Title

Hardware Modeling and Efficient Architectural Exploration for Machine Learning Accelerators

Author

Tang, Tianqi

Publication Date

2022

University of California
Santa Barbara

Hardware Modeling and Efficient Architectural Exploration for Machine Learning Accelerators

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Tianqi Tang

Committee in charge:

Professor Yuan Xie, Chair
Professor Yufei Ding
Professor Zheng Zhang
Professor Jonathan Balkind
Dr. Norman P. Jouppi

June 2022

The Dissertation of Tianqi Tang is approved.

Professor Yufei Ding

Professor Zheng Zhang

Professor Jonathan Balkind

Dr. Norman P. Jouppi

Professor Yuan Xie, Committee Chair

June 2022

Hardware Modeling and Efficient Architectural Exploration for Machine Learning Accelerators

To my friends.

Acknowledgements

First of all, I would like to thank my advisor, Prof. Yuan Xie, sincerely. I have learnt a lot from his broad knowledge in the field of computer architecture and CAD/EDA, his passion and open-mindedness to keep exploring the unknowns, his courage and good taste to work on real problems, and his kindness to support young people. Words are pale to say all my gratitude. Hope I would have opportunity to forward what I learnt from Prof. Xie to the next-generation researchers some day in the future.

Many thanks to Prof. Yufei Ding, Prof. Zheng Zhang, Prof. Jonathan Balkind, and Dr. Norm Jouppi for serving in my committee and giving valuable feedbacks to this thesis. I would like to thank Dr. Sheng Li, Dr. Lifeng Nai, Dr. Summer Deng, and Dr. Peter Tang, for their insightful mentorship during my internship at Google and Facebook. I would like to thank Dr. Shengcheng Wang from Alibaba, for his feedbacks on Chp. 6.

It takes a village to raise a Ph.D. I would like to express my gratitude to all the people I met in UCSB, all the members I collaborated with in the SEAL group, all the friends I made from all sorts of academic and industrial events, all the co-authors and the anonymous reviewers of my papers. Thank you for keeping me accompanied in this long journey, encouraging me to make all the successful and clumsy attempts, witnessing all my sweet and struggling memories. Especially, Shuangchen Li, Xing Hu, Xueqi Li, Fengbin Tu, Jiayi Huang, Maohua Zhu, Dylan Stow, Abanti Basak, Peng Gu, Xinfeng Xie, Boyuan Feng, Liu Liu, Wenqin Huangfu, Nan Wu, Zhaodong Chen, Jilan Lin.

Last but not least, I would like to thank my husband, Gushu Li, for all the joys and tears we had from Beijing to Goleta. I would like to thank my best sisterhoods from Xutong Wang, Wei Li, and Yi Ding. I am the last one among the four who got the degree. Anyway, I made it, finally. I would like to thank my parents, Hongyi Tang and Huimin Qi, whose words and actions motivate me to be a better person.

Curriculum Vitæ

Tianqi Tang

Education

2022	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara.
2017	M.S. in Electronic Engineering, Tsinghua University.
2014	B.S. in Electronic Engineering, Tsinghua University.

Publications

[C1]. Boyuan Feng, **Tianqi Tang**, Yuke Wang, Zhaodong Chen, Zheng Wang, Shu Yang, Yuan Xie, Yufei Ding. "Faith: An Efficient Framework for Transformer Verification on GPUs." *USENIX Annual Technical Conference (USENIX ATC)*, 2022.

[C2]. **Tianqi Tang**, Sheng Li, Lifeng Nai, Norm Jouppi, Yuan Xie. "NeuroMeter: An Integrated Power, Area, and Timing Modeling Framework for Machine Learning Accelerators." *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[C3]. Yi Cai, **Tianqi Tang**, Lixue Xia, Ming Cheng, Zhenhua Zhu, Yu Wang, Huazhong Yang. "Training Low Bitwidth Convolutional Neural Network on RRAM." *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018.

[C4]. Fang Su, Wei-Hao Chen, Lixue Xia, Chieh-Pu Lo, **Tianqi Tang**, Zhibo Wang, Kuo-Hsiang Hsu, Ming Cheng, Jun-Yi Li, Yuan Xie, Yu Wang, Meng-Fan Chang, Huazhong Yang, Yongpan Liu. "A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory." *Symposium on VLSI Circuits* , 2017.

[C5]. **Tianqi Tang**, Lixue Xia, Boxun Li, Yu Wang, Huazhong Yang "Binary Convolutional Neural Network on RRAM." *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[C6]. Wenqin Huangfu, Lixue Xia, Ming Cheng, Xiling Yin, **Tianqi Tang**, Boxun Li, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, Huazhong Yang. "Computation-Oriented Fault-Tolerance Schemes for RRAM Computing Systems." *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[C7]. Lixue Xia, **Tianqi Tang**, Wenqin Huangfu, Ming Cheng, Xiling Yin, Boxun Li, Yu Wang, Huazhong Yang "Switched by Input: Power Efficient Structure for RRAM-based Convolutional Neural Network." *Design Automation Conference (DAC)*, 2016.

[C8]. Lixue Xia, Boxun Li, **Tianqi Tang**, Peng Gu, Xiling Yin, Wenqin Huangfu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, Huazhong Yang. "MNSIM: Simulation Platform for Memristor-based Neuromorphic Computing System." *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2016.

[C9]. Yu Wang, Lixue Xia, **Tianqi Tang**, Boxun Li, Song Yao, Ming Cheng, Huazhong Yang. "Low power Convolutional Neural Networks on a chip." *International Symposium on Circuits and Systems (ISCAS)*, 2016.

[C10]. Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, **Tianqi Tang**, Ningyi Xu, Sen Song, Yu Wang, Huazhong Yang. "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network." *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016.

[C11]. Yung-Hsiang Lu, Alan M Kadin, Alexander C Berg, Thomas M Conte, Erik P DeBenedictis, Rachit Garg, Ganesh Gingade, Bichlien Hoang, Yongzhen Huang, Boxun Li, Jingyu Liu, Wei Liu, Huizi Mao, Junran Peng, **Tianqi Tang**, Elie K Track, Jingqiu Wang, Tao Wang, Yu Wang, Jun Yao. "Rebooting Computing and Low-Power Image Recognition Challenge." *International Conference on Computer-Aided Design (ICCAD)*, 2015.

[C12]. **Tianqi Tang**, Lixue Xia, Boxun Li, Rong Luo, Yiran Chen, Yu Wang and Huazhong Yang "Spiking Neural Network with RRAM: Can We Use It for Real-World Application?" *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2015.

[C13]. Deming Zhang, Lang Zeng, Yuanzhuo Qu, Zhang Mengxing Wang, Weisheng Zhao, **Tianqi Tang**, Yu Wang. "Energy-Efficient Neuromorphic Computation based on Compound Spin Synapse with Stochastic Learning." *International Symposium on Circuits and Systems (ISCAS)*, 2015.

[C14]. Yu Wang, **Tianqi Tang**, Lixue Xia, Boxun Li, Peng Gu, Huazhong Yang, Hai Li, Yuan Xie. "Energy Efficient RRAM Spiking Neural Network for Real Time Classification." *Great Lakes Symposium on VLSI (GLSVLSI)*, 2015.

[C15]. Peng Gu, Boxun Li, **Tianqi Tang**, Shimeng Yu, Yu Cao, Yu Wang, Huazhong Yang. "Technological Exploration of RRAM Crossbar Array for Matrix-Vector Multiplication." *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015.

[C16]. **Tianqi Tang**, Rong Luo, Boxun Li, Hai Li, Yu Wang, Huazhong Yang. "Energy Efficient Spiking Neural Network Design with RRAM Devices." *International Symposium on Integrated Circuits (ISIC)*, 2014.

[J1]. Jilan Lin, Cheng-Da Wen, Xing Hu, **Tianqi Tang**, Chao Lin, Yu Wang, Yuan Xie. "Rescuing RRAM-based Computing from Static and Dynamic Faults." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020.

[J2]. Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, **Tianqi Tang**. "A Survey of Accelerator Architectures for Deep Neural Networks." *Engineering*, 2020.

[J3]. Yi Cai, **Tianqi Tang** (**Equal Contribution**), Lixue Xia, Boxun Li, Yu Wang, Huazhong Yang. "Low Bit-width Convolutional Neural Network on RRAM." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.

[J4]. Lixue Xia, Boxun Li, **Tianqi Tang**, Peng Gu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, Huazhong Yang. "MNSIM: Simulation Platform for Memristor-based Neuromorphic Computing System." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAS)*, 2018.

[J5]. Lixue Xia, Wenqin Huangfu, **Tianqi Tang**, Xiling Yin, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, Huazhong Yang. "Stuck-at Fault Tolerance in RRAM Computing Systems." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, 2017.

[J6]. Huizi Mao, Song Yao, **Tianqi Tang**, Boxun Li, Jun Yao, Yu Wang "Towards Real-Time Object Detection on Embedded Systems." *IEEE Transactions on Emerging Topics in Computing (TETC)*, 2016. **(Invited Technical Report for LPIRC'15 Winner)**

[W1]. **Tianqi Tang**, Yuan Xie. "Cost-Aware Exploration for Chiplet-Based Architecture with Advanced Packaging Technologies." *The 1st International Workshop on High Performance Chiplet and Interconnect Architectures (HiPChips)*, in conjunction with *the 49th International Symposium on Computer Architecture (ISCA)*, 2022.

[W2]. **Tianqi Tang**, Sheng Li, Yuan Xie, Norm Jouppi "MLPAT: A Power, Area, Timing Modeling Framework for Machine Learning Accelerators." *The 1st International Workshop on Domain Specific System Architecture (DOSSA)*, in conjunction with *the 51st International Symposium on Microarchitecture (MICRO)*, 2018.

Please Note: Text, tables, and figures from these papers are used and appear in this dissertation.

Abstract

Hardware Modeling and Efficient Architectural Exploration for Machine Learning Accelerators

by

Tianqi Tang

The innovation in computer architecture and the development of simulation tools are influencing each other mutually. The booming of machine learning (ML) invokes new modeling tools for novel architectures and emerging applications. Meanwhile, the continuous evolution of ML models drives the need to know how well the domain-specific accelerators can be adapted to a broad spectrum of ML workloads with satisfying performance and high utilization at the early design stage.

To address those challenges, this thesis focuses on hardware modeling and efficient architectural exploration of ML accelerators. The hardware modeling is conducted from two perspectives. First, this thesis develops NeuroMeter, an integrated power, area, and timing modeling framework for ML accelerators. It enables the runtime analysis of system-level performance and efficiency at the early design stage. Second, this thesis develops the cost model with an emphasis on the 2.5D integration and chiplet system. Leveraging the proposed hardware modeling frameworks, this thesis explores the efficient architectural design for ML workloads under different scenarios. Two broad classes of architectures are explored, i.e., the brawny design, which adopts small numbers of large cores; and the wimpy design, which adopts large numbers of small cores. This thesis explores the pros and cons of these two classes of architectures; and proposes a reconfigurable systolic array-based architecture that has the advantages of both these two architectures with negligible overheads.

Contents

Chapter 1

Introduction

1.1 Motivation

Recently, machine learning (ML) techniques have been widely applied in many application domains. For example, the convolutional neural networks (CNNs) [1, 2, 3] have been used in image processing tasks like classification [4], object detection [5], and autonomous driving [6]; the attention models or transformers [7, 8] achieve the state-of-the-art performance on many natural language processing (NLP) tasks; the graph neural networks (GNN) [9] have been used for distilling knowledge from the unstructured data and been widely applied in the fields of recommendation system [10], protein structure prediction [11], and traffic prediction [12].

To facilitate the powerful ML techniques under different scenarios, different application-specific accelerators are developed because a general-purpose processor usually cannot meet up all the user requirements. In terms of user scenarios, some accelerators are specialized for data center, for example, Google's tensor processing unit (TPU) [13, 14, 15, 16], Meta's Zion and Kings Canyon [17, 18], and Alibaba's Hanguang [19]; others are designed for edge/embedded system, for example, Google's EdgeTPU [20] and nVidia's Jetson-Series [21]. In terms of functionalities, some accelerators are designed for training [14, 15] while some others are designed for inference [13, 16, 17, 19].

Despite the success of these ML accelerators, the time to design and tape out an accelerator can be much longer than that of the emergence of new ML algorithms/models. Therefore, an accelerator must be able to accommodate the ever-evolving ML models with programmability. This drives the need to evaluate how well the domain-specific accelerators can be adapted to a broad spectrum of ML workloads with satisfying performance and high utilization at the early design stage. Taking CNN as an example, the operators of depthwise and pointwise convolution have taken a larger and larger proportion in the emerging CNN models [2, 3]. These emerging operators have different operational intensities and memory access patterns from the traditional convolution when running on the array-based or vector-based ML accelerators [16, 22, 23] proposed previously. New tools, such as analytical hardware modeling tools, are naturally invoked for the new architectures and applications. These tools are expected to enable fast yet accurate power, area, and timing modeling for the early-staged design decision.

In addition to the evolution of ML algorithms, more hardware techniques are also emerging. In particular, the technology of chiplet and package-level heterogeneous integration [24, 25, 26, 27, 28] has been adopted in recent years with the slowdown of Moore's Law. The chiplet-based System-in-Package (SiP) technology enables more design flexibility via various inter-chiplet connections and heterogeneous integrations. However, it is not known how to convert such flexibility into cost efficiency, which is critical when making a design decision. An analytical cost model is in need to help the designer select the interposer, figure out the design partition granularity, and determine the technology node for cost-efficient chiplet-based SiP design. The cost model could be a good supplement beyond the analytical hardware model of power, area, and timing. It helps the user to evaluate whether the design flexibility of the chiplet-based SiP technology can be converted into cost efficiency or cost feasibility at the early design stage.

1.2 Contribution

The primary goal of this thesis research is to advance the ML accelerator design methodology with collaborative contributions on hardware modeling and cost analysis. Leveraging the tools developed in this thesis research, architectural innovations are proposed to accommodate a broad spectrum of emerging ML models.

1. **Analytical Hardware Modeling** This thesis develops NeuroMeter [29], an integrated power, area, and timing modeling framework for ML accelerators. With the integrated electrical characteristics models (e.g., resistance-capacitance path model and Elmore delay model [30]) of typical circuit structures in it, NeuroMeter, the proposed modeling tool, allows the user to input the high-level architectural specifications and technology enablers alone. NeuroMeter can formulate the possible circuit representations of ML accelerators, explore the design space internally and transparently, and finally generate a fast and accurate estimation on power, area, and chip timing. It enables the runtime analysis of system-level performance and efficiency when the runtime activity factors are provided at the pre-register-transfer-level (pre-RTL) design stage.

2. **Cost-Aware Analysis on Chiplet and Heterogeneous Integration** This thesis develops an analytical cost model [31] that can estimate the cost of the 2.5D chiplet-based SiP systems under various interconnection options and technology nodes. The cost modeling is used as a critical supplement beyond the analytical modeling of power, area, and chip timing. It helps the user evaluate whether the design flexibility enabled by various inter-chiplet connections and heterogeneous integrations can be converted into cost efficiency at the early design stage. Leveraging the proposed analytical cost model, a series of case studies are conducted to explore the cost characteristics of the 2.5D

chiplet-based SiP systems in different scenarios. By analysing the case study results, several observations are made on the interposer selection, design partition granularity, and technology node adoption for cost-efficient chiplet-based SiP design.

3. Efficient Architecture Exploration By combining the power, area, and timing results of NeuroMeter with performance simulation, this thesis explores the manycore ML accelerator design in different scenarios; and proposes new architectures which are listed as follows:

First, the brawny vs wimpy study shows that brawny designs with 64x64 systolic arrays are the most performant and efficient for inference tasks in the 28nm datacenter architectural space with a 500mm^2 die area budget. The exploration also reveals important tradeoffs between performance and efficiency. For datacenter accelerators with small batch-sized inference, a tiny (∼16%) sacrifice of performance can lead to more than a 2x efficiency[1] improvement.

Second, this thesis also conducts a case study on energy efficiency (TOPS/Watt) implications of sparsity on different ML accelerators to showcase NeuroMeter's capability to model a wide range of accelerator architectures. The results show that despite its relatively low energy efficiency, it is easier for wimpier accelerator architectures to benefit from sparsity processing.

Third, this thesis proposes a reconfigurable systolic array (RSA) based CNN accelerator, which introduces quite a small area and power overhead as well as minor changes in control logics. The internal reconfigurability of the proposed RSA enables both the high performance of small numbers of brawny systolic arrays and the high utilization of large numbers of wimpy systolic arrays. Especially, it increases the array utilization largely for the operators of depthwise convolution with low operational intensity. The results show

---

[1] Efficiency is measured by the proxy metric of TOPS/TCO, where TOPS is the tera-operations per second and TCO is the total cost of ownership.

that the performance of ResNet [1] and EfficientNet [3] achieves 1.25x-2.68x gain on the proposed RSA-based accelerator compared with the TPU-v2-styled baseline [14].

1.3 Future Influence and Impact

The experiences from this thesis research naturally lead to a wide range of future directions. Looking forward, the borderline between software optimization and architecture-level design is blurring. There is a large space to explore for hardware-software-algorithm co-optimization. This thesis research addresses the urge for the early-staged hardware modeling for domain-specific accelerator design; and showcases examples of hardware-and-software co-design space for ML accelerators to address a broad spectrum of workload characteristics. The thesis research will foster the exploration on the co-design workflow and the automation toolset to increase the productivity on domain-specific accelerator design for future work.

