

ABSTRACT

ZHAO, QIANLI. An Analytical Framework for PPA Estimation of Systolic Array-Based Hardware Accelerators. (Under the direction of Dr. Rhett Davis).

Computing power is essential for artificial intelligence (AI), as it enables the training and inference of complex and large-scale models. However, conventional general-purpose processors such as GPUs are often inefficient and costly for AI workloads, which require high parallelism and low memory latency. Systolic array-based hardware accelerators are a promising alternative for computer vision applications, as they can achieve high performance and energy efficiency by exploiting varied data-reuse methods. Systolic arrays are special-purpose processors that consist of an array of multiply-accumulate (MAC) units, which communicate operands and results using local register-to-register communication only, thus avoiding the overhead of accessing external memory. However, designing and optimizing systolic array-based accelerators is challenging, as it involves many trade-offs among performance, power, and area (PPA). For example, the size, shape, and direction of the systolic array can affect the throughput, latency, and resource utilization of the accelerator. Moreover, different AI applications may have different requirements and preferences for the PPA metrics, which makes it difficult to find a one-size-fits-all solution.

In this paper, we present a method that can facilitate the development of systolic array-based accelerators for AI applications. Our method can perform fast PPA analysis for different configurations and architectures of systolic arrays, and provide insights and guidance for the accelerator designers. Additionally, we present the first simulation study of using systolic array-based accelerators to train MoCo[15]. We explore the impact of different systolic array parameters, such as size and bus bandwidth, on the PPA of the

accelerator. We also compare the PPA of our systolic array-based accelerator with that of a GPU-based solution, and show that our accelerator can achieve significant advantages in terms of performance, power, and energy efficiency. Our results show that our systolic array-based accelerator can achieve significant PPA advantages over the GPU-based solution. In particular, when the systolic array size is 512x512, the systolic based accelerator with weight stationary algorithm can train MoCo[15] with nearly 6.93x less energy than the GPU, while the training time is only 1.26x longer than the GPUs based system. This demonstrates that systolic array-based accelerators are well-suited for training large-scale self-supervised learning models such as MoCo[15], and can potentially enable new applications and scenarios for AI such as transformer models.

© Copyright 2024 by Qianli Zhao

All Rights Reserved

An Analytical Framework for PPA Estimation of Systolic Array-Based Hardware Accelerators.

by
Qianli Zhao

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina
2024

APPROVED BY:

Rhett Davis
Committee Chair

Paul Franzon

Tianfu Wu

Mengmeng Zhu

DEDICATION

To my family: my parents and my wife.

BIOGRAPHY

Qianli Zhao was born in August 1991, Suzhou, Anhui, China. He received bachelor and master's degrees from ECE departments of Southern Polytechnic State University and University of Central Florida in 2014 and 2017 respectively. During his bachelors, he worked as a teaching assistant for Robotics class for Dr. Ying Wang.

During his Ph.D. he had three graduate internship experiences. He worked as a graduate intern in the Physical Design team at Intel(May 2021 - August 2021). He also worked as a graduate intern in the CAD Performance Verification team at Intel(June 2022 - August 2022). Last but not least, he worked as a graduate intern in the Post-silicon Power Analysis team at Ampere Computing(May 2023 - August 2023). His research interests include PPA analysis of systolic array based hardware accelerators for different deep learning algorithms.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Rhett Davis, for his invaluable guidance, encouragement, and support throughout my PhD journey. He has been a great mentor and a friend, who always inspired me to pursue excellence in research and life. He not only helped me to develop my research skills and critical thinking, but also provided me with many resources to help improve my research efficiency. I am also thankful to my committee members, Dr. Paul Franzon, Dr. Tianfu Wu, and Dr. Mengmeng Zhu, for their constructive feedback and helpful suggestions at different stages of my PhD. They have always been supportive and responsive, and challenged me to improve the quality of my dissertation..

Also, I owe my deepest appreciation to my wife, Molly Zhang, for her unconditional love, emotional comfort, and practical support throughout these five years. She has been my rock and my sunshine, who always stood by me and cheered me up. She has sacrificed a lot for me and our family, and I cannot thank her enough. I am also indebted to my parents, Weitao Zhao and Meiling Zhang, for their everlasting love and encouragement. They have always believed in me and supported me in pursuing my dreams. They have given me the best gift of all: education.

We would like to express our gratitude to the developers of ChatGPT, an advanced language model, which significantly contributed to the optimization of English sentences in this paper. Its ability to generate fluent and coherent text greatly enhanced the clarity and readability of our work.

TABLE OF CONTENTS

LIST OF TABLES	VII
LIST OF FIGURES	VIII
Chapter 1: Introduction	1
1.1 Motivation	1
1.2 Contribution	4
1.3 Outline	5
1.4 Abbreviations	6
Chapter 2: Background	8
2.1 Fundamental of Artificial Neural Networks	8
2.2 Self-Supervised learning	14
2.2.1 Introduction	14
2.2.2 MoCo	18
2.3 High Level Synthesis	21
2.4 Systolic Array Overview	23
2.4.1 Systolic Array Architecture	23
2.4.2 Data Reuse Methods	23
Chapter 3: State-of-the-art	31
3.1 State-of-the-art Frameworks for Systolic Array based Hardware Accelerators ...	31
3.2 State-of-the-art Hardware Simulators	37
Chapter 4: Model for PPA Estimation	41
4.1 Introduction	41
4.2 Related Tools	42
4.3 PPA Estimation	43
4.3.1 Performance Estimation	44
4.3.2 Power & Area Estimation	46
4.3.3 Summary of the Estimation	50
4.4 MoCo Simulation	51

	VI
4.4.1 Performance Estimation	51
4.4.2 Power & Area Estimation	55
4.4.3 Summary of MoCo Simulation	56
4.5 Result	62
4.6 Analysis and Conclusion	71
4.6.1 Systolic Array Analysis	72
4.6.2 Systolic Array vs GPU	75
Chapter 5: Conclusion and Future Work	81
5.1 Summary of Contribution	81
5.2 Future Work	83
Reference	85
Appendices	93
Appendix A ResNet50 Forward Propagation Configuration on ScaleSim	94
Appendix B ResNet50 Backward Propagation Configuration on ScaleSim	96
Appendix C ResNet50 Loss Function Configuration on ScaleSim	98

LIST OF TABLES

Table 3.1 Summary of State-of-the-art DNN Simulators	40
Table 4.1 Simulation Configuration	52
Table 4.2 An Example of ScaleSim Input	53
Table 4.3 An Example of ScaleSim Output	54
Table 4.4 Logits Calculation	54
Table 4.5 Key Information in Testbench.cpp	55
Table 4.6 Synthesis Result using 45 nm Technology	56
Table 4.7 Default Training Information of MoCo	56
Table 4.8 Volta GPU Key Configuration	57
Table 4.9 MoCo One Mini-batch Training Information in a 16x16 Systolic Array	58
Table 4.10 MoCo Whole Training Information in a 16x16 Systolic Array (45 nm)	59
Table 4.11 MoCo Whole Training Information in a 16x16 Systolic Array(12 nm)	61
Table 4.12 PPA Results to Train MoCo base on a 16x16 Systolic Array	61
Table 4.13 MoCo Training Information on Different Systems	70

LIST OF FIGURES

Figure 2.1 AN's Working Principle	9
Figure 2.2 Elementary constituents of CNNs	10
Figure 2.3 ResNet-50[20] Architecture	13
Figure 2.4 Deep Learning	17
Figure 2.5 Supervised Learning	17
Figure 2.6 Self-Supervised Learning	18
Figure 2.7 MoCo	19
Figure 2.8 Pseudocode of MoCo[15]	22
Figure 2.9 Generic Architecture of a Systolic Array Accelerator	24
Figure 2.10 WS	27
Figure 2.11 IS	28
Figure 2.12 OS	29
Figure 4.1 Performance Estimation Model	45
Figure 4.2 Power & Area Estimation Model	47
Figure 4.3 Whole Estimation Model	50
Figure 4.4 AN's Working PrincipleArea vs Systolic Array Size	64
Figure 4.5 MoCo Training Time vs Systolic Array Size	65
Figure 4.6 Power vs Systolic Array Size	65
Figure 4.7 Energy Consumption vs Systolic Array Size.....	67
Figure 4.8 SRAM Bandwidth Requirements vs Systolic Array Size	68
Figure 4.9 DRAM Bandwidth Requirements vs Systolic Array Sizer	68
Figure 4.10 Systolic Array usage percentage vs Systolic Array Size	69
Figure 4.11 Systolic Array vs GPU(Energy Consumption)	76
Figure 4.12 Systolic Array vs GPU(Training Time)	77
Figure 4.13 Systolic Array vs GPU(Required DRAM BW)	78

CHAPTER 1

Introduction

1.1 Motivation

Artificial intelligence (AI) is the field of computer science that aims to create machines and systems that can perform tasks that normally require human intelligence and abilities. AI has been rapidly developing and expanding its applications in various domains of society, such as education, social networks, health care, entertainment, security, and business. Some examples of AI systems are virtual assistants, self-driving cars, facial recognition, recommender systems, and chatbots[1, 2, 3, 4, 5, 6, 7, 8, 9]. These systems can provide convenience, efficiency, accuracy, and safety for human users, as well as generate new knowledge and insights from large amounts of data.

One of the key factors that drive the progress of AI is the availability and quality of data, which can be used to train and evaluate AI models[10]. In the domain of natural language processing (NLP), which deals with the analysis and generation of natural language, it has been shown that larger models tend to achieve higher accuracy and performance on various tasks, such as text summarization, machine translation, and sentiment analysis[11]. GhatGPT[12], a massive pre-trained language model with 175 billion parameters, is one of the state-of-the-art examples of this trend. Many researchers believe that a similar phenomenon will occur in the domain of computer vision (CV), which deals with the analysis and generation of visual information, such as images and videos. However, this also implies that more data and computational resources are needed to train and run these large models, which poses a significant challenge for the development of AI.

Training large AI models requires a huge amount of computational resources, which are often scarce and expensive[14]. Most of the current AI models are trained using graphics processing units (GPUs), which are specialized hardware devices that can perform parallel computations efficiently. However, GPUs also consume a lot of energy and generate a lot of heat, which limits their scalability and sustainability. For instance, one of the most popular Self-supervised learning algorithms – MoCo[15] requires 8 high performance GPUs to train for 53 hours, in order to complete 200 epochs training with ImageNet-1M[16] as the dataset. During the process, the 8 GPUs consume a lot of energy, and according to calculation, it needs to consume about 106000 joules of energy to finish above training. Therefore, alternative hardware solutions are needed to enable more efficient and effective AI training.

One of the promising candidates for such solutions is the systolic array, a hardware architecture that consists of a grid of processing elements that can perform matrix operations in a synchronized and pipelined manner. Systolic arrays have several advantages over GPUs, such as lower power consumption, higher memory bandwidth, and simpler design and implementation. These advantages make systolic arrays suitable for AI training, especially for deep neural networks, which rely heavily on matrix multiplications[17].

However, developing a systolic array-based hardware accelerator from scratch is a costly and time-consuming process, which requires a lot of expertise and resources[17]. Moreover, there is a lack of effective tools and methods to design and evaluate systolic array-based hardware accelerators for different AI applications[17]. This leads to the

following research question: How can we facilitate the design and evaluation of systolic array-based hardware accelerators for AI training?

In this work, we propose a model that can address this research question. Our model can provide researchers with a fast and accurate estimation of the power, performance, and area (PPA) of different systolic array configurations, without the need to build the actual hardware. This can help researchers to explore the design space and optimize the trade-offs of systolic arrays for different AI applications. Moreover, we demonstrate the feasibility and effectiveness of our model by simulating the training process of MoCo[15], a state-of-the-art self-supervised learning method for CV, using a systolic array-based hardware accelerator. We compare the results with those obtained using a GPU, and show that our systolic array-based hardware accelerator can achieve a 6.57x reduction in energy consumption and only a 1.26x increase in training time, under the condition that the systolic array size is 512 x 512 and the bus bandwidth requirement is 39.99 GB/s.

1.2 Contribution

The main contribution of this paper is to develop a model that can quickly estimate the PPA of hardware accelerators based on systolic array architecture. Systolic array is a popular design choice for implementing neural network algorithms on hardware, as it can achieve high parallelism and throughput with low latency and memory footprint. However, designing and optimizing systolic array hardware accelerators is a challenging task, as it involves many trade-offs among PPA, algorithm, and input parameters. Therefore, a fast and accurate PPA estimation tool is essential for exploring the design space and evaluating the feasibility of different systolic array configurations. The specific contributions of this paper are as follows:

1. We propose an estimation model to predict the performance of systolic array hardware accelerators. The performance model is scalable, which is able to predict the performance of the accelerator for different sizes of input, weight, and output, and thus for new neural networks trained on the same systolic array accelerator(Chapter 3 shows why the existing methods are not sufficient).
2. We use high-level synthesis (HLS) methods to estimate the area and power of the systolic array hardware accelerators. By using HLS, we can avoid the tedious and error-prone process of manual hardware design, and obtain the area and power estimates in a short time.
3. The model is flexible, and can perform PPA estimation for different sizes of systolic array and different algorithms' training processes. The platform can support various manufacturing technologies for different designs.

4. The model fully completes the training prediction of one of the most popular models: MoCo[15], that is, using a specific size of systolic array to train MoCo[15], and obtaining the PPA prediction of the systolic array.
5. This paper also compares the results of our model predicting the PPA of the systolic array hardware accelerator training MoCo[15] with the PPA of the GPU training MoCo[15]. We then analyze the advantages and disadvantages of each platform. We also discuss the factors that affect the PPA comparison, such as the systolic array size and the DRAM bandwidth.

1.3 Outline

The remaining chapters of this dissertation will organized as follows:

Chapter 2 introduces the fundamentals of artificial neural networks, including CNNs. It then focuses on self-supervised learning, a novel and powerful technique for learning from unlabeled data, and MoCo[15], a state-of-the-art self-supervised learning method that we use as our target algorithm for simulation and estimation. Finally, it reviews some basic concepts of HLS, a technique that we use to estimate the area and power of the systolic array hardware accelerators. It also discusses the most advanced data reuse methods, which are essential for improving the performance and energy efficiency of systolic array hardware accelerators.

Chapter 3 surveys the state-of-the-art platforms for PPA estimation of systolic array hardware accelerators.

Chapter 4 describes the design and implementation of our evolving model, which consists of two main components: a performance estimation model and an HLS-based area and power estimation tool. It also explains how we use our model to estimate the PPA of the systolic array hardware accelerator for training MoCo[15], and how we validate the accuracy of our estimates by comparing them with the results from a physical design tool and a power meter.

Chapter 5 summarizes the main contributions and findings of this dissertation, and reflects on what I have learned from this project. It also suggests some possible directions for future work to further improve and extend the model and its applications.

1.4 Abbreviations

AI	Artificial Intelligence
NLP	NLP
AN	Artificial neuron
ANNs	Artificial neural networks
CV	Computer Vision
CNNs	Convolutional Neural Networks
DNNs	Deep Neural Networks
DCNNs	Deep Convolutional Neural Networks
MoCo	Momentum Contrast for Unsupervised Visual Representation Learning
RISCV	Reduced Instruction Set Computer Five
CPU	Central Processing Unit
GPU	Graphics Processing Unit

TPU	Tensor Processing Unit
PE	Processing Unit
NoC	Network on Chip
ASIC	Application-Specific Integrated Circuit
PPA	Power, Performance and Area
SRAM	Static Random-Access Memory
DRAM	Dynamic Random-Access Memory
MAC	Multiplication and Accumulation
Fmap	Feature Map
HLS	High Level Synthesis
RTL	Register Transfer Level
WS	Weight Stationary
IS	Input Stationary
OS	Output Stationary
DSE	Design Space Exploration
CGRA	Coarse-Grained Reconfigurable Array
BOP	Bit Operations Performed
DDDG	Dynamic Data Dependence Graphs
TLM	Transaction-Level Modeling

CHAPTER 2

Background

This chapter introduces the working principle of artificial neural networks, as well as various types of popular neural networks and their applications. Next, we describe our target neural network category, self-supervised learning, and a specific neural network, MoCo[15]. Then we introduce high-level synthesis, which is the method we use in our research. At the end of this chapter, we give an overview of systolic arrays.

2.1 Fundamental of Artificial Neural Networks

Artificial neural networks are computational models inspired by the structure and function of the human brain. They consist of a large number of interconnected processing units, called neurons, that can learn from data and perform complex tasks. ANNs have been widely used in various fields, such as computer vision, natural language processing, speech recognition, and biomedical engineering.

Figure 2.1 illustrates the working principle of an AN, which is a computational model of a biological neuron and the fundamental unit of an ANN. An AN has three main elements: synaptic weights, a threshold, and an activation function[18]. The synaptic weights (W) are associated with each input and affect the input vector (X) by multiplying it with the weight matrix ($WT X$). The weighted-input ($WT X$) is then added to an internal threshold, called bias, which can influence the output activation. The activation function then transforms the weighted-input into the output, which is sent to other neurons in a similar way as the biological neuron sends the sum of the incoming signal to other neurons through the axon. ANNs use different kinds of activation functions

(Sigmoid, Tanh, ReLU, etc.) to introduce non-linearity in ANN. An ANN has one input layer, one output layer, and one or more hidden layers. DNNs are neural networks with more than one hidden layer. For simplicity, the rest of the dissertation will use neuron and neural network to refer to artificial neuron (AN) and artificial neural network (ANN) respectively.

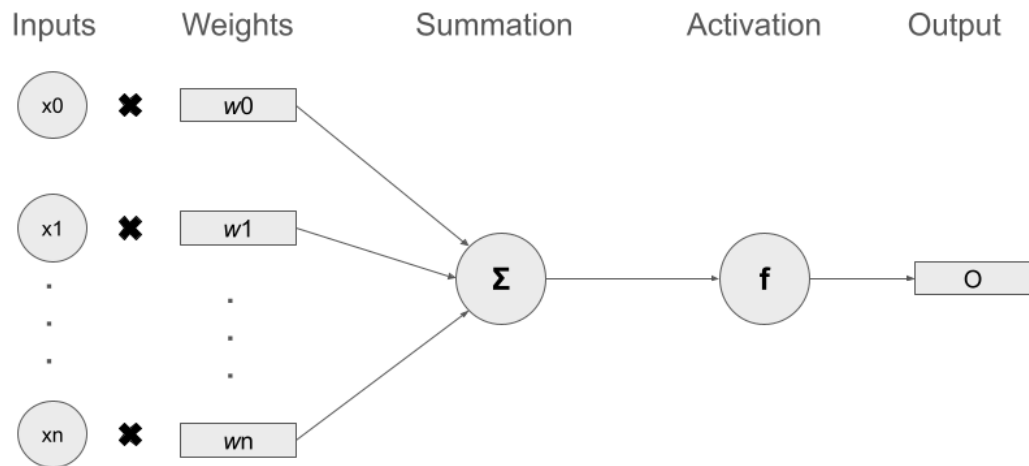


Figure 2.1 AN's Working Principle

There are many types of ANNs, each with its own characteristics and applications. Some of the most popular types are:

1. Convolutional neural networks (CNNs): These are a special type of FFNNs, where the neurons are arranged in a grid-like structure, and each neuron is connected to a local region of the previous layer. CNNs can exploit the spatial structure of the input data, such as images or videos, and perform feature

extraction and classification tasks. CNNs are widely used for image recognition, face detection, and object segmentation.

2. Recurrent neural networks (RNNs): These are a type of ANNs, where the information can flow in both directions, and the neurons can have feedback connections to themselves or to other neurons. RNNs can capture the temporal dynamics of the input data, such as sequences or time series, and perform tasks such as natural language generation, machine translation, and speech synthesis.
3. Long short-term memory (LSTM) networks: These are a special type of RNNs, where the neurons have a memory cell that can store and retrieve information over long periods of time. LSTM networks can overcome the problem of vanishing or exploding gradients, which often occurs in RNNs, and learn long-term dependencies in the input data. LSTM networks are widely used for text summarization, sentiment analysis, and video captioning.

Among the many types of ANNs, convolutional neural networks (CNNs) are one of the most popular and powerful ones, especially for computer vision tasks. Each neuron in CNN is connected to a local region of the previous layer. CNNs can exploit the spatial structure of the input data, such as images or videos, and perform feature extraction and classification tasks. CNNs are widely used for image recognition, face detection, object segmentation, and many other applications.

