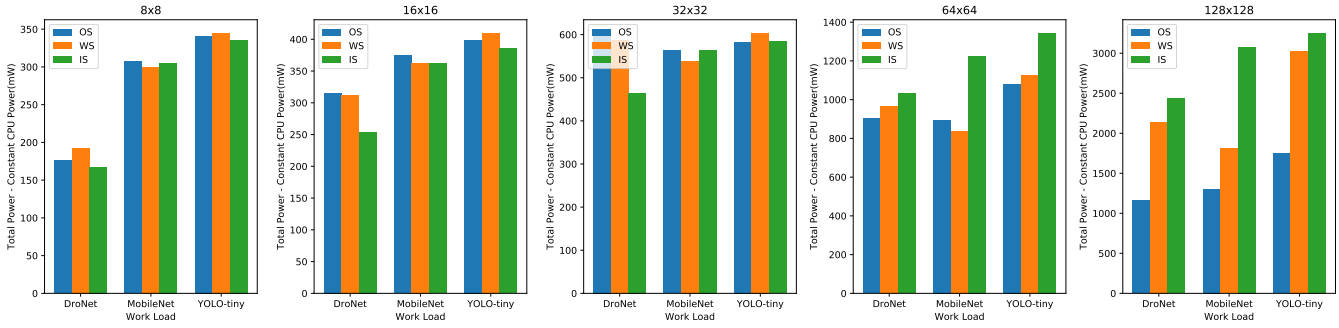


Figure 5: Power consumption with respect to dataflow type

consumes the least amount of power while Input Stationary (IS) consumes the most. The benefits brought by OS is offset by the fact that OS occupies the largest area due to extra SRAM bank usage.

4.2 Effect of SRAM Sizing

There are three types of SRAMs used in our model: Input Feature Map SRAM (IfmapSRAM), Filter SRAM (FSRAM) and Output Feature Map SRAM (OfmapSRAM). These SRAMs are used to temporarily hold data for either the input pixels, filter weights or output pixels. SRAM sizes have a different impact on the system performance, especially chip area and power. In this section, we will closely study their relationships.

Figure 7 first depicts the memory system power consumption with respect to SRAM size at a fixed systolic array size (256x256). Notice the vertical axis here is the sum of power consumed by SRAM and DRAM. This is done to highlight the fundamental memory system trade-off: If SRAM's size increases (thus SRAM's power consumption increases), then the CPU will fetch data from DRAM less frequently, leading to a lower DRAM power consumption. Overall, we found that the power reduced by less visiting the DRAM is offset by the power increased by incremental SRAM size. The total power consumed by the memory system still increases with the SRAM size.

Similarly, Figure 7 also depicts the change of SRAM area estimates with SRAM size when the array size is 256x256 . The simulated result is not surprising - SRAM area is scaled proportionally with its size.

4.3 Effect of Shape of Array

When the number of PE and dataflow are fixed, it is interesting to explore the effect of array shape on the overall system performance. With a total of 16384 elements, we used AutoDSE to adjust the size from 8x2048 to 2048x8 (i.e. from a wide array to a narrow array). As shown in Figure 8, we find a few interesting insights.

The first insight is that different workload exhibits different sensitivity to the shape of the array. Regardless of the workflow, DroNet is the least sensitive to array shape, followed by MobileNet. YOLO-tiny is the most sensitive workload. Second, the square aspect ratio (256x256) tends to perform very well compared to short-wide and tall-narrow array shapes.

4.4 Insights from Utilization

Utilization refers to the ratio between the active PEs involved in the computation and the total number of PEs for a specific network layer. Utilization is one of the key metrics in designing hardware accelerators as it can be used to track system performance and provides invaluable insights in diagnosing system bottleneck. Utilization rate implies a fundamental design trade-off: if the utilization for a given workload is too high (i.e. close to 100%), then the hardware has the risk to be stalled when dealing with a more computational-intensive task. On the other hand, if the utilization is too low, then a great portion of the hardware is not active and it introduces unnecessary hardware costs.