The hardware analytical modeling tool proposed in this thesis can be further developed to support more advanced technology nodes and more complex 2.5D/3D integrated systems. As the technology node scaling down, the circuit-level electrical characteristics models (e.g., resistance-capacitance path model and Elmore's model [30]) for FinFET [32] and gate-all-around (GAA) nanosheet [33] are required if the advanced technology nodes ($\leq$7nm) are planned to support in the future hardware modeling tools. In addition to the single-chip level power, area, and timing (PAT) modeling tool proposed in this thesis, future work may extend it to a chiplet system level PAT modeling framework which supports 2.5D (or 3D) integration with the inter-chiplet (or inter-tier) connections with resistance-inductance-capacitance (RLC) path modeling as well as the interposers and the substrates.

On top of PAT analysis, future work will explore the integration cost as well to see whether the advantages of new technologies like emerging devices, 2.5D or 3D integration, can ultimately be translated into cost feasibility at the early design stage. Besides the manufacturing cost and the package cost discussed in the analytical cost model in this thesis, the testing cost and the cooling cost also play an important role in making the early-stage design decision. With the consideration of the 2.5D chiplet system or 3D integration, the thermal issue may become critical compared with the 2D monolithic system. Besides power, area, timing, and cost, future work may consider the thermal issue collaboratively at the early design stage.

Chapter 2

Background and Related Work

In this chapter, Sec. 2.1 first discusses the fundamental methodologies of analytical hardware modeling on power, area, timing, and cost. Sec. 2.2 then conducts the literature review for the architectural level innovations on the systolic array related computing components for machine learning (ML) accelerators to highlight the uniqueness of the reconfigurable architecture proposed in this thesis.

2.1 Analytical Hardware Modeling

Due to the high cost of prototyping, simulation plays an important role in the advanced computer architecture research. The urge of simulation tools has stimulated the research and development in hardware modeling in the last few decades. According to the functionality, the methodology of hardware modeling and the corresponding simulation tools can be divided into two major classes, including:

The first class is the **cycle-accurate, or event-accurate modeling**, including SimpleScalar [34], Gem5 [35], GPGPU-Sim [36, 37], MGPUSim [38]. These tools can collect the performance counters or activity factors of different hardware components. The second class is the **analytical modeling**, including CACTI [39], McPAT [40], and NVSim [41]. These tools consider more from a hardware perspective and conduct the circuit-level modeling for the estimation on power, silicon area, chip timing, and cost. These tools also provide the component-level power and timing, which can be used collectively with the performance counters or activity factors from the former cycle/event-accurate modeling tools to analyze the runtime performance and efficiency.

This thesis mostly focuses on the second class, i.e., analytical hardware modeling. We aim to build the hardware analytical tool from the perspective of (1) power, area, and timing; and (2) cost. Leveraging the built-up analytical hardware modeling tools, researchers are able to seek for the architectural innovations which enable high performance and efficiency of the emerging ML workloads. The rest of this section introduces the related work of the modeling of power, area, and time as well as cost respectively.

2.1.1 Power, Area, and Timing Modeling

CACTI [39] is the first analytical modeling framework for cache and memory arrays. McPAT [40] and NVSim [41] use the same analytical modeling methodology as CACTI. They build up the modeling framework for manycore general-purpose processors and the non-volatile memory arrays respectively.

Eyeriss [42], Eyeriss-v2 [43], and MAESTRO [44] provide dataflow analysis and modeling framework for ML accelerators. NNest [45] provides a generalized spatial architecture framework for exploring the design space of ASIC-based ML inference accelerators. Scale-Sim [46] provides a cycle-accurate performance simulator for systolic CNN accelerators through on-chip and off-chip memory access traces. Interstellar [47] uses Halide's algorithm and scheduling primitives [48] to express different ML accelerator architectures. Aladdin [49], Minerva [50], PolySA [51] and other work [52] provide different frameworks with (semi) HLS-level capabilities. Accelergy [53] and Timeloop [54] together provide an ecosystem to model ML accelerators. Besides providing modeling tools, previous work like NVDLA [55] open source the RTL codes of typical ML accelerator designs; and this kind of work boosts the modeling ecosystem from another perspective. With the simultaneous and analytical modeling of the power, area, and timing of the widely-adopted ML accelerator micro-architectures, the proposed NeuroMeter in this thesis advances the state-of-the-art and provides foundational support for the hardware modeling ecosystem.

2.1.2 Cost Modeling

Besides the analytical model of power, area, and timing, which estimates the performance and efficiency, the analytical model of cost is another significant tool to make reasonable design choices at the early design stage. This is because all the potential advantages of the proposed architectural innovations and the introduction of the advanced technologies ultimately have to be translated into either cost savings or cost feasibility when evaluating a design strategy. In this thesis, the cost model is mainly built for exploring the chiplet-based SiP.

Previously, cost model has been used in evaluating the TSV-based 3D architecture [56] or the silicon interposer based 2.5D integrated system [57, 58]. These works cannot be adopted in the chiplet-based SiP design because the heterogeneity, which is the key to the advantage of the chiplet-based SiP, never appears in previous technologies that only support homogeneous integration. To our best, this thesis makes the first attempt to build a cost model for chiplet-based SiP design space exploration.

2.2 Systolic Array

Systolic array [59, 60] was originally proposed as an architectural concept of a high-performance, special-purpose very-large-scale integration (VLSI) compute system for matrix operations by H.T.Kung in 1978. A systolic array is made up of a set of systolic cells, which are interconnected with each other according to the predefined topology, and each of which is capable of performing some simple operation. The architecture of systolic array has the potential to achieve high performance on matrix operations due to the regular communication pattern and simple control structure in the systolic array itself. In the recent years of "Cambrian exploration" [61] of ML accelerators, systolic array has regained a lot of attentions from both academia and industry since Google introduced the systolic array in its first generation of Tensor Processing Unit (TPU) [13]. Several questions have been raised when adapting the systolic array into the ML accelerators; and motivated a lot of architectural explorations on the systolic array based accelerators from the perspective of hardware-and-software co-design. Among all these questions, two specific questions are addressed in this thesis.

- What is the proper shape of systolic array?
- How to keep the performance and efficiency of the systolic array when the ML models continue to evolve?

The first question is summarized as a "brawny vs wimpy" question in this thesis. Sec. 2.2.1 conducts the literature review on the related "brawny vs wimpy" studies. The second question is addressed with the proposal of the systolic array with reconfigurable interconnections (RSA) in this thesis. Sec. 2.2.2 surveys the related work on the architectural innovations of ML accelerators with the emphasis on reconfigurable structures or components.

2.2.1 Brawny vs Wimpy Study

The brawny vs wimpy study [62] has been conducted extensively, with aspects including latency [63], throughput [64], energy efficiency [65], interconnect [66], heterogeneity [67, 68], and workload characteristics [69] in the CPU design space. With the growing ML workloads and the increasing deployments of ML accelerators in the datacenter, a similar "brawny vs wimpy" question has been raised in domain specific hardware. Kung et. al. [70] have studied the latency of accelerators with different systolic array sizes.

2.2.2 Reconfigurable Architecture for ML Accelerators

Several works introduce the reconfigurability into their architectural design. Planaria [71] proposes a fissionable architecture. Scale-out Systolic Array [72] proposes a reconfigurable systolic array with butterfly-based interconnections. The reconfigurability does not only enable the satisfying performance on various workloads, but also enhances the flexibility of the resource assignment in the multi-tenant scenarios.

Chapter 3

Power, Area, and Timing Modeling for ML Accelerators

3.1 Introduction and Contribution Overview

This chapter aims to establish NeuroMeter, an integrated power, area, and timing modeling framework for ML accelerators. NeuroMeter models the detailed architecture of ML accelerators and generates a fast and accurate estimation on power, area, and chip timing. Meanwhile, it also enables the runtime analysis of system-level performance and efficiency when the runtime activity factors are provided. Especially, when combined with an external performance simulator via its flexible and extensible interface, NeuroMeter enables a comprehensive study of architecture, system performance (TOPS), power efficiency (TOPS/Watt), and cost efficiency (TOPS/TCO[1]). With its new capabilities, NeuroMeter empowers the architects with a fast yet accurate modeling framework for exploring emerging manycore ML accelerators in a large architectural design space. Neu-

---

[1] TCO is the abbreviation for "Total Cost of Ownership". The experiment in the latter of this thesis uses TOPS/mm^4/Watt as a proxy metric for TOPS/TCO.

roMeter's micro-architecture model includes fundamental components of ML accelerators, including systolic array based tensor units (TU), reduction trees (RT), and 1D vector units (VU). To ensure accuracy, a rigorous validation on NeuroMeter results is conducted on both the component level and the whole-chip level. Our validation shows that NeuroMeter achieves high modeling accuracy, with overall power and area estimation errors below 10% and 17% respectively when validated against TPU-v1 [13], TPU-v2 [14, 15], and Eyeriss [42].

The main contributions of NeuroMeter include:

- Firstly, unlike prior ML accelerator modeling frameworks, which either model the power, area, or timing in isolation, or require other electronic design automation (EDA) tools to work collaboratively, NeuroMeter is the first framework to simultaneously model the power, area, and timing analytically at the architecture level.

- Secondly, NeuroMeter supports detailed modeling of critical architectural components of ML accelerators, including 2D systolic array based tensor units, reduction trees, 1D vector units, vector register files, and beyond.

- Thirdly, compared to previous modeling frameworks such as McPAT [40], NeuroMeter increases the architectural abstraction level. For example, it only requires users to configure high level architecture; meanwhile, it automatically scales and configures dependent hardware resources. As another example, it only requires users to configure high-level design targets, such as TOPS; meanwhile, it automatically searches for the optimal clock rate.

- Fourthly, NeuroMeter significantly advances the state-of-the-art, enhancing the architectural modeling ecosystem of ML accelerators for the community. Recent work such as Accelergy [53] and Timeloop [54] provide an ecosystem for architecture level

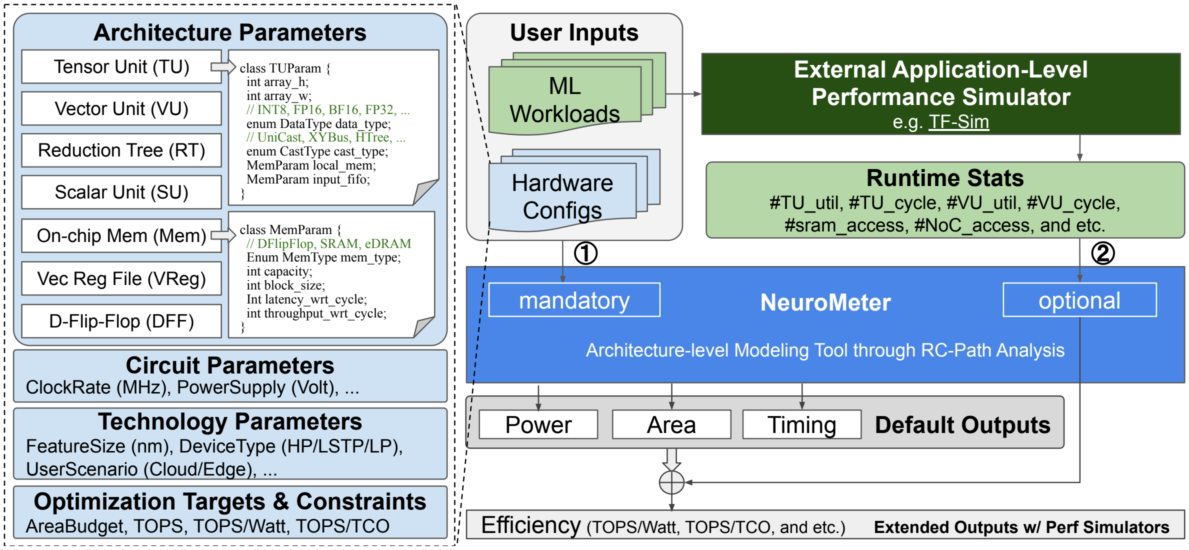
ML accelerator modeling. Timeloop [54] is an automatic design exploration tool, requiring fast energy consumption evaluations. Such fast energy consumption evaluations are supported by a high-level modeling tool, Accelergy [53], which relies on CACTI and lookup-table based energy models. However, the community still lacks an accurate analytical architecture modeling for the whole accelerator architecture to analytically model all accelerator components in the way CACTI does for memory arrays. NeuroMeter bridges this gap by providing a consistent analytical modeling methodology for the entire accelerator chip, building a strong foundation for Accelergy, Timeloop, among others [45, 46, 73], to form a robust and coherent ecosystem. Meanwhile, with its modular structure, NeuroMeter can also be used as a standalone framework, if the users choose to.

The rest of this chapter is organized as follows: Sec. 3.2 gives an overview of NeuroMeter; Sec. 3.3 describes the modeling methodology; Sec. 3.4 validates NeuroMeter against three ML accelerators, including TPU-v1 [13], TPU-v2 [14, 15], and Eyeriss [42].; Sec. 3.5 summarizes this chapter.

3.2 Framework Overview

NeuroMeter is an integrated power, area, and timing modeling framework for ML accelerators. Fig. 3.1 gives an overview of NeuroMeter and highlights its input/output interface. There are two types of inputs to NeuroMeter:

- Mandatory Input: the accelerator hardware configuration, which NeuroMeter uses to construct and optimize the target accelerator.
- Optional Input: the runtime statistics, which NeuroMeter uses to conduct runtime analysis. Usually, this type of input comes from an external application-level per-



```
class TUParam {
  int array_h;
  int array_w;
  // INT8, FP16, BF16, FP32, ...
  enum DataType data_type;
  // UniCast, XYBus, HTree, ...
  enum CastType cast_type;
  MemParam local_mem;
  MemParam input_fifo;
}
```

```
class MemParam {
  // DFlipFlop, SRAM, eDRAM
  Enum MemType mem_type;
  int capacity;
  int block_size;
  Int latency_wrt_cycle;
  int throughput_wrt_cycle;
}
```

Figure 3.1: Overview of NeuroMeter Framework (© 2021 IEEE)

formance simulator, for example TF-Sim [74]. Leveraging these two types of inputs collaboratively, NeuroMeter enables system performance and efficiency analysis.

NeuroMeter by default outputs the power, area, and timing of target ML accelerators based on their specified hardware configuration.

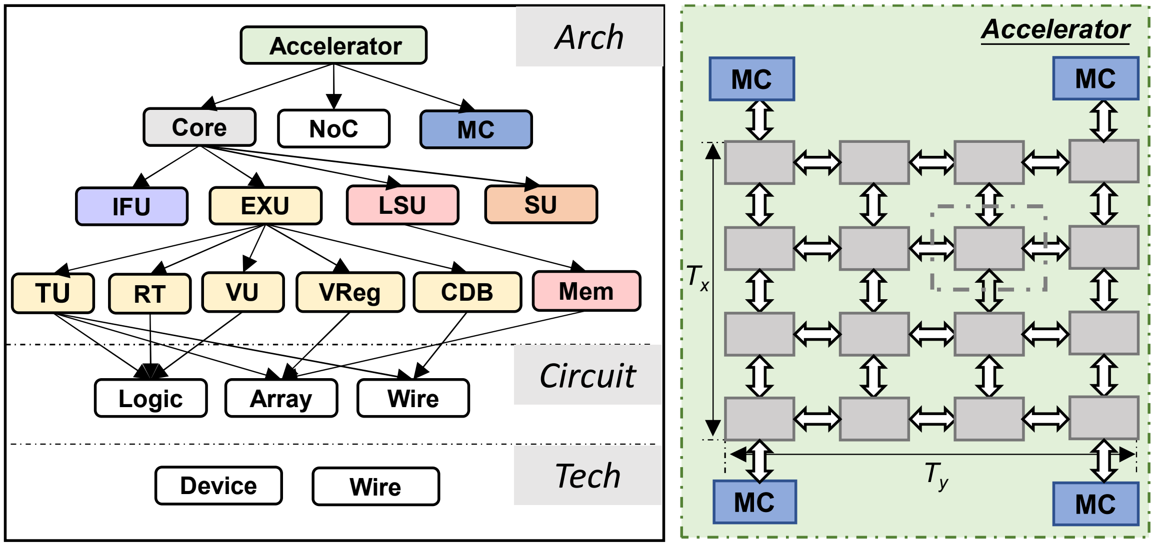
NeuroMeter allows users to specify the parameters at the architecture, circuit, and technology level, as well as the optimization targets and constraints, as shown in Fig. 3.1. Besides the essential parameters, such as the core count, clock rate, power supply voltage, and technology node, it only requires the user to provide the high-level configurations of critical hardware components without worrying about the low-level configurations. For example, when the user configures the computing components of the accelerator, they only need to configure critical parameters, such as the tensor unit's array height/width, the data type of the multiplication-accumulation unit, and the type of inner array interconnection, the tool itself will automatically help the user figure out the dependent hardware components, including the vector register file, the scalar unit, and the interconnection overheads between different components. When the user configures the on-chip memory, they only need to configure the parameters of capacity, block size, target latency, and the target throughput. The tool will automatically set the low-level parameters (such as the number of banks, the number of the read/write ports) via its internal optimizer.

By default, NeuroMeter requires the input of system-level performance (i.e., peak TOPS) as the optimization target (or a minimal value of it as a design constraint). TOPS/Watt and TOPS/TCO are also allowed to feed in as alternative optimization targets or design constraints. Given the system-level performance constraints, NeuroMeter conducts the component-level timing analysis using an Elmore delay model [30]. Once a design is found to meet the optimization targets and design constraints, NeuroMeter finalizes an internal chip representation to get the chip-level thermal design power (TDP), silicon area, and their component-level breakdowns. NeuroMeter also outputs the timing information of the electrical signal propagation delay (e.g., Elmore Delay) and the cycle time per component to help the user find out the hardware critical path.