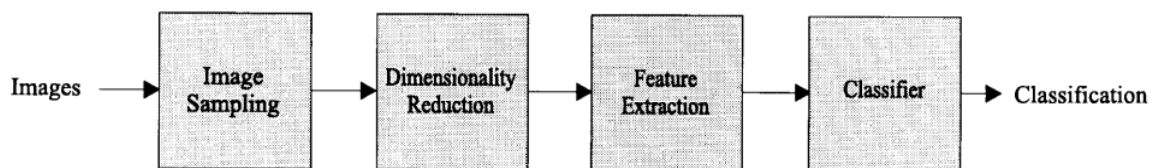


Figure 2.2 Elementary constituents of CNNs[19]

CNNs are deep learning algorithms that are especially suitable for image recognition and classification tasks. Figure 2.2[19] illustrates the basic elementary constituents of CNNs and workflow of CNNs. The basic idea of CNNs is to use convolutional kernels to slide over the image, extract the local features of the image, then reduce the dimension of the features through the pooling layer, and finally perform classification through the classifier. During the image sampling phase at the very beginning, the original image data is first sampled to reduce the data size and complexity. There are various sampling methods, such as scaling, cropping, rotating, flipping, etc. The next step is dimensionality reduction, and in this stage, some techniques (such as principal component analysis PCA) are used to reduce the dimension of the data, further simplifying the model calculation. The purpose of dimensionality reduction is to remove redundancy and noise in the data, and retain the most important information. The following step is feature extraction. During this phase, convolutional layers and pooling layers are used to extract the key features of the image. The convolutional layers are composed of multiple convolutional kernels, each of which can detect a certain feature in the image, such as edges, textures, shapes, etc. The pooling layers are to downsample the output of the convolutional layer, which can reduce the number and spatial size of the features, while maintaining the invariance of the features. The feature extraction process can be repeated multiple times, forming multiple convolutional layers and pooling layers, to extract higher-level features. The next stage is the classifier. The extracted features are fed into the classifier for training in this step, common classifiers are fully connected layers, SVM, etc. The function of the classifier is to map the features to different

categories, such as cats, dogs, humans, etc. The output of the classifier is the probability distribution of each category, indicating the possibility of the image belonging to each category. Classification: Finally, the probability distribution of each category is obtained, and the category with the highest probability is selected as the output result. For example, if the probability of cat is 0.8, the probability of dog is 0.1, and the probability of human is 0.1, then the output result is cat.

Our target neural network uses one of the most popular CNNs as the encoder, which is ResNet-50[20]. ResNet-50[20] is a deep convolutional neural network (CNN) architecture that was developed by Microsoft Research in 2015. It is a variant of the popular ResNet architecture, which stands for “Residual Network”. The “50” in the name refers to the number of layers in the network, which is 50 layers deep. ResNet-50[20] uses the concept of residual learning to overcome the problem of vanishing gradients and degradation of accuracy in very deep networks. Residual learning means that each layer learns the difference between the input and the output, rather than the output itself. This difference is called the residual, and it is added to the input through a shortcut connection. This way, the network can preserve the information from previous layers and avoid losing it during the forward and backward propagation.

The architecture of ResNet-50[20] is shown as Figure 2.3[21], and it is divided into four main parts: the convolutional layers, the identity block, the convolutional block, and the fully connected layers. The convolutional layers consist of a 7x7 convolution with 64 filters and a stride of 2, followed by a 3x3 max pooling layer with a stride of 2. The identity block is a residual block that does not change the dimensions of the input and output, and it has three convolutional layers: 1x1 with 64 filters, 3x3 with 64 filters,

and 1x1 with 256 filters. The convolutional block is a residual block that changes the dimensions of the input and output, and it has four convolutional layers: 1x1 with 64 filters and a stride of 2, 3x3 with 64 filters, 1x1 with 256 filters, and a 1x1 convolution with 256 filters and a stride of 2 on the shortcut connection. The fully connected layers consist of an average pooling layer, a 1000-node softmax layer, and a classification output.

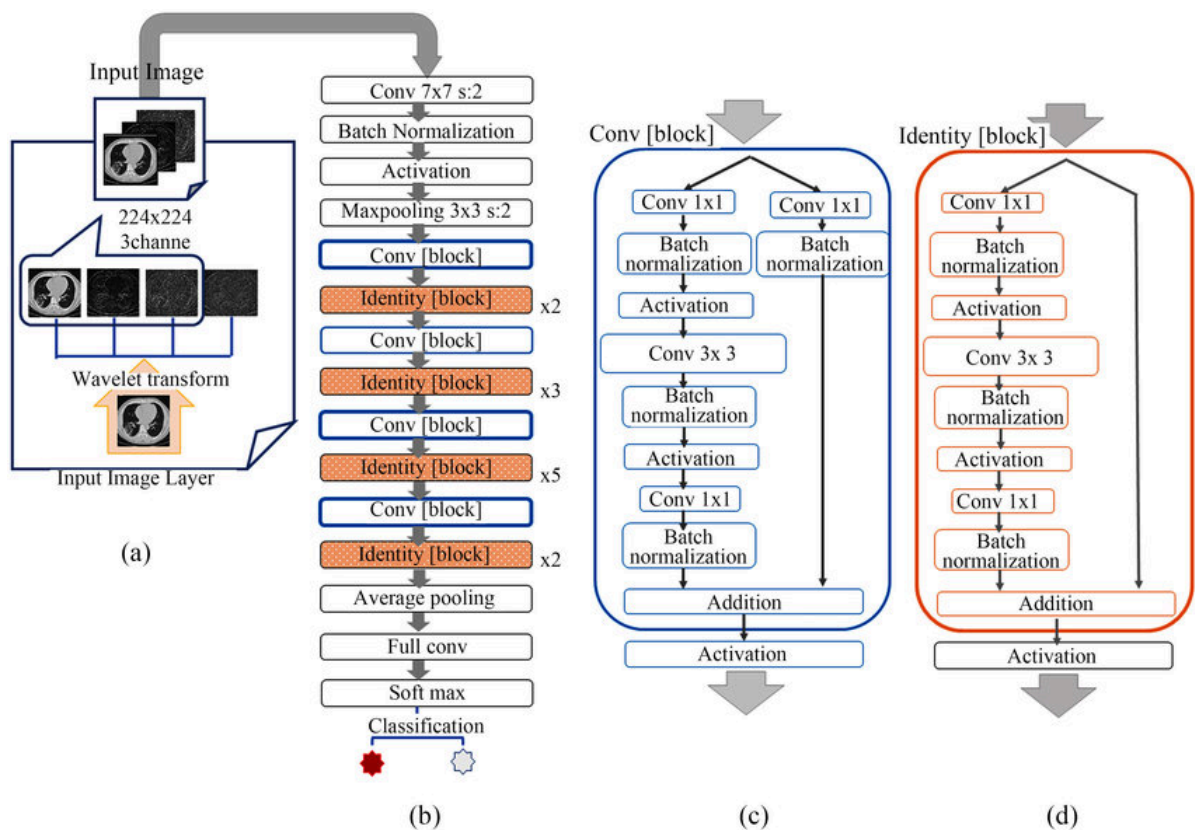


Figure 2.3[21] ResNet-50[20] Architecture: (a) Input Layer, (b) ResNet-50[20] overall structure. The symbols of x2, x3, x5 in the figure are the number of blocks, (c) Structure of a convolution block, (d) Structure of an identity block

ResNet-50[20] has 3.8×10^9 floating point operations (FLOPs) and 25.6 million parameters. ResNet-50[20] is a powerful and efficient model for image recognition and classification tasks. It has achieved state-of-the-art results on several benchmarks, such as ImageNet, COCO, and Pascal VOC. It has also been widely used as a backbone for other computer vision applications, such as object detection, semantic segmentation, and face recognition, that is why our target neural network — MoCo[15] uses it as the encoder.

2.2 Self-Supervised Learning

In recent times, self-supervised learning has emerged as a significant trend in the field of machine learning. The inception of this trend can be traced back to the introduction of Momentum Contrast (MoCo) by Kaiming He et al.[15], which sparked extensive discourse in the scientific community. This was further reinforced by Yann LeCun(Chief AI Scientist at Meta)’s proclamation at the AAAI conference, asserting self-supervised learning as the future of machine learning. The appeal of self-supervised learning lies in its ability to circumvent the labor-intensive and costly process of data labeling. It introduces a paradigm shift in model training, enabling the use of unlabeled data, thereby revolutionizing traditional supervised learning methodologies. This has profound implications for the efficiency and scalability of machine learning models.

2.2.1 Introduction

Self-supervised learning is a machine learning technique that uses unlabeled data to train models for tasks that usually require labeled data. Instead of relying on human annotations, self-supervised models generate their own labels from the data itself, such as

predicting the next word in a sentence, the rotation of an image, or the speaker of an audio clip. Self-supervised learning differs from supervised learning in that it does not need external supervision, and from unsupervised learning in that it optimizes performance against a ground truth.

Fig 2.4 illustrates the composition of deep learning in terms of algorithm category and how self-supervised learning fits into deep learning. As shown in Fig 2.4, machine learning can be classified into supervised learning, unsupervised learning, and reinforcement learning. Self-supervised learning is a form of unsupervised learning that aims to learn a general feature representation from unlabeled data for downstream tasks. Currently, self-supervised learning can be categorized into two types: generative methods and contrastive methods. Generative methods mainly focus on the reconstruction error of pixel space, often using the loss of pixel labels. They are mainly represented by autoencoders and their later variants [22]. Besides the generative methods, there is another type of methods based on contrastive learning. This type of method does not require the model to reconstruct the original input, but expects the model to distinguish different inputs in the feature space. Contrastive methods need to build distance metrics on feature space and obtain various prediction results through feature invariance [22]. They do not need pixel-level reconstruction, and thus the optimization becomes easier. Of course, these methods are not flawless, because the data lacks labels, so the main challenge is how to select and construct positive samples and negative samples.

Fig 2.5 and Fig 2.6 further demonstrate the distinction between supervised learning and self-supervised learning. Fig 2.5 depicts the basic procedure of supervised learning. In supervised learning, all input data are labeled, and the input is fed into a

convolutional neural network (CNN in Fig 2.5) to produce output (prediction). Then the output is compared with the input label (ground truth) by a predefined loss function, and the discrepancy between input label and neural network's output is used to update the neural network's parameters (weights) until they are sufficiently close. Fig 2.6 presents the basic procedure of self-supervised learning. In contrast to supervised learning, the input data in self-supervised learning are unlabeled and there are two convolutional neural networks in the procedure (CNN 1 and CNN 2 in Fig 2.6). As Fig 2.6 shows, the input (unlabeled) is fed into two parallel neural networks (CNN 1 and CNN 2 in Fig 2.6), and the input data before being fed into CNN 1 and CNN 2 is augmented. The output 1 generated by CNN 1 is regarded as ground truth while the Output 2 serves as prediction. Then, similar to supervised learning, a predefined loss function will assist in updating the CNN 1's parameters (weights) by comparing the difference between the ground truth and prediction. There are several methods to update CNN 2, but this paper will focus on the momentum method proposed by MoCo[15], and the details will be discussed later.

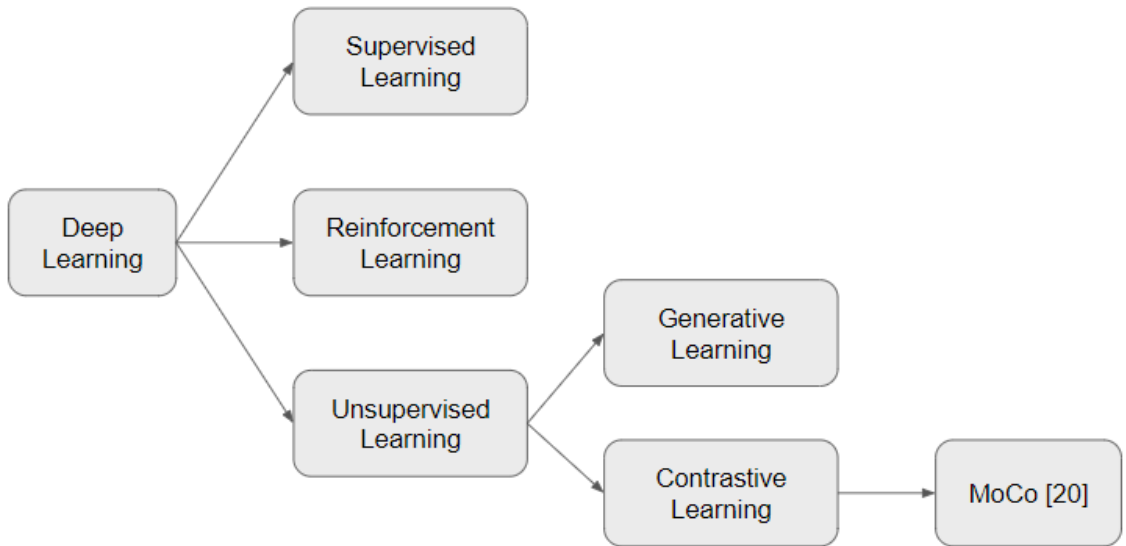


Figure 2.4 Deep Learning

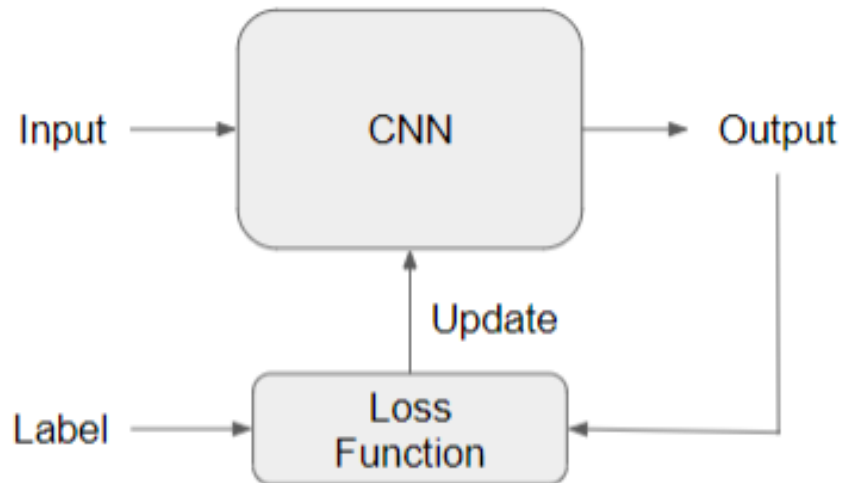


Figure 2.5 Supervised Learning

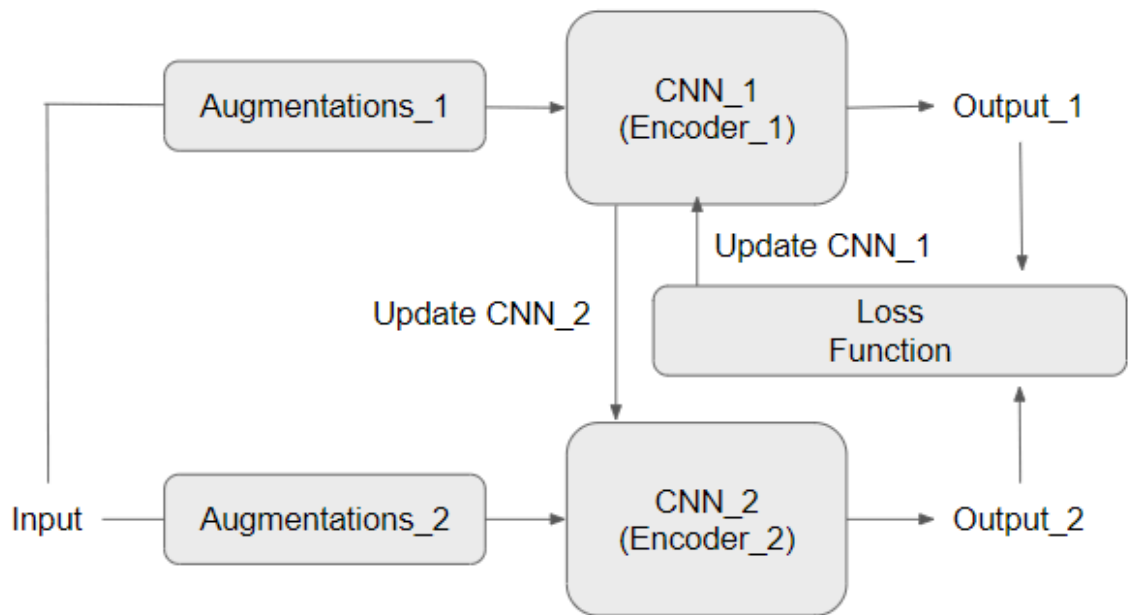


Figure 2.6 Self-Supervised Learning

2.2.2 MoCo

MoCo[15] introduces an effective architecture for contrastive learning. This model leverages the power of unsupervised learning to extract features that, when applied to ImageNet[23] classification tasks, can surpass the performance of traditional supervised learning methods. Drawing inspiration from natural language processing tasks, MoCo[15] encodes image data into query vectors (Q) and key vectors (K), as illustrated in Figure 2.7. Each encoding comprises a single positive sample and multiple negative samples. The model learns feature representation by comparing losses. MoCo[15] employs two neural networks to encode data: an encoder for the query and a momentum encoder for the key. The query encoder is tasked with encoding the abstract

representation of the current instance, while the momentum encoder encodes the abstract representation of multiple instances, including the current one.

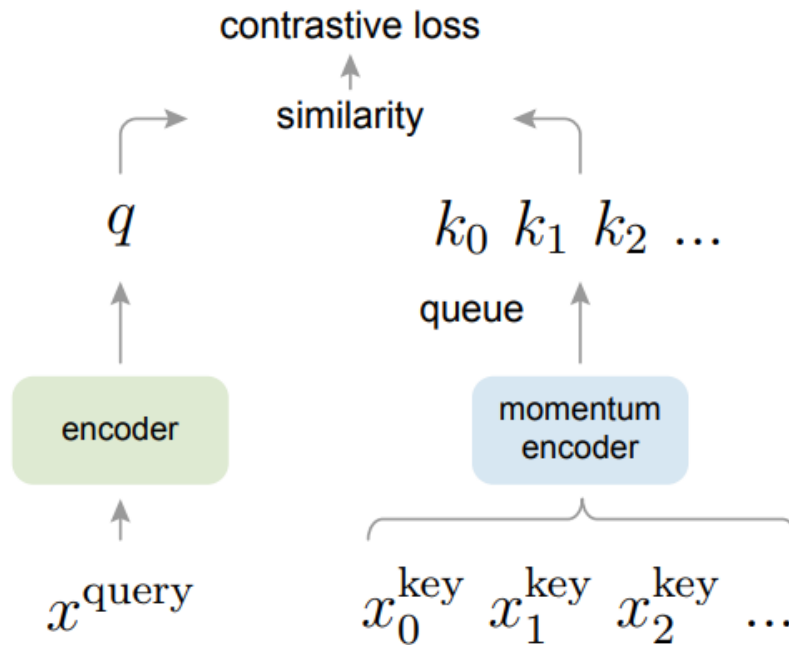


Figure 2.7 MoCo[20]

For each instance, the model aims to maximize the coding results of its encoder and momentum encoder, and minimize the coding results of other instances in the momentum encoder. This is achieved using a loss function, denoted as Equation (1).

$$Lq = -\log \frac{\exp(q \times k_+ / \tau)}{\sum_{i=0}^{\kappa} \exp(q \times k_i / \tau)} \quad (1)$$

Where τ represents a temperature hyper-parameter. The summation is performed over one positive and κ negative sample. Intuitively, this loss function represents the

logarithmic loss of a (K+1)-way softmax-based classifier that attempts to classify the query as the key.

Figure 2.8 presents a PyTorch-style pseudocode of the MoCo[15], which serves as a valuable resource for comprehending the fundamental steps involved in the MoCo[15] process. As depicted in Figure 2.8, the initial step involves the initialization of two convolutional neural networks, namely the encoder and the momentum encoder. Subsequently, two versions of augmented input data are fed into the encoder and the momentum encoder respectively, facilitating forward propagation. Momentum encoder is detached because it will not perform back-propagation later on. The ensuing step involves the computation of the distance between the current instance and the positive sample(positive logits), as well as the distance between the current instance and the negative samples(negative logits). A predefined loss function(Equation (1)) is then employed to update the parameters (weights) of the encoder (query network), adhering to the principle of maximizing the coding results of its encoder and momentum encoder for the current instance, while minimizing the coding results of other instances in the momentum encoder. Upon updating the query encoder, the momentum encoder (key network) is updated based on the updated query encoder, as per Equation (2).

$$\Theta_k = m \times \Theta_k + (1 - m) \times \Theta_q \quad (2)$$

Here, Θ_q and Θ_k represent the query encoder and key encoder respectively, and $m \in [0, 1)$ is a momentum coefficient. Notably, only the parameters Θ_q are updated via back-propagation. The momentum update in Equation (2) ensures that Θ_k evolves more smoothly than Θ_q . Consequently, despite the keys in the queue being encoded by different

encoders (in different mini-batches), the difference among these encoders can be minimized. Experimental results indicate that a relatively large momentum (e.g., $m = 0.999$, our default) outperforms a smaller value (e.g., $m = 0.9$), suggesting that a slowly evolving key encoder is crucial for effectively utilizing a queue[20].