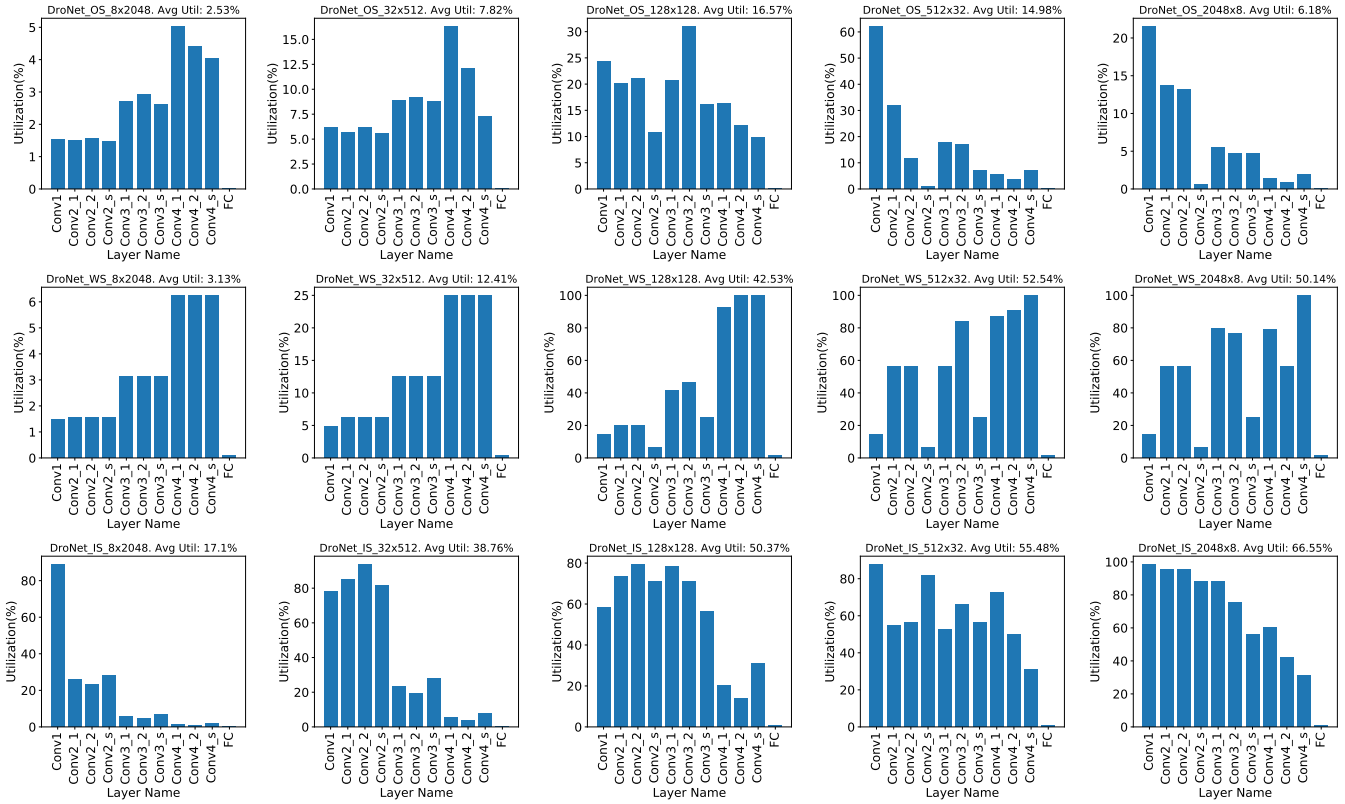


Figure 6: Layer-wise utilization for DroNet

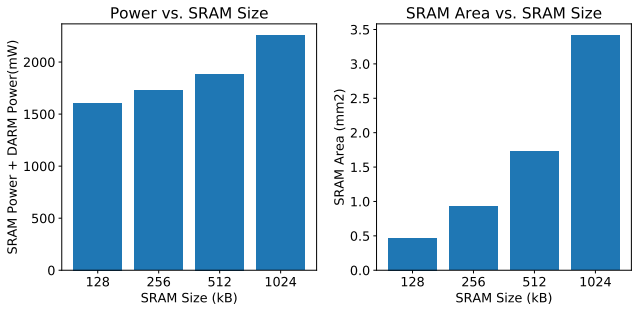


Figure 7: SRAM size impact to power and area

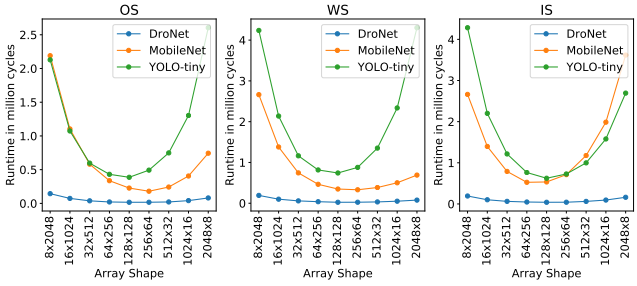


Figure 8: Cycle time with respect to array shape

Considering hardware utilization at the simulation phase allows architects to tune system hyperparameters and hardware configurations for optimized performance (see Section 5). As shown in Figure 6, we have demonstrated the layer-wise hardware utilization for systolic arrays with the same number of PEs but different shapes and dataflows under the same workload (i.e. DroNet).

The results of this experiment are very interesting. Among all three workflows under test, despite that OS has the lowest runtime and power consumption, it also has the lowest utilization rate. It implies that hardware resource reduction may be possible to reduce the hardware cost and yet remain a similar level of performance. Among all layers, fully-connected layers have the lowest utilization rate. This finding highlights one of the well-known disadvantages of systolic arrays. This makes sense as the convolutional layers are much more computationally complex and thus are more resource-hungry. As the array shape shifts from 8x2048 to 2048x8, the average utilization rate for OS generally increases first and then decreases. For IS and WS, the average utilization rate increases rapidly as the shape changes. Combining the results from 4.3, we could conclude that square-shaped systolic arrays tend to have the least runtime and relatively high utilization rate.

5 MATHEMATICAL FREQUENCY SCALING MODEL

In this section, we propose a per-layer frequency scaling method that takes both performance and power into account. Leveraging this architectural optimization method, we try to meet the real-time and energy-efficient constraints at the same time.