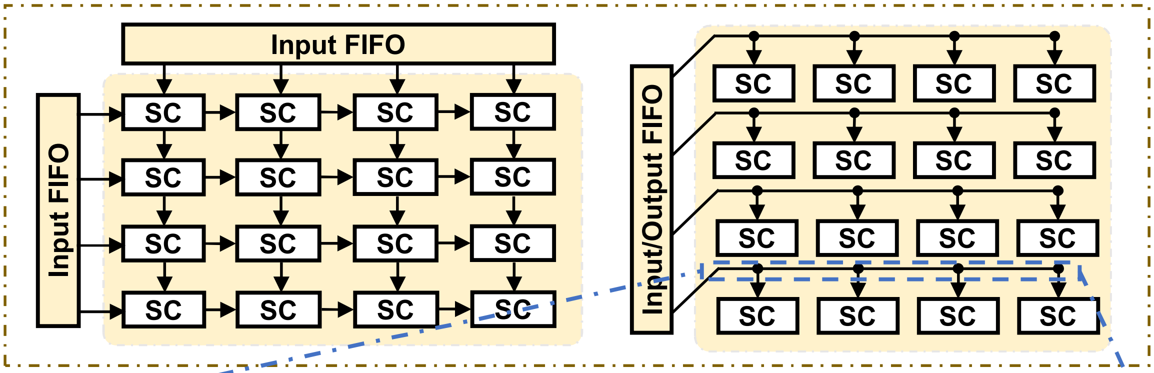
When given the runtime statistics of the target ML model running on the accelerator, NeuroMeter also combines the inputs of runtime statistics on hardware utilization and activity factors for micro-architecture components with the chip-level TDP and silicon area to generate the end-to-end runtime estimation of performance[2], power, and efficiency of the target accelerators running specified ML models. NeuroMeter decouples the performance simulation from the architecture modeling, so that it can be flexibly paired with any external performance simulation framework for comprehensive ML accelerator research.

---

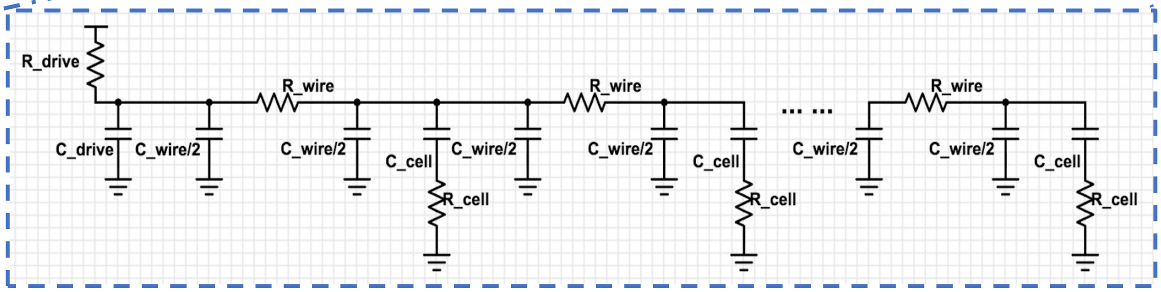
[2] The word "Performance" here represents the program execution time (i.e., end-to-end latency) and/or throughput.



(a) Modeling Methodology & Architectural Breakdown (b) Multi-core Accelerator with 2D Mesh NoC



(c) Different Types of TU. Left: Uni-Cast, Right: Multi-Cast



(d) Analytical Model of cell-to-cell Interconnection in Multi-Cast TU

Figure 3.2: NeuroMeter's Top-Down Modeling Methodology (© 2021 IEEE)

3.3 Modeling Methodology

NeuroMeter follows a top-down modeling methodology. As shown in Fig. 3.2(a), high-level blocks are divided into lower-level sub-blocks and finally mapped onto the circuit-level models of compute logic units, memory arrays, and hierarchical wires, with the backend technology devices and wiring parameters. This section introduces the modeling methodology of NeuroMeter at the architecture level (Sec. 3.3.1) as well as the circuit and technology level (Sec. 3.3.2).

3.3.1 Architecture-Level Modeling

At the chip architecture level, NeuroMeter models a multi-core ML accelerator. Fig. 3.2(b) gives an example of a multi-core accelerator with a 2D-mesh Network-on-Chip (NoC), while other types of NoCs are also supported, including bus, ring, and H-tree. Other peripheral blocks, including Memory Controllers (MCs) and DMA controllers, are also modeled.

At the core architectural level, NeuroMeter breaks down a single core into an Instruction Fetch Unit (IFU), a Load-and-Store Unit (LSU), an EXecution Unit (EXU), and a Scalar Unit (SU) for control. An IFU in ML accelerators is usually lightweight, unlike the complicated front-end circuit in high performance general-purpose processors. An LSU in ML accelerator includes on-chip memory (Mem) and data/control paths to off-chip memory. The most critical component is EXU, which is further broken down into multiple functional units, i.e., 2D systolic array based Tensor Unit (TU), Reduction Tree (RT), 1D Vector Unit (VU), Vector Register file (VReg), and Central Data Bus (CDB). Each unit is discussed below.

Tensor Unit (TU) is a generic 2D systolic array made up of three parts, (1) an array of systolic cells (SCs), each one of which consists of a multiplication-accumulation (MAC) unit or a fused multiply-add (FMA) unit as well as a D-Flip-Flop (DFF) or SRAM based local buffer; (2) wires connecting nearby systolic cells; (3) DFF/SRAM based I/O FIFOs. Our tool supports TUs with various types of interconnections between systolic cells and I/O FIFOs. Fig. 3.2(c) exemplifies two types of inner-TU interconnections, including unicast as in Google's TPU-v1, and multicast (or X/Y bus) as in Eyeriss. For systolic arrays (or unicast TUs) we support modeling of both weight-stationary and output-stationary dataflow with a flexible systolic cell configuration. At the circuit level, MAC units inside the systolic cells are pre-simulated through EDA tools, while the DFF/SRAM based local buffers, I/O FIFO, and the cell-to-cell interconnections are modeled analytically. Fig. 3.2(d) illustrates the multicast inner-TU interconnection as an example, i.e., the interconnection is decomposed into several segments of wires that are abstracted into the π-RC model; the output resistance of the I/O FIFO and the input resistance of the systolic cells are extracted as the drive and the load of the RC path respectively.

Reduction Tree (RT) is made up of three parts, (1) a N-input 1D MAC array (Which is similar as in VU); cascaded by (2) a $\log(N)$-layered adder tree; (3) the (optional) DFFs between the two nearby layers to satisfy the timing constraints if needed. In the default configuration, we assume that each layer uses an array of 2-by-1 adders in the adder tree. The users can customize the type of the adder and the level of the adder tree according to their design requirements. The RT is broadly used in sparsity-aware accelerator designs [73, 75, 76] since it has more flexible workload mapping compared with the 2D array based TU.

Vector Unit (VU) processes 1D vector operations, such as pooling, activation, and different variants of normalization. It also merges the partial sums when one TU is not large enough to hold the whole Conv2D or MatMul operator without tiling. Moreover, in some ML accelerators [77] without 2D TUs, VUs are the main processing elements. Such accelerators can be well supported by NeuroMeter. The vector register file (VReg) is the center for data exchange inside VU as well as between VU and TU. In the default architectural configuration, the number of the VU lanes and the vector width of VReg match the TU array length; and each TU/VU has private read/write VReg ports. For the core with single VU and single TU, VReg is configured as 4 read ports and 2 write ports to support dual issue width. Meanwhile, multiple TUs can be configured to share one group of read/write VReg ports. In that case, the external performance tool has to exclude the mapping based on independent data to different TUs, or include the extra cost when data broadcast is not applicable.

Scalar Unit (SU) is mainly used for auxiliary operations in the control flow, e.g., address calculation. Leveraging McPAT's configuration, SU is by default configured as a simplified "ARM Cortex-A9 core" which only has the instruction fetch unit (w/o branch prediction logic), integer register file, ALU, and LSU, with the rest of the core removed. It can also be easily reconfigured to other architectures.

On-chip Memory (Mem) models the storage units, which hold the weights and feature maps on the chip. It can be configured as a software-managed scratchpad memory, which is commonly used in many ML ASICs, or a cache hierarchy. The cell type of Mem can be selected from DFF, SRAM, and eDRAM. According to the throughput requirements, Mem is always configured as multi-banked. Based on the architectural configurations, Mem can be modeled as a unified structure where weights and activations are stored together as in TPU-v1, or as a dedicated structure where each bank has its own functionality as in Eyeriss.

Central Data Bus (CDB) models the interconnection between different components within the core, especially the wires connecting VReg and other functional components, including TU, VU, and Mem. Wires are assumed to route around the functional components, and their length is estimated by the square root of the functional component area. When the length is large, wires are pipelined to meet the throughput requirement.

3.3.2 Circuit and Technology-Level Modeling

NeuroMeter models the power, area, and timing of the hardware components analytically and simultaneously. Similar to McPAT [40], NeuroMeter maps the architectural components to basic logic gates and regular circuit blocks, including computing arrays (e.g., TU, VU), memory arrays (e.g., DFF, SRAM, and eDRAM), interconnects (e.g., router, link, and bus), and regular logic (e.g., decoder and dependency-checking unit). These circuit blocks are then mapped to fundamental analytical resistance-capacitance (RC) ladder/trees and layout models to compute timing, area, and energy at different technology nodes.

However, an analytical approach does not work well for complex structures that have custom layouts, such as the MAC logic in the TU, VU, and SU. For these components, NeuroMeter currently takes an empirical modeling approach, which utilizes curve fitting to build a parameterizable numerical model for the area and power of complex components. The empirical model is based on the synthesis results from Design Compiler using the RTL models from Berkeley Hardware Floating Point Unit Library [78] with the technology backend of FreePDK [79, 80] libraries.

3.4 Validation

The primary focus of NeuroMeter is fast yet accurate power and area modeling at the architectural level when given the target system performance (i.e., peak TOPS). To ensure the accuracy of NeuroMeter, rigorous validations are conducted at both the *component* level and the whole *chip* level. At the *component* level, NeuroMeter's power, area, and timing results are validated against the synthesis results from Chisel [81] with the FreePDK45 library [80]. The validation against EDA tools shows that NeuroMeter's prediction is within a 15% area error margin, which provides strong confidence for the component level modeling accuracy. Considering that power depends highly on the block activity factors, it is rigorously validated at the chip level under the assumption of average power traces. At the *chip* level, the validation is conducted against TPU-v1 [13], TPU-v2 [14, 15], and Eyeriss [42]. NeuroMeter demonstrates satisfying modeling accuracy, with about 10% and 17% error margins on overall power and area respectively against the three real ML accelerators. It is important to note that chip-to-chip power variation in modern microprocessors [82] is comparable to the magnitude of the power validation errors of NeuroMeter.

Fig. 3.3 shows TPU-v1's validation results of power and area, at a 28nm technology node with a 700MHz target clock rate. At the chip level, the modeling results of overall power (i.e., TDP) and area have <5% and <10% error respectively, compared with the published TDP (75W) and area (<331mm^2). At the component level, TPU-v1 contains four major parts: (1) a MAC-based Systolic Array for matrix multiplication; (2) a Unified Buffer & Weight FIFO for activation and weights; (3) an Accumulator Buffer for partial sums; and (4) an Activation Pipeline for other operations. NeuroMeter models the systolic array by the TU with a unicast interconnection; models the unified buffer, accumulator buffer, and the weight FIFO by the Mem; and models the activation pipeline with the VU. As shown in Fig. 3.3(a), NeuroMeter produces accurate area modeling results (within 2% relative error) for the systolic array and the accumulator buffer; but over-estimates the relative area of the unified buffer[3] by ∼10%, and the error ratio will enlarge to about 30% if considering the unified buffer alone. This even larger error rate may be due to the lack of knowledge of optimized placement-and-routing for

---

[3] Relative area is used in the validation. It is difficult to make an apple-to-apple comparison on the absolute area because the published overall area of TPU-v1 only gives an upper bound of 331mm^2. Taking the ground truth of the overall chip area as 331mm^2, NeuroMeter has at most 20% absolute estimation error on the unified buffer (NeuroMeter: 108mm^2 vs Ground truth: 87mm^2. The ground truth of unified buffer is calculated by the published overall area with its manually measured proportion based on TPU-v1's published block diagram.)

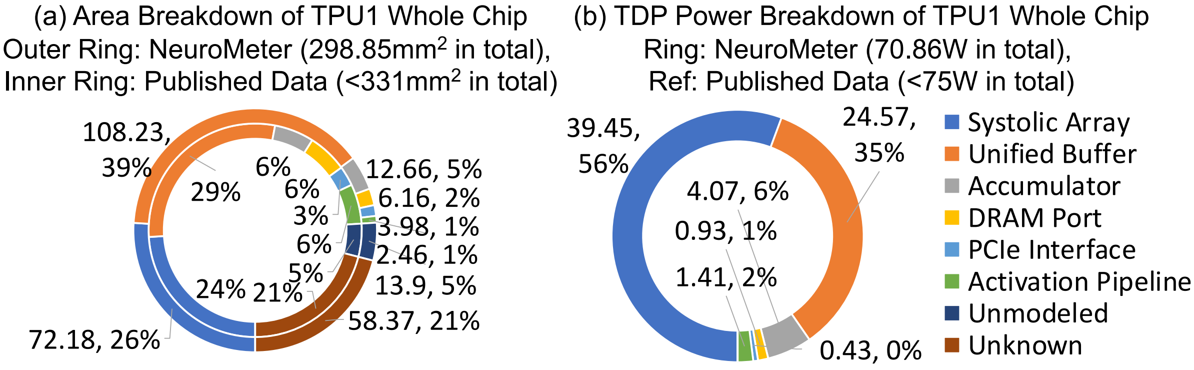
the interconnect between systolic array and unified buffer in TPU-v1. We also model the peripheral interfaces including DRAM port (6.0% vs 2.8%) and PCIe interface (3.0% vs 1.8%). We currently do not model host interface, controller, and misc I/O, with 5% in total. The unknown components in TPU-v1 occupy ∼21% of the chip area, and we use the same percentage as white space in our area overall estimation. Although no published data exists to compare against, the NeuroMeter power breakdown is shown in Fig. 3.3(b), where the systolic array is the biggest power consumer with 56% of the total chip power.

Fig. 3.4 shows the area validation of TPU-v2 at an assumed 16nm technology node[4] with a 700MHz target clock. At the chip level, the modeling results of area (513mm^2) have at most 17% error compared with the published area ($<$ 611mm^2); and the modeling results of TDP (255W) have ∼9.1% error compared with the published TDP (280W). Similar to TPU-v1, NeuroMeter models the MXU, Vector Unit, and Vmem by systolic array based TU, VU, and Mem respectively. We would like to highlight that our simulation results show that TPU-v2 requires two read ports and one write ports per bank, and this is automatically searched by NeuroMeter with the given throughput requirement. Furthermore, we also modeled the Inter-Chip Interconnection (ICI) link and switch (12% vs 5%) by the components of Network Interface Unit (NIU) and NoC given the bisectional bandwidth at 496Gb/s per direction. Other peripheral components, including HBM ports (9% vs 5%) and PCIe Controllers (2% vs 2%) are also modeled. We currently do not model transpose unit, RPU, and misc datapath, with 11% in total. The unknown components (which probably includes the inter-component interconnection) in TPU-v2 occupy ∼21% of the chip area, and we use the same percentage as white space in our overall area estimation.

Fig. 3.5 shows Eyeriss's validation results of power and area, at a 65nm technology

---

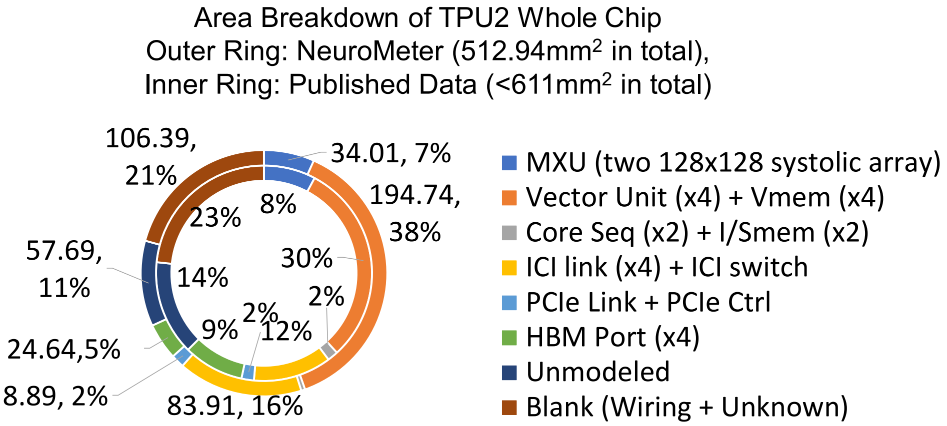
[4] According to the published information [14], TPU-v2's technology node is greater than 12nm.