2.3 High Level Synthesis

High-Level Synthesis (HLS) is a transformative approach that has revolutionized the field of digital circuit design. By translating high-level programming languages into hardware description languages, HLS has significantly improved the efficiency and effectiveness of the design process[24]. In this dissertation, we will utilize the HLS method to generate Register-Transfer Level (RTL) code, which is then used to estimate power consumption and area utilization of the systolic array based hardware accelerator.

The process of HLS involves several intricate steps. Initially, the high-level source code is analyzed and optimized for the target architecture. This is followed by the allocation of resources, such as adders, multipliers, and registers. Subsequently, the operations are scheduled over time, and finally, the RTL code is generated[25]. Various tools are available for HLS, including Xilinx Vivado HLS, Cadence Stratus HLS, and Mentor Catapult HLS. These tools offer a wide range of features, such as support for various input languages, optimization directives, and interfaces for IP integration[24].

The advantages of HLS are manifold. It allows designers to work at a higher level of abstraction, thereby reducing the complexity of the design process¹. HLS also enables faster design iterations, as changes can be made at the high-level source code and propagated to the RTL code. Furthermore, HLS tools often provide optimization

directives that allow designers to guide the synthesis process towards specific goals, such as minimizing power consumption or area utilization[24]. HLS is a powerful technique that can significantly enhance the efficiency and effectiveness of digital circuit design. By generating RTL code from high-level source code, HLS allows designers to focus on algorithmic aspects, leaving low-level implementation details to the synthesis tool. This not only improves productivity but also opens up new opportunities for optimization and innovation. In this project, we will leverage these benefits of HLS to generate RTL code and perform power and area estimation, thereby driving the design and optimization of our digital circuits[24].

```

# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
    x_q = aug(x) # a randomly augmented version
    x_k = aug(x) # another randomly augmented version

    q = f_q.forward(x_q) # queries: NxK
    k = f_k.forward(x_k) # keys: NxK
    k = k.detach() # no gradient to keys

    # positive logits: NxK
    l_pos = bmm(q.view(N,K), k.view(N,K))

    # negative logits: NxK
    l_neg = mm(q.view(N,K), queue.view(K,K))

    # logits: Nx(1+K)
    logits = cat([l_pos, l_neg], dim=1)

    # contrastive loss, Eqn. (1)
    labels = zeros(N) # positives are the 0-th
    loss = CrossEntropyLoss(logits/t, labels)

    # SGD update: query network
    loss.backward()
    update(f_q.params)

    # momentum update: key network
    f_k.params = m*f_k.params+(1-m)*f_q.params

    # update dictionary
    enqueue(queue, k) # enqueue the current minibatch
    dequeue(queue) # dequeue the earliest minibatch

```

Figure 2.8 Pseudocode of MoCo[15]

2.4 Systolic Arrays Overview

This section provides a comprehensive overview of the primary components of a systolic array accelerator and the principal mechanism for reducing energy consumption: data reuse.

A significant portion of this section is dedicated to the concept of data reuse, a key strategy in reducing energy consumption in systolic array accelerators. By reusing data across multiple operations, the need for expensive data movements is minimized, leading to substantial energy savings. Subsequently, we delve into the primary types of data reuse: weight stationary (WS), input stationary (IS), and output stationary (OS). Each type has its unique advantages and disadvantages in terms of energy efficiency, computational speed, and implementation complexity. A thorough comparison and analysis of these types will provide readers with a deeper understanding of their applicability in different scenarios

2.4.1 Systolic Array Architecture

Figure 2.9 shows a generic architecture of a systolic array based hardware accelerator. As depicted in Figure 2.9, the Convolutional Core of the systolic array hardware accelerator is composed of several integral components, including the SRAM, Activation function, and the Systolic Array.

The SRAM/Buffer plays a pivotal role in the architecture by storing input data, weight data, intermediate data, and output data. It is designed to ensure that data is readily accessible for processing, thereby minimizing latency and maximizing the

system's throughput. The buffer interfaces with the DRAM, which serves as a repository for larger datasets that are fetched for processing when required.

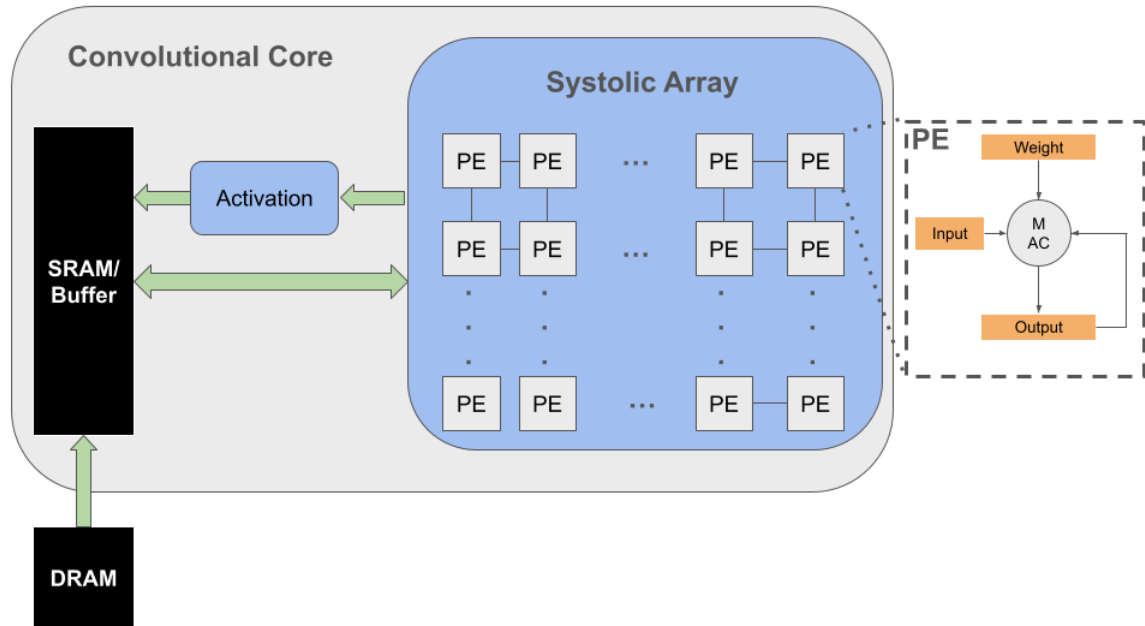


Figure 2.9 Generic Architecture of a Systolic Array Accelerator

Situated within the Convolutional Core, the Systolic Array is a grid of PEs that collaboratively execute matrix multiplications, a fundamental operation in neural network computations. Each PE is interconnected in a manner that facilitates efficient data transfer and parallel processing capabilities, significantly accelerating computational tasks. The existence of the Systolic Array enables a multitude of computational tasks to be processed concurrently, thereby greatly enhancing computational efficiency.

Each PE within the Systolic Array is intricately linked to an PE module, which comprises several registers for weight, input, and output, as well as the MAC unit. The MAC unit within each PE rapidly performs multiplications and accumulations, operations that are essential in neural network computations. The presence of the PE module allows

each PE to perform computations independently, further augmenting the computational efficiency of the entire system.

2.4.2 Data Reuse Methods

In the realm of systolic array hardware accelerators, particularly for Convolutional Neural Networks (CNNs), data reuse is a pivotal strategy. It optimizes computational efficiency and performance by minimizing memory access and data migration, which are often performance and energy consumption bottlenecks. Three prevalent data reuse methods are Weight Stationary, Input Stationary, and Output Stationary, each with unique principles, advantages, and disadvantages.

The term “stationary” in the context of data-flow refers to the matrix that is “anchored” to a specific PE. Consequently, “Weight Stationary” denotes the mapping strategy where each pixel of the kernel (weight) is allocated to a specific PE. In an ideal scenario with unlimited resources, the number of PEs would be equivalent to the number of output pixels. To realize this, all computations required for generating a specific output are executed on the designated PE, with the necessary operands being streamed in every cycle. The reduction operation is performed in place, eliminating the need for additional communication between the MAC units for the generation of the specified pixel. However, in a practical scenario where the number of computational elements is constrained, resources are time-multiplexed. Upon the generation of an output pixel by a PE, the result is transferred to memory, and the PE is subsequently reassigned to compute another pixel. This approach ensures efficient utilization of limited computational resources.

According to [26], the Weight Stationary (WS) method keeps weights stationary in the processing elements. In accordance with the aforementioned nomenclature convention, the term “Weight Stationary data-flow” is used to describe a mapping process in which each element of the weight matrix is uniquely assigned to a specific MAC unit. Upon mapping a set of weights onto the array, these weights remain stationary until all computations involving them are completed. In each cycle, the input elements that need to be multiplied with the currently mapped weights are streamed, and the resulting partial sums are stored within the array. The reduction process, which often spans multiple cycles, involves the communication of these partial sums across the MAC units present in the array. This mapping in the Weight Stationary data-flow is illustrated in Figure 2.10. The mapping process is executed in two stages. Initially, each column is allocated to a specific filter. For a given column, the elements of the assigned filter matrix are introduced from the top edge until each PE in the column has one element. Following the placement of filter elements, the pixels of the input feature map are then introduced from the left edge. During this phase, the partial sums for a given output pixel are generated in each cycle. For a specific output pixel, the corresponding partial sums are distributed over a column. These partial sums are subsequently reduced over the given column in the next n cycles, where n represents the number of partial sums generated for a specific pixel. The weight pixels remain in the array until all computations requiring these values as operands are completed. Once the computations associated with the mapped weight are finished, the mapping process is repeated with a new set of weights.

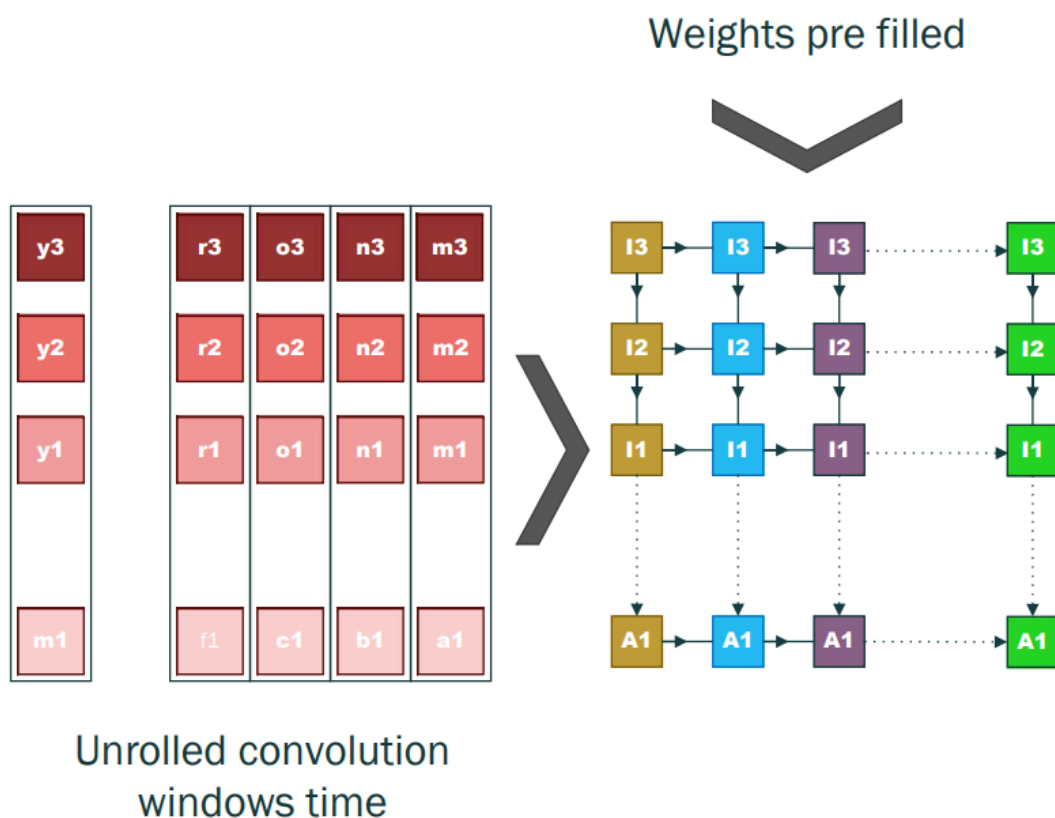


Figure 2.10 WS

The Input Stationary dataflow, as the name suggests, is a mapping similar to Weight Stationary (WS), where pixels of the input feature map (IFMAP) are “anchored” to the PEs and elements of the weight matrices are streamed in. This mapping is schematically represented in Figure 2.11. Like WS, this mapping also occurs in two stages. However, in this case, each column is assigned to a convolution window, which is defined as the set of all pixels in the IFMAP necessary to generate a single output feature map (OFMAP) pixel. For a given column, the pixels corresponding to a specific convolution window are streamed in from the top edge. Once the input pixels are introduced, the elements of the weight matrices are streamed in from the left edge. Reduction, similar to WS, is performed over a given column, and the convolution

windows are retained until all computations requiring these elements are completed. The array is then remapped with elements belonging to new convolution windows. This dataflow offers the advantage of lower SRAM bank requirements compared to Output Stationary (OS). However, the cost and runtime compared to WS vary depending on the workload.

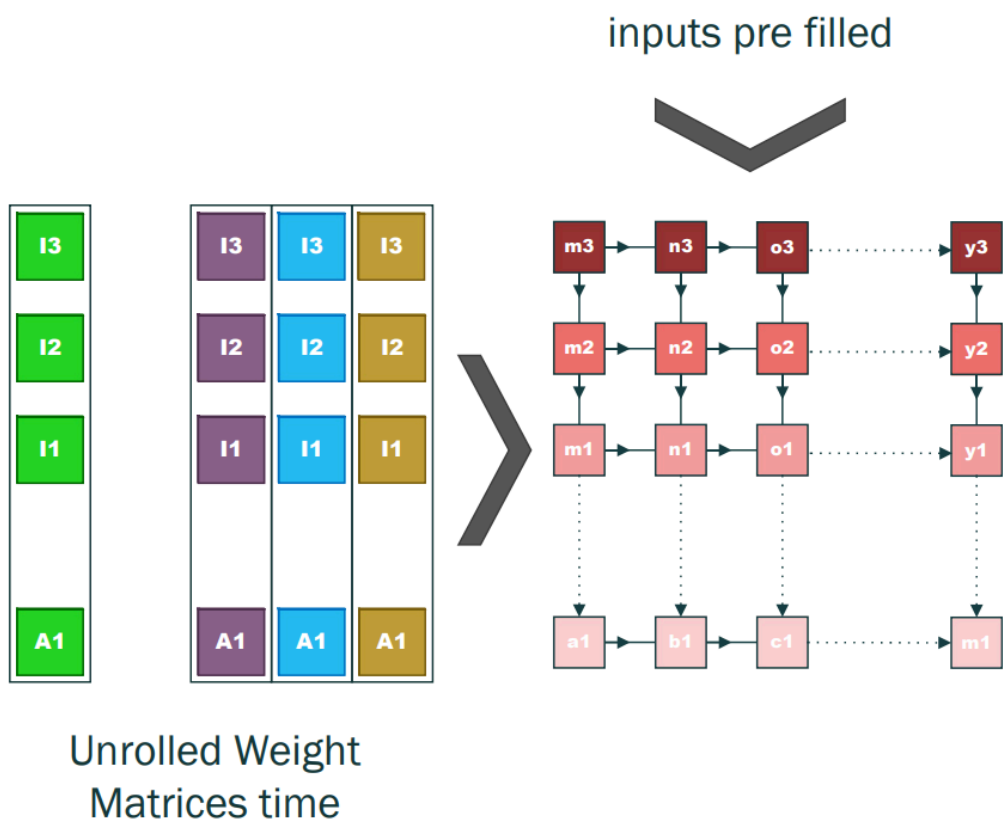


Figure 2.11 IS

Figure Figure 2.12 presents the schematic representation of the data-flow. The data is introduced from the left and top edges of the array. The left edges stream in the input pixels, while the top edge streams in pixels from the filter or weight matrices. In a specific column, the PEs in each row are tasked with generating adjacent output pixels within a single channel. However, each column generates pixels corresponding to

different output channels. The data-flow model operates under the assumption that the generated outputs can be transferred out of the array without causing a computational stall. However, in practical implementations, this may not always hold true, potentially leading to a runtime that exceeds the calculated value. This discrepancy underscores the importance of considering real-world constraints when designing and implementing such systems.

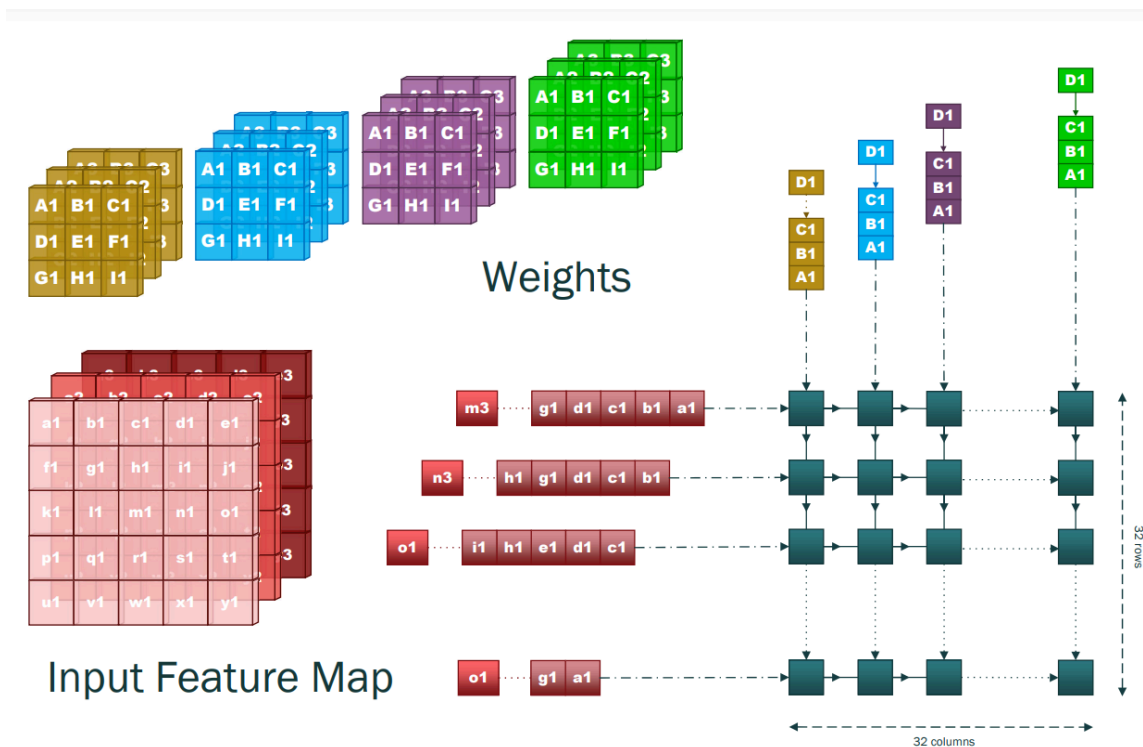


Figure 2.12 OS

Within the scope of our project, we have chosen to implement the WS method in our simulation. This decision is substantiated by several key factors. Primarily, the WS method offers significant advantages in situations where convolutional reuse and filter reuse are critical. By maintaining the weights in a stationary state within the processing elements, we can optimize the utilization of these reuses. This approach considerably

diminishes bandwidth consumption, thereby augmenting the operational throughput of our systolic array hardware accelerator. Furthermore, the WS method intrinsically facilitates the efficient execution of matrix multiplications, a core operation in neural network computations. By curtailing the movement of weights, we can ensure that the most frequently utilized data is immediately available for processing, thereby mitigating latency and further enhancing the system's throughput.

CHAPTER 3

State-of-the-art

This chapter is composed of two distinct sections, each focusing on state-of-the-art developments in specific areas of hardware technology.

The first section provides an in-depth exploration of the latest frameworks for systolic array hardware accelerators. It delves into the intricacies of these frameworks, shedding light on their design principles, operational mechanisms, and the advantages they offer in the realm of hardware acceleration.

The second section pivots to the cutting-edge hardware simulators currently shaping the industry. It offers a comprehensive overview of these advanced tools, discussing their functionalities, capabilities, and the role they play in hardware design and testing.