5.1 Motivation

Adopting inherently compute-intensive DNNs on resource- and energy-constrained UAVs often creates multiple challenges. First, they have complex and non-linear control and require continuous input to the rotors from the flight controller at a frequency of 50 Hz-100 Hz to hover [8, 22], which imposes real-time constraints. For an agile flight with very high maneuverability, the flight controller must operate at about 1 kHz [9]. Hence, the performance of deep RL is bounded by the closed-loop frequency of 100 Hz to 1 kHz, which translates to 1ms-10ms real-time constraint.

The second challenge is that UAVs are constrained by the amount of energy available on board. The UAV's size determines the on-board battery capacity. But irrespective of the UAV size, all UAVs must have the same navigation capabilities. For example, a nano-UAV like the CrazyFlie must have the same autonomous navigation capabilities as a mini-UAV like DJI-Mavic Pro. But the power available for compute for nano-UAVs is typically less than 1 Watt.

However, real-time performance and energy efficiency often come into conflict. This is because the former requires reserving sufficient resources for guaranteeing latency even in the worst case; while the latter often desires to allocate just enough resource that barely meets the needs of the current job.

From the previous analysis in section 4, we show that a unique characteristic of DNNs is that they consist of multiple layers, each of which exhibits different computation complexity and characteristic. We also notice that the energy usage pattern differs dramatically among different layers and under different system configurations. However, existing energy/latency optimization techniques have two main limitations: 1) they are usually layer-oblivious and cannot deal with these huge layer-differences very well; 2) they target the general workload that may not work for DNN-based UAVs since they do not explore and exploit per-layer characteristics. Therefore, we propose a layer-wise architectural optimization method.

5.2 Layer-wise Frequency Scaling Model

Inspired by Dynamic Voltage and Frequency Scaling (DVFS), we explore frequency scaling for different layers based on the characteristic of each layer (e.g. utilization, cycles, SRAM access time) to consider both real-time and energy-efficient constraint. Since the frequency of DRAM is fixed, we only optimize SRAM and PE power and performance, and DRAM power and performance are constant values for each specific network and thus unaffected by the frequency scaling method.

PE Power

From [17] we know that optimizing for a widely-adopted 8-bit precision for CNN inference, each PE consumes 0.3pJ per cycle at 1GHz, so the PE power of layer i is

$$P_{PE_i} = AS \times f_i \times u_i \times 0.3 \tag{1}$$

where, AS is the array size of a systolic array, f_i is the frequency of layer i, u_i is the systolic array's utilization of layer i.

To have high energy efficiency, we aim to achieve $\frac{P_{PE}(same)}{P_{PE}(scaling)} > 1$, where $P_{PE}(same)$ is the original design (using the same frequency for all layers), $P_{PE}(scaling)$ is the power with per-layer frequency scaling.

Assuming the DNN has n layers, and f is the frequency for all layers in the original design, expanding equation (1), we have

$$\frac{P_{PE}(same)}{P_{PE}(scaling)} = \frac{AS \times f \times \sum_{i=1}^{n} u_i}{AS \times \sum_{i=1}^{n} f_i u_i} = \frac{f \times \sum_{i=1}^{n} u_i}{\sum_{i=1}^{n} f_i u_i} > 1 \tag{2}$$

On one hand, in order to achieve $\frac{f \times \sum_{i=1}^{n} u_i}{\sum_{i=1}^{n} f_i u_i} > 1$ to the greatest extent, we should make $\sum_{i=1}^{n} f_i u_i$ as small as possible. On the other hand, considering the performance of overall systems, we cannot make all of f_i as small as possible, but some $f_i > f$ and some $f_i < f$ instead. Apparently, a larger u_i term has a larger impact, meaning we should assign smaller f_i to this term. Similarly, a smaller u_i term has a smaller impact, meaning we can assign a relative larger f_i to this term without making the denominator value explode. In that case, we can achieve lower power without hurting performance too much.

Hence, we can conclude:

- The layer with higher PE utilization should have lower frequency; The layer with lower PE utilization should have a higher frequency.

SRAM Power

The SRAM (dynamic) power of layer i is calculated as

$$P_{SRAM_i} = \frac{E_{SRAM_i}}{T_{SRAM_i}} = \frac{Rd_i \times E_{Rd} + Wr_i \times E_{Wr}}{(Rd_i + Wr_i)/f_i} \tag{3}$$

where E_{SRAM_i} is the SRAM energy of layer i, T_{SRAM_i} is the SRAM time of layer i, Rd_i is the the number of SRAM read access for IFMAP scratchpad and Filter scratchpad of layer i, Wr_i is the number of SRAM write access for OFMAP scratchpad of layer i, E_{Rd} is SRAM read energy per access, E_{Wr} is SRAM write energy per access, f_i is the frequency of layer i (GHz). Wr_i and E_{Rd} can be derived from CACTI, Rd_i (Wr_i) can be derived by Read bytes/Read interface (Write bytes/Write interface).

To have high energy efficiency, we aim to achieve $\frac{P_{SRAM}(same)}{P_{SRAM}(scaling)} > 1$, where $P_{SRAM}(same)$ is the original design (use the same frequency for all layers), $P_{SRAM}(scaling)$ is the power with per-layer frequency scaling.