(a) Area Breakdown of TPU1 Whole Chip
Outer Ring: NeuroMeter (298.85mm^2 in total),
Inner Ring: Published Data (<331mm^2 in total)

(b) TDP Power Breakdown of TPU1 Whole Chip
Ring: NeuroMeter (70.86W in total),
Ref: Published Data (<75W in total)

Figure 3.3: Area and Power Break Down of TPU-v1 Published Data [13] vs NeuroMeter Simulation Results. TPU-v1 @ 700MHz with 0.86V power supply is fabricated at 28nm. Architecture parameters used in the model are: Systolic Array Size: 256x256; Accumulator: 256 int32 adders; Unified Buffer: 24MB, dual banks, one read port and one write port; Accumulator Buffer: 4MB, 4k blocks per bank, dual ports; PCIe Gen3x16: 14GB/s. Notice: The ring only shows the relative percentage of different hardware components; the ring diameter has nothing to do with the absolute power/area.(© 2021 IEEE)

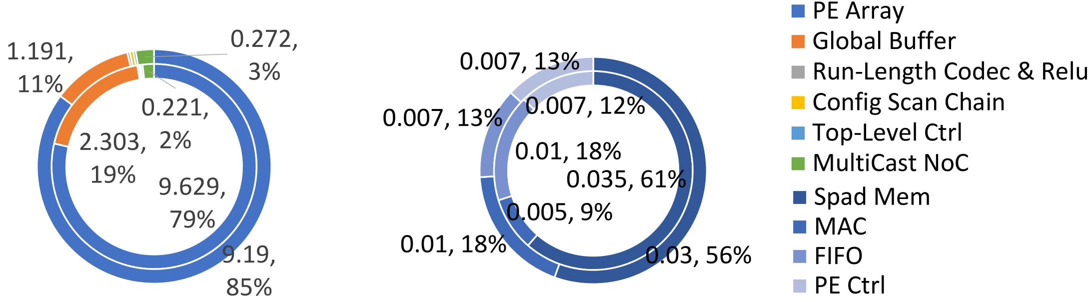


Area Breakdown of TPU2 Whole Chip
Outer Ring: NeuroMeter (512.94mm^2 in total),
Inner Ring: Published Data (<611mm^2 in total)

Figure 3.4: Area Break Down of TPU-v2 Published Data [14] vs NeuroMeter Simulation Results. TPU-v2 @ 700MHz with 0.75V power supply assumes to be fabricated at 16nm. Architecture parameters used in the model are: MXU: two 128x128 systolic arrays with BF16 multiplier and FP32 adder; VMem: 8MB, quad-banks, with two read ports and one write port. Notice: The ring only shows the relative percentage of different hardware components; the ring diameter has nothing to do with the absolute power/area. (© 2021 IEEE)

(a) Area Breakdown of Eyeriss Whole Chip
Outer Ring: NeuroMeter (10.87mm² in total),
Inner Ring: Published Data (12.25mm² in total)

(b) Area Breakdown of Eyeriss Single PE
Outer Ring: NeuroMeter (0.054mm² in total),
Inner Ring: Published Data (0.057mm² in total)



(c) Runtime Power Breakdown of AlexNet-Conv1
Outer Ring: NeuroMeter (356mW in total),
Inner Ring: Published Data (320mW in total)

(d) Runtime Power Breakdown of AlexNet-Conv5
Outer Ring: NeuroMeter (197mW in total),
Inner Ring: Published Data (227mW in total)

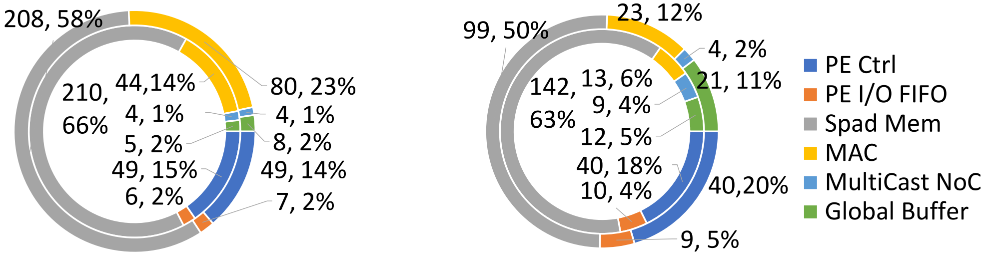


Figure 3.5: Area and Power Break Down of Eyeriss-v1 Published Data [42] vs NeuroMeter Simulation Results.Eyeriss @ 200MHz with 1.0V power supply is fabricated at 65nm. Architecture parameters used in the model are – PE Array Size: 14x12; Local Buffer per PE: 448byte SRAM, 72byte registers, PE I/O FIFO transferring between 32bit to 8bit; Global Buffer: 108kB, 27 banks in total, dual ports. Notice: The ring only shows the relative percentage of different hardware components; the ring diameter has nothing to do with the absolute power/area. (© 2021 IEEE)

node with a 200MHz target clock rate. As shown in Fig. 3.5(a) and (b), the area modeling of the single PE and the overall results have <5% and <15% error respectively. At the single PE level, Eyeriss's PE is modeled by NeuroMeter's systolic cells in the TU. At the chip level, Eyeriss's three major components, i.e., PE Array, Global Buffer, and MultiCast NoC, are modeled by the TU, Mem, and inner-TU connection respectively as introduced in Sec. 3.3.1. Other chip-level components, including Run-Length Code & ReLU, Config Scan Chain, and Top-Level Ctrl, are also modeled. As shown in Fig. 3.5(b), the relative area breakdown of PE array is overestimated by ∼7%, which may result from the limited knowledge of the exact MAC logic model in use. The relative area breakdown of the global buffer is under-estimated by ∼7%, which may be due to the insufficient knowledge of the outside-bank overhead. Compared with TPU-v1, the area breakdown of the PE array in Eyeriss is much larger than that of the systolic array in TPU-v1. Both of them are modeled by the TU in NeuroMeter, but Eyeriss introduces a heavier local buffer design, i.e., every PE has the local scratchpad memory and register files to support the row-stationary dataflow.

We also validate the runtime power[5] against the report from Eyeriss when running publicly available ML models. As shown in Fig. 3.5(c) and (d), the overall power has 11% over-estimation and 13% under-estimation respectively when running AlexNet-Conv1 and AlexNet-Conv5 layers. The differences of the runtime power in these two layers may result from the insufficient knowledge of the zero-skipping and clock-gating operation in Eyeriss. To be consistent with the published data, the power consumption breaks down into the following six components, including (1) MAC logic, (2) local buffer (Spad Mem), (3) PE I/O FIFO, (4) PE controller, (5) multicast NoC, and (6) global buffer; and the first five components are the internal structures of the PE array. The unmodeled components include chip I/O pads and top-level control and are not shown in Fig. 3.5. Since NeuroMeter does not model the clock network as a separate component, we amortize the power breakdown of the clock network into other components. Similar to the TDP in TPU-v1, the PE array in Eyeriss takes the major proportion of the runtime power consumption. Unlike TPU-v1's TDP, the global buffer in Eyeriss takes a much smaller proportion. This shows the difference between TDP and the runtime power consumption.

---

[5] In order to decouple the error of hardware modeling from the error of performance analysis, we calculate the activity factor based on the published data of the processing time, the number of active PEs, the percentage of zero input feature maps, and the number of global buffer accesses.

3.5 Summary

NeuroMeter is an architectural analytical framework for simultaneously modeling power, area, and chip timing for emerging ML accelerators. It models all major architectural components of emerging ML accelerators, including TU, VU, on-chip Mem, NoC, MemCtrl, host interface, and beyond. Moreover, its analytical model of TU and VU captures the key difference between emerging ML accelerators and the mainstream CPUs. Its analytical modeling methodology generates fast and accurate modeling results without relying on EDA tools. Validations show a reasonable agreement between NeuroMeter and published data for both datacenter-oriented (TPU-v1/v2) and mobile/edge-oriented (Eyeriss) state-of-the-art ML accelerators. NeuroMeter empowers architects with a fast yet accurate exploration of the large and diverse design space of modern manycore ML accelerators. When combined with performance simulations via its flexible and extensible interface, NeuroMeter enables broader architecture study with comprehensive metrics such as TOPS/Watt, TOPS/TCO.

Chapter 4

Case Studies on Design Space Exploration for ML Accelerators

4.1 Introduction and Contribution Overview

This chapter is dedicated to use two case studies to showcase the capacity of NeuroMeter, which is established in Chp. 3, in modeling a wide range of diverse ML accelerator architectures; how NeuroMeter would be involved in the design space exploration of ML accelerators; and how NeuroMeter would further inspire the architectural innovation, especially in the software-and-hardware co-design for emerging ML workloads.

With the recent "Cambrian explosion" [61] of ML accelerators, two clear design paths have emerged. One path is a "Brawny" design that uses a few large cores such as Google's TPU (single core with a 256x256 systolic array in TPU-v1 [13]; dual cores, each with one or multiple 128x128 systolic arrays in TPU-v2,v3,v4i [14, 15, 16]), while the other path is a "Wimpy" design that uses a sea of small cores such as nVidia's Volta (640 TensorCores with 64 FMAs per clock per TensorCore [22]) and Ampere (512 TensorCores with 1024 FMAs per clock per TensorCore and hardware supports for structural sparsity

[23]). While both design paths have proven to be successful and inspired many subsequent designs, there is no in-depth quantitative understanding about the essence and rationale of either design path. To bridge this gap, this chapter conducts a comprehensive and consistent study on the design space and tradeoffs of "Brawny and Wimpy" for the datacenter inference accelerators. The study reveals that for datacenter chips with a 500mm^2 silicon area budget, a dual-core accelerator with four 64x64 systolic arrays per core has superior efficiency and performance on inference tasks among 28nm design points, despite relatively lower utilization. Moreover, the study also reveals important tradeoffs among different design targets. For example, for datacenter accelerators with low batch inference, a small ($\sim$ 16%) sacrifice of performance (achieved TOPS) can lead to more than 2x improvement of efficiency (achieved TOPS/TCO).

Based on these choices of accelerator architectures, another case study is conducted on energy efficiency (TOPS/Watt) implications of sparsity on both systolic-array and reduction-tree based ML accelerators to showcase NeuroMeter's capability to model diverse ML accelerator architectures. The results show that despite their relatively lower energy efficiency, it is easier for wimpier accelerator architectures to benefit from sparsity processing.

In the rest of this chapter, Sec. 4.2 introduces how the operator-level mapping is conducted by the performance simulator; Sec. 4.3 conducts a case study on brawny and wimpy manycore ML accelerators to showcase NeuroMeter's functionality to model the datacenter inference scenarios; Sec. 4.4 conducts a sparse-oriented case study to showcase NeuroMeter's functionality to model diverse architectures and support various workloads; and Sec. 4.5 summarizes the insights discovered from the two case studies.

4.2 Performance Simulation and Operator Mapping

As explained in Chp. 3, besides the hardware modeling tool alone, an external performance simulator is required when the user evaluates the workload-aware runtime performance and efficiency. The quality of application-level performance simulation and operator mapping will affect the final results of runtime performance and efficiency. The purpose of this section is NOT to propose an optimal operator mapping methodology that fulfills all the architectures. Instead, this section aims to clarify a reasonable mapping methodology (yet probably a non-optimal one) and apply it to all the architectures in the proposed design space. We try our best to make a fair evaluation of the different architectures by decoupling the impact on performance when different mapping methodologies are applied on different architectures.

The case studies in this chapter leverage TF-Sim [74] to simulate the application-level performance. TF-Sim [74] intakes both the compute graph of the given workload and the architectural parameters of the given accelerator. It first partitions the whole compute graph into multiple operators according to the internal integrated or user-customized operator partition/fusion rules; then calculates the operator-wise runtime statistics (e.g., the compute cycle, utilization, the number of memory access of each involved hardware components) according to the internal integrated or user-customized methodologies of operator mapping. The achieved runtime statistics would be further fed into NeuroMeter to evaluate the runtime performance and efficiency.

Batched matrix-matrix multiplication (Batched MatMul) is an important basic operator for ML workloads. In the rest of this section, Sec. 4.2.1 will first analyse how to map the batched MatMul onto the Tensor Units which are made up with homogeneous systolic arrays (HSAs); Sec. 4.2.2 will further discuss how other operators, including batched matrix-vector multiplication (Batched MatVec), traditional 2D convolution

(Conv2d), and depthwise convolution (DConv2d), are first tiled into batched MatMul and then mapped onto HSAs. The operator mapping described in this section will give the reader a better understanding on how performance simulation works collaboratively with the proposed hardware modeling tool, NeuroMeter; and how the case studies are conducted in the later of this chapter.

4.2.1 The Mapping of Batched MatMul

When mapping the workload of batched **MatMul(M, N, K)**, i.e. $C_{M\times N} = A_{M\times K} \cdot B_{K\times N}$, with batch size **b** onto the X-by-X-sized single HSA based Tensor Unit, the tiling length X_t needs to be determined such that the original batched MatMul is tiled into $b \cdot \lceil \frac{M}{X_t} \rceil \cdot \lceil \frac{N}{X_t} \rceil \cdot \lceil \frac{K}{X_t} \rceil$ blocks of unit operators $\mathbf{MatMul(X_t,\ X_t,\ X_t)}$. Comparing HSA's array length X and the tiling length X_t, there are three cases:

(1) When $X_t == X$, the unit operator $\mathbf{MatMul(X_t,\ X_t,\ X_t)}$ can be directly map onto the HSA and it takes X cycles to process one unit operator.

(2) When $X_t > X$, the X-by-X HSA is still unable to hold it. In this case, the value of X_t need to be further tiled to make the unit operator able to be held up by the HSA with the given size.

(3) When $X_t < X$, besides the input matrix of the unit operator, the input buffer of the X-by-X-sized HSA has to be filled up with padding zeros. This leads to extra ineffective FMA operations with at least one of the input operators as zero. In this circumstance, although HSA processes **MatMul(X,X,X)** by $2X^3$ operations in X cycles, only the $2X_t^3$ operations from $\mathbf{MatMul(X_t,\ X_t,\ X_t)}$ conduct the effective calculations. To reduce the ineffective operations which have zero-valued input operators, we can compose multiple $\mathbf{MatMul(X_t,\ X_t,\ X_t)}$ into one **MatMul(X,X,X)**. When an integer $n_t = \lceil \frac{X}{X_t} \rceil$, we will fill the input matrices of the n_t operators of $\mathbf{MatMul(X_t,\ X_t,\ X_t)}$

into the n_t X_t-by-X_t submatrices of **MatMul(X,X,X)** on the diagonal of its input matrix. Then it takes X cycles to process n_t unit operations $\mathbf{MatMul(X_t, X_t, X_t)}$ into one **MatMul(X,X,X)** in parallel. The output of these n_t $\mathbf{MatMul(X_t, X_t, X_t)}$ unit operators are exactly lined up on the diagonal submatrices of the output matrix **MatMul(X,X,X)**. In this way, n_t unit operators are processed independently and parallelly.

To sum up, Eq. 4.1 shows T_{MatMul}, the time consumption when the operator of batched MatMul with the tiling length X_t is computed by the X-by-X-sized HSA-based Tensor Unit, i.e.,

$$T_{MatMul}(X_t) = (\lceil \frac{b \cdot \lceil \frac{M}{X_t} \rceil \cdot \lceil \frac{N}{X_t} \rceil \cdot \lceil \frac{K}{X_t} \rceil}{n_t} \rceil + 2) \cdot X \cdot T_{HSA} \tag{4.1}$$

where T_{HSA} is the cycle time of the HSA; and the symbol $\lceil \cdot \rceil$ is the integer roundup. The term $2 \cdot X \cdot T_{HSA}$ is the time overhead when streaming in the first block of input data and streaming out the last block of output data. For simplification, we assume both X and X_t are power-of-two numbers. In this way, we will sweep all the power-of-two numbers from 2 to X to find the optimal value of X_t to minimize T_{MatMul}.

4.2.2 The Mapping of Other Tensor Operators

The operator of batched matrix-vector multiplication (Batched MatVec) can be regarded as a special case of batched **MatMul(M, N, K)** where $N = 1$. We can search the optimal tiling length in the same way as that of batched MatVec.

The operator of traditional 2D convolution (Conv2d) has the input tensors of batch size B, each with the shape of (I_x, I_y, I_z); the output tensors of batch size B, each with the shape of (O_x, O_y, O_z); and Oz kernel tensors, each with the 3D shape of (K_x, K_y, K_z). It can be regarded as the batched $\mathbf{MatMul(B, O_z, I_z)}$ with the new batch size of

$K_x \cdot K_y \cdot I_x \cdot I_y$.

The operator of depthwise convolution (DConv2d) has the input tensors of batch size B, each with the shape of (I_x, I_y, I_z); the output tensors of batch size B, each with the shape of (O_x, O_y, O_z); and Oz kernel tensors, each with the 2D shape of (K_x, K_y). Different from Conv2d, DConv2d has $K_z = 1$. If we would like to map DConv2d similarly like traditional Conv2d, we will zero pad the 2D convolution kernel into a 3D counterpart by setting $K_z = I_z$. After zero padding, DConv2d can be mapped in a similar way like that of Conv2d, i.e., the batched $\mathbf{MatMul(B,\ O_z,\ I_z)}$ with the new batch size of $K_x{\cdot}K_y{\cdot}I_x{\cdot}I_y$. However, in this case, only the elements on the diagonal of the kernel matrix are non-zero while all the other elements are zero. When DConv2d is further mapped onto the systolic array, only the systolic cells on the diagonal are conducting the effective calculations, while the other systolic cells can be clock-gated or configured into the idle mode because at least one of the multiplicands is zero. Assuming the systolic array is sized of X-by-X, then only X systolic cells are active, which makes the systolic array in low utilization. The larger size of the systolic array, the lower the array utilization.

4.3 Brawny vs Wimpy Study on Homogeneous Many-core ML Accelerators

Of the many types of ML accelerators that have emerged, one type can be classified as having relatively “Brawny” core designs that use a few large systolic arrays such as Google’s TPU (a single core with a 256x256 systolic array [13] in TPU-v1 or one/dual cores, each with one or multiple 128x128 systolic arrays in TPU-v2,v3,v4i [14, 15, 16]). Another class are designs based on relative “Wimpy” cores that use a sea of small computing arrays or vector processing units such as nVidia’s Volta architecture (640 TensorCores with 64 FMAs per clock per TensorCore [22]) and Ampere architecture (512 TensorCores with 1024 FMAs per clock per TensorCore and hardware supports for structural sparsity [23]). Intuitively, the brawny design is believed to have an advantage of high performance, especially when the tensor size is large enough; while the wimpy design is believed to have an advantage of high utilization without sacrificing performance by using sophisticated compiler and runtime software. However, there is no in-depth and comprehensive study to quantify these hypotheses, partly because of the lack of tools.