3.1 State-of-the-art Frameworks for Systolic Array Based Hardware Accelerators

Timeloop[31] serves as a Design Space Exploration (DSE) framework specifically tailored for Convolutional Neural Networks (CNNs). It has the capability to emulate a variety of accelerators, including but not limited to NVDLA. The primary focus of Timeloop[31] is the analysis of convolution layers. The inputs for Timeloop[31] encompass a workload description, which includes parameters such as input dimensions and weight values, a description of the hardware architecture, which includes components like arithmetic modules, and hardware constraints. In lieu of a cycle-accurate simulator, Timeloop[31] leverages the deterministic behavior of data transfers to conduct analytical analyses. In terms of energy models, Timeloop[31] incorporates models for memory,

arithmetic units, and wire/network, all of which are based on the TSMC 16nm FinFET technology. While Timeloop[31] is a powerful tool, it does have some potential drawbacks[32]. Firstly, Timeloop[31] uses the deterministic behavior of data transfers for analysis instead of a cycle-accurate simulator. This might lead to discrepancies between the performance and energy efficiency predictions of Timeloop[31] and the actual results when the hardware is run. Additionally, while Timeloop[31] can emulate various accelerators, it primarily focuses on the analysis of convolution layers, which may not provide a comprehensive evaluation of other types of neural network layers. Lastly, Timeloop[31]'s energy models are based on TSMC 16nm FinFET technology, which may not accurately predict the energy consumption of hardware accelerators using other process technologies.

Accelergy[32] provides a mechanism for estimating the energy consumption of accelerators without necessitating a comprehensive hardware description. This is achieved by utilizing a library of fundamental components. Accelergy[32] employs a high-level architectural description to encapsulate the characteristics of circuit behavior, such as memory reads. It takes into account the quantity of memory reads and the pattern of memory access, which can either be random or involve repetitive access to the same address. Accelergy [32], a potent energy estimation tool, is not without its limitations. Its energy models, derived from a library of basic components, may not accurately represent the energy consumption of intricate or bespoke hardware architectures. Moreover, Accelergy[32]'s energy estimation, which is not predicated on a comprehensive hardware description, may expedite design space exploration but could potentially compromise the accuracy of energy estimations relative to methods employing a complete hardware

description. Finally, Acclergy[32] conducts energy estimation at an initial design stage, which, despite facilitating rapid design space exploration, may not accurately mirror the energy consumption of the ultimate hardware design, particularly if substantial modifications are implemented during the design process.

Gemmini [33], an open-source systolic array generator, facilitates the evaluation of deep-learning architectures. It produces a custom ASIC accelerator, specifically designed for matrix multiplication, utilizing a systolic array architecture. Furthermore, Gemmini[33] is seamlessly integrated with the RISC-V Rocket ecosystem [34]. Despite its strengths, Gemmini [33] is not without limitations. Primarily, the ASIC accelerator it generates, which is predominantly based on a sparse array architecture for matrix multiplication, may not cater to deep learning models necessitating computations beyond the conventional systolic array. Additionally, while Gemmini [33] is seamlessly integrated with the RISC-V Rocket ecosystem, its interoperability with other hardware or software ecosystems may be constrained.

MLPAT[43] is a comprehensive framework designed for the modeling of power, area, and timing characteristics of machine learning accelerators. It is capable of modeling various components including systolic arrays, on-chip memory, and the activation pipeline. MLPAT[43] supports a variety of precision types, facilitating the validation of the trade-off between accuracy and precision. It also accommodates different data-flows, such as WS and OS. The framework accepts inputs such as the accelerator architecture, the circuit design, and the technology used. Based on these inputs, MLPAT[43] generates an optimized chip representation and provides a detailed

report on key metrics such as area, power, and performance. This functionality makes MLPAT[43] a powerful tool for the design and analysis of machine learning accelerators.

MAESTRO[44] is a specialized framework designed to characterize and analyze the hardware used in neural networks. It enables the determination of the hardware cost associated with implementing a desired architecture. MAESTRO[44] incorporates a domain-specific language for describing the dataflow, which allows for the specification of parameters such as the number of PEs, memory size, and Network-on-Chip bandwidth. The framework primarily generates results that are centered on performance analyses. In recent studies, MAESTRO[44] has been utilized to estimate the trade-offs between execution time and energy efficiency for CNN models, including but not limited to VGG[45] and AlexNet[46]. This capability makes MAESTRO[44] a valuable tool in the optimization of neural network hardware implementations

Heidorn et al.[47] introduce an analytical model designed to estimate throughput and energy relative to specific hardware constraints. They propose a DSE to ascertain the limits of the accelerator architecture in terms of throughput, the number of parallel operations, and memory. The authors further propose an accelerator to evaluate the model, which incorporates tile-local memory, a bus, and a CGRA. Each CGRA comprises a two-dimensional array of PEs, and the accelerator can incorporate multiple CGRAs to facilitate parallel processing. This approach allows for a comprehensive evaluation of the accelerator's performance under various conditions.

Zhao et al.[48] introduce an analytical performance predictor designed to estimate energy, throughput, and latency for ASIC and FPGA. The predictor leverages DNN models, hardware architecture, types of dataflows, and hardware cost relative to a

technology node. The results are generated using AlexNet[46] and SkyNet CNN models, in conjunction with Eyeriss[50], an FPGA implementation from [49], and the synthesized results of a proposed accelerator. This comprehensive approach allows for a detailed analysis of the performance characteristics of various hardware configurations.

DNNExplorer[51] is a framework designed for DSE of Machine Learning accelerators. It provides support for machine learning frameworks such as Caffe[37] and PyTorch, in addition to three distinct accelerator architectures. The architecture is also compatible with WS and IS dataflows. DNNExplorer[51] employs analytical models to estimate both performance and hardware configuration, making it a versatile tool for the design and analysis of Machine Learning accelerators.

Interstellar[52] is a DSE framework that leverages the Halide language to generate hardware and compare various accelerators. It supports different dataflows, such as WS, OS, and RS in 2D arrays, as well as MAC tree schemes. The authors present a systematic methodology to delineate the design space of DNN accelerators using Halide. Additionally, the framework is designed to optimize the memory hierarchy, further enhancing the efficiency of the accelerators.

DeepOpt[53] is a DSE framework that facilitates the exploration of ASIC implementations of systolic hardware accelerators for CNNs. The primary objective of this DSE is to minimize the number of memory accesses, taking into account hardware characteristics such as on-chip SRAMs and the number of parallel PEs. DeepOpt[53] employs a search tree to schedule the convolution process, enabling the minimization of memory accesses by modeling memory access patterns (weight and output stationary)

and pruning branches from the search tree. This approach allows for efficient memory management and optimized performance in the implementation of CNNs.

Karbachevsky et al.[54] introduce a method for estimating area and power values, which is based on the BOP metric[55]. The BOP metric quantifies the number of bit operations needed to execute a calculation, as determined by the input bit size, output bit size, number of inputs, and number of outputs. According to the authors, the BOP metric enables the estimation of the area and power demands of accelerator hardware with high precision during the early stages of the design process. Furthermore, the method can illustrate the trade-off between the number of PEs and the bottlenecks induced by parameter quantization, such as memory bandwidth or computational resources. This approach provides valuable insights for optimizing the design and performance of hardware accelerators.

Ferianc et al.[56] introduce a method aimed at enhancing the performance of DSE analyses. This method employs a Gaussian process regression model, parameterized by the features of the accelerator and the target CNN, such as filter, channel, and data parallelism. The method is adept at predicting hardware latency and energy. It has been compared with machine learning-based methods for performing DSE, including linear regression, Gradient Tree Boosting, and Neural Networks. This comparison provides a comprehensive evaluation of the method's effectiveness in the context of DSE analyses.

Aladdin[57] is a pre-RTL framework designed for modeling the power-performance of accelerators. It provides estimates for performance, power, and area. The infrastructure of Aladdin employs DDDG to depict accelerators. The DDDG is generated from a C program, enabling Aladdin[57] to report on the program

dependencies and resource constraints. This approach allows for a comprehensive analysis of the accelerator's performance under various conditions.

3.2 State-of-the-art Hardware Simulators

SCALE-Sim[30] is a cycle-accurate simulator specifically designed for systolic arrays. It allows for the adjustment of micro-architectural features such as array size, aspect ratio, scratchpad memory size, and data flow mapping strategy. Additionally, it enables the configuration of system integration parameters, including memory bandwidth. This feature will be used to estimate the bandwidth of our model as the size of the systolic array changes. However, SCALE-Sim[30] has certain limitations. It is exclusively designed for forward propagation, making it unsuitable for training simulations. It also does not support padding during the simulation of convolution operations. Finally, it is unable to predict the power or area of the design.

AccTLMSim[35] is a pre-RTL, cycle-accurate simulator specifically developed for CNN accelerators, grounded on SystemC transaction-level modeling (TLM). The simulator is engineered to maximize throughput performance given a defined on-chip SRAM size. An accelerator is employed to validate the functionality of the simulator. It's crucial to highlight that AccTLMSim[35] predominantly focuses on performance, neglecting power or area considerations. Nevertheless, it does exhibit some potential limitations. Firstly, akin to all behavioral level designs utilizing SystemC, AccTLMSim[35] has restricted parameterization mechanisms and limited adaptability for hardware synthesis. Secondly, AccTLMSim[35] is primarily performance-oriented, disregarding power or area.

STONNE[36] serves as a cycle-accurate architectural simulator for CNNs, facilitating comprehensive evaluation. It collaborates with the Caffe framework[37] to produce CNNs and simulates the MAERI accelerator[38]. The primary emphasis of the results is on performance and hardware utilization. STONNE[36] adopts the Accelergy[32] energy estimation methodology for area and energy approximation, considering fundamental modules like adders for energy computation. This strategy guarantees a thorough and precise assessment of the system's efficiency and effectiveness. Nevertheless, it utilizes Accelergy[32] as the power estimation model, suggesting potential inaccuracies in representing the energy consumption of complex or custom hardware architectures.

According to [26], Table 3.1 provides a summary of the works reviewed. The second column indicates whether the work integrates with high-level CNN modeling frameworks, such as TensorFlow and Caffe[37]. The third and fourth columns pertain to the metrics evaluated. The third column presents metrics based on fundamental components, such as MAC units and register files. The fourth column displays the metrics evaluated concerning the entire convolution process, which is our original contribution.

Maestro[44] does not support accelerator simulation, thereby limiting the performance evaluation (e.g., throughput). SCALE-Sim[30] does not provide results for power or energy. MLPAT[43] and Timeloop[23] offer PPA based on basic operations, such as adders and multipliers. However, methods that rely on operation counting do not take into account the interconnection of these operators (e.g., 1D or 2D systolic arrays or adder trees), leading to imprecise hardware metrics. Works [47] and [48] present

analytical results for power, performance, and area. Additionally, SimuNN[48] considers features like the dataflow type, which can contribute to power consumption. A similar approach is adopted by Aladdin[57]. Works such as Gemmini[33], Interstellar[38], DeepOpt[53], and [56] have limited PPA analyses. Furthermore, [54] posits that the primary effect of changing the circuit frequency is to reduce power consumption. However, this is not entirely accurate since logical synthesis with different frequency constraints yields different area values. STONNE[36] and SimuNN[48] present flows integrated with frameworks to model CNNs. However, SimuNN[48]’s energy estimation is based on basic elements and does not consider data movement through the accelerator. While STONNE[36] does not address power estimation, the authors argue that it is possible to integrate STONNE[36] with Accelergy[32] (which only evaluates power). DNNExplorer[51] also allows frameworks to model CNNs but lacks PPA analyses. Both SCALE-Sim[30] and AccTLMSim[35] lack integration with frameworks to model CNNs.

Table 3.1 Summary of State-of-the-art DNN Simulators

Simulators	Integration with CNN Frameworks	Evaluation Metrics	PPA Analysis
MLPAT[43]	NO	PPA	NO
Maestro[44]	NO	Performance	NO
Timeloop[23]	NO	PPA	NO
Accelergy[32]	NO	Power	NO
[47]	NO	PPA	NO
[48]	NO	PPA	NO
DNNExplorer[51]	Caffe, PyTorch	Performance	NO
Gemmini[33]	NO	Performance	NO
Interstellar[38]	NO	Power	NO
DeepOpt[53]	NO	Power, Performance	NO
[54]	NO	Area	NO
[56]	NO	Power, Performance	NO
Aladdin[57]	NO	PPA	NO
ScaleSim[30]	NO	Performance, Area	NO
STONNE[36]	Caffe	Performance	NO
SimuNN[48]	TensorFlow	PPA	NO
AccTLMSim[35]	TensorFlow	Performance	NO

CHAPTER 4

Model for PPA Estimation

In this chapter, we delve into the intricacies of a sophisticated, evolvable model specifically designed for PPA estimation of systolic arrays based hardware accelerators.

4.1 Introduction

This chapter is meticulously structured into five main sections. The first section(4.2) serves as a primer to the suite of tools that are instrumental in the implementation of the PPA estimation model. These tools, which form the foundation of our model, enable it to adapt and evolve in response to varying requirements, thereby enhancing the accuracy of the estimations it produces. The second section(4.3) provides an in-depth analysis of how the model estimates the performance, power, and area of systolic array-based hardware accelerators. Each of these aspects plays a pivotal role in gauging the efficiency and effectiveness of the hardware design, and their accurate estimation is crucial in the optimization of the design process. The performance estimation component of the model provides insights into the operational efficiency of the hardware design, while the power estimation component evaluates the energy consumption. The area estimation component, on the other hand, provides valuable information about the spatial requirements of the design. Together, these three components provide a holistic view of the hardware design's viability and efficiency. Section 4.4 of this paper elucidates the simulation of MoCo[15] training process using our model. This simulation provides a comprehensive understanding of the model's

capabilities and its application in real-world scenarios. In the fourth section(4.5), we present the PPA results obtained when a specifically configured systolic array is employed to train MoCo[15]. These results are then juxtaposed with the outcomes from MoCo's[15] default training hardware, the GPU. This comparative analysis offers valuable insights into the model's performance and its potential advantages over traditional training hardware. The penultimate section[4.6] of this chapter encapsulates the conclusions drawn from our model and the experimental results. This section synthesizes the key findings of our research, providing a succinct summary of the model's capabilities and its implications for the field of systolic array-based hardware accelerators. It serves as a culmination of our rigorous exploration and analysis, paving the way for future research in this domain.

This chapter, therefore, offers a comprehensive exploration of the PPA estimation model, shedding light on its underlying mechanisms, its applications, and its potential impact on the field of hardware design. It is an invaluable resource for those seeking to understand and leverage the power of PPA estimation in the realm of systolic array-based hardware accelerators.

4.2 Related Tools

The workflow is bifurcated into two primary components: the Performance model and the Power & Area model, each employing a distinct set of tools and methodologies. This dissertation introduces a comprehensive workflow for a PPA Estimation Model, as depicted in Figure 4.4.

The Performance Model leverages Catapult HLS for the transformation of high-level language descriptions into RTL designs. Within the realm of Catapult HLS, SCVerify, an intrinsic tool, is utilized for RTL simulations. These simulations validate the functionality and performance of the design, ensuring its alignment with the initial high-level descriptions. Transitioning to the Power & Area model, the workflow incorporates DesignCompiler[28]. DesignCompiler[28] emerges as a pivotal tool for power prediction, renowned for its robustness and versatility, encapsulates an array of IP cores optimized for power, performance, and area metrics. This ensures that the designs are not only functional but also efficient in terms of power consumption and spatial requirements. ScaleSim[30] is a simulator designed for systolic array-based accelerators. It is primarily used for Convolution, Feed Forward, and any layer that utilizes General Matrix to Matrix Multiplication. ScaleSim[30] here is employed to elucidate the correlation between the performance of systolic arrays of varying dimensions and that of a fundamental systolic array as the array size is incrementally amplified. This approach provides a comprehensive understanding of the performance implications associated with scaling up systolic arrays.

4.3 PPA Estimation

In this section, we will concentrate on detailing how our model is utilized to estimate three critical parameters: performance, power, and area of a systolic array based hardware accelerator. Our model, built with a comprehensive architecture and advanced algorithms, is adept at providing precise estimations of these parameters.

We will explore the complexities of the model, discussing its structure, the fundamental principles that govern its functionality, and the methodologies it employs for performance, power, and area estimation. This comprehensive understanding will provide a solid foundation for interpreting the results presented in the subsequent sections of this paper.

4.3.1 Performance Estimation

In this subsection, we will provide a comprehensive exposition of how our model is employed to estimate the performance of systolic array-based hardware accelerators. The performance estimation stands as the most innovative aspect of this dissertation, and we will delve into the details of how artificial intelligence (AI) principles are leveraged for this purpose. We will elucidate the AI methodologies used in our model, and how they contribute to the accurate estimation of performance. The advantages of our performance model will be highlighted, explaining why it outperforms traditional models and how it continues to evolve for better accuracy and efficiency.

This section will provide a deep understanding of our model's unique capabilities, setting the stage for the subsequent results and discussions in this dissertation. The insights gained here will underscore the novelty and significance of our work in the field of hardware accelerator performance estimation.

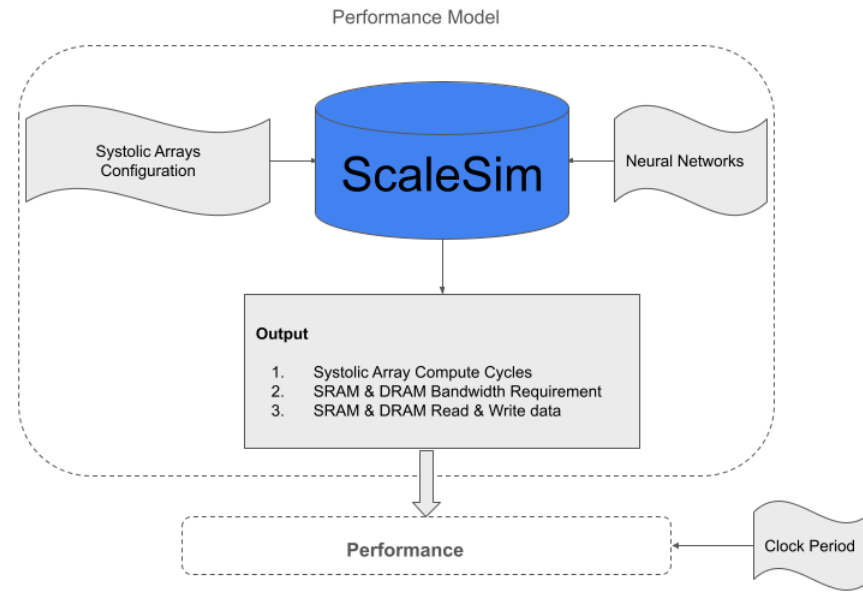


Figure 4.1 Performance Estimation Model

As depicted in Figure 4.1, the Performance Model is an exhaustive and methodical procedure developed to assess the performance of systolic arrays in neural networks utilizing ScaleSim[30]. The model commences with the neural network algorithm and configurations such as systolic array size, data reuse technique, and SRAM size. These configurations are subsequently input into ScaleSim[30], a simulation environment engineered to scrutinize and evaluate their performance metrics.

ScaleSim[30] is a simulation environment that incorporates various parameters from neural networks to deliver a thorough analysis. It assesses the compute cycles of systolic arrays, SRAM & DRAM bandwidth requisites, and read & write data volume operations. Compute cycles denote the number of cycles necessitated by the systolic array to execute a specific computation. This metric is pivotal as it directly influences the speed and efficiency of the computation. A reduced number of compute cycles signifies a more efficient configuration. The bandwidth requisites of the SRAM and DRAM are also

assessed. Bandwidth denotes the volume of data that can be transferred between the memory and the processor per time unit. Elevated bandwidth facilitates faster data transfer, which consequently leads to expedited computations. The volume of data read from and written to the SRAM and DRAM is another significant metric. This provides an indication of the memory usage of the computation. Efficient memory usage is crucial for the overall performance of the computation.

The output from ScaleSim[30] is subsequently utilized to assess compute cycles within a specified clock period originating from HLS (details will be illustrated in the Power & Area Estimation section) to calculate the time required for the given neural network inference or training. All the relative outputs including time, SRAM & DRAM bandwidth will be utilized as performance metrics. The SRAM & DRAM read & write data volume will be utilized for energy consumption later on.

4.3.2 Power & Area Estimation

In this subsection, we will provide a comprehensive exposition of how our model is employed to estimate the performance of systolic array-based hardware accelerators. The Power & Area Model, as depicted in Figure 4.2, is a comprehensive tool that guides efficient power consumption and area estimation.

As depicted in Figure 4.2, the proposed methodology initiates with the construction of systolic arrays, specifically utilizing a 16x16 CHIMERA[29]-based systolic array architecture. The significance of this initial configuration is twofold: it forms the structural basis for the hardware accelerator and serves as the foundation for our prediction model. The systolic arrays are arranged in a parallel configuration, a

strategic design choice that enables comprehensive and accurate training of the base prediction model. This, in turn, augments the precision of performance estimations for systolic array-based hardware accelerators.

The entire configuration, which includes both the systolic array and the neural network, is encapsulated within a C++ codebase. The codebase is structured such that the primary interface for the configuration is a .cpp file, which is located at the top level. This .cpp file is exhaustive, capturing all relevant information derived from the configuration process.