Assuming the DNN has n layers, and f is the frequency for all layers in the original design, expanding equation (3), we have

$$\frac{P_{SRAM}(same)}{P_{SRAM}(scaling)} = \frac{\frac{E_{Rd} \times \sum_{i=1}^{n} Rd_i + E_{Wr} \times \sum_{i=1}^{n} Wr_i}{(\sum_{i=1}^{n} Rd_i + \sum_{i=1}^{n} Wr_i)/f}}{\frac{E_{Rd} \times \sum_{i=1}^{n} Rd_i + E_{Wr} \times \sum_{i=1}^{n} Wr_i}{\sum_{i=1}^{n} (Rd_i + Wr_i)/f_i}} \tag{4}$$

$$= \frac{\sum_{i=1}^{n} ((Rd_i + Wr_i)/f_i)}{(\sum_{i=1}^{n} Rd_i + \sum_{i=1}^{n} Wr_i)/f} \tag{5}$$

$$= \frac{f}{\prod_{i=1}^{n} f_i} \times \frac{\sum_{i=1}^{n} ((Rd_i + Wr_i)/f_i \times \prod_{k=1}^{n} f_k)}{\sum_{i=1}^{n} Rd_i + \sum_{i=1}^{n} Wr_i} \tag{6}$$

$$= \frac{A}{B} \times \frac{\sum_{i=1}^{n} ((Rd_i + Wr_i)/f_i \times B)}{C} \tag{7}$$

$$= D \times \sum_{i=1}^{n} \frac{(Rd_i + Wr_i)}{f_i} \tag{8}$$

$$> 1 \tag{9}$$

On one hand, in order to achieve $\sum_{i=1}^{n} \frac{(Rd_i+Wr_i)}{f_i} > 1$ to the greatest extent, we should make $\frac{(Rd_i+Wr_i)}{f_i}$ as large as possible. On the other hand, considering the performance of overall systems, we cannot make all of f_i as small as possible, but some $f_i > f$ and some $f_i < f$ instead. Apparently, a larger $(Rd_i + Wr_i)$ term has a larger impact, which means we should assign a smaller f_i to this term. Similarly, a smaller u_i term has smaller impact, which means we can assign a relative larger f_i to this term without influence the total value too much. In that case, we can achieve lower power without hurting performance too much.

Hence, we can conclude:

- The layer with less SRAM read and write access should have a lower frequency; The layer with more SRAM read and write access should have a higher frequency.

Performance

The SRAM and systolic array performance of layer i is calculated as

$$t_{PE_i+SRAM_i} = \frac{cycle_{PE_i+SRAM_i}}{f_i} \tag{10}$$

To have better real-time performance, we aim to achieve $\frac{t_{PE+SRAM}(same)}{t_{PE+SRAM}(scaling)} > 1$, where $t_{PE+SRAM}(same)$ is the original design (use the same frequency for all layers), $t_{PE+SRAM}(scaling)$ is the performance with per-layer frequency scaling.

Assuming the DNN has n layers, and f is the frequency for all layers in the original design, expanding equation (10), we have

$$\frac{t_{PE+SRAM}(same)}{t_{PE+SRAM}(scaling)} = \frac{\frac{\sum_{i=1}^{n} cycle_i}{f}}{\sum_{i=1}^{n} \frac{cycle_i}{f_i}} \tag{11}$$

$$= \frac{\prod_{i=1}^{n} f_i}{f} \times \frac{\sum_{i=1}^{n} cycle_i}{\sum_{i=1}^{n}(cycle_i/f_i \times \prod_{k=1}^{n} f_k)} \tag{12}$$

$$= \frac{A}{B} \times \frac{C}{\sum_{i=1}^{n}(cycle_i/f_i \times A)} \tag{13}$$

$$= D \times \frac{1}{\sum_{i=1}^{n}(cycle_i/f_i)} \tag{14}$$

$$> 1 \tag{15}$$

On one hand, in order to achieve $D \times \frac{1}{\sum_{i=1}^{n}(cycle_i/f_i)} > 1$ to the greatest extent, we should make $\sum_{i=1}^{n}(cycle_i/f_i)$ as small as possible. On the other hand, considering the performance of overall systems, we cannot make all of f_i as small as possible, but some $f_i > f$ and some $f_i < f$ instead. Apparently, a larger $cycle_i$ term has a larger impact, which means we should assign a lager f_i to this term to avoid this term explode. Similarly, a smaller u_i term has a smaller impact, which means we can assign a relative small f_i without exploding total value.

Hence, we can conclude:

- The layer with less SRAM+PE process cycles should have a lower frequency; The layer with more SRAM+PE process cycles should a have higher frequency.

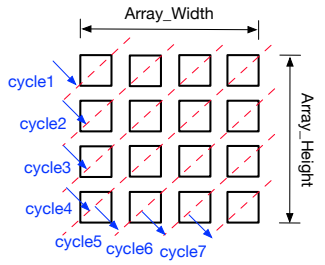


Figure 9: Propagation delay penalty in systolic array

5.3 Determine Factor

From previous mathematical analysis, we get three main layer-wise insights to achieve both higher performance and higher energy-efficient:

- Utilization ↑⟹ Frequency ↓
- #SRAM Access (Read + Write) ↑⟹ Frequency ↓
- #SRAM+PE Cycle ↓⟹ Frequency ↓

These insights motivate us to determine the frequency of each layer based on its utilization, SRAM access and cycle characteristic. To facilitate this process, we define a Determine Factor (DF_i) of layer i to analyze the frequency quantitatively:

$$DF_i = \frac{\mathbf{Norm}(utilization_i) \times \mathbf{Norm}(SRAMaccess_i)}{\mathbf{Norm}(cycles_i)} \tag{16}$$

where $\mathbf{Norm}(utilization_i)$ means the ratio of layer i utilization to the maximum utilization of all layers. $\mathbf{Norm}(SRAMaccess_i)$ and $\mathbf{Norm}(cycles_i)$ have similar meaning.