To bridge this gap and to showcase the capability of NeuroMeter, detailed analyses are conducted to compare brawny and wimpy manycore ML accelerator designs. Interestingly, the brawny vs wimpy design tradeoffs have been a critical topic in CPU design and date back to decades ago as summarized in previous work [62]. This section aims to foster a comprehensive and systematic study of brawny and wimpy design tradeoffs on the ML accelerator frontier.

In the study described in this section, the brawny accelerator architecture uses fewer cores with large systolic array based TU(s) per core, while the wimpy accelerator architecture uses more cores with small systolic array based TU(s) per core. The rest of the on-chip resources are scaled proportionally as the systolic array size changes. While NeuroMeter models both training and inference accelerators, this case study focuses on the inference accelerators in the data center scenarios.

4.3.1 Experiment Methodology and Setup

The study focuses on the general architecture of manycore ML accelerators shown in Fig. 4.1. All cores are connected by a 2D mesh NoC. Each core has a systolic array based tensor unit (TU) for matrix operations; meanwhile, each core also has a vector unit (VU) for vector operations. Each core may also have a scalar unit (SU) for control

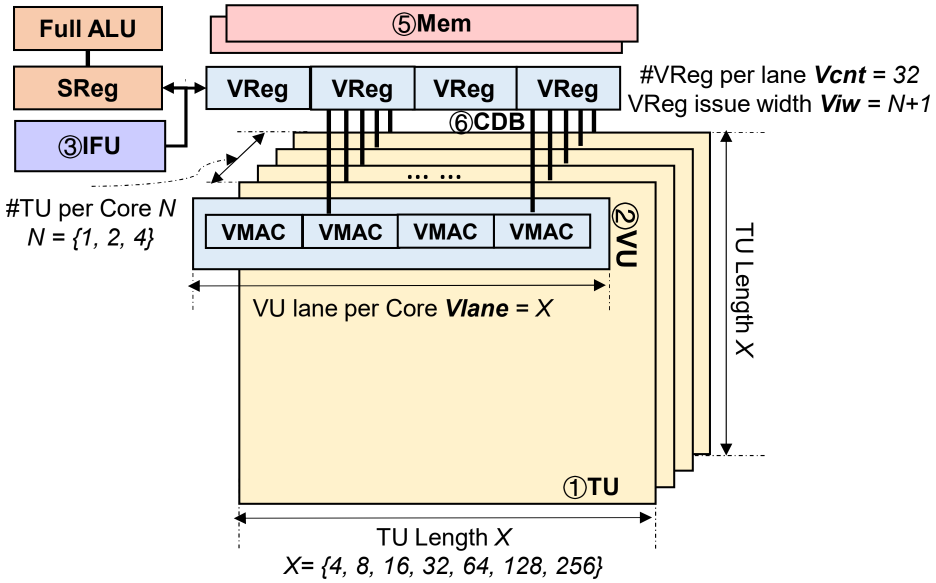


Figure 4.1: Single Core Architecture of the Datacenter Inference Chips (© 2021 IEEE)

path because of the high throughput of TUs in the core. Each core has a portion of the distributed on-chip memory (Mem). A vector register file (VReg) is the data exchange hub among TU, VU, and Mem. The central data bus (CDB) connects VReg and other components inside the core.

Architecture Design Space and Chip Modeling

Since brawny and wimpy is a continuous spectrum in the design space, we denote each architectural design point by a four-element tuple (X, N, T_x, T_y), where X is the TU length that defines how brawny or wimpy the architecture is; N is the number of TU in each core; T_x and T_y are the 2D mesh NoC topology to connect all the cores. Given each tuple of such a design point, NeuroMeter automatically scales and sets the dependent hardware parameters such as the number of VU lanes, the VReg issue width, and VReg port count accordingly as shown in Fig. 4.1.

To some extent, chip architecting can be considered as an optimization problem, where

Table 4.1: Architecture Configuration of Datacenter Design Space

Constraint
Tech Node = 28nm, Freq = 700MHz; Area/Power Budget = 500mm^2/300W.
Optimization Target
TOPS Upper Bound = 92TOPS.
Design Space (X, N, T_x, T_y)
TU array length X = {4, 8, 16, 32, 64, 128, 256}; #TU per tile N = {1, 2, 4}; TU data type = Int8.
Mem capacity = 32MB.
NoC bisectional bandwidth = 256GB/s; Ring when #Tile on chip $T_x * T_y \leq 4$, 2D-Mesh when $T_x * T_y \geq 8$; Off-chip bandwidth = 700GB/s (HBM).

Table 4.2: Characteristics of ML Workloads Used in Case Study

Workload	ResNet	Inception	NasNet
#MAC Op	7.8G	5.7G	23.8G
#Data	5.72M	2.93M	5.35M
#Param	23.7M	22.0M	84.9M

we try to maximize performance under a given budget on chip area and power. Thus, we pick reasonable optimization targets and design constraints to make the design space exploration manageable. Particularly, as shown in Table 4.1, for datacenter inference accelerators, we constrain the die area to 500mm^2 and TDP to 300W based on recent data center ML accelerators [13, 83]. The memory subsystem is configured with 32MB of software managed on-chip memory distributed to all cores and 700GB/s off-chip HBM bandwidth, similar to Google's TPU-v2/v3 [14]. Note that TPU-v2/v3 are designed for both training and inference [83]. We then use NeuroMeter to sweep the design space to optimize the TOPS for each design point of (X, N, T_x, T_y) with dependent hardware parameters automatically scaled proportionally to the design point parameters as shown in Fig. 4.1 and Table 4.1.

Before setting the ranges of the implicit hardware parameters in Fig. 4.1, we explore a larger design space of systolic array centric architectures, including a larger number of

TUs per core, multiple TUs sharing VReg read/write ports, and other types of inner-TU interconnection. We prune the design points that exceed the area/power budgets or have extremely low performance. We only take the design points that meet the perf/power/area requirements into the second round for further workload-aware analysis. To make the design space manageable, we finally set the range of TU length (X) from 4 to 256. NeuroMeter automatically sets one VU per core with its lane number the same as the TU array length. NeuroMeter reserves two read ports and one write port in the VReg for each functional unit. The number of TUs in each core (N) determines how many total ports are required for each VReg, where a large N leads to an overhead explosion of VReg. For example, with eight 4x4 TUs per core, the VReg area and power overhead is 12.7% and 24.9% of the core. To avoid such an overhead explosion of VReg, N is capped at 4. The distributed on-chip memory is automatically multi-banked by NeuroMeter to satisfy the timing constraints determined by the target TOPS and clock frequency. The total core count (the product of T_x and T_y) is maximized to achieve the peak TOPS target while under the area and power constraint. For the convenience of evenly partitioning the neural network model, we assume T_x and T_y to be the power-of-2 numbers. To make the overall layout close to square, we assume that T_x is equal to or half of T_y.

Machine Learning Models

The datacenter case study uses three widely adopted CNN models, including ResNet-50 (abbrev. ResNet) [84], Inception-v3 (abbrev. Inception) [85], and NasNet-A-Large (abbrev. NasNet) [86]. Table 4.2 summarizes the characteristics of these ML models, including the compute (#MAC Op/frame), the peak transient memory footprint per frame (#Data), and the model size (#Param, quantized into Integer8).

Performance Simulation and Efficiency Modeling

TF-Sim [74] is used to simulate the performance of the ML models running on the target accelerators. TF-Sim first takes the computational graph (e.g., tfGraph [87]) of a given ML model and the same architecture configurations previously used as the inputs to NeuroMeter. Then, the simulator generates the performance of the ML model running on the target accelerators and the statistics for architecture components. The component level statistics are fed to NeuroMeter for computing runtime power and energy. The end-to-end performance (e.g., throughput and latency of inference) is used together with NeuroMeter's output on chip area and (runtime) power to compute energy efficiency and cost efficiency. The cost efficiency (i.e., TOPS/TCO) is approximated as TOPS/mm^4/Watt, where power (Watt) is an approximation of operational expenditures (OpEx) and area squared (mm^4) is an approximation of capital expenditures (CapEx) because silicon die cost grows roughly as the square of the die area [88].

TF-Sim supports advanced runtime graph scheduling and optimization, following the best practices in modern ML compiler/runtime such as XLA [89]. Especially for wimpy architectures, TF-Sim considers how to reduce the extra overhead of partial sum merging and weight/activation broadcast when a single TU is not large enough to map the whole operation without tiling. Moreover, TF-Sim also supports optimizations to improve parallelism, such as Space-to-Batch [90], Space-to-Depth [91], and double memory buffering. Fig. 4.2 shows the significant improvement of the simulated performance with the supported software optimizations, especially on small batch sizes. For wimpy designs, the operation is always too large to map on single TU without tiling. The mapping strategy considers how to reduce the extra overhead of partial sum merging and weight/activation broadcast.

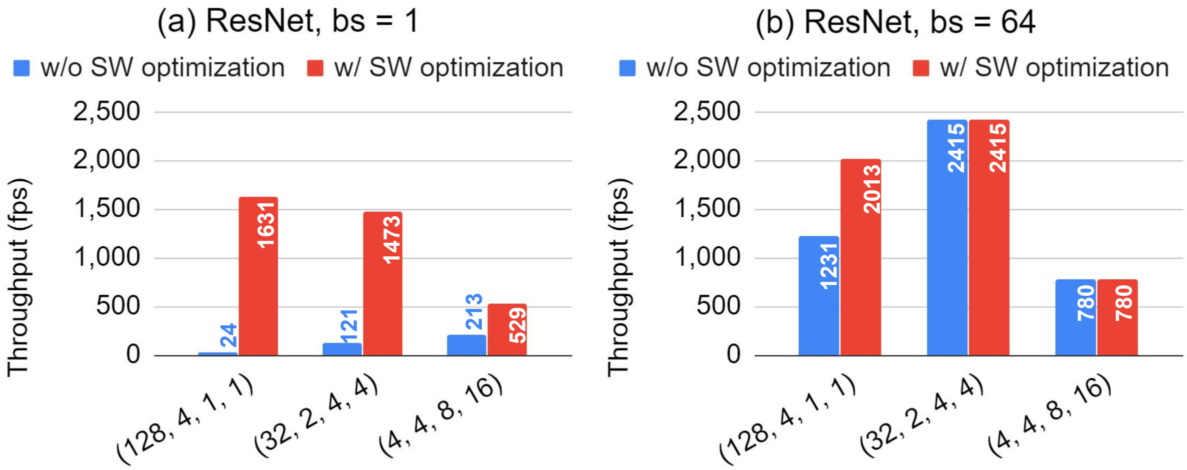


Figure 4.2: Throughput Before and After Software Optimization (© 2021 IEEE)

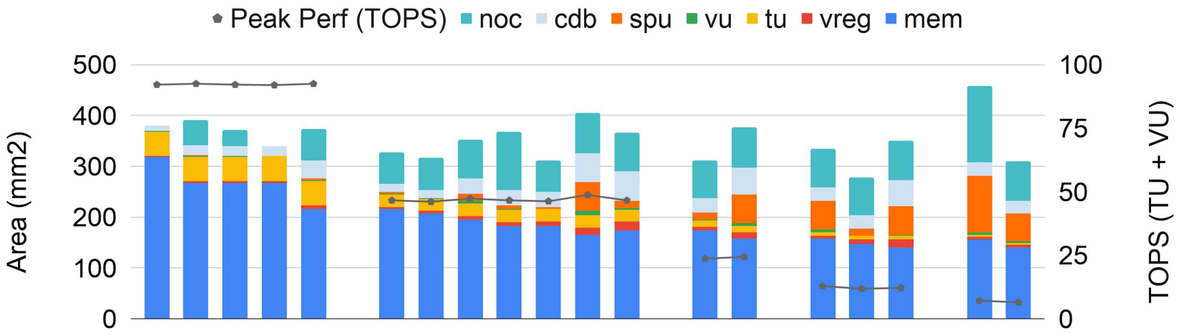
4.3.2 Results: Datacenter Inference Accelerator

This subsection first explores the design space using the chip area and TDP, then analyzes the runtime performance and efficiency by using NeuroMeter in conjunction with TF-Sim [74], a performance simulator. The study reveals important insights for ML inference accelerator designs, which otherwise cannot be discovered in a fast-yet-accurate way without NeuroMeter.

Chip Thermal Design Power and Area

Fig. 4.3 shows the die area and chip thermal design power (TDP) for the representative design points in the design space as defined in Table 4.1. As shown in Fig. 4.3(a), the on-chip memory consumes the largest portion of the die area among all architecture components. While die areas of all the design points are within the area budget of 500mm^2, the wimpier the accelerator is, the larger die area it needs to optimize for the target peak performance of 92TOPS. This is because wimpy designs need more cores as each core having smaller TUs, which in turn needs more interconnect such as NoC and CDB and more control logic such as scalar cores. However, even with an extra area

(a) Area Breakdown & Peak TOPS



(b) TDP Breakdown,
Relative TDP Efficiency (TOPS/W) & Relative Peak Cost Efficiency (TOPS/mm^4/W)

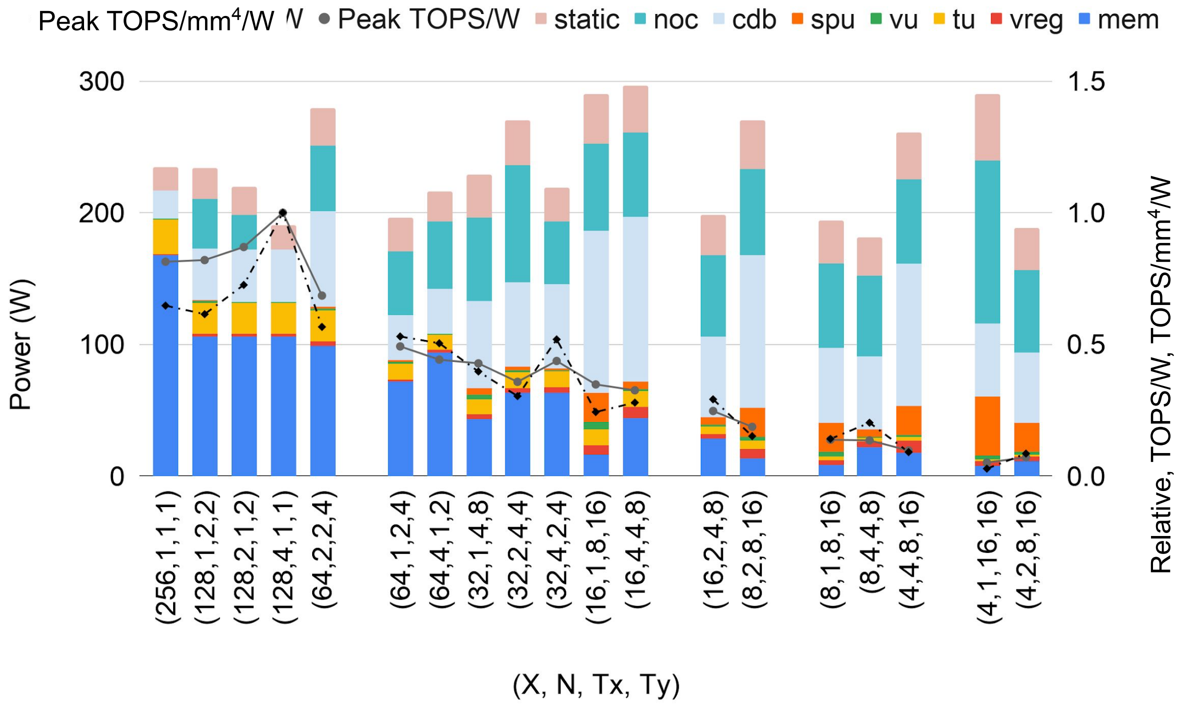


Figure 4.3: Area, TDP Breakdown, Peak TOPS, and Relative Power, Cost Efficiency in DataCenter Inference Chips. Figure (a) and (b) share the same x-axis that indicates the design point defined in Table 4.1. The subclusters are bins of peak TOPS. (© 2021 IEEE)

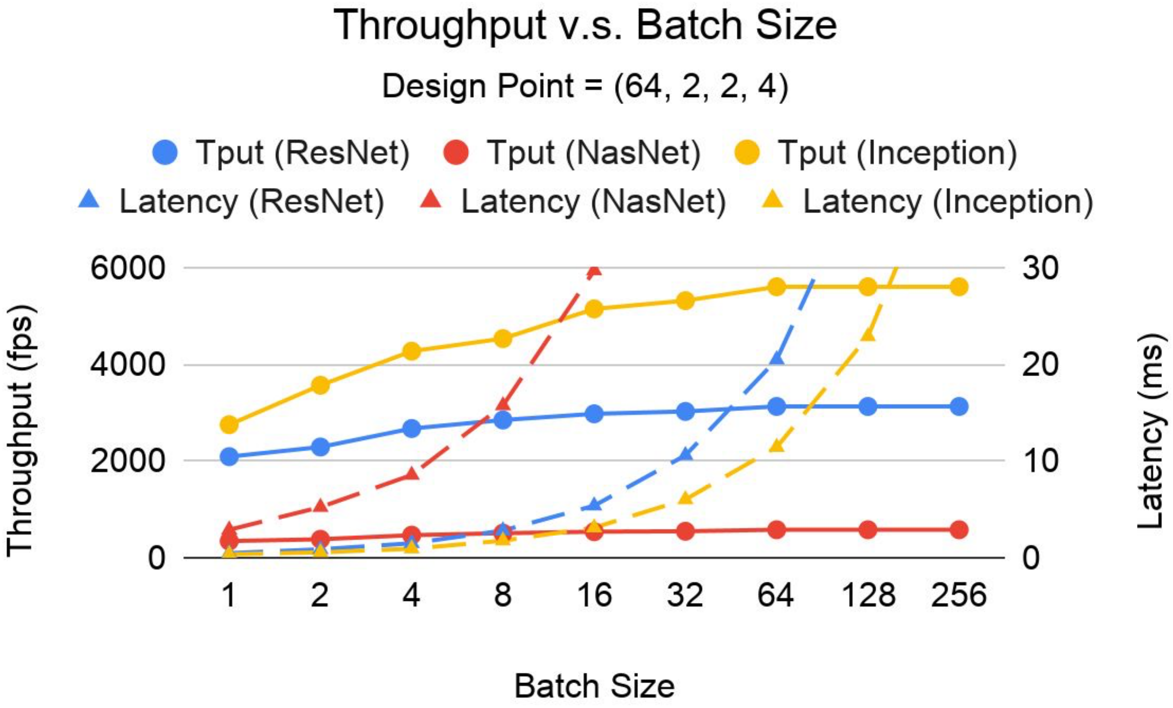


Figure 4.4: Performance on Different Batch Size. Throughput is measured as frame per sec (fps) that essentially is TOPS as operation per frame is constant. (© 2021 IEEE)

budget, the wimpy design still cannot achieve the same peak performance as the brawny cores. For example, the wimpy accelerators with 4x4 TUs have comparable or larger die areas than brawny designs with TUs sized of 64x64 to 256x256, but only less than 1/12 of peak TOPS of that of the brawny accelerators.