The encapsulation of the configuration within this .cpp file ensures a seamless transition to subsequent steps, thereby facilitating an efficient and streamlined workflow. This encapsulation strategy contributes to the robustness and reproducibility of the methodology, key attributes in the realm of technical research.

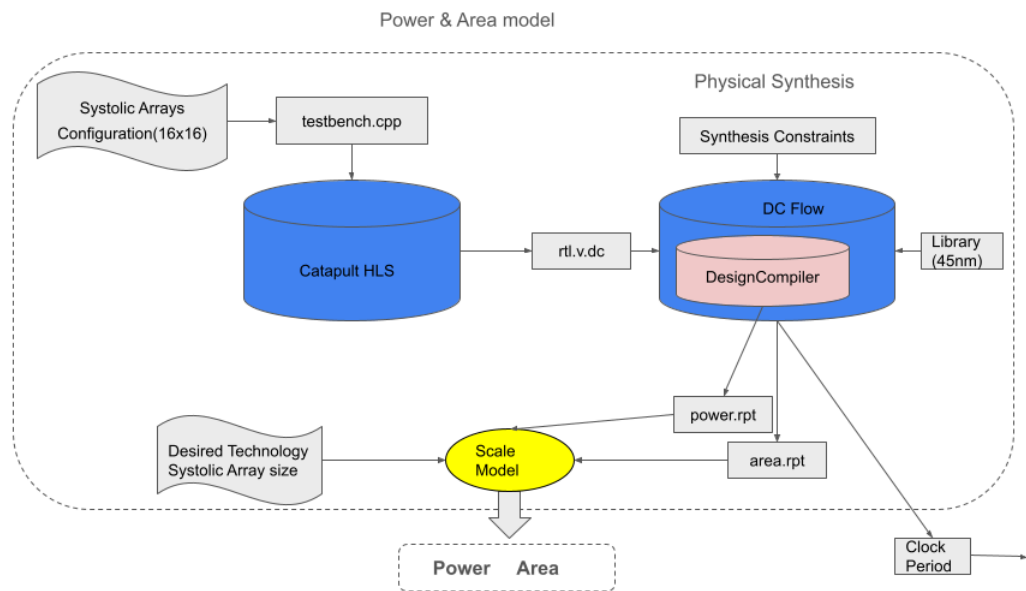


Figure 4.2 Power & Area Estimation Model

The .cpp file, which encapsulates the configuration of the systolic array and the neural network, is introduced into the Catapult HLS platform. The first operation rtl.v performed by the Catapult HLS platform is the generation of an rtl.v file. This file, derived from the .cpp file, serves a crucial role in estimating the power consumption and area of the systolic array, a topic that will be elaborated upon in the subsequent subsection. This code is then fed into the DesignCompiler, which is influenced by specific synthesis constraints (such as clock period, utilization and so on) and library elements. The DesignCompiler performs the task of physical synthesis and generates two reports: one for power consumption report (power.rpt) and the other one is area report, (area.rpt). These reports contain detailed information about the designed circuit, which is the basic systolic array design (16x16). The attained clock period will be incorporated into the performance model to estimate the time required for inference or training for a specified task.

Scale Model is designed to provide an estimation of power and area based on a few inputs including the desired systolic array size & desired technology for chip manufacturer and the base systolic arrays (16x16) power and area reports. Scale Model has two main functions: systolic array size scale and technology scale.

In terms of systolic array size scale, the estimations of power and area are derived from Equations 4 and 5, respectively. In equation 4, P_{new} represents the power of the desired size systolic array, while P_{base} denotes the power of the base systolic array. Similarly, S_{new} and S_{base} correspond to the sizes of the desired and base systolic arrays, respectively. Equation 5, on the other hand, deals with the area of the systolic array. Here, A_{new} signifies the area of the desired systolic array, and A_{base} represents the area of the

base systolic array. The terms S_{new} and S_{base} in equation 5 have the same implications as in equation 4.

$$P_{new} = P_{base} \times \left(\frac{S_{new}}{S_{base}}\right)^2 \quad (4)$$

$$A_{new} = A_{base} \times \left(\frac{S_{new}}{S_{base}}\right)^2 \quad (5)$$

Our assertion that the increments in power and area are quadratic in relation to the size increment is rooted in the structural organization of the systolic array. Specifically, the systolic array is arranged in a simple square configuration. As the size of the systolic array expands, the number of PEs experiences a quadratic increase. Consequently, this leads to a quadratic increase in both power and area relative to the size increment. This relationship underscores the inherent scalability of systolic arrays and their potential for efficient design optimization in high-performance computing systems.

In terms of technology scaling, we employ the method proposed in [39], which provides an efficient and rapid approach to estimate area, power, and speed (clock frequency) for a new technology based on an existing one.

The Scale Model is instrumental in scaling both power and area parameters, drawing upon the new data as well as the power and area reports of the base systolic array (16x16) with 45 nm technology. This model affords the flexibility to estimate the Power, Performance, and Area (PPA) of a systolic array-based hardware accelerator system across various sizes of systolic arrays and manufacturing technologies.

4.3.3 Summary of the Estimation

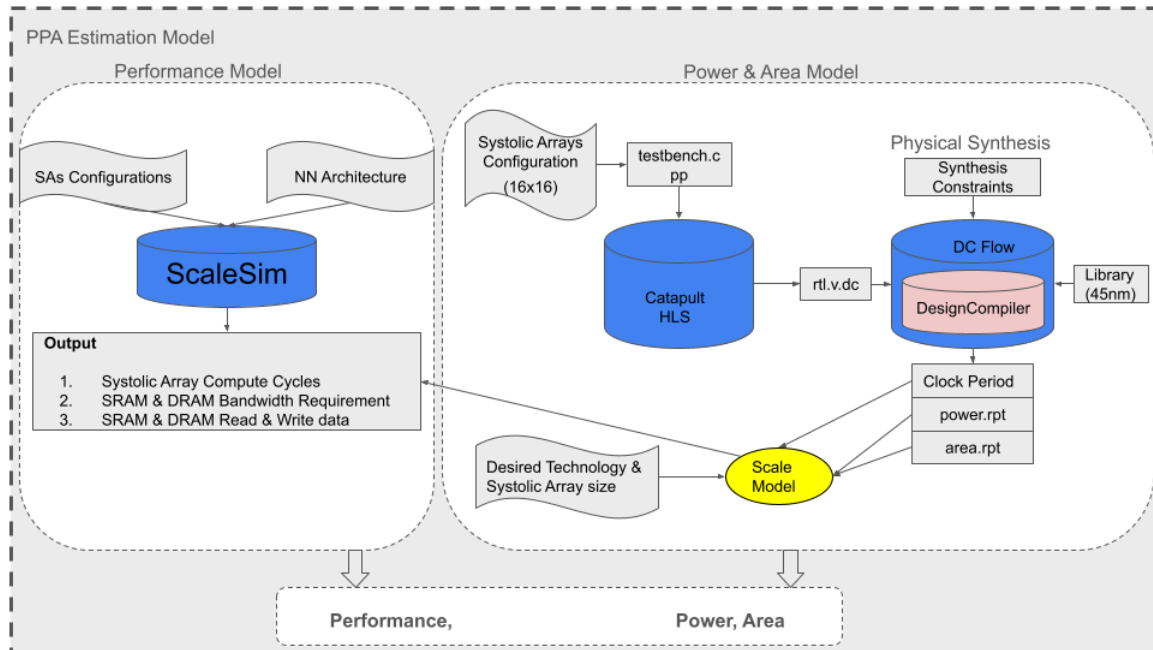


Figure 4.3 Whole Estimation Model

The entire workflow of the Power, Performance, and Area (PPA) Estimation is depicted in Figure 4.3. The Performance Estimation Model utilizes ScaleSim[30] to estimate compute cycles, SRAM & DRAM bandwidth requirement and SRAM & DRAM read & write data volume. In parallel, the Power & Area Estimation Model uses the RTL description (rtl.v) generated by HLS as the base systolic array description for power and area estimation. This model provides a comprehensive analysis of the power consumption and area utilization of the systolic array. Finally, all the information, including performance, power, and area estimations, is fed into the Scale Model. The desired architecture of the systolic array as well as the desired technology for manufacture are also inputted into the Scale Model. The Scale Model then provides an accurate estimation of the desired systolic array with desired technology in terms of performance, power, and area.

In essence, this workflow represents a holistic approach to systolic array design and optimization, ensuring that the final design meets the desired specifications in terms of performance, power consumption, and area utilization. This methodology underscores the importance of accurate estimation models in the design and optimization of complex electronic systems.

4.4 MoCo Simulation

This chapter provides a comprehensive examination of our model's approach to estimating the PPA of a system during the training of Momentum Contrast MoCo[15] on a hardware accelerator based on a systolic array. The primary focus is on how our model can effectively estimate the PPA of a system under these specific conditions. The chapter will delve into the intricacies of the process, explaining the underlying principles and methodologies involved.

To provide a concrete understanding, we will use a 16x16 systolic array as a case study. This example will guide through the complete process of PPA estimation during MoCo[20] training, highlighting the practical aspects and real-world implications of our model's approach.

4.4.1 Performance Estimation

The objective of performance estimation is to ascertain the training duration for MoCo[15] when employing a systolic array-based hardware accelerator. The performance model delineated in Section 4.3.1 will be utilized to estimate the requisite training time

for the three principal stages of MoCo[15]: forward propagation, computation of logits and loss, and backward propagation and weights update.

Prior to the commencement of the simulation, Table 4.1 delineates the fundamental parameters for the simulation configuration. As discerned from Table 4.1, a 16x16 systolic array will be utilized. The dimensions of the Input Buffer, Weight Buffer, and Output Buffer in our simulation are 512 KB, 512 KB, and 256 KB respectively. Furthermore, the data reuse strategy employed is weight stationary.

Table 4.1 Simulation Configuration

Object	Value
ArrayHeight	16
ArrayWidth	16
lmapSramSz	512KB
FilterSramSz	512KB
OfmapSramSz	256KB
Dataflow	WS

To accurately simulate the forward propagation, we employ the use of ScaleSim[30], a well-regarded tool in the field. In the study conducted by MoCo[15], ResNet-50[20] was utilized as the encoder. Drawing inspiration from this, we will employ ScaleSim[30] to simulate the forward propagation process of ResNet-50[20].

In our simulation, we input the same parameters into ScaleSim[30] as were used in ResNet-50[20]. An example of these parameters is presented in Table 4.2, which is the input layer of ResNet50[20]. Given the specified input, ScaleSim[30] is capable of determining the dimensions of the input, weight, and output matrices, as well as the number of channels for each. Additionally, it can ascertain the stride value. With this

information, ScaleSim then executes the forward propagation process, resulting in the generation of the desired output.

Table 4.2 An Example of ScaleSim Input

Layer name	IFMAP Height	IFMAP Width	Filter Height	Filter Width	Channels	Num Filter	Strides
Conv1	224	224	7	7	3	64	2

The output from ScaleSim[30] is summarized in Table 4.3. This table encapsulates all the information we require from the ScaleSim[30] simulation for our study. By analyzing this output, we can gain valuable insights into the forward propagation process, thereby advancing our understanding of this critical aspect of neural network operation. Table 4.3 presents a comprehensive overview of the data volume for both read and write operations in SRAM and DRAM, the bandwidth requirements for these operations, and the compute cycles necessary for executing the forward propagation of the ResNet-50[20] input layer. The data volume of read and write operations in SRAM and DRAM is a crucial factor in estimating power consumption in the Power & Area estimation model. Moreover, the bandwidth requirements for SRAM and DRAM read and write operations constitute a key estimation in our study. Lastly, the compute cycles will be instrumental in estimating the total training time in subsequent stages of our research. Table 4.3 provides the results of the forward propagation simulation for the initial layer of ResNet-50[20]. However, it is important to note that this is merely the beginning of our analysis. We will employ ScaleSim[30] to extend this simulation to encompass all layers of ResNet-50, adhering to the same methodology.

Table 4.3 An Example of ScaleSim Output

SRAM_read_bytes	SRAM_write_bytes	SRAM Read BW	SRAM Write BW	DRAM_read_bytes_Filter	DRAM_read_bytes_inftmap
7603840	7539716	14.76936397	14.8949749	113895	9408
DRAM_write_bytes_outfmap	DRAM IFMAP Read BW	DRAM Filter Read BW	DRAM OFMAP Write BW	Compute cycles(total one image)	Utilization
976480	0.2231061103	0.01842909949	1.912802622	480188	91.88%

The subsequent phase of our study involves the simulation of logits calculation. In the context of MoCo[15], two types of logits are computed: the positive logit and the negative logit. Mathematically, the computation of logits can be viewed as a matrix multiplication operation. The parameters for the input and output matrices involved in this operation are delineated in Table 4.4.

Table 4.4 Logits Calculation

Logit	Matrix_Input_0	Matrix_Input_2	Matrix_Output
Positive	256x1x128	256x128x1	256x1
Negative	256x128	128x65536	256x65536

For our simulation, we will utilize ScaleSim[30] to perform the logits calculation. This will be based on a 16x16 systolic array, as illustrated in this example. The output generated from this simulation is expected to bear resemblance to the data presented in Table 4.3. The results will be used for power calculations and training time estimation later on.

The final stage in our performance model simulation involves the processes of backward propagation and weight updates. Prior to this, the calculation of loss, which is

based on logits, is executed via software. Subsequently, we employ ScaleSim[30] to simulate the backward propagation and weight update processes, utilizing a systolic array for this purpose. The output generated from these processes is expected to align closely with the data presented in Table 4.3.

4.4.2 Power & Area Estimation

The estimation of power and area, as depicted in Figure 4.2, is a crucial aspect of our study. To achieve this, we will employ the HLS method. Our first step in this process involves the use of the testbench.app to delineate the specifics of the systolic array. Key configurations, such as the architecture of the systolic array, the size of the SRAM, the method of data reuse, and the algorithm to be trained (in this case, ResNet-50[20]), are all summarized in Table 4.5.

Table 4.5 Key Information in Testbench.cpp

Object	Value
ArrayHeight	16
ArrayWidth	16
IfmapSramSz	512KB
FilterSramSz	512KB
OfmapSramSz	256KB
Dataflow	WS
Algorithm	ResNet-50[20]

As delineated in Table 4.5, it is imperative that the description within testbench.cpp aligns with the configuration of the ScaleSim[30] simulation. Subsequently, the Catapult HLS tool is utilized to generate the corresponding RTL code. This RTL code is then forwarded to the DesignCompiler, which in turn generates

power.rpt and area.pt. For the physical synthesis process, a 45 nm technology is employed. The primary synthesis result is presented in Table 4.6, which will be instrumental for subsequent power and area estimations.

Table 4.6 Synthesis Result using 45 nm Technology

Object	Value
Technology	45 nm
Fastest clock period can be archived	11 ns
Power	123.98 mWatts
Area	2516016 mm ²

4.4.3 Summary of MoCo Simulation

Drawing upon the comprehensive data presented in sections 4.4.1 and 4.4.2, we are equipped to conduct a simulation of the training procedure for the MoCo[15] model. Table 4.7 provides a detailed overview of the fundamental training parameters associated with the MoCo[15] model.

Table 4.7 Default Training Information of MoCo

Dataset	ImageNet(1281167 images)
Mini-batch Size	256
Encoder	ResNet-50
Momentum Coefficient	0.999
Key Size	65536
Number of Epoch	200
Hardware Resources	8 32GB NVIDIA Volta100 GPUS
Training Time	53 hours

As delineated in Table 4.7, the MoCo[15] model underwent a training regimen spanning 200 epochs. This process was facilitated by the computational power of eight NVIDIA Volta 100 GPUs, each equipped with 32 GB of memory. The entire training

procedure required a substantial time investment, clocking in at 53 hours. For future comparative analyses, Table 4.8 provides a detailed breakdown of the key specifications of the V100 GPU.

Table 4.8 Volta GPU Key Configuration

Object	Value
Technology	12 nm
SRAM	128 x 84 KB
DRAM	32 GB
Band width(NVLink)	300 GB/s
Power	250 Watts

Table 4.9 consolidates all the pertinent data from preceding tables, providing a comprehensive overview for the purpose of conducting PPA estimation using a 16x16 systolic array-based hardware accelerator. As detailed in Table 4.9, the fast clock period, power, and area metrics are derived from the DesignCompiler, while the clock cycle data, as well as the read and write data volume information for both DRAM and SRAM, are sourced from ScaleSim[30]. This amalgamation of data facilitates a thorough and accurate PPA estimation, thereby enabling a more informed evaluation of the hardware accelerator's performance.

Table 4.9 MoCo One Mini-batch Training Information in a 16x16 Systolic Array

Systolic Array Size	16x16
Technology	45 nm
Fastest clock period can be archived	11 ns
Power	123.98 mWatts
Area	2516016 mm ²
Clock cycles for forward propagation(one mini-batch)	398735093760
Clock cycles for logits/loss(one mini-batch)	542770440
Clock cycles for backward propagation & weights update(one mini-batch)	298392700
SRAM read data volume(one mini-batch)	1514073454640 bytes
SRAM write data volume(one mini-batch)	2212389207600 bytes
Required SRAM read BW	5.75 GB/s
Required SRAM write BW	5.4 GB/s
DRAM read data volume(one mini-batch)	10447727392 bytes
DRAM write data volume(one mini-batch)	11407114768 bytes
Required DRAM read BW	1.77 GB/s
Required DRAM read BW	0.01GB/s

MoCo[15] utilizes the ImageNet[16] dataset, which comprises 1,281,167 images, and undergoes training for 200 epochs. To estimate the total training clock cycles and data transfer volume between SRAM and DRAM, we refer to Equation (6). In this equation, ‘a’ represents the parameters that include the number of clock cycles and data transfer volume between SRAM and DRAM for each mini-batch (comprising 256 images), as detailed in Table 4.9. ‘A’ signifies the parameters that include the number of clock cycles and data transfer volume between SRAM and DRAM for the entire MoCo[15] training. ‘I’ denotes the number of images in the ImageNet[16] dataset, ‘i’ represents the number of images in each mini-batch, and ‘n’ is the number of epochs. By

applying these parameters in Equation (6), we can derive a comprehensive understanding of the computational demands of the MoCo[15] training process.

$$A = a \times \frac{I}{i} \times n \quad (6)$$

Upon application of Equation (6), we obtain a comprehensive set of training data for the MoCo[15] model, as facilitated by a 16x16 Systolic Array hardware accelerator. This data is systematically presented in Table 4.10. This table serves as a valuable resource for understanding the computational dynamics of the MoCo[15] model's training process when implemented on the specified hardware accelerator.

Table 4.10 MoCo Whole Training Information in a 16x16 Systolic Array (45 nm)

Systolic Array Size	16x16
Technology	45 nm
Fastest clock period can be archived	11 ns
Power	123.98 mWatts
Area	2516016 mm ²
Clock cycles for forward propagation(200 epoch)	1.99607E+15
Clock cycles for logits/loss(200 epoch)	2717108822640
Clock cycles for backward & weights update(200 epoch)	1493753856200
SRAM read data volume(200 epoch)	7.57945E+15 bytes
SRAM write data volume(200 epoch)	1.10752E+16 bytes
Required SRAM read BW	5.75 GB/s
Required SRAM write BW	5.4 GB/s
DRAM read data volume(200 epoch)	52301323324352 bytes
DRAM write data volume(200 epoch)	57104016528608 bytes
Required DRAM read BW	1.77 GB/s
Required DRAM write BW	0.01GB/s

The results obtained with a 45 nm technology are presented in Table 4.10.

However, considering that the NVIDIA Volta is fabricated using a 12 nm process, it

becomes necessary to adjust the results from Table 4.10 to reflect this technology for a more equitable comparison. This adjustment is achieved by employing the scaling technology method[39], which yields the revised results displayed in Table 4.11.

Table 4.11 MoCo Whole Training Information in a 16x16 Systolic Array(12 nm)

Systolic Array Size	16x16
Technology	12nm
Fastest clock period can be archived	1.1518 ns
Power	35.4087 mWatts
Area	322566 mm ²
Clock cycles for forward propagation(200 epoch)	1.99607E+15
Clock cycles for logits/loss(200 epoch)	2717108822640
Clock cycles for backward & weights update(200 epoch)	1493753856200
SRAM read data volume(200 epoch)	7.57945E+15 bytes
SRAM write data volume(200 epoch)	1.10752E+16 bytes
Required SRAM read BW	5.75 GB/s
Required SRAM write BW	5.4 GB/s
DRAM read data volume(200 epoch)	52301323324352 bytes
DRAM write data volume(200 epoch)	57104016528608 bytes
Required DRAM read BW	1.77 GB/s
Required DRAM write BW	0.01GB/s

Upon comparing Tables 4.10 and 4.11, it becomes evident that the transition from 45 nm to 12 nm technology impacts certain parameters of the systolic array, including the fastest clock, power, and area. However, it is noteworthy that the clock cycles for identical operations, as well as the volume of SRAM & DRAM data transfer and the corresponding read & write bandwidth requirements, remain unaffected by this technological shift. This observation underscores the robustness of these parameters to changes in fabrication technology.