Therefore, based on our analysis, the layer with a larger DF will be assigned a lower frequency, and the layer with a smaller DF will be assigned a higher frequency.

5.4 Frequency Scaling Penalty

Scaling frequency among each layer is not free, and it will bring some cycle delay during the switching process. To model this delay penalty, we assume that systolic array should not have bubbles and if a bubble enters the SA, we have to wait for it to flush it out of the systolic array. In that case, when the layer i frequency f_i switches to layer $i+1$ frequency f_{i+1}, the process of layer $i+1$ will not start until all the process of layer i has finished in the systolic array. As usual, the global input clock signal of SoC has one fixed frequency, then we use a phase-locked loop (PLL) as the clock divider to generate different frequencies used by different layers. PLL has a delay when changing to a new phase and generating a new frequency, represented as $cycle_{PLL}$. As figure 9 shows, for a single cycle bubble to propagate through the systolic array, it takes (Array_Height + Array_Weight - 1) cycles to flush out. So the total delay cycle for is (Array_Height + Array_Weight - 1) ×$cycle_{PLL}$, and the delay time for switching from f_i to f_{i+1} is

$$t_{PLL\_delay_i} = \frac{(Array\_Height + Array\_Width - 1) \times cycle_{PLL}}{f_i} \tag{17}$$

Therefore, assuming the whole network has n different frequencies, then the total frequency switching delay of the whole neural

network is

$$t_{PLL\_delay} = \sum_{i=1}^{n} t_{PLL\_delay_i} \tag{18}$$

$$= \sum_{i=1}^{n-1} \frac{(Array\_Height + Array\_Width - 1) \times cycle_{PLL}}{f_i} \tag{19}$$

To emphasize here, if there's no frequency switching between two adjacent layers ($f_i = f_{i+1}$), then there's no switching penalty in that case.

6 EVALUATION

In this section, we perform per-layer frequency scaling on three different networks and validate our analysis. In particular, we try to answer the following questions:

- Does the per-layer frequency scaling method work? Will it achieve both shorter latency and higher energy-efficiency?
- How does the layer granularity influence the power and performance results?
- How does the PLL delay influence the power and performance results? Are these overhead negligible?
- Does the Determine Factor mathematical model work well?

6.1 Experiment Setup

We use the same simulator framework proposed in Section 5. The only change is we adopt non-volatile memory (NVM) to replace off-chip DRAM. The NVM has the same cycles and access numbers as DRAM, but it consumes 1.79pJ/bit and 86.53pJ/bit per access at 133MHz and ReadEDP optimization target [23]. To effectively evaluate per-layer frequency scaling, we exclude fixed CPU power and only model memory and PE power. In our case we will have 20 cycle bubble due to PLL delay [30], which means $cycle_{PLL} = 20$ in equation(17).

6.2 Frequency Scaling Results

We perform layer-wise frequency scaling on network template mentioned in Section 3. The parameter of the network is: array height: 64; array width: 64; # layer: 6Conv + 6FC; # filter: 48; input image size: 224x224x3; dataflow: ws; IFMAP SRAM size: 2048KB; Filter SRAM size: 512KB; OFMAP SRAM size: 512KB.

Table 1 shows per-layer cycles, utilization and access times results of the network. We can notice that Conv1, Conv2 and Conv3 have similar higher utilization and access time, Conv4, Conv5 and Conv6 are lower than the first three convolution layers. All FC layers are similar and have the lowest utilization, cycles and access time.

Based on these analyses, we experiment on three different frequency scaling schemes:

1) **2-cluster**: One frequency for Conv1-Conv6, one frequency for FC1-FC6;

2) **3-cluster**: One frequency for Conv1-Conv3, one frequency for Conv4-Conv6, one frequency for FC1-FC6;

3) **4-cluster**: One frequency for Conv1-Conv2, one frequency for Conv3-Conv4, one frequency for Conv5-Conv6, one frequency for FC1-FC6;

The baseline is all layers are processed in 1GHz. Based on analysis in Section 5, we calculate Determine Factor (*DF*) of each scheme, shown in table 2, 3, 4.