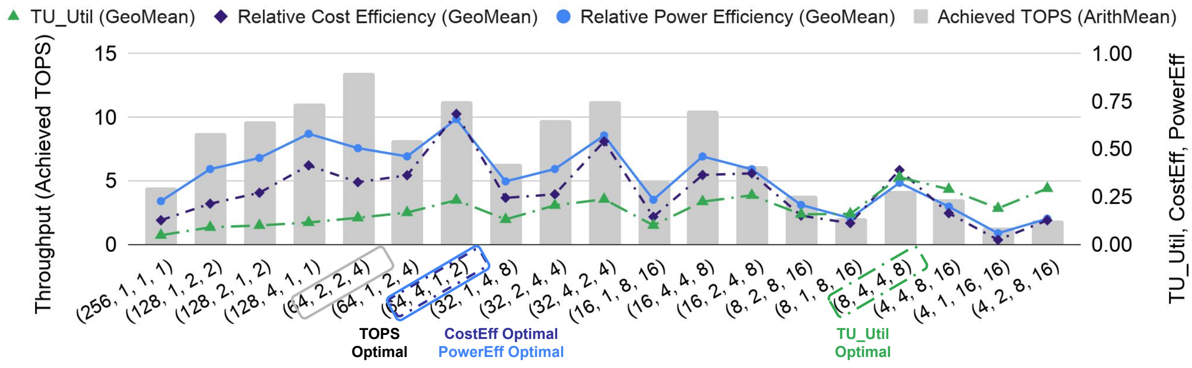
TDP analysis shown in Fig. 4.3(b) demonstrates a similar trend, where on-chip memory burns a big portion of the total power. Wimpy designs consume more power on interconnects and control flow logic than brawny designs. Fig. 4.3(b) also shows that the design point of (128, 4, 1, 1), i.e., the single-core accelerator with four 128x128 TUs in the core has the best peak TOPS/Watt and TOPS/TCO. In summary, brawny datacenter accelerator designs have the better area, TDP, and efficiency w.r.t. peak performance. Next, we will discuss more insights on the sweet spots of inference accelerator architectures w.r.t. runtime performance and efficiency.

Runtime Performance, Efficiency, and Trade-Offs

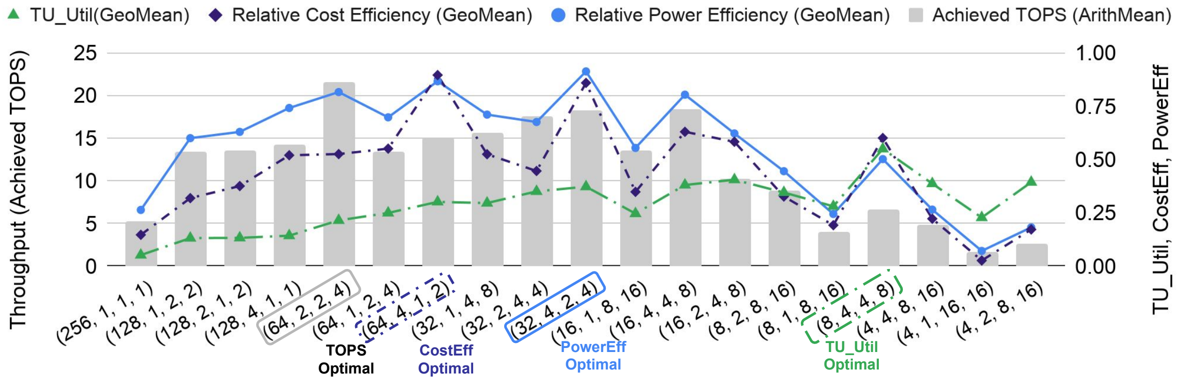
Datacenter inference accelerators are designed to maximize throughput, i.e., frames per second (fps) that essentially is TOPS as operation per frame is constant, when satisfying the latency requirements. Batch size is an important factor for runtime throughput and latency. For example, Fig. 4.4 shows the relationship between performance, including both throughput and latency, and batch size for the design point of (64, 2, 2, 4), i.e., an inference accelerator with 2x4 cores with each core having two 64x64 TUs. For all ML models, we can observe significant throughput improvements when the batch size switches from 1 to 64. This is because even with advanced graph optimizations, the accelerator still suffers from low utilization at a small batch size.

Fig. 4.4 also shows the boundary on batch size for real-time tasks. Concretely, real-time online inference is assumed to have a latency constraint of 10ms based on production requirements from Google [13] and Facebook [92]. Therefore, the upper-bound batch sizes

(a) AvgPerf (ResNet, Inception, NasNet), Small Batch, bs= 1



(b) AvgPerf (ResNet, Inception, NasNet), Medium Batch, 10ms-Latency Bounded



(c) AvgPerf (ResNet, Inception, NasNet), Large Batch, bs=256

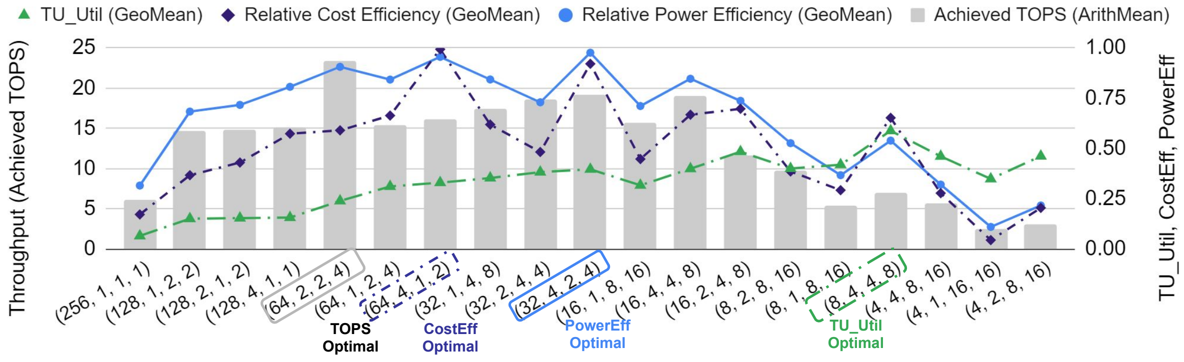


Figure 4.5: Average Runtime Performance of ResNet, Inception, and NasNet of DataCenter Inference Chips. (a) bs=1, (b) bs = bounded by 10ms latency, (c) bs=256. The relative cost/power efficiency in right y-axes in three subfigures are normalized against the largest absolute values in Subfigure (c). The colored boxes in the x-axes are optimal points for different design targets. (© 2021 IEEE)

to meet the 10ms latency requirement are 16, 4, 32 respectively for ResNet, NasNet, and Inception with the given design point. Thus, in the subsequent study, the same approach is used to determine the maximum batch sizes that maximize the throughput while meeting the latency requirements. Such batch size is called as "latency limited batch size" (aka medium batch size). The study also includes "batch size of 1" (aka small batch size) for the optimal latency but low throughput scenarios (e.g., extremely low latency service) and "batch size of 256" (aka large batch size) with very high throughput but also high latency for offline inference service that does not impose latency Service Level Objectives (SLOs).

Fig. 4.5 shows the average performance and efficiency of the three datacenter workloads. Fig. 4.5(a)-(c) represents the small, medium, and large batch, respectively. Each subfigure analyzes four metrics, including throughput (achieved TOPS), TU utilization (achieved TOPS/Peak TOPS), normalized cost efficiency (achieved TOPS/TCO, aka TOPS/mm^4/Watt), and normalized energy efficiency (achieved TOPS/Watt). Arithmetic mean is used for averaging the throughput and geometric mean is used for averaging other metrics as they are all ratios. Two important insights can be observed as follows:

Firstly, an important insight observed from Fig. 4.5 is that the optimal design varies w.r.t. optimization targets, which is difficult to discover without tools like NeuroMeter. For all the three batch size categories, the wimpy design with 32 cores and four 8x8 TUs per core, i.e., $(X, N, T_x, T_y) = (8, 4, 4, 8)$, always has the highest TU utilization. However, it is the brawny design with 8 cores and two 64x64 TUs per core that has the highest throughput because of its much higher peak TOPS than the wimpy design and thus compensates for the low TU utilization. The cost-efficiency optimized design is similar to the throughput-optimized design as they both prefer 64x64 TUs, except the former prefers fewer yet larger cores to reduce the NoC area overhead. The energy-efficiency optimized architecture also prefers brawny design with a slight drop in TU size from 64x64 to 32x32 with both medium and large batch size. This is because the energy consumption of systolic arrays scales quadratically with the length of the TU. These discoveries also carry an important conclusion: while wimpy designs have higher utilization, it is the brawny designs (with 64x64 and/or 32x32 TU size) that have the highest performance and efficiency.

Secondly, an important tradeoff exists among the brawny designs, where a large improvement of efficiency can be gained with a small sacrifice on throughput. As shown in Fig. 4.5(a), when choosing the efficiency-optimized design, i.e., (64, 4, 1, 2), over the throughput-optimized design, i.e., (64, 2, 2, 4), the target accelerator gains 2.1x cost-efficiency improvements and 1.3x power-efficiency improvement, with less than 16% sacrifice on sustainable achieved TOPS. This is because, compared to the efficiency-optimized design, the throughput-optimized design has more cores to distribute and balance computation but requires longer and more power-hungry inter-core NoC. Similar tradeoffs also exist in the medium and large batch size configurations as shown in Fig. 4.5(b) and (c). These tradeoffs provide important design guidance for architecting ML inference accelerators with different design priorities.

Summary of the Key Observations and Insights

The key observations and insights from the brawny and wimpy manycore ML accelerators are summarized as follows:

First, for datacenter inference chips, on-chip memory takes the largest die area among all architectural components. On-chip memory is also a major power consumer. However, on-chip interconnect starts to dominate the power consumption as the accelerators have more and more relatively wimpier cores.

Second, wimpy designs have higher utilization because smaller TUs are easier to schedule and parallelize with sophisticated software; meanwhile, brawny designs achieve better performance and efficiency for datacenter inference chips because they have less overhead from control logic and long distance on-chip interconnect.

Third, the optimal design varies w.r.t. the optimization target. There are also important tradeoffs among the selection of design targets for an architect. For example, for relatively brawny designs, we can achieve substantial benefits in efficiency with only a small sacrifice in throughput.

4.4 Case Study on Sparsity Implication

To showcase NeuroMeter's capability in modeling a wide range of diverse ML accelerator architectures, this section conducts a case study on implications of sparsity on both tensor-unit (TU) based and reduction-tree (RT) based ML accelerators. Leveraging the previous results of the latency-bounded design space exploration in Fig. 4.5(b), we pick the power efficient optima with 32x32 TUs (abbrev. TU32), and the utilization optima with 8x8 TUs (abbrev. TU8) to further explore their performance and efficiency when dealing with sparse workloads. Thanks to NeuroMeter's flexible capability in supporting different architectures, we use the RT-based architecture with the same OPS per compute unit as the corresponding systolic arrays, including 1024-to-1 RT (abbrev. RT1024) and 64-to-1 RT (abbrev. RT64).

A synthetic SpMV microbenchmark with different element-wise sparsities is generated manually for a weight matrix of $M \times N$ and the batched vectors of $N \times K$, where $M, N \geq 1024$, and the batch size $K \geq 32$, to ensure sufficient parallelism for TU/RT utilization. The sparse weight matrices use the Compressed Sparse Row (CSR) format [93], including the non-zero elements and the row/column indices. The batched vectors are assumed dense in this case study; and SpMSpV [94] (i.e., Sparse-Matrix-Sparse-

Vector-Multiplication) is beyond the scope of this case study.

Since TF-Sim, the performance simulator paired with NeuroMeter in Sec. 4.3, does not support sparse operations, we develop a simple roofline model similar to that in [95] for runtime performance estimation, which is then combined with NeuroMeter to generate power and energy efficiency results. The modified simple roofline model is shown in the equations below:

$$t_{\mathrm{d}} = \max(t_{\mathrm{d\_comp}}, t_{\mathrm{d\_bw}}) = \max(\frac{C}{F}, \frac{S_V + S_W}{B});$$
$$t_{\mathrm{s}} = \max(t_{\mathrm{s\_comp}}, t_{\mathrm{s\_bw}}) = \max(\frac{\alpha \cdot y \cdot C}{F}, \frac{S_V + \beta \cdot x \cdot S_W}{B});$$

$$\begin{aligned}\mathrm{EnergyEfficiencyGain} &= \frac{(TOPS/Watt)_{\mathrm{s}}}{(TOPS/Watt)_{\mathrm{d}}} \\ &= \frac{(C/t_{\mathrm{s}})/Power_{\mathrm{s}}}{(C/t_{\mathrm{d}})/Power_{\mathrm{d}}} \\ &= \frac{Power_{\mathrm{d}} \cdot t_{\mathrm{d}}}{Power_{\mathrm{s}} \cdot t_{\mathrm{s}}}\end{aligned}$$

where t_{d} the runtime for dense MV; and $t_{\mathrm{d\_comp}}$ and $t_{\mathrm{d\_bw}}$ are the compute time and the memory time for the dense MV, respectively. According to the roofline model, the overall runtime t_d is the maximum of these two terms. Similarly, t_{s} is the SpMV runtime; and $t_{\mathrm{s\_comp}}$ and $t_{\mathrm{s\_bw}}$ are the SpMV runtime bound by compute and memory bandwidth, respectively. The symbol C (in OPs) is the computational operations required in the dense MV; S_V and S_W (both in bytes) are the size of the batched input/output vectors and the weight matrix respectively without considering sparsity; F (in OPs/sec) and B (in bytes/sec) are the compute capability and memory bandwidth of the accelerator, respectively.

The symbol x represents the non-zero ratio of the weight matrix, i.e., the lower the x, the higher the sparsity of the weight matrix. The symbol y is the reduction factor of the total compute operation, and it is determined by the non-zero ratio x and the distribution of zero elements. Particularly, the systolic array based TU conducts block-wise zero-skipping to reduce computation, i.e., if the zero elements form a block of the size of TU's systolic array and align on the systolic array loading boundary, then this all-zero block can be skipped for computation. For the whole sparse matrix, it is assumed to be partitioned into TU-sized blocks and evenly mapped to all the on-chip systolic arrays with the block-wise zero skipping. Similarly, RT conducts vector-size zero-skipping. The symbols α and β denote the compute and storage overheads of sparse representations, respectively. α is optimistically set to be one, assuming the overhead of loading and decompressing CSR weight matrix can be overlapped with the computing time of systolic arrays and reduction trees. Depending on sparsity, data type, and the size of the weight matrix, β is a value between 2.0 and 2.5 in this case study. It is determined by CSR encoding overhead. First, the whole weight matrix is tiled into 256x256-sized submatrices. Then, each Int8 non-zero element requires an extra byte for column indexing; each tiled row requires an extra byte for inner-submatrix row indexing; and each submatrix requires two bytes for tile indexing.

The energy efficiency gain is the ratio of energy efficiency (i.e., OPS/Watt [96]) between the SpMV and its dense counterpart. Since the SpMV and its dense counterpart are considered to achieve the same effective operations, i.e., MxNxK, the energy efficiency gain is simplified to SpMV's runtime energy reduction compared to its dense counterpart. Considering the goal is to showcase NeuroMeter's capability to model power, area, and timing of a wide range of different ML accelerator architecture, more sophisticated techniques to maximize the performance benefits of sparsity are beyond the scope of this case study. This simple roofline analytical performance model is then paired with NeuroMeter to study final energy efficiency implications.