Table 4.11 provides insights into the training time (derived from the clock period and compute cycles) and the area of the hardware accelerator. To facilitate a detailed comparison with the training Power, Performance, and Area (PPA) of the NVIDIA Volta GPU, it is essential to understand the energy consumption of SRAM and DRAM. According to references [40, 41], the read and write energy consumption of SRAM with 12 nm technology are 0.0002116 picojoule/byte and 0.000023675 picojoule/byte, respectively. Contemporary DRAM consumes approximately 1 picojoule to read/write per byte of data[42]. With this information at hand, we can present the final PPA result for training MoCo[15] on a 16x16 systolic array hardware accelerator, as depicted in Table 4.12.

Table 4.12 PPA Results to Train MoCo base on a 16x16 Systolic Array

Object	Value(Systolic Array)	Volta GPU(8 pieces)
Size of systolic Array	16x16	
Tech(nm)	12	
clock_period(ns)	1.1518 3	
Area of Systolic Array	322566	
Power of Systolic Array(mWatt)	135.4	
MoCo training Time(hour)	168445.7	53
Systolic Array Energy Consumption(J)	5812.5	
SRAM Energy Consumption(J)	1.888	
DRAM Energy Consumption(J)	21881.07	
Total Energy Consumption(j)	27693.5	106000
Required SRAM read BW(GB/s)	5.75	
Required SRAM write BW(GB/s)	5.4	
Required DRAM read BW(GB/s)	1.77	
Required DRAM write BW(GB/s)	0.01	

As inferred from Table 4.12, to train MoCo[15] using ImageNet[16] for 200 epochs on a 16x16 systolic array hardware accelerator system, it would require approximately 168445.7 hours and consume around 27693.5 joules of energy. The corresponding SRAM and DRAM read & write bandwidth requirements are also outlined in Table 4.12. When compared to Volta GPUs, the use of a 16x16 systolic array hardware accelerator for training MoCo[15] results in an energy saving of about 75%, albeit at the cost of a training time that is roughly 3197 times longer. This comparison underscores the trade-offs involved in the choice of hardware accelerators for machine learning training tasks.

4.5 Result

The results presented in Section 4.4, encompassing aspects such as energy consumption, training time, area, and bandwidth requirement, are predicated on the use of a 16x16 systolic array. However, the training time, which exceeds 151 years, is deemed impractical for training MoCo[15] using the aforementioned system. Consequently, this section will investigate alternative systolic array sizes to ascertain the feasibility of training MoCo[15] on a systolic array-based hardware accelerator. This exploration aims to identify configurations that strike a balance between computational efficiency and practicality.

We will employ the same methodology (as depicted in Figure 4.3) to estimate the Power, Performance, and Area (PPA) for various systems with differing systolic array sizes. The procedure will closely mirror the steps delineated in Section 4.4. The systolic array sizes under consideration include 16x16 (as discussed in Section 4.4), 32x32,

64x64, 128x128, 256x256, and 512x512. We refrain from extending our analysis to larger systolic arrays because our investigations revealed that training MoCo[15] on a 1024x1024 systolic array-based hardware accelerator system results in an average systolic array utilization of less than 50%. This underutilization is particularly pronounced during forward propagation, where the average usage dips below 25% for 75% of the time duration.

Figure 4.4 focuses on the correlation between the systolic array size and the chip area, highlighting how increasing the array size influences the physical footprint of the chip. Figure 4.5, on the other hand, explores the relationship between the systolic array size and the total time required for training MoCo[15](log version), shedding light on the impact of array size on training efficiency. Lastly, Figure 4.6 examines how the power consumption of the chip varies with the size of the systolic array, providing a crucial perspective on the energy efficiency of different array configurations.

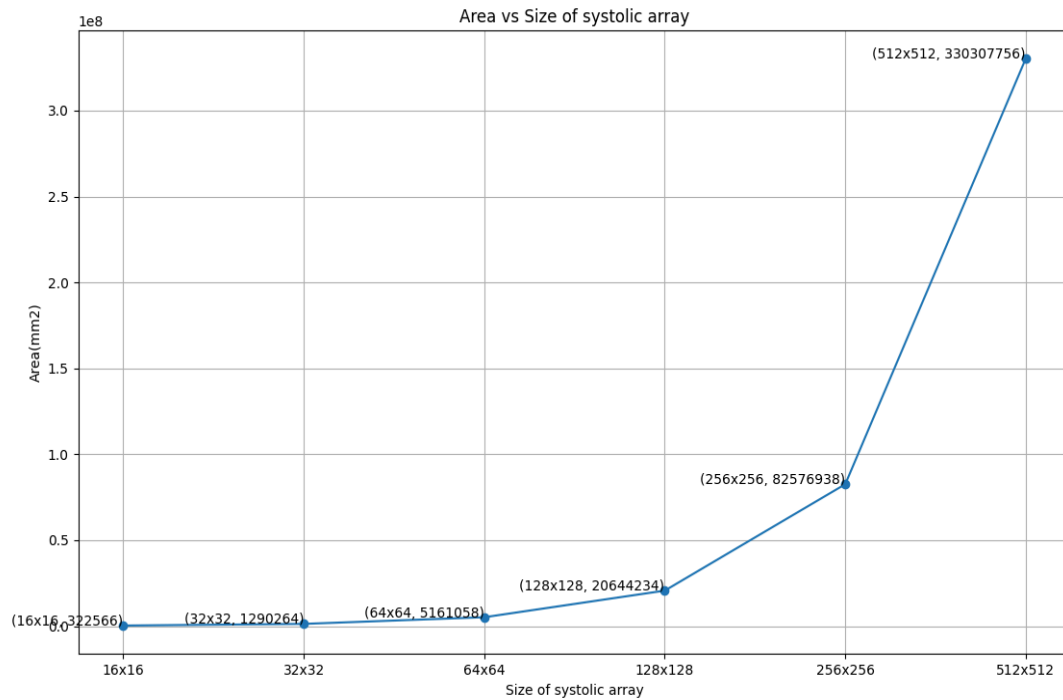


Figure 4.4 Area vs Systolic Array Size

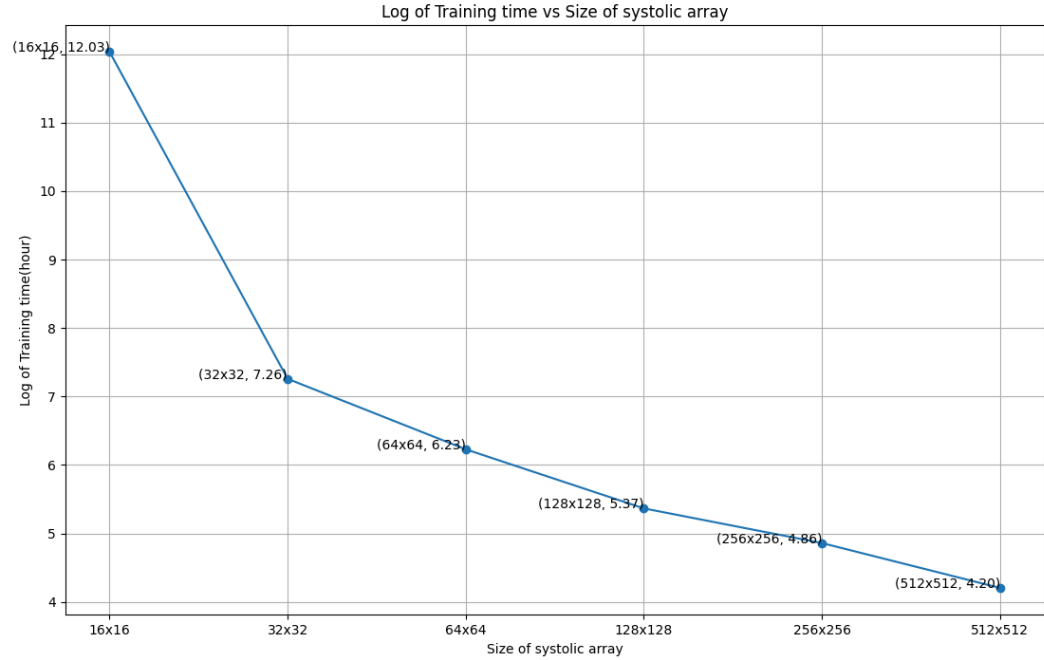


Figure 4.5 MoCo Training Time vs Systolic Array Size

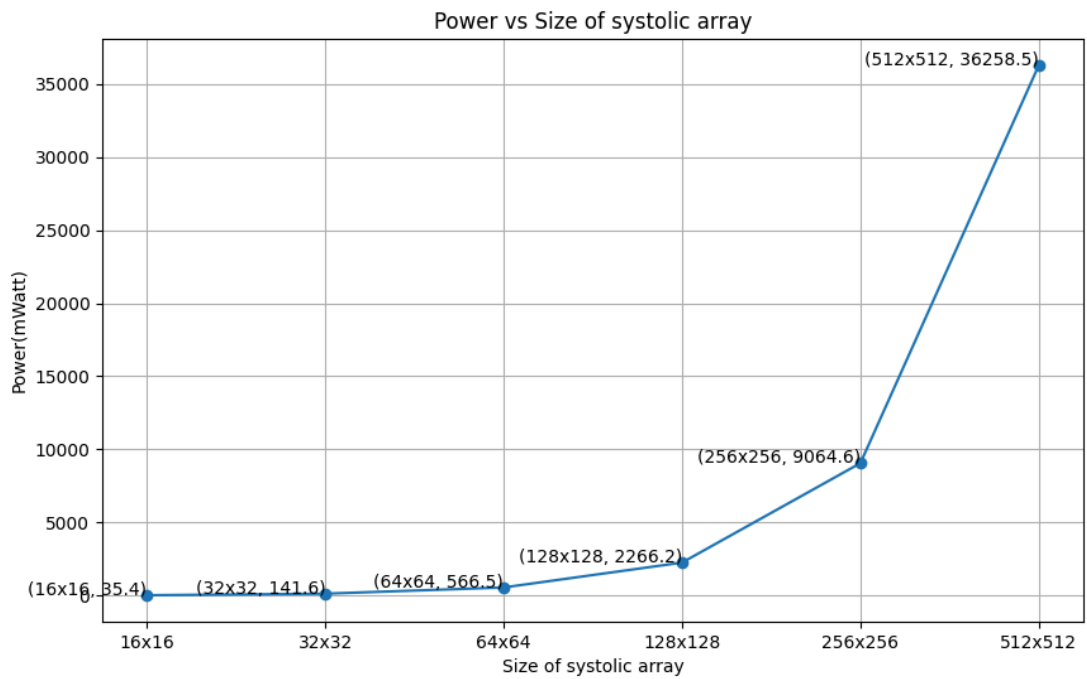


Figure 4.6 Power vs Systolic Array Size

Figure 4.7(a) focuses on the correlation between the systolic array size and SRAM energy consumption, shedding light on the impact of array size on the energy efficiency of the SRAM components. Figure 4.7(b) explores the relationship between the systolic array size and DRAM consumption, providing insights into how the DRAM's energy usage is influenced by the array size. Figure 4.7(c) examines the energy consumption of the systolic array itself as a function of its size, offering a direct measure of the array's energy efficiency. Finally, Figure 4.7(d) presents the relationship between the size of the systolic array and the total power consumption of the system. This figure provides a comprehensive view of the system's overall energy efficiency, taking into account all components and their respective energy consumptions.

These analyses collectively contribute to a deeper understanding of the energy characteristics of systolic array-based hardware accelerators, paving the way for more informed decisions in the design and utilization of such systems for machine learning tasks. They underscore the importance of selecting an appropriate systolic array size to ensure efficient energy utilization of the hardware accelerator.

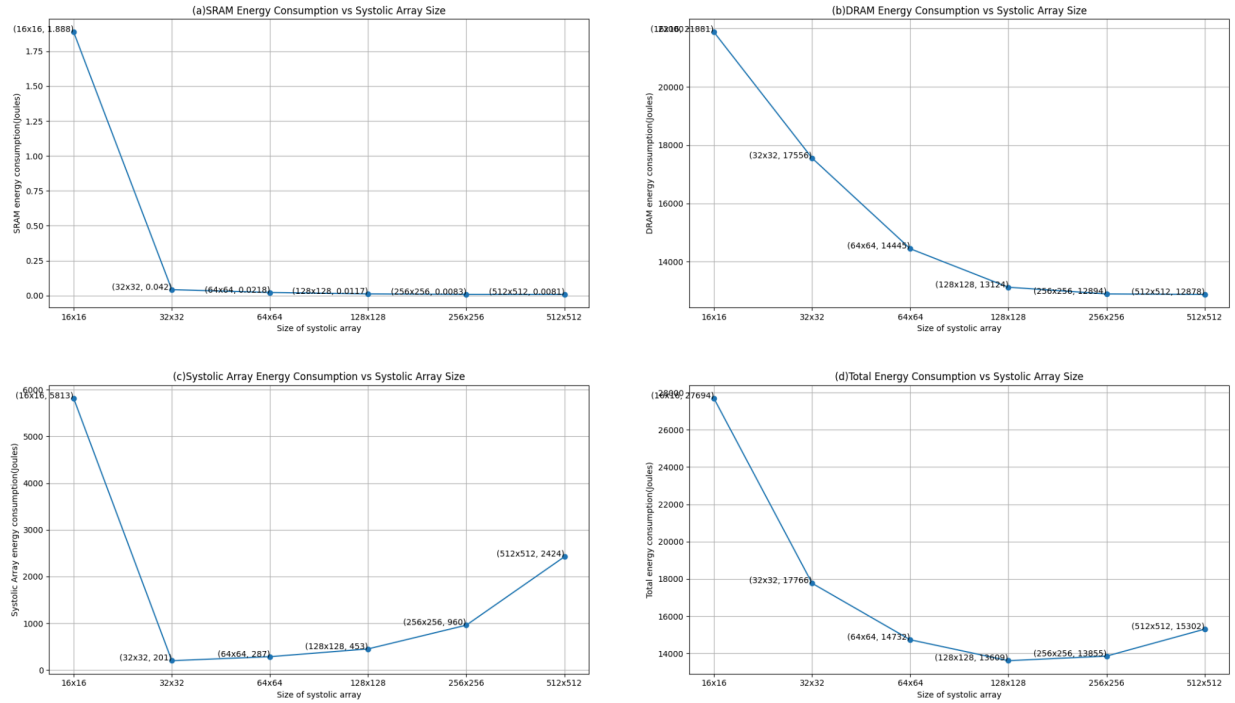


Figure 4.7 Energy Consumption vs Systolic Array Size

Figure 4.8 delves into the relationship between the size of the systolic array and the bandwidth requirements of the SRAM. This figure provides a comprehensive understanding of how the size of the systolic array influences the SRAM's bandwidth needs, thereby shedding light on the interplay between hardware configuration and memory performance.

On the other hand, Figure 4.9 investigates the correlation between the size of the systolic array and the bandwidth requirements of the DRAM. This analysis offers valuable insights into how the DRAM's bandwidth needs are affected by the size of the systolic array, highlighting the importance of array size in optimizing memory performance.

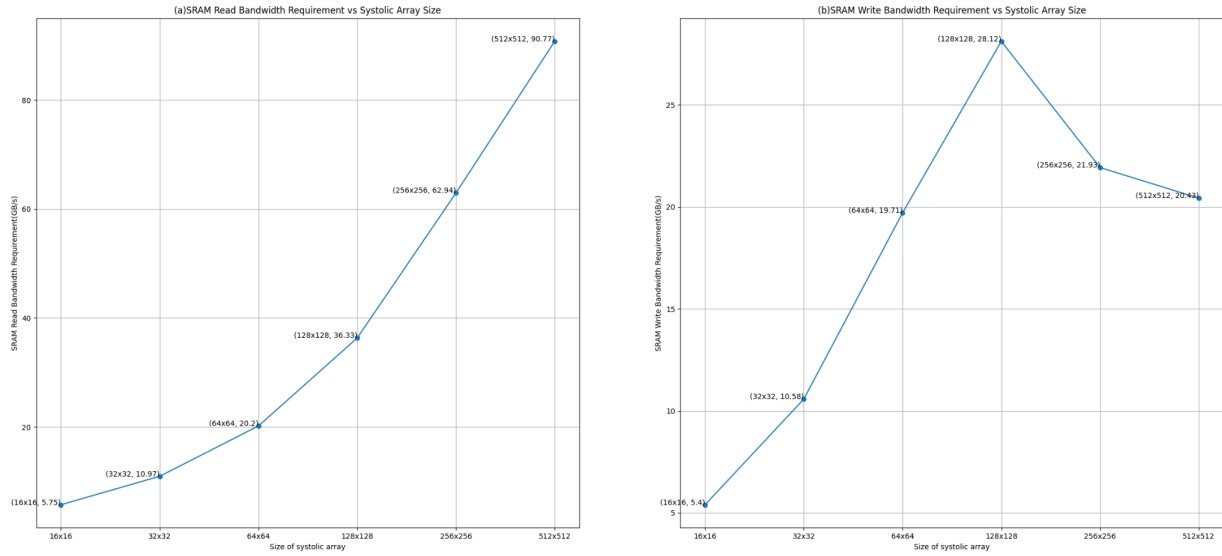


Figure 4.8 SRAM Bandwidth Requirements vs Systolic Array Size

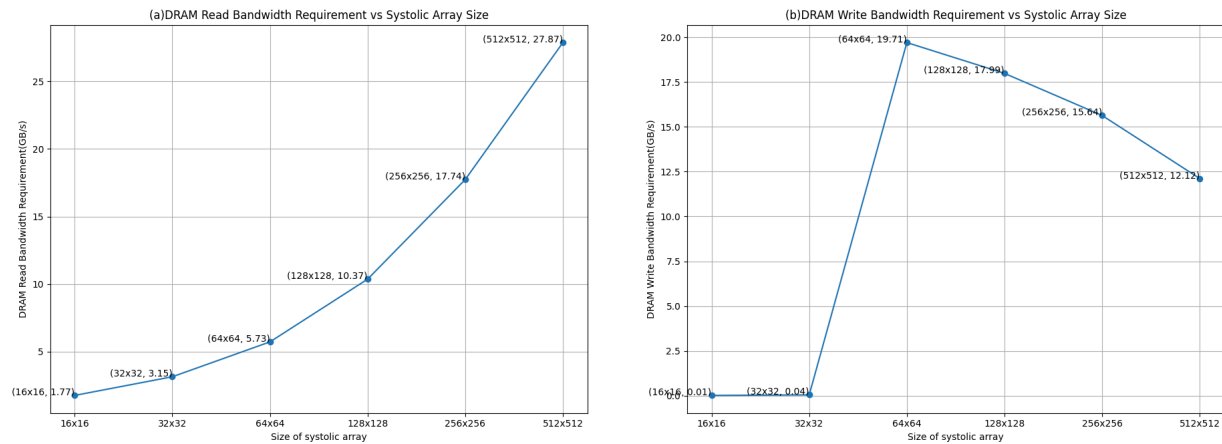


Figure 4.9 DRAM Bandwidth Requirements vs Systolic Array Size

As depicted in Figure 4.10, we observe the average utilization of systolic arrays of varying dimensions during the training process of the MoCo[15] model. This analysis provides insights into the computational efficiency and resource allocation of the hardware architecture, thereby informing future design and optimization strategies for

deep learning applications.