Table 1: Cycles, utilization and # access of network

Layer	Cycles	Utilization	SRAM access
conv1	49358	74.7%	55213
conv2	82131	74.5%	149057
conv3	19955	74.0%	35993
conv4	4835	71.1%	8456
conv5	1359	53.9%	2019
conv6	504	25.4%	511
fc1	834	1.1%	616
fc2	96	0.3%	40
fc3	96	0.3%	40
fc4	96	0.3%	40
fc5	96	0.3%	40
fc6	96	0.3%	40

Table 2: Determine Factor (DF) of 2-cluster scheme

Layer	Norm(C)	Norm(U)	Norm(Acc)	DF
conv1-conv6	1	1	1	1
fc1-fc6	0.0083	0.0069	0.0032	0.0177

Table 3: Determine Factor (DF) of 3-cluster scheme

Layer	Norm(C)	Norm(U)	Norm(Acc)	DF
conv1-conv3	1	1	1	1
conv4-conv6	0.0442	0.6741	0.0457	0.6520
fc1-fc6	0.0043	0.0058	0.0017	0.0148

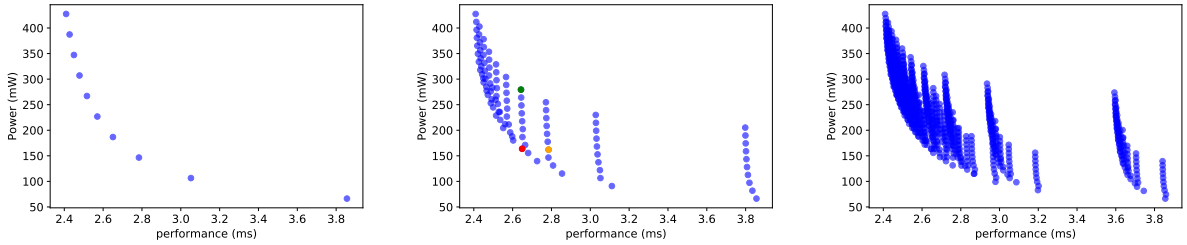
Table 4: Determine Factor (DF) of 4-cluster scheme

Layer	Norm(C)	Norm(U)	Norm(Acc)	DF
conv1-conv2	1	1	1	1
conv3-conv4	0.1885	0.9730	0.2176	0.8430
conv5-conv6	0.0141	0.5318	0.0124	0.6083
fc1-fc6	0.0033	0.0058	0.0013	0.01443

The layer with smaller DF value should assign higher frequency, so we intuitively set the frequency of FC layers larger than baseline 1GHz, frequency of Conv layers smaller than baseline 1GHz.

For 2-cluster scheme, we set the frequency of FC layers as 1.3GHz, 1.5GHz and 1.7GHz separately, and sweep frequency of Conv layers from 0.1GHz to 1GHz (step 0.1GHz). The performance and power results are shown in figure 11. We can notice that the points overlap under three different cases, which means the frequency difference of FC layers has minimal effect for the whole network. Therefore, we will fix FC Layer at 1.5GHz for the following experiments.

Then we perform layer-wise frequency scaling experiments for 2-cluster, 3-cluster and 4-cluster cases. We sweep the frequency of Conv layers from 0.1GHz to 1GHz, step by 0.1GHz each, figure 10 shows the power and performance results of three different schemes.



(a) 2-cluster frequency scaling. (b) 3-cluster frequency scaling. (c) 4-cluster frequency scaling.

Figure 10: Power and Performance results for different layer-wise frequency scaling schemes

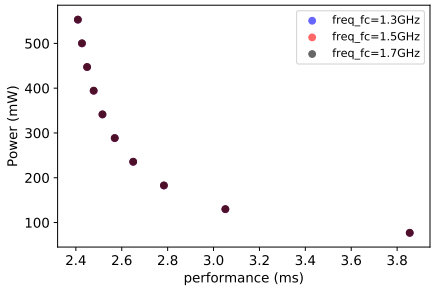


Figure 11: Power and performance under different frequency of FC layers

- **Does the per-layer frequency scaling method works? Will it achieve relatively both shorter latency and higher energy-efficiency?**

From figure 10, we notice that we can get a Pareto-curve of power and performance through frequency scaling, which can help us meet real-time and power constraints at the same time.

In addition, there are huge power and performance improvements after layer-wise frequency scaling. In figure 10 (b), red point ([0.5GHz, 0.1GHz, 1.5GHz]) has same performance as green point ([0.4GHz, 1GHz, 1.5GHz]), but only has 0.57x power consumption. Red point ([0.5GHz, 0.1GHz, 1.5GHz]) has same power as orange point ([0.3GHz, 0.4GHz, 1.5GHz]), but achieve 0.95x latency reduction. [x,y,z] represents the frequency of each layer cluster. That observation reveals there's a huge space to perform optimization.

- **How does the layer granularity influence the power and performance results?**

Compare the results of three different frequency cluster schemes. Table 5 shows the optimal power consumption under various real-time constraints, and we can see 4-cluster schemes consume the lowest power under all three constraints, followed by 3-cluster and 2-cluster, which indicates fine-grained frequency scaling can achieve better results. But we also notice that 3-cluster and 4-cluster results are closer compared to 2-cluster and 3-cluster, which indicates the benefits of fine-grained will decrease with the increase of granularity. Table 6 shows the optimal performance under various power constraints, and we can notice a similar trend.