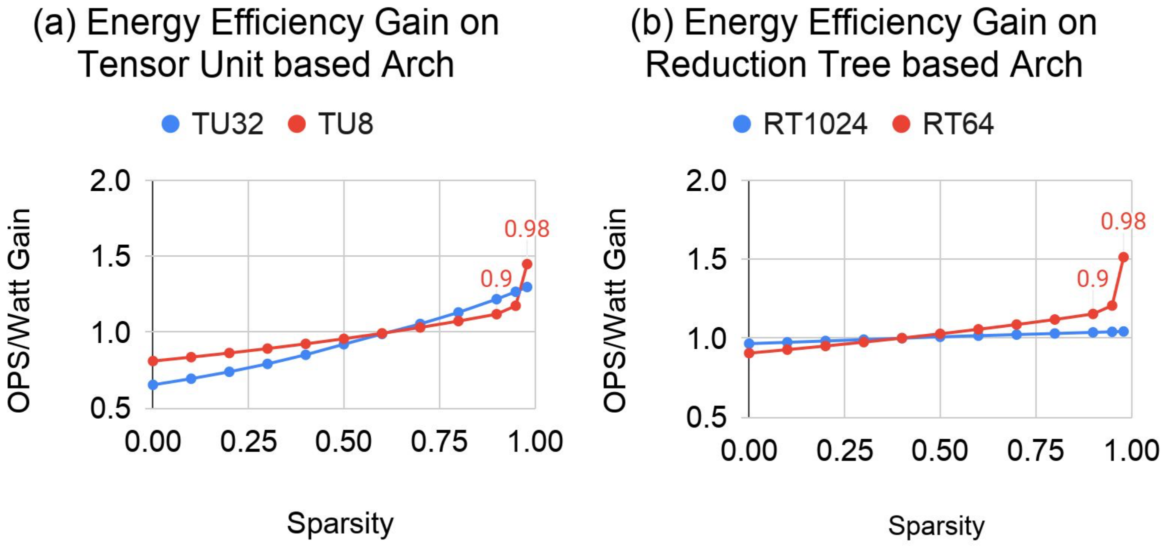


Figure 4.6: The Energy Efficiency Gain of Sparse over Dense computation at Different Sparsity Levels on (a) Tensor Unit and (b) Reduction Tree based Architectures. For each architecture, its energy efficiency at different sparsities is normalized against that of the baseline dense processing on the same architecture. Thus, it indicates an improvement when the energy efficiency gain is larger than one. (© 2021 IEEE)

Fig. 4.6 shows the energy efficiency gain of sparse over dense under different sparsity levels for different architectures and configurations. For all designs, the energy efficiency increases as the sparsity grows. However, compared to the dense counterpart, the energy efficiency only benefits from sparsity when the sparsity level is larger than 0.5. This is because the power saving from the block/vector-wise zero skipping is limited and is unable to amortize the extra data transfer of CSR encoding when sparsity is low. Moreover, as shown in Fig. 4.6, a clear transition point can be observed when sparsity is 0.9 in TU8 and RT64; while the efficiency grows slowly in a low slope in TU32 and RT1024. This implies that the brawny design gets efficiency benefits mostly from the reduced CSR encoding as sparsity grows rather than the block/vector-wise zero skipping.

Clearly, despite its relatively lower absolute energy efficiency as shown in Fig. 4.5, a wimpier architecture with fine-grained computing units can benefit from element-wise sparsity more than a brawnier coarse-grained architecture.

4.5 Summary

By combining the power, area, and timing results of NeuroMeter with performance simulation, this chapter explores the manycore ML accelerator design in multiple scenarios. The first case study explores the brawny and wimpy designs in the datacenter inference scenario. The results show that brawny designs with 64x64 systolic arrays are the most performant and efficient for inference tasks in the 28nm datacenter architectural space with a 500mm^2 die area budget. The results also reveal important tradeoffs between performance and efficiency. For datacenter accelerators with low batch inference, a small (∼16%) sacrifice of performance can lead to more than a 2x efficiency improvement (in achieved TOPS/TCO). The second case study showcases NeuroMeter's capability to model a wide range of accelerator architectures by studying the energy efficiency (TOPS/Watt) implications of sparsity on different ML accelerators. The results show that despite its relatively low energy efficiency, it is easier for the wimpier architectures to benefit from sparsity processing.

Chapter 5

RSA: A Reconfigurable Systolic Array based CNN Accelerator

5.1 Introduction and Contribution Overview

This chapter is dedicated to a reconfigurable systolic array (RSA) design for the emerging convolutional neural networks (CNNs) which introduce the operator of depthwise convolution (DConv2d). Different from the traditional CNN workloads (e.g. ResNet [1] and Inception [85]), the emerging CNN models with the DConv2d layers (e.g. NasNet [86], MNasNet [97], EfficientNet [3, 98], and EfficientNet-X [99]) have lower operation numbers (#OPs) and lower operational intensities. Compared with the widely-adopted brawny homogeneous systolic array (HSA) based ML accelerators, the proposed architecture of RSA in this chapter, increases the utilization of the array-based computing units effectively. The increasing array utilization successfully converts the decreasing #OP of the emerging CNN model to a shorter latency and a higher throughput during runtime.

Brawny HSA-based accelerators, for example, Google's TPU-series [13, 14, 15, 16], have achieved satisfying performance on multiple ML workloads. However, as more and more ML models and emerging operators are proposed, it is observed that not all emerging CNN models can achieve as good performance as the traditional CNN models [1, 85]. Especially, although some emerging CNN models [3, 86, 97, 98, 99] achieve similar levels of accuracy with fewer #OPs by introducing the DConv2d layers, the reduced #OP does not effectively turn into a shorter execution time during runtime. Take EfficientNet-B0 [3] as an example, although DConv2d only takes less than 10% in the overall convolution #OP, our analysis shows that DConv2d takes nearly 70% of the overall convolution execution time. Further, we find that DConv2d's unsatisfying performance mainly comes from a large amount of extra ineffective multiplications when padding zeros to tile DConv2d into loops of large-sized MatMuls during mapping the DConv2d onto the brawny HSA-based accelerators. The ineffective multiplication has "zero" as one multiplier input of the systolic cell, and further leads to an extremely low array utilization of the brawny HSAs during runtime.

Inspired by the brawny vs wimpy study as in Sec. 4.3, an intuitive solution is first explored by replacing the architecture of small numbers of brawny HSAs with the many-core architecture of large numbers of wimpy HSAs. However, the results show that the performance improvement of DConv2d cannot amortize the performance degradation of the traditional convolution (Conv2d) in the wimpy HSA design. On one hand, the wimpy architecture indeed improves the systolic array utilization, and later results to DConv2d's performance improvement. On the other hand, the reduction of the overall TOPS leads to Conv2d's performance degradation. This is because the wimpy architecture introduces extra hardware overheads of the control logics and inter-core connections given the same area and power budget as the brawny counterpart.

This chapter proposes an RSA-based CNN accelerator, where the systolic arrays are allowed to configure into different working modes thanks to the reconfigurability of the inner-array connections. The reconfigurable architecture can achieve both the high performance of brawny systolic arrays and the high utilization of wimpy systolic arrays at the same time.

The contribution of this chapter is as follows:

1) An RSA-based CNN accelerator is proposed. It introduces quite small area and power overheads as well as minor changes on control logics while achieving both the high performance of the brawny design and the high array utilization of the wimpy design.

2) The hardware modeling of the proposed RSA architecture is built; and integrated into NeuroMeter, the hardware modeling framework established in Chp. 3.

3) The results show that the proposed RSA-based accelerator achieves 1.25x-2.68x performance gains on ResNet and EfficientNet compared with the TPU-v2 styled baseline.

In the rest of this chapter, Sec.5.2 discusses the HSA-based architecture and shows different sweet points of array length on different workloads; Sec.5.3 proposes RSA, the systolic array with reconfigurable interconnections; Sec.5.4 gives the hardware modeling of RSA; Sec.5.5 shows the experiment results; Sec. 5.6 summarizes the chapter.

5.2 Motivation

The standard systolic array has been adopted by Google's four generations of cloud TPUs [13, 14, 15, 16]. TPU-v1 adopts the 256x256 systolic array; while the follow-up generations of TPU-v2, TPU-v3, and TPU-v4i adopt the 128x128 systolic arrays. No matter what size of the systolic array is adopted in the matrix multiplication units, the existing TPUs have one common feature, i.e., all the systolic arrays on one chip have the same size. We call this type of architecture as a *manycore* ***homogeneous systolic array (HSA)*** *based accelerator*. A question is raised, i.e., what is the proper size of the systolic array? The brawny vs wimpy study in Sec. 4.3 has explored the performance of manycore HSA-based accelerators in the datacenter scenarios. This section further

Table 5.1: Hardware Configurations and Peak TOPS of Manycore HSA-based Accelerators with Different Array Lengths

Tech Node = 28nm, Freq = 700MHz, Area/Power Budget = 500mm^2/300W			
Idx	Len(HSA)	#Core	Peak TOPS
Design 0	128	4	92.47
Design 1	64	8	46.59
Design 2	32	32	47.31
Design 3	16	128	48.74
Design 4	8	128	12.90
Design 5	4	256	7.17

shows different runtime performances when different ML models run on the manycore HSA-based accelerator with the given array size. Especially, we observe that traditional Conv2d and the emerging DConv2d prefer the HSAs of different sizes. This observation implies that the HSA with a specific size cannot meet up the performance requirements of all the CNN workloads. It further inspires the proposal of the systolic arrays with reconfigurable interconnections (RSA), which will be later discussed in Sec. 5.3.

Following the hardware configuration in the brawny vs wimpy study in Sec. 4.3, one design point per systolic array length is picked out to represent a typical design of the manycore HSA-based accelerator with the specific systolic array length. Table 5.1 shows the selected designs of the manycore HSA-based accelerator where each core is assumed to have one SA with a specific size. The single-core architecture is shown in Fig. 4.1; the topology of 2D mesh is assumed as the inter-core connection with 256GB/s bi-sectional bandwidth; the off-chip memory bandwidth is assumed 700GB/s in total. The core number is determined under the area/power budget of 500mm^2/300W with the technology node of 28nm and the clock rate of 700MHz. Fig. 5.1 compares the performance of EfficientNet-B0 [3] with batch size 128 on manycore HSA-based accelerators with different array lengths. It is observed that:

First, the median HSA-based accelerators (i.e., 32x32, 16x16) get much better performance on the overall latency compared with the brawny HSA (i.e., 128x128, 64x64) and the wimpy HSA-based accelerators (i.e., 8x8, 4x4) as shown in Fig. 5.1(a). To further look at the operator-wise performance, the brawny HSA-based accelerators perform the worst on DConv2d among all the design points. It uses about 70% of the overall latency to process DConv2d, which only takes up 8.9% of the overall #OP. This implies that the TPU-styled brawny HSA-based accelerator may not work well for the workloads which introduce the DConv2d operator, even if DConv2d takes only a very small proportion. Meanwhile, the wimpy HSA-based accelerators perform the worst on Conv2d. This implies that it is unable to get satisfying performance by directly shrinking the size of each HSA and increasing the core number.

Second, Conv2d and DConv2d have different performance sweet spots, i.e., Conv2d gets the optimal latency on the 32x32 HSA-based accelerator; while DConv2d gets the optimal latency on the 16x16 HSA-based accelerator. This implies that one size for all arrays may not win on all the operator types.

Third, the optimal points of different metrics diverge on DConv2d as shown in Fig. 5.1(b), i.e., the optimal point of runtime TOPS for DConv2d layers on 16x16-HSA based accelerator, while the array utilization keeps increasing as the array length gets smaller. Considering the peak TOPS reduces monotonically from the brawny HSA-based accelerators to the wimpy counterparts shown in Table 5.1, the unsatisfying performance of the manycore wimpy HSA-based accelerators results from the insufficient computing power due to the extra high area overhead when building up the manycore architecture.

The above observations inspire us that: (1) A high peak TOPS does not necessarily mean satisfying performance, yet an extremely low peak TOPS will lead to poor performance; (2) One size for all arrays cannot work well on all workloads; (3) Due to the high array utilization on both Conv2d and DConv2d, wimpy HSA-based architecture has the potential of achieving better performance. If the architecture provides more computing power, the end-to-end performance will improve prominently.

5.3 Architectural Design

As illustrated in Fig. 5.2(a), the systolic array with reconfigurable interconnections, RSA, splits the original standard systolic array into multiple *basic subarrays*. Additionally, RSA introduces the two-by-two crossbars as reconfigurable interconnections between the nearby basic subarrays, and input/output memory banks. For each basic subarray, besides the interconnections between nearby systolic cells, RSA introduces a pair of vertical/horizontal bypass interconnects for each basic subarray to allow the dataflow to route aside the current basic subarray.

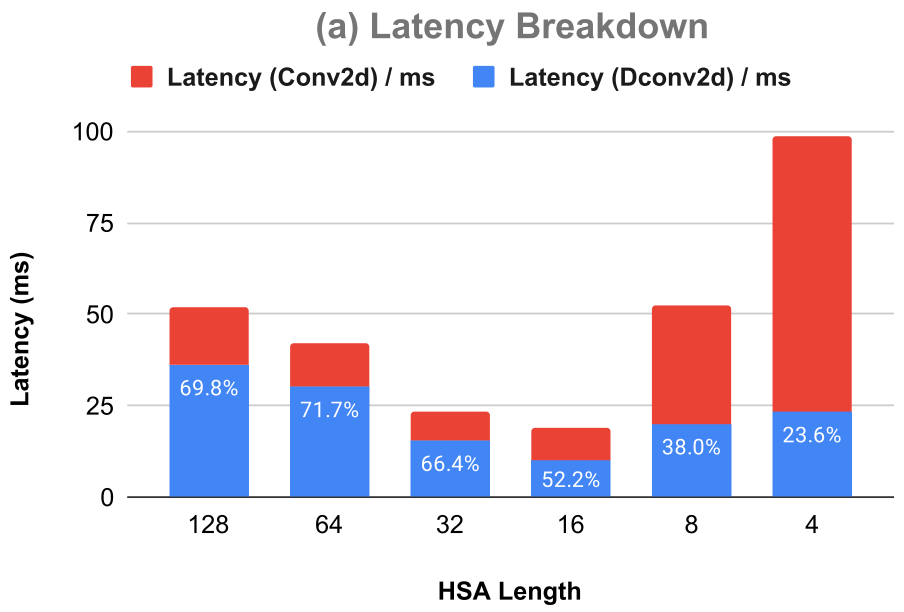
One or multiple basic subarrays can make up an *active array*, whose constituting systolic cells are working together in a systolic way. To better illustrate how to reconfigure RSA in different working modes, we assume the original systolic array is sized of 128x128, and each basic subarray is sized of 32x32, with 16 basic subarrays in total. Fig. 5.2(b) illustrates how it is configured to work as a 128x128 standard systolic array where all the 16 basic subarrays are active and make up one large active array; Fig. 5.2(c) illustrates how it is configured to work as dual independent 64x64 active arrays when the rest eight 32x32 basic subarrays are idle; Fig. 5.2(d) illustrates how it is configured to work as four independent 32x32 active arrays when the rest twelve 32x32 basic subarrays are idle.

To parameterize RSA's architecture, we have:

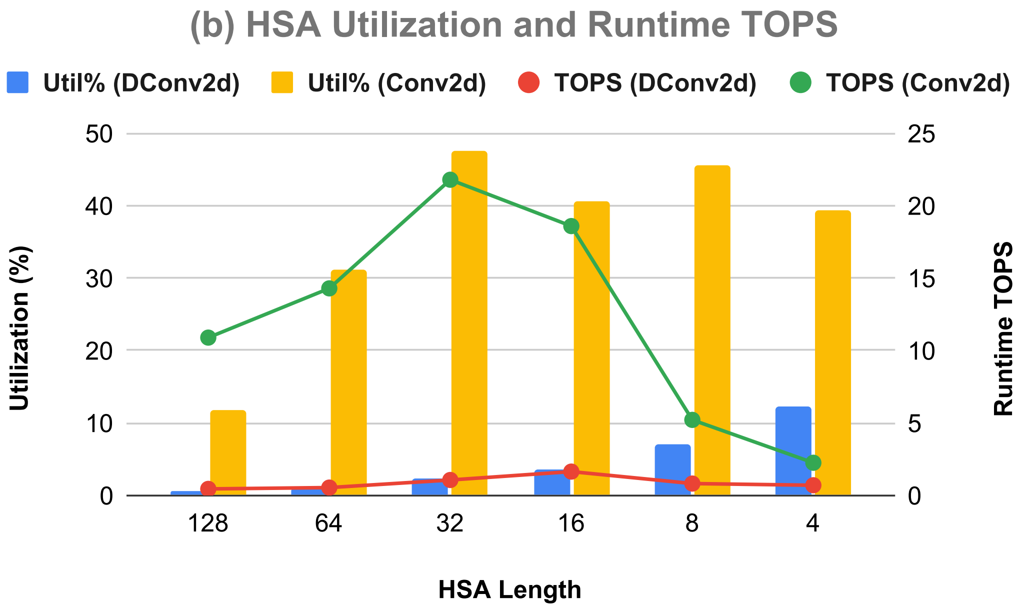
(1) The overall systolic array has $X \times X$ systolic cells in total.

(2) Each basic subarray has $X_{base} \times X_{base}$ systolic cells, then the total number of basic subarrays, N_{BSA} equals $|X/X_{base}|^2$.

(3) In one working mode, each active array has $X_{act} \times X_{act}$ systolic cells; and equivalently, each active array is made up of $|X_{act}/X_{base}|$ basic subarrays. The active arrays



((a)) Latency Breakdown of Conv2d and DConv2d, where the percentage on the bar represents the proportion of DConv2d's latency.



((b)) Array Utilization and Runtime TOPS of DConv2d and Conv2d.

Figure 5.1: Operator-wise Performance Analysis of EfficientNet-B0 with Batchsize = 128 on Manycore HSA-base Accelerators with Different HSA Lengths.

have to be placed on the diagonal of the overall systolic array. There are $|X/X_{act}|$ active arrays in total (equivalently $|X \cdot X_{act}/X_{base}^2|$ basic subarrays are in working), while the rest $(N_{BSA} - |X \cdot X_{act}/X_{base}^2|)$ basic subarrays are idle.

(4) X, X_{act}, and X_{base} are all power-2 numbers, where $X_{base} \leq X_{act} \leq X$. There are $(\log_2 X - \log_2 X_{base} + 1)$ possible values for X_{act}, where each value corresponds to one working mode. On one extreme where $X_{act} = X_{base}$, RSA is configured to have only X/X_{base} basic subarrays on the diagonal in the working mode. On the other extreme where $X_{act} = X_{base}$, RSA is configured to work functionally the same as the original $(X \times X)$-sized systolic array. In the rest of this chapter, if no further explanation, RSA(X, X_{base}) is used to describe the hardware architecture of RSA; and RSA(X, X_{base}, X_{act}) is used to describe the working mode when the active array of RSA is configured in the length of X_{act} during runtime.