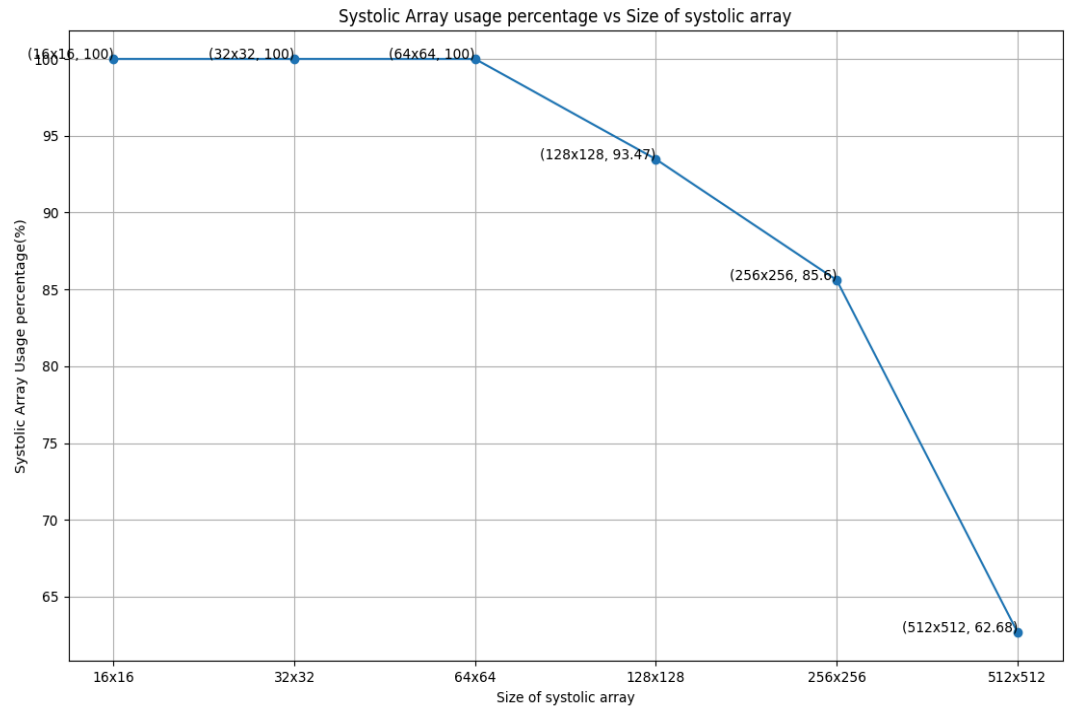


Figure 4.10 Systolic Array usage percentage vs Systolic Array Size

To streamline the analysis and foster a holistic comprehension, Table 4.13 encapsulates all salient data. This tabular compilation serves as a comprehensive repository of key performance indicators pertinent to systolic array systems of varying dimensions. The metrics encapsulated include the duration of the training phase, power utilization, the physical footprint of the systolic array, and the associated memory bandwidth requirements for both SRAM and DRAM read/write operations. This consolidation of data facilitates an efficient comparison and evaluation of the performance and resource demands of different systolic array configurations, thereby aiding in the selection and design of optimal hardware architectures for specific deep learning applications.

Table 4.13 MoCo Training Information on Different Systems

Size of systolic Array	Tech(nm)	clock_period(ns)	Area of Accelerator	MoCo training Time(hour)	Power of Accelerator (mWatt)
16x16	12	1.5158	322566	1688446	35.4
32x32	12	1.5158	1290264	1492	141.5
64x64	12	1.5158	5161056	507	566.5
128x128	12	1.5158	20644234	214	2266.2
256x256	12	1.5158	82576938	129	9064.6
512x512	12	1.5158	330307756	64	36258.5
Size of systolic Array	Accelerator Energy Consumption(J)	SRAM Energy Consumption(J)	DRAM Energy Consumption(J)	Total Energy Consumption(j)	Average usage of Systolic Arrays(%)
16x16	5812.5	1.888	21881	27693.5	100
32x32	201	0.042	17565	17766	100
64x64	287.2	0.033	14445	14732.2	100
128x128	485.3	0.012	13124	13609.3	93.47
256x256	960.6	0.008	12894	13854.9	85.6
512x512	2424.2	0.006	12878	15302.2	62.68
Size of systolic Array	Required SRAM read BW(GB/s)	Required SRAM write BW(GB/s)	Required DRAM read BW(GB/s)	Required DRAM write BW(GB/s)	
16x16	5.75	5.4	1.77	0.01	
32x32	10.97	10.58	3.15	0.04	
64x64	20.2	19.71	5.73	19.71	
128x128	36.33	28.12	10.37	17.99	
256x256	62.94	21.93	17.74	15.64	
512x512	90.77	20.43	27.87	12.12	

This tabular representation allows for a quick comparison across different systolic array sizes, providing a holistic view of how these parameters vary with the size of the systolic array. It underscores the trade-offs involved in selecting the size of the systolic

array and provides valuable insights into the performance and efficiency characteristics of different systolic array configurations.

By presenting this information in a consolidated manner, Table 4.13 aids in the decision-making process when designing and utilizing systolic array-based hardware accelerators for machine learning tasks. It highlights the importance of considering multiple factors - not just one or two metrics - in order to choose the most suitable systolic array size for a given application. This comprehensive approach ensures that the selected systolic array size optimally balances the various performance and efficiency considerations.

4.6 Analysis and Conclusion

Section 4.5 presents PPA results for various systolic array systems of different sizes. This section is divided into two main parts to provide a thorough examination of the subject matter.

The first part delves into an in-depth analysis of different systems, each characterized by a unique systolic array size. This analysis aims to elucidate the performance characteristics and efficiency metrics of these systems, highlighting how the size of the systolic array influences key parameters such as power consumption, computational performance, chip area and required SRAM & DRAM bandwidth. This part of the section provides valuable insights into the trade-offs involved in selecting the size of the systolic array and underscores the impact of this decision on the overall performance and efficiency of the hardware accelerator.

The second part of the section shifts focus to a comparative study between systolic array-based hardware accelerator systems and GPU-based systems, specifically in terms of their PPA characteristics. This comparison aims to provide a balanced perspective on the strengths and weaknesses of each type of system, thereby aiding in the decision-making process when choosing between these two types of hardware accelerators for specific applications. This part of the section underscores the importance of considering multiple factors - not just one or two metrics - in order to choose the most suitable hardware accelerator for a given task.

4.6.1 Systolic Array Analysis

The analysis of the Systolic Array will encompass key metrics such as energy consumption, training time, and area, based on the results obtained from training MoCo[15] as detailed in Section 4.5.

Firstly, the energy consumption during the training phase of MoCo[15] is primarily attributed to three main components in the system: the SRAM, and DRAM, Systolic Array. Figure 4.7 provides a comprehensive view of the energy consumption of these three components as the size of the systolic arrays varies. Specifically, Figure 4.7(a) illustrates the SRAM energy consumption for training MoCo[2] as a function of the systolic array size. Similarly, Figure 4.7(b) and Figure 4.7(c) depict the DRAM energy consumption and the Systolic Array energy consumption, respectively, for training MoCo[2] as the size of the systolic array changes. Lastly, Figure 4.7(d) presents the total energy consumption of the system for training MoCo[2] as a function of the systolic array size. As can be inferred from Figure 4.7, both SRAM and DRAM energy consumption

decrease as the size of the systolic array increases. However, the energy consumption of the Systolic Array exhibits a more complex behavior. It decreases when the size of the array increases from 16x16 to 32x32, but it increases when the size of the array further increases beyond 32x32. The total energy consumption of the system reaches its minimum when the size of the systolic array is 128x128. The reason is the usage percentage of systolic arrays. As illustrated in Figure 4.10, we examine the average utilization of systolic arrays of various dimensions during the training of the MoCo[15]. The figure reveals that when the size of the systolic array is less than or equal to 64x64, the average utilization is at its maximum capacity of 100%. However, for systolic arrays larger than 64x64, there is a noticeable decline in utilization as the size increases. This trend elucidates the rise in total energy consumption observed during the training of the MoCo[15] model when the size of the systolic array exceeds 128x128. Despite not achieving full utilization for larger systolic arrays, all PEs are presumed to be operational. Consequently, systolic arrays larger than 64x64 lead to an unnecessary expenditure of hardware resources. Specifically, a 512x512 systolic array results in a wastage of approximately 36.32% of hardware resources and corresponding energy consumption during the training of the MoCo[15] model. This observation underscores the importance of selecting an appropriate systolic array size to balance computational efficiency and resource utilization. In light of energy consumption considerations, systolic array dimensions of 128x128, 256x256, and 512x512 emerge as optimal configurations. This finding accentuates the criticality of judiciously selecting a systolic array size that strikes a balance between energy efficiency and computational prowess. It is imperative to note that the estimations for both usage and energy consumption are predicated on the

MoCo[15] model. Consequently, the implementation of alternative algorithms may yield divergent results. This is due to the inherent differences in computational complexity and efficiency among various algorithms, which can significantly impact the overall system usage and energy consumption. Therefore, any extrapolation or application of these estimations should be undertaken with due consideration of the underlying algorithmic framework. Such a balanced approach facilitates more enlightened decision-making in the design and deployment of systolic array-based hardware accelerators, thereby optimizing machine learning tasks. This not only enhances the performance of machine learning models but also contributes to the broader goal of energy-efficient computing in the realm of artificial intelligence.

The subsequent aspect to consider is the training time, which can be straightforwardly inferred from Figure 4.5. As the size of the systolic array increases, the training time consistently decreases. However, the rate of decrease diminishes as the array size grows, primarily due to the utilization of the systolic array. When training MoCo[15], if the size of the systolic array exceeds 256×256 , our observations indicate that the utilization of the systolic array falls below 100% for nearly half of the training time. Therefore, in terms of training time for MoCo[15] on a systolic array-based hardware accelerator system, the size of the systolic array must be larger than 64×64 to be considered practical.

The next aspect to consider is the area of the systolic array, as depicted in Figure 4.4. It is observed that all sizes are practical, indicating that the area of the systolic array does not pose a significant constraint in the selection of the array size. However, it is important to note that the area of the systolic array is a crucial factor in the overall size

and cost of the hardware accelerator, and therefore, it should be carefully considered in the design and selection of the systolic array size.

Figure 4.8 and 4.9 provide a comprehensive overview of the SRAM & DRAM read & write bandwidth requirements for different sizes of systolic arrays respectively. It is observed that all sizes are practical, indicating that the bandwidth requirements do not pose a significant constraint in the selection of the array size.

Upon considering all relevant factors, including energy consumption, training time, area, and bandwidth requirements for both SRAM & DRAM, it is determined that systolic arrays of sizes 256x256 and 512x512 are suitable for training MoCo[15]. These sizes strike a balance between computational efficiency and practicality, making them ideal choices for this specific task. These selected systolic array sizes will be further analyzed in the following section, where they will be compared with a GPU-based system in terms of PPA for training MoCo[15]. This comparative analysis aims to provide a balanced perspective on the strengths and weaknesses of each type of system, thereby aiding in the decision-making process when choosing between these two types of hardware accelerators for specific applications.

4.6.2 Systolic Array vs GPU

As previously discussed, systolic arrays of sizes 256x256 and 512x512 have been selected for comparison with the default training system for MoCo[15] — a system based on GPUs. We will conduct a comparative analysis between the systolic array-based system and the GPU-based system in terms of PPA.

Figure 4.11 offers a comparative perspective on the energy consumption of a 256x256 systolic array-based hardware accelerator system, a 512x512 systolic array-based hardware accelerator system, and an 8-Volta GPU-based system during the training of MoCo[15]. It provides a clear visual representation of the energy savings achieved by the systolic array-based systems when compared to the GPU-based system. These energy savings, which amount to 86.93% and 85.56% for the 256x256 and 512x512 systolic array-based systems respectively, underscore the potential of systolic array-based hardware accelerators as energy-efficient alternatives to traditional GPU-based systems for machine learning tasks. This significant improvement in energy efficiency could have far-reaching implications for the design and utilization of hardware accelerators in machine learning.

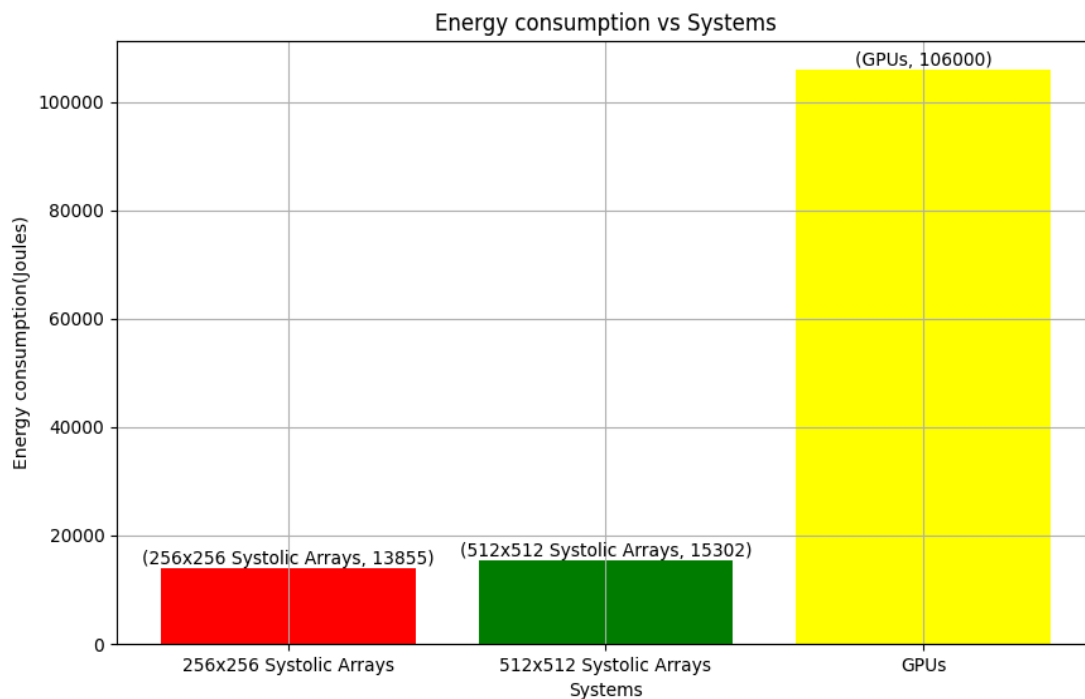


Figure 4.11 Systolic Array vs GPU(Energy Consumption)

Figure 4.12 provides a comparative analysis of the training time required by a 256x256 systolic array-based hardware accelerator system, a 512x512 systolic array-based hardware accelerator system, and an 8-Volta GPU-based system during the training of MoCo[15]. From the figure, it is evident that the 256x256 systolic array-based system requires 2.43 times more training time compared to the GPU-based system. Similarly, the 512x512 systolic array-based system necessitates 1.26 times more training time compared to the GPU-based system.

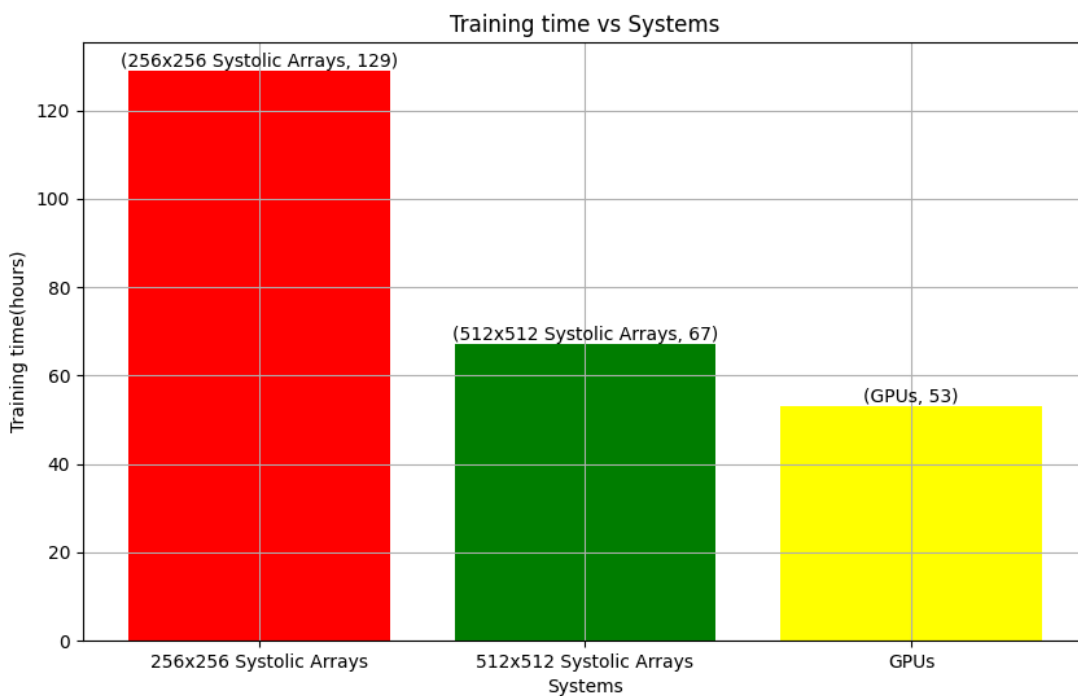


Figure 4.12 Systolic Array vs GPU(Training Time)

Figure 4.13 presents a comparative study of the DRAM bandwidth requirements for a 256x256 systolic array-based hardware accelerator system, a 512x512 systolic array-based hardware accelerator system, and an 8-Volta GPU-based system during the training of MoCo[15]. The figure clearly illustrates that the GPU-based system has

significantly higher bandwidth requirements compared to the systolic array-based system, with the difference being nearly 27-fold. This stark contrast underscores the efficiency of systolic array-based systems in terms of bandwidth utilization. Despite their larger size, these systems demonstrate a remarkable ability to manage data flow effectively, thereby reducing the demand on the DRAM bandwidth. This is particularly beneficial in machine learning tasks such as training MoCo[15], where efficient data management can significantly impact the overall performance and speed of the training process.

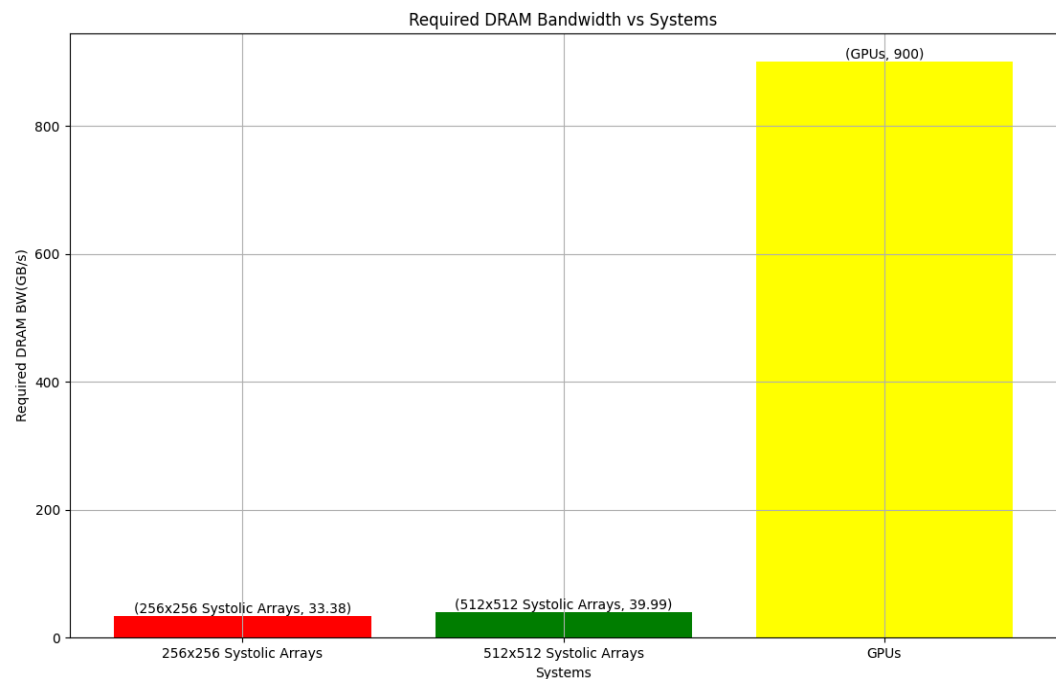


Figure 4.13 Systolic Array vs GPU(Required DRAM BW)

In the context of training MoCo[15], a systolic array-based system exhibits extraordinary advantages over a GPU-based system in terms of energy consumption. For instance, when training MoCo[15] on a 512x512 systolic array-based system, the energy efficiency can reach up to 6.99 times that of a GPU-based system. This represents a

significant leap in energy efficiency, highlighting the potential of systolic array-based systems in energy-constrained environments. When considering DRAM bandwidth requirements, the systolic array-based system demonstrates even more pronounced advantages. Specifically, when training MoCo[15], a 512x512 systolic array-based system requires only 3.67% of the DRAM bandwidth needed by a GPU-based system. This stark contrast underscores the efficiency of systolic array-based systems in managing data flow and memory usage. However, in terms of training time, the GPU-based system does hold a certain advantage. For instance, when training MoCo[15], a GPU-based system can save up to twice the amount of time compared to a 512x512 systolic array-based system. This highlights the trade-offs involved in choosing between these two types of systems, with the GPU-based system offering faster training times at the expense of higher energy consumption and bandwidth requirements.

In summary, while both systolic array-based and GPU-based systems have their own strengths and weaknesses, the choice between the two should be guided by the specific requirements of the task at hand. Factors such as energy consumption, memory bandwidth requirements, and training time should all be considered in order to select the most suitable system for a given application. This comprehensive approach ensures that the selected system optimally balances the various performance and efficiency considerations, thereby paving the way for more informed decisions in the design and utilization of hardware accelerators for machine learning tasks.

It's also worth noting that the choice of hardware accelerator can have significant implications for the scalability and cost-effectiveness of machine learning operations. For instance, while GPU-based systems may offer faster training times, their higher energy

consumption and bandwidth requirements could lead to increased operational costs over time. On the other hand, while systolic array-based systems may require more training time, their superior energy efficiency and lower bandwidth requirements could result in substantial cost savings in the long run.