Table 5: Lowest power consumption under real-time constraint for different frequency scaling schemes

	<2.6ms	<2.8ms	<3.2ms
2-cluster	226.832mW	146.59mW	106.46mW
3-cluster	187.89mW	139.60mW	90.89mW
4-cluster	187.97mW	131.32mW	82.88mW

Table 6: Lowest latency under power constraint for different frequency scaling schemes

	<100mW	<200mW	<300mW
2-cluster	3.86ms	2.65ms	2.52ms
3-cluster	3.11ms	2.59ms	2.45ms
4-cluster	2.98ms	2.56ms	2.45ms

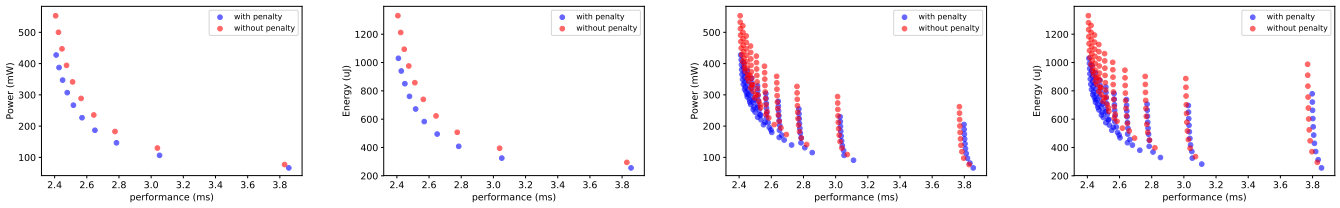
- **How does the PLL delay influence the power and performance results? Are these overhead negligible?**

Figure 12 compares frequency scaling results under with and without switching penalty. We can notice that if there no penalty during frequency switching, the latency of the whole system will be reduced, but the power and energy will be increased, and the difference is not minimal. Most energy-related literature might assume that frequency change and DVFS, in general, are free, but our results show that frequency scaling might not have a negligible overhead.

- **Does the Determine Factor mathematical model work well?**

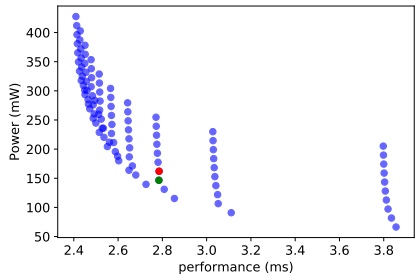
To validate the effectiveness of our DF mathematical model, we compare the points directly derrived from DF model and the optimal point got after sweeping the frequency parameters. Our previous Determine Factor analysis shows that for 3-cluster, the ratio of conv1-3:conv4-6:fc1-6 is 1:0.65:0.015. If we set real-time constraint < 2.8ms, then based on DF analysis, we will choose 0.3GHz, 0.4GHz, 20GHz for con1-3, conv4-6 and fc1-6 separately. Considering frequency limits, we choose (0.3GHz, 0.4GHz, 1.5GHz), which is red point in figure 13 (a). The optimal green point derived from sweeping parameters is (0.3GHz, 0.3GHz, 1.5GHz). We can see the red is very close to the optimal green point, which indicates that our simple DF model works well.

For 4-cluster analysis, our Determine Factor analysis shows that conv1-conv2:conv3-conv4:conv5-conv6 is 1:0.843:0.608:0.015. If we

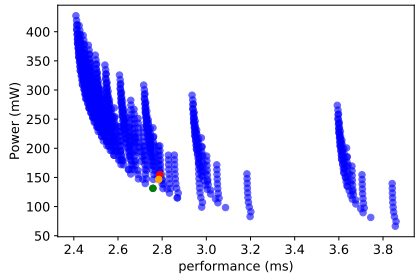


(a) 2-cluster performance vs. power. (b) 2-cluster performance vs. energy. (c) 3-cluster performance vs. power. (d) 3-cluster performance vs. power.

Figure 12: Frequency scaling results under with and without penalty



(a) 2-cluster frequency scaling. < 2.8ms constraint, green point is the optimal point and red point is directly derived from DF model



(b) 3-cluster frequency scaling. < 2.8ms constraint, green point is the optimal point, red and orange point are directly derived from DF model

Figure 13: Evaluation of DF mathematical model

set real-time constraint < 2.8ms and fix the frequency of fc layer is 1.5GHz, then based on DF analysis, we will choose 0.3GHz, 0.3GHz, 0.4GHz, 1.5GHz or 0.4GHz, 0.4GHz, 0.6GHz, 1.5GHz for con1-2, conv3-4, conv5-6 and fc1-6 separately. That corresponds to the red point and orange point in figure 13 (b). The optimal green point got from sweeping parameters is (0.4GHz, 0.2GHz, 0.2GHz, 1.5GHz). We can see the red and orange points are very close to the optimal green point, which further indicates our simple DF model works well. We can get the rough optimal frequency combination without performing a thorough and time-consuming sweeping process.

We also conduct similar experiments on the other two networks: 1) 7 Conv + 7 FC layers, 128x64 systolic array, and 2) yolotiny, 128x128 systolic array. The results shows the same trend as the plots shown before, due to the limited space, we won't show all of them.

7 CONCLUSION

In this paper, we presented AutoDSE, which is a simulation tool that provides high flexibility that allows architects to quickly prototype hardware accelerators by sweeping architectural parametric configurations and network topologies, and provides run time, layer-by-layer hardware utilization, SRAM bandwidth usage, power and area analysis. We believe AutoDSE could serve as a great tool in aiding future CNN accelerator design.

By running AutoDSE, we performed a few preliminary experiments with the hope to draw some design insights. We first adjusted the dataflow type for each network topology, and measured runtime and power estimates. Among all dataflows, Output Stationary (OS) achieved the fastest runtime and consumed a moderate amount of power. We also varied the array size from 8x2048 to 2048x8, and observed that square-shaped arrays tend to have the least runtime. Increased SRAM size leads to increased power consumption and area estimates. Layer-wise hardware utilization is the lowest for systolic arrays with OS, which indicates that we could have the potential to further optimize hardware accelerators using OS as the primary dataflow.

We explore the per-layer frequency scaling on different neural networks, and demonstrate the per-layer frequency scaling can take both power and performance into account and achieve better results. We also demonstrate a more fine-grained frequency scaling will bring more benefits than coarse-grained schemes, but the benefits will decrease as the increase of layer granularity. In terms of penalty, we show that frequency switching is not free, as mentioned in previous work. We should consider the overhead when applying this method. Finally, we build a Determine Factor mathematical model based on the power and performance calculation. We demonstrate that the optimal point derived from the mathematical model is very close to the global optimal point which is produced after sweeping all parameters. This provides us a fast way to find optimal frequency characteristics for each layer.

There are some directions for future work: (1) integrate SCALE-Sim with Gem5-Aladdin for full-system simulation; (2) automate the design space exploration with machine learning methods, such as Bayesian Optimization or Reinforcement Learning; (3) Apply per-layer frequency scaling and voltage scaling on real SoC or processors and make it more realistic; (4) Explore more memory technologies design space explorations for mobile DNNs using our framework.

8 ACKNOWLEDGEMENT

The authors gratefully acknowledge Prof. David Brooks, Iulian Valentin Brumar and Lillian Pentecost for helpful discussion, and Glenn Holloway for technical support.

REFERENCES

[1] Micron ddr4 power calculator. https://www.micron.com/~/media/documents/products/power-calculator/ddr4_power_calc.xlsm, 2016.
[2] Integrated circuit design, Nov 2019.
[3] What is considered a small android phone in 2019, and what are your options?, Dec 2019.
[4] R. Begum, M. Hempstead, G. Srinivasa, and G. Challen. Algorithms for cpu and dram dvfs under inefficiency constraints. In *Proceedings of the 34th IEEE International Conference on Computer Design, ICCD 2016*, Proceedings of the 34th IEEE International Conference on Computer Design, ICCD 2016, pages 161–168, United States, 11 2016. Institute of Electrical and Electronics Engineers Inc.
[5] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *SIGARCH Comput. Archit. News*, 44(3):367–379, June 2016.
[6] Y.-L. Chen, M.-F. Chang, C.-W. Yu, X.-Z. Chen, and W.-Y. Liang. Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems. *Sensors*, 18:3068, 09 2018.
[7] S. Collange, D. Defour, and A. Tisserand. Power consumption of gpus from a software perspective. In *International Conference on Computational Science*, pages 914–923. Springer, 2009.
[8] A. Faust, H. Chiang, N. Rackley, and L. Tapia. Avoiding moving obstacles with stochastic hybrid dynamics using pearl: Preference appraisal reinforcement learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 484–490, May 2016.
[9] D. Gurdan, J. Stumpf, M. Achtelik, K. Doth, G. Hirzinger, and D. Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1 khz. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 361–366, April 2007.
[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.
[11] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi. Air learning: An AI research platform for algorithm-hardware benchmarking of autonomous aerial robots. *arXiv:1906.00421*, 2019.
[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
[13] H. Kung and C. E. Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282. Society for Industrial and Applied Mathematics, 1979.
[14] C. Kyrkou, G. Plastiras, T. Theocharides, S. I. Venieris, and C.-S. Bouganis. Dronet: Efficient convolutional neural network detector for real-time uav applications. *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar 2018.
[15] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, 2010.
[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
[17] H. Li, M. Bhargava, P. N. Whatmough, and H.-S. P. Wong. On-chip memory technology design space explorations for mobile deep neural network accelerators. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 131:1–131:6, New York, NY, USA, 2019. ACM.
[18] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques. In *Proceedings of the International Conference on Computer-Aided Design*, pages 694–701. IEEE Press, 2011.
[19] W.-Y. Liang, P.-T. Lai, and C. W. Chiou. An energy conservation dvfs algorithm for the android operating system. 2011.
[20] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
[21] K. Lyytinen and Y. Yoo. Ubiquitous computing. *Communications of the ACM*, 45(12):63–96, 2002.
[22] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The grasp multiple micro-uav testbed. *IEEE Robotics Automation Magazine*, 17(3):56–65, Sep. 2010.
[23] L. Pentecost, M. Donato, B. Reagen, U. Gupta, S. Ma, G.-Y. Wei, and D. Brooks. Maxnvm: Maximizing dnn storage density and inference efficiency with sparse encoding and error mitigation. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 769–781, New York, NY, USA, 2019. ACM.
[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2015.
[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
[26] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna. Scale-sim: Systolic cnn accelerator simulator, 2018.
[27] R. R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997.
[28] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. *SIGARCH Comput. Archit. News*, 42(3):97–108, June 2014.
[29] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.
[30] F. u. Rahman, R. Pamula, A. Boora, X. Sun, and V. Sathe. 19.1 computationally enabled total energy minimization under performance requirements for a voltage-regulated 0.38-to-0.58v microprocessor in 65nm cmos. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 312–314, Feb 2019.
[31] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic. The risc-v instruction set manual, volume i: Base user-level isa. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, 116, 2011.