Furthermore, the choice of hardware accelerator can also impact the feasibility of deploying machine learning models in resource-constrained environments. For example, systolic array-based systems, with their lower energy consumption and bandwidth requirements, could be more suitable for deployment in edge computing scenarios, where resources are typically limited.

In conclusion, the choice between systolic array-based and GPU-based systems for training MoCo[15] involves a complex interplay of various factors. By carefully considering these factors and understanding the trade-offs involved, it is possible to make an informed decision that optimizes performance, efficiency, cost-effectiveness, and scalability. This, in turn, can help pave the way for the broader adoption and deployment of machine learning models in a wide range of applications and environments.

CHAPTER 5

Conclusion and Future Work

This section primarily encapsulates the key contributions of the paper and provides a forward-looking perspective on future research directions.

5.1 Summary of Contribution

This paper makes a substantial contribution to the field of hardware accelerators for machine learning, particularly in the context of training the MoCo[15] model. The primary contribution is the development of a scalable estimation model designed to predict the performance of systolic array hardware accelerators. This model is capable of predicting the performance of the accelerator for different sizes of input, weight, and output, thereby enabling the prediction of performance for new neural networks trained on the same systolic array accelerator.

To estimate the area and power of the systolic array hardware accelerators, the paper employs high-level synthesis (HLS) methods. The use of HLS allows for the avoidance of the tedious and error-prone process of manual hardware design, enabling the quick and accurate estimation of area and power. This approach significantly streamlines the design process, making it more efficient and less prone to errors.

The model is characterized by its flexibility, capable of performing Power, Performance, and Area (PPA) estimation for different sizes of systolic arrays and different algorithms' training processes. This flexibility extends to the platform's ability to support various manufacturing technologies for different designs, further enhancing its

applicability. This adaptability makes the model a versatile tool for performance prediction, capable of accommodating a wide range of scenarios and requirements.

The paper also presents a detailed analysis of the training prediction for MoCo[15] using a specific size of systolic array, obtaining the PPA prediction of the systolic array. This analysis provides valuable insights into the performance characteristics of systolic array hardware accelerators when training MoCo[15]. It offers a detailed breakdown of the training process, highlighting the key factors that influence the performance of the systolic array.

Furthermore, the paper conducts a comparative analysis of the PPA of the systolic array hardware accelerator training MoCo[15] with the PPA of the GPU training MoCo[15]. This comparison elucidates the advantages and disadvantages of each platform, providing a balanced perspective on their respective strengths and weaknesses. The paper also discusses the factors that affect the PPA comparison, such as the systolic array size and the DRAM bandwidth, providing a comprehensive understanding of the trade-offs involved in the selection of hardware accelerators for machine learning tasks.

In conclusion, this paper makes a significant contribution to the field of machine learning hardware accelerators, providing valuable insights and paving the way for future research in this area. Its findings have the potential to inform the design and utilization of hardware accelerators, ultimately contributing to the advancement of machine learning applications. The paper's comprehensive approach to performance prediction and comparative analysis provides a solid foundation for future research, promising to drive further advancements in the field of hardware accelerators for machine learning..

5.2 Future Work

The findings of this paper not only contribute to the current understanding of systolic array-based hardware accelerators but also pave the way for numerous future research opportunities.

One promising direction for future research is the exploration of other types of hardware accelerators. Given the diverse range of machine learning tasks, each with its unique computational requirements, there is a need to investigate the suitability of various hardware accelerators for these different tasks. This could involve a comparative analysis of different hardware accelerators, examining their performance, energy efficiency, and cost-effectiveness across a range of machine learning tasks.

Another potential area of research is the development of more efficient algorithms specifically designed for systolic array-based systems. Given the unique architecture of systolic arrays, there is an opportunity to develop algorithms that can leverage this architecture to further enhance performance and energy efficiency. This could involve exploring new algorithmic techniques or optimizing existing ones to better align with the characteristics of systolic arrays.

Additionally, future work could also focus on ways to further optimize the design and utilization of systolic arrays. This could involve exploring new manufacturing technologies that could potentially reduce the cost or improve the performance of systolic arrays. Alternatively, it could involve developing more advanced estimation models that can provide more accurate predictions of the Power, Performance, and Area (PPA) of systolic array-based hardware accelerators.

Furthermore, the approach used in this paper to estimate PPA for training MoCo[15] on a systolic array-based hardware accelerator system could be extended to other learning algorithms. For instance, future research could investigate the use of this approach to estimate PPA for transformer learning algorithms when trained on a systolic array-based hardware accelerator system. This could provide valuable insights into the performance and efficiency of systolic array-based hardware accelerators for a broader range of machine learning tasks.

In conclusion, the findings of this paper open up a multitude of avenues for future research, promising to drive further advancements in the field of hardware accelerators for machine learning. By continuing to explore these avenues, we can look forward to a future where machine learning tasks can be performed more efficiently, effectively, and sustainably.

REFERENCES

- [1] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, "Disease prediction by machine learning over big data from healthcare communities," *IEEE Access*, vol. 5, pp. 8869–8879, 2017. [2] S. Ghoshal and S. S. Ghoshal, "Product review platform based on social connections," January 26 2018. US Patent App. 10/007,936.
- [2] S. Ghoshal and S. S. Ghoshal, "Product review platform based on social connections," January 26 2018. US Patent App. 10/007,936.
- [3] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.
- [4] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, ACM, 2014.
- [5] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.
- [6] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 IEEE international conference on*, pp. 6645–6649, IEEE, 2013.
- [7] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

- [8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.
- [9] A. Abeshu and N. Chilamkurti, “Deep learning: the frontier for distributed attack detection in fog-to-things computing,” *IEEE Communications Magazine*, vol. 56, no. 2, pp. 169–175, 2018.
- [10] Abhinav Jain, Hima Patel, Lokesh Nagalapatti, Nitin Gupta, Sameep Metha, Shanmukha Guttula, Shashank Mujumdar, Shazia Afzai, Ruhi Sharma Mittal, Vitobha Munigala, “Overview and Importance of Data Quality for Machine Learning Tasks” *KDD '20: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* August 2020 Pages 3561–3562
- [11] Yaoqing Yang, Ryan Theisen, Liam Hodgkinson, Joseph E. Gonzales, Kanna Ramchandran, Charles H. Martin, Michale W. Mahoney, “Test Accuracy vs. Generalization Gap: Model Selection in NLP without Accessing Training or Testing Data ” in *KDD '23: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* August 2023 Pages 3011–3021
- [12] OpenAI. (2023). ChatGPT(Mar 14 version)[Large language model].
- [13] Junyi Cai, Hao Zeng, Anming Li, Eric W.T. Ngai, “Deep learning in computer vision: A critical review of emerging techniques and application scenarios”
- [14] Chunlei Chen, Peng Zhang, Huixiang Zhang, Jiangyan Dai, Yugen Yi, Huihui Zhang, Yonghui Zhang, “Deep Learning on Computational-Resource-Limited Platforms: A Survey” *Volume 2020 | Article ID 8454327*

- [15] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, “Momentum contrast for unsupervised visual representation learning,” CoRR, vol. abs/1911.05722, 2019.
- [16] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. 2009. p. 248–55.
- [17] Rui Xu, Sheng Ma, Yang Guo, Dongsheng Li, “A Survey of Design and Optimization for Systolic Array-based DNN Accelerators” ACM Computing Surveys Volume 56, Issue 1 Article No.: 20, pp 1–37
- [18] Dey, Sumon, “ Design of a Scalable, Configurable, and Cluster-based Hierarchical Hardware Accelerator for a Cortically Inspired Algorithm and Recurrent Neural Networks” NCSU dissertation 2019]
- [19] Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997) “Face recognition: A convolutional neural-network approach.” IEEE transactions on neural networks 8 (1): 98-113.
- [20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” CoRR, vol. abs/1512.03385, 2015. [Online].
- [21] Matsuyama, E. (2020) A Deep Learning Interpretable Model for Novel Coronavirus Disease (COVID-19) Screening with Chest CT Images. Journal of Biomedical Science and Engineering, 13, 140-152. doi: 10.4236/jbise.2020.137014.
- [22] L. Jing and Y. Tian, “Self-supervised visual feature learning with deep neural networks: A survey,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 11, pp. 4037–4058, 2021.

- [23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009, pp. 248–255.
- [24] P. Coussy, D. D. Gajski, M. Meredith and A. Takach, "An Introduction to High-Level Synthesis," in *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 8-17, July-Aug. 2009, doi: 10.1109/MDT.2009.69.
- [25] A. Qin, M. Shen and N. Xiao, "A Practical High-Level Synthesis Framework," *2021 IEEE 14th International Conference on ASIC (ASICON)*, Kunming, China, 2021, pp. 1-4, doi: 10.1109/ASICON52560.2021.9620230.
- [26] L. R. Juracy, A. de Morais Amory and F. G. Moraes, "A Fast, Accurate, and Comprehensive PPA Estimation of Convolutional Hardware Accelerators," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 5171-5184, Dec. 2022, doi: 10.1109/TCSI.2022.3204932.
- [27] Catapult-C Manual and C/C++ Style Guide, Mentor Graph., Wilsonville, OR, USA, 2004.
- [28] Kurup, P., & Abbasi, T. (1995). Design Re-Use using DesignWare. In: Logic Synthesis Using Synopsys®. Springer, Boston, MA
- [29] M. Giordano et al., "CHIMERA: A 0.92 TOPS, 2.2 TOPS/W Edge AI Accelerator with 2 MByte On-Chip Foundry Resistive RAM for Efficient Training and Inference," 2021 Symposium on VLSI Circuits, Kyoto, Japan, 2021, pp. 1-2, doi: 10.23919/VLSICircuits52068.2021.9492347.
- [30] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-SIM: Systolic CNN accelerator," *Comput. Res. Repository*,

vol. abs/1811.02883, no. 1, pp. 1–11, 2018.

[31] A. Parashar et al., “Timeloop: A systematic approach to DNN accelerator evaluation,” in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.

(ISPASS), Mar. 2019, pp. 304–315

[32] Y. Nellie Wu, J. S. Emer, and V. Sze, “Accelergy: An architecturelevel energy estimation methodology for accelerator designs,” in Proc.

IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD), Nov. 2019, pp. 1–8.

[33] H. Genc et al., “Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration,” in Proc. 58th ACM/IEEE

Design Autom. Conf. (DAC), Dec. 2021, pp. 769–774.

[34] K. Asanovic et al., “The rocket chip generator,” Univ. California,

Los Angeles, CA, USA, Tech. Rep. UCB/EECS-2016-17, 2016

[35] S. Kim et al., “Transaction-level model simulator for communication limited accelerators,” Comput. Res. Repository, vol. abs/2007.14897,

no. 1, pp. 1–11, 2020.

[36] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna,

“STONNE: A detailed architectural simulator for flexible neural network accelerators,” Comput. Res. Repository, vol. 2006, no. 1, pp. 1–8, 2020

[37] (2022). Caffe. [Online]. Available: <https://caffe.berkeleyvision.org/>

[38] H. Kwon, A. Samajdar, and T. Krishna, “MAERI: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects,” ACM Special Interest

Group Program. Lang. Notices, vol. 53, no. 2, pp. 461–475, 2018.

- [39] Stillmaker, A, and Baas, B, (2017). Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm. *Integration, the VLSI Journal*, 58, 74-81
- [40] S. Barua, U. H. Irin, M. M. Azmir, M. M. A. Bappy and S. Alam, "In 12nm FinFET Technology, performance analysis of low power 6T SRAM layout designs with two different topologies," 2022 IEEE 31st Microelectronics Design & Test Symposium (MDTS), Albany, NY, USA, 2022, pp. 1-5, doi: 10.1109/MDTS54894.2022.9826987
- [41] Javad Mohagheghi, Behzad Ebrahimi, Pooya Torkzadeh, "Single-ended 6T SRAM Cell with Low Power/Energy Consumption and High Stability".
10.22060/EEJ.2022.20479.5432
- [42] Saeed Kargar¹, Faisal Nawab, "Challenges and future directions for energy, latency, and lifetime improvements in NVMs" *Distributed and Parallel Databases* 41(3):1-27
DOI:10.1007/s10619-022-07421-x
- [43] T. Tang and Y. Xie, "MIPat: A power area timing modeling framework for machine learning accelerators," in *Proc. IEEE Int. Workshop Domain Specific Syst. Archit. (DOSSA)*, Aug. 2018, pp. 1–3.
- [44] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, May 2019, pp. 754–768.
- [45] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.

- [46] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.
- [47] C. Heidorn, F. Hannig, and J. Teich, “Design space exploration for layerparallel execution of convolutional neural networks on CGRAs,” in *Proc. 23th Int. Workshop Software Compilers Embedded Syst. (SCOPEs)*, 2020, pp. 26–31
- [48] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, “DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 1593–1597
- [49] C. Hao et al., “FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge,” in *Proc. ACM/IEEE Design Automat. Conf. (DAC)*, Mar. 2019, pp. 1–6.
- [50] Chen, Y., Krishna, T., Emer, J., & Sze, V. (2016). Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*
- [51] X. Zhang, H. Ye, and D. Chen, “Being-ahead: Benchmarking and exploring accelerators for hardware-efficient AI deployment,” *Comput. Res. Repository*, vol. abs/2104.02251, no. 1, pp. 1–12, Jun. 2021.
- [52] X. Yang et al., “Interstellar: Using halide’s scheduling language to analyze DNN accelerators,” in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, 2020,

pp. 369–383

[53] S. D. Manasi and S. S. Sapatnekar, “DeepOpt: Optimized scheduling of CNN workloads for ASIC-based systolic deep learning accelerators,” in Proc. ACM/IEEE Asia South Pacific Design Automat. Conf. (ASPDAC), May 2021, pp. 235–241

[54] A. Karbachevsky et al., “Early-stage neural network hardware performance analysis,” MDPI Sustainability, vol. 13, no. 2, p. 717, 2021.

[55] C. Baskin et al., “UNIQ: Uniform noise injection for non-uniform quantization of neural networks,” ACM Trans. Comput. Syst., vol. 37, nos. 1–4, pp. 1–15, 2021.

[56] M. Ferianc et al., “Improving performance estimation for design space exploration for convolutional neural network accelerators,” MDPI Electron., vol. 10, no. 4, pp. 1–14, 2021.

[57] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, “Aladdin: A preRTL, power-performance accelerator simulator enabling large design space exploration of customized architectures,” in Proc. ACM Int. Symp. Comput. Archit. (ISCA), 2014, pp. 97–108.

APPENDICES

Appendix A

ResNet50 Forward Propagation Configuration on ScaleSim

Layer name	IFMAP Height	IFMAP Width	Filter Height	Filter Width	Channels	Num Filter	Strides
Conv1	224	224	7	7	3	64	2
CB2a_1	56	56	1	1	64	64	1
CB2a_2	56	56	3	3	64	64	1
CB2a_3	56	56	1	1	64	256	1
CB2s	56	56	1	1	64	256	1
IB2b_1	56	56	1	1	256	64	1
IB2b_2	56	56	3	3	64	64	1
IB2b_3	56	56	1	1	64	256	1
IB2c_1	56	56	1	1	256	64	1
IB2c_2	56	56	3	3	64	64	1
IB2c_3	56	56	1	1	64	256	1
CB3a_1	56	56	1	1	256	128	2
CB3a_2	28	28	3	3	128	128	1
CB3a_3	28	28	1	1	128	512	1
CB3s	56	56	1	1	256	512	2
IB3b_1	28	28	1	1	512	128	1
IB3b_2	28	28	3	3	128	128	1
IB3b_3	28	28	1	1	128	512	1
IB3c_1	28	28	1	1	512	128	1
IB3c_2	28	28	3	3	128	128	1
IB3c_3	28	28	1	1	128	512	1
IB3d_1	28	28	1	1	512	128	1
IB3d_2	28	28	3	3	128	128	1
IB3d_3	28	28	1	1	128	512	1
CB4a_1	28	28	1	1	512	256	2
CB4a_2	14	14	3	3	256	256	1
CB4a_3	14	14	1	1	256	1024	1
CB4s	28	28	1	1	512	1024	2
IB4b_1	14	14	1	1	1024	256	1
IB4b_2	14	14	3	3	256	256	1
IB4b_3	14	14	1	1	256	1024	1

IB4c_1	14	14	1	1	1024	256	1
IB4c_2	14	14	3	3	256	256	1
IB4c_3	14	14	1	1	256	1024	1
IB4d_1	14	14	1	1	1024	256	1
IB4d_2	14	14	3	3	256	256	1
IB4d_3	14	14	1	1	256	1024	1
IB4e_1	14	14	1	1	1024	256	1
IB4e_2	14	14	3	3	256	256	1
IB4e_3	14	14	1	1	256	1024	1
IB4f_1	14	14	1	1	1024	256	1
IB4f_2	14	14	3	3	256	256	1
IB4f_3	14	14	1	1	256	1024	1
CB5a_1	14	14	1	1	1024	512	2
CB5a_2	7	7	3	3	512	512	1
CB5a_3	7	7	1	1	512	2048	1
CB5s	14	14	1	1	1024	2048	2
IB5b_1	7	7	1	1	2048	512	1
IB5b_2	7	7	3	3	512	512	1
IB5b_3	7	7	1	1	512	2048	1
IB5c_1	7	7	1	1	2048	512	1
IB5c_2	7	7	3	3	512	512	1
IB5c_3	7	7	1	1	512	2048	1
FC6	1	1	1	1	2048	1000	1

Appendix B

ResNet50 Backward Propagation Configuration on ScaleSim

Layer name	IFMAP Height	IFMAP Width	Filter Height	Filter Width	Channels	Num Filter	Strides
Conv1	224	224	56	56	3	64	2
CB2a_1	56	56	56	56	64	64	1
CB2a_2	56	56	56	56	64	64	1
CB2a_3	56	56	56	56	64	256	1
CB2s	56	56	56	56	64	256	1
IB2b_1	56	56	56	56	256	64	1
IB2b_2	56	56	56	56	64	64	1
IB2b_3	56	56	56	56	64	256	1
IB2c_1	56	56	56	56	256	64	1
IB2c_2	56	56	56	56	64	64	1
IB2c_3	56	56	56	56	64	256	1
CB3a_1	56	56	28	28	256	128	2
CB3a_2	28	28	56	56	128	128	1
CB3a_3	28	28	28	28	128	512	1
CB3s	56	56	28	28	256	512	2
IB3b_1	28	28	28	28	512	128	1
IB3b_2	28	28	28	28	128	128	1
IB3b_3	28	28	28	28	128	512	1
IB3c_1	28	28	28	28	512	128	1
IB3c_2	28	28	28	28	128	128	1
IB3c_3	28	28	28	28	128	512	1
IB3d_1	28	28	28	28	512	128	1
IB3d_2	28	28	28	28	128	128	1
IB3d_3	28	28	28	28	128	512	1
CB4a_1	28	28	14	14	512	256	2
CB4a_2	14	14	28	28	256	256	1
CB4a_3	14	14	14	14	256	1024	1
CB4s	28	28	14	14	512	1024	2
IB4b_1	14	14	14	14	1024	256	1
IB4b_2	14	14	14	14	256	256	1
IB4b_3	14	14	14	14	256	1024	1

IB4c_1	14	14	14	14	1024	256	1
IB4c_2	14	14	14	14	256	256	1
IB4c_3	14	14	14	14	256	1024	1
IB4d_1	14	14	14	14	1024	256	1
IB4d_2	14	14	14	14	256	256	1
IB4d_3	14	14	14	14	256	1024	1
IB4e_1	14	14	14	14	1024	256	1
IB4e_2	14	14	14	14	256	256	1
IB4e_3	14	14	14	14	256	1024	1
IB4f_1	14	14	14	14	1024	256	1
IB4f_2	14	14	14	14	256	256	1
IB4f_3	14	14	14	14	256	1024	1
CB5a_1	14	14	7	7	1024	512	2
CB5a_2	7	7	14	14	512	512	1
CB5a_3	7	7	7	7	512	2048	1
CB5s	14	14	7	7	1024	2048	2
IB5b_1	7	7	7	7	2048	512	1
IB5b_2	7	7	7	7	512	512	1
IB5b_3	7	7	7	7	512	2048	1
IB5c_1	7	7	7	7	2048	512	1
IB5c_2	7	7	7	7	512	512	1
IB5c_3	7	7	7	7	512	2048	1
FC6	1	1	1	1	2048	1000	1

Appendix C**ResNet50 Loss Function Configuration on ScaleSim**

Layer name	IFMAP Height	IFMAP Width	Filter Height	Filter Width	Channels	Num Filter	Strides
Positive	256	128	128	1	1	1	1
Negative	256	128	128	1	1	65536	1