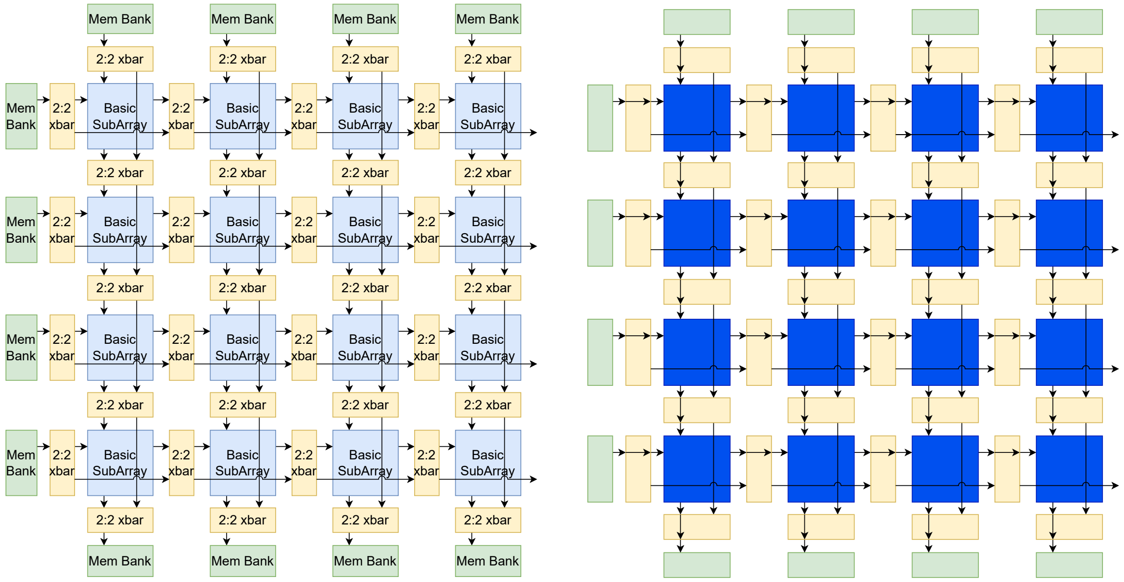
5.4 Hardware Modeling

5.4.1 Area

Besides the basic subarrays which can be directly modeled by the standard systolic arrays, RSA contains the 2-by-2 crossbars and the bypass interconnections as well. Assume A_{BSA} and A_{Xbar} denote the area of one $X_{base} \times X_{base}$-sized basic systolic subarray and one 2-by-2 crossbar [1]; $|p_{wire}|$ and $|DW|$ denote the wire pitch and the bitwidth of the bypass interconnections; each segment of the bypass interconnection aside one basic

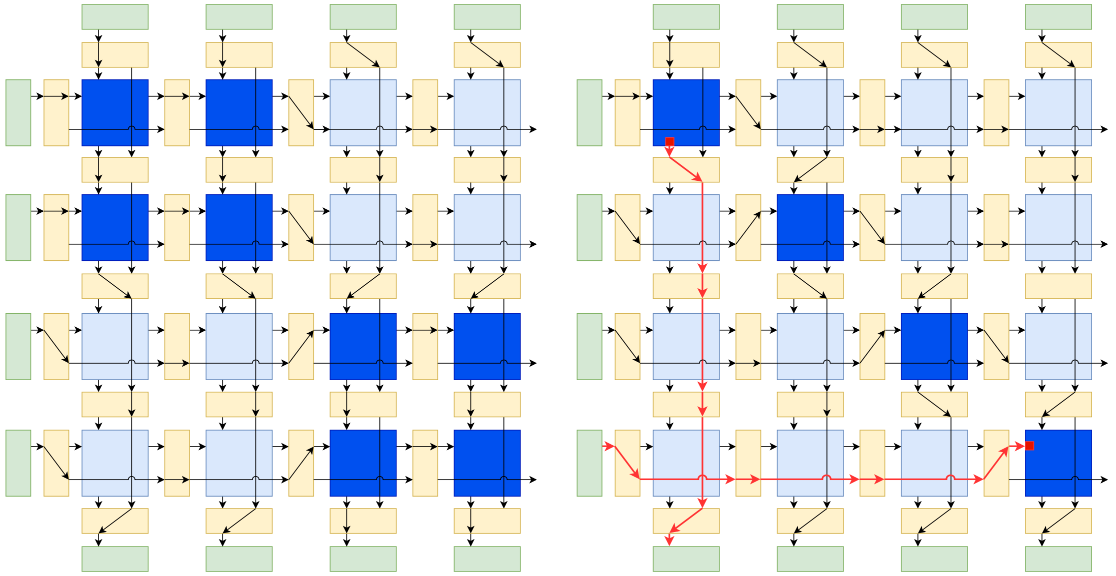
---

[1] The power, area, and timing model of the standard systolic, crossbar, and wire can refer to previous works of NeuroMeter [29] and McPAT [40].



((a)) RSA's architecture: systolic array with reconfigurable interconnections between basic subarrays.

((b)) RSA configured to work as a 128x128 standard systolic array, with all the sixteen 32x32 basic subarrays active.



((c)) RSA configured to work as two independent 64x64 active arrays, with the rest eight 32x32 basic subarrays idle.

((d)) RSA configured to work as four independent 32x32 active arrays, with the rest twelve 32x32 basic subarrays idle. The red marks show the two possible critical paths.

Figure 5.2: Reconfigurable Systolic Array (RSA): Its architecture and how it is configured into different modes, where the dark blue blocks are the active systolic subarrays, the light blue blocks are the idle ones; the light yellow blocks are 2:2 crossbars; the light green blocks are memory banks.

subarray has the length of $\sqrt{A_{BSA}}$. Then, RSA's area can be calculated as in Eq. 5.1.

$$\begin{aligned} A_{RSA} = A_{BSA} \cdot N_{BSA} + A_{Xbar} \cdot (N_{BSA} + \frac{X}{X_{base}}) \\ + |p_{wire}| \cdot \sqrt{A_{BSA}} \cdot 2N_{BSA} \cdot |DW| \end{aligned} \tag{5.1}$$

5.4.2 Timing

In a standard systolic array, the critical path of one cycle is made up of the systolic cell itself, while the critical path of RSA includes the systolic cell, the bypass interconnections, and the crossbars on the bypass interconnections. The red mark in Fig. 5.1(c) illustrates the two possible critical paths of RSA, where the small red block represents one systolic cell and the red arrows represent the critical path inside the crossbars and bypass interconnections. Assuming T_{SC}, T_{Xbar}, and T_{Wire} denote the latency of one systolic cell, one 2-by-2 crossbar, and the bypass interconnection with unit length, respectively, RSA's cycle time can be calculated in Eq. 5.2.

$$T_{RSA} = T_{SC} + T_{Xbar} \cdot \frac{X}{X_{base}} + T_{Wire} \cdot \sqrt{A_{BSA}} \cdot (\frac{X}{X_{base}} - 1) \tag{5.2}$$

5.4.3 Operator Mapping

Among the three parameters of RSA, the parameters of X and X_{base} are determined during hardware design; while X_{act} is determined by the workload characteristics during compilation. Similar to the operator mapping in Sec. 4.2, we will first tile the given MatMul operator into the blocks of unit MatMuls, where the unit MatMul matches the X_{act}-by-X_{act} active array size; and then operate these unit MatMul blocks with the parallelism of X/X_{act}. Eq. 5.3 shows the time consumption t_{MatMul} when processing batched **MatMul(M, N, K)** in the active array length of X_{act}. And we will pick out the optimal active array length where the shortest execution time is achieved during compilation. Although Eq. 5.3, where MatMul(M,N,K) tiles into unit operators of MatMul(X_{act}, X_{act}, X_{act}), looks similar to Eq. 4.1, where MatMul(M,N,K) tiles into unit operators of MatMul(X_t, X_t, X_t), the time consumptions in these two equations are quite different. This is because – Eq. 5.3 is able to process one MatMul(X_{act}, X_{act}, X_{act}) in X_{act} cycles since other basic subarrays on that column are all bypassed; while Eq. 4.1 has to process one MatMul(X_t, X_t, X_t) in X cycles to go through all systolic cells in the column due to the strict one-cast datapath.

$$T_{MatMul}(X_{act}) = (\lceil \frac{b \cdot \lceil \frac{M}{X_{act}} \rceil \cdot \lceil \frac{N}{X_{act}} \rceil \cdot \lceil \frac{K}{X_{act}} \rceil}{X/X_{act}} \rceil + 2) \cdot X_{act} \cdot T_{RSA} \tag{5.3}$$

5.4.4 Effective TOPS

When RSA runs the workload of batched-MatMul(M, N, K), where $M = N = K = X_{act}$, RSA will switch to the working mode with active arrays lengthed X_{act}, only the active arrays on the diagonal are in work while the other subarrays are idle and can be clock gated. Therefore, RSA's throughput can be calculated as in Eq. 5.4.

$$TOPS_{RSA} = \lfloor \frac{X}{X_{act}} \rfloor \cdot 2X_{act}^2 \cdot f_{RSA} = 2\frac{X \cdot X_{act}}{T_{RSA}} \tag{5.4}$$

When the same workload runs on HSA, the original batched-MatMul will be composed into MatMul(X,X,X). Although the nominal runtime TOPS increases to $2 \cdot X^2 \cdot f_{HSA}$. However, only partial of the runtime TOPS are effective, which correspond to the subarrays on the diagonal and have the value of $(2 \cdot X \cdot X_{act} \cdot f_{HSA})$, are effective; others correspond to the ineffective operations with input values of zero.

5.4.5 Dynamic Power

RSA's dynamic power depends on RSA's working mode because the idle subarrays and the unselected bypass interconnections can be clock-gated. Assuming $P_{dyn}(\text{BSA})$, $P_{dyn}(\text{Xbar})$, and $P_{dyn}(\text{Wire})$ denote the dynamic power of one basic subarray, one 2-by-2 crossbar, and the single-bit bypass interconnection with unit length, respectively. Then, RSA's dynamic power, when each active array has the size of $X_{act} \times X_{act}$, can be calculated as in Eq. 5.5.

$$\begin{aligned} P_{dyn}(\text{RSA}) = {} & P_{dyn}(\text{BSA}) \cdot |X \cdot X_{act}/X_{base}^2| \\ & + P_{dyn}(\text{Xbar}) \cdot (N_{BSA} + X/X_{base}) \\ & + P_{dyn}(\text{Wire}) \cdot \sqrt{A_{BSA}} \cdot \frac{X - X_{act}}{X_{base}} \cdot 2 \cdot |DW| \end{aligned} \tag{5.5}$$

5.4.6 Static Power

Assume $P_{leak}(\text{BSA})$, $P_{leak}(\text{Xbar})$, and $P_{leak}(\text{Wire})$ denote the static power of one basic subarray, one 2-by-2 crossbar, and the single bit bypass interconnection with unit length, respectively. Then, RSA's static power can be calculated as in Eq. 5.6.

$$\begin{aligned} P_{leak}(\text{RSA}) = {} & P_{leak}(\text{BSA}) \cdot N_{BSA} \\ & + P_{leak}(\text{Xbar}) \cdot (N_{BSA} + X/X_{base}) \\ & + P_{leak}(\text{Wire}) \cdot \sqrt{A_{BSA}} \cdot 2N_{BSA} \cdot |DW| \end{aligned} \tag{5.6}$$

5.5 Experiment

The architecture of TPU-v2 [14, 15] is used as a baseline which has dual cores each with single 128x128 standard systolic array per core. The original HSA-based Tensor Unit in TPU-v2 is replaced with the proposed RSA in 5.3. The proposed RSA's hardware

Table 5.2: NeuroMeter Estimated Area of Dual RSAs (the left six columns) and HSAs (the rightmost column), where RSA(X, X_{base}) is Configured as $X = 128$ and X_{base} Ranges from 4 to 64; HSA is Configured as in TPU-v2 (16nm@700MHz).

X_{base}	4	8	16	32	64	RSA(GeoMean)	**HSA(128)**
Area (mm^2)	72.2	72.1	72.4	73.2	73.0	72.6	**56.6**

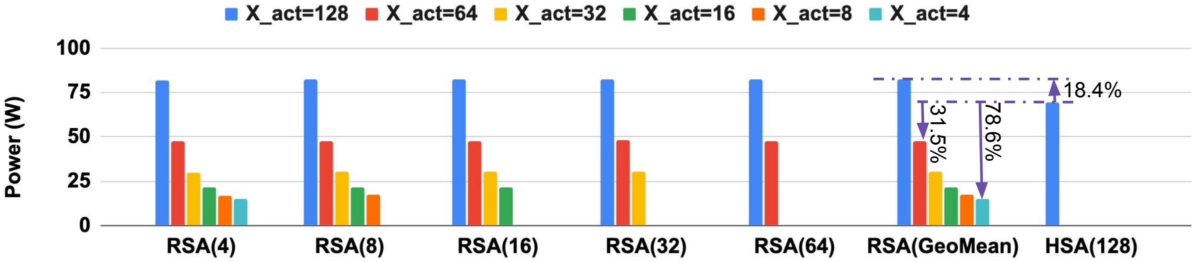


Figure 5.3: NeuroMeter Estimated Peak Power of RSA and HSA, where HSA is Configured as in TPU-v2 (16nm@700MHz).

modeling in Sec. 5.4 is integrated to NeuroMeter for the chip-level hardware simulation. Sec. 5.5.1 explores the peak performance, area, and power of the proposed RSA; Sec. 5.5.2 further explores RSA's runtime performance and efficiency with the given workloads.

5.5.1 Hardware Analysis

Table 5.2 and Fig. 5.3 show the area and power of the proposed RSA with different granularities of basic subarrays compared with the baseline of the dual 128x128 standard HSAs, whose configuration is the same as TPU-v2 under the 16nm technology node and 700MHz clock rate simulated by NeuroMeter.

Table 5.2 shows the areas of five groups of RSA(X, X_{base}) with different granularities of basic subarrays. Here, X, the length of the overall systolic array, is configured as 128 for each group; and X_{base}, the length of the basic subarrays, ranges from 4 to 64. The baseline HSA(128) represents TPU-v2's dual 128x128 systolic arrays. The observation shows that RSAs under different granularities of basic subarrays have negligible differences in area. Based on NeuroMeter's simulation, the geometric mean area of the five RSAs (i.e.,

72.6mm^2) introduces 28% area overhead compared with HSA(128) (i.e., 56.6mm^2).The published area of HSA(128) of TPU-v2 (i.e., 50.4mm^2) is listed aside to demonstrate the credibility of NeuroMeter's simulation results. Although RSA introduces nearly 30% area overhead on Tensor Unit itself in the experiment results, the original Tensor Units only take 24%, 8%, 11%, and 11% respectively in the published four generations of TPUs [13, 14, 15, 16]. Considering the RSA introduces negligible changes on dathpath and control logics of the original HSA-based architecture, RSA does not introduce other area overhead besides Tensor Unit itself. In summary, the proposed RSA introduces about 5% area overhead on the whole-chip level.

Fig. 5.3 illustrates the peak power of five groups of RSA(X, X_{base}) with different granularities of basic subarrays. The bars of different colors show RSA's peak power when the active array is configured as different lengths (X_{act}). Because the length of the active array is unable to be configured smaller than the length of the basic subarray, the active array has fewer available configurations as the basic subarray gets larger. Similar to the observation from the area side in Table 5.2, the peak power of the same active array configuration has negligible difference among different basic subarray sizes. Comparing the geometric mean of different RSAs and the baseline of HSA(128), we find that the proposed RSA introduces 18.4% power overhead when the active array is configured as 128x128, while RSA achieves 31.5% to 78.5% power efficiency when the active array is configured from 64x64 to 4x4.

5.5.2 Runtime Performance

Table 5.3 compares the runtime TOPS, latency, and throughput before and after replacing TPU-v2's HSA-based tensor units with RSAs. Three workloads are involved, including EfficientNet-B0 (abbrev. E0), EfficientNet-B7 (abbrev. E7), and ResNet-50

Table 5.3: Runtime Performance Comparison Before and After Replacing TPU-v2's HSA-based Tensor Units with RSAs.

Workload	Array Type	Runtime TOPS	Latency (ms)	Throughput (fps)
E0, BS=1	HSA	1.81	0.8	1,250
E0, BS=1	RSA	4.85	0.3	3,333
E0, BS=128	HSA	3.61	51.72	2,475
E0, BS=128	RSA	6.06	30.8	4,156
E7, BS=1	HSA	5.75	13.12	76
E7, BS=1	RSA	9.65	7.82	128
E7, BS=128	HSA	9.05	1067.88	120
E7, BS=128	RSA	11.31	854.59	150
ResNet, BS=1	HSA	14.76	1.06	943
ResNet, BS=1	RSA	22.76	0.69	1,449
ResNet, BS=128	HSA	22.59	88.55	1,446
ResNet, BS=128	RSA	35.79	55.87	2,291

Table 5.4: Runtime Performance Comparison on DConv2d and Conv2d Before and After Replacing TPU-v2's HSA-based Tensor Units with RSAs.

		Runtime TOPS		Latency (%)		OP Count (%)	
Workload	Array Type	DConv2d	Conv2d	DConv2d	Conv2d	DConv2d	Conv2d
E0, BS=1	HSA	0.22	6.3	73.75	26.25	8.81	91.19
E0, BS=1	RSA	0.63	13.52	66.67	33.33	8.81	91.19
E7, BS=1	HSA	0.36	24.97	69.78	30.22	4.93	95.07
E7, BS=1	RSA	0.69	29.88	77.37	22.63	4.93	95.07
E0, BS=128	HSA	0.46	10.88	78.13	21.88	8.81	91.19
E0, BS=128	RSA	0.69	24.41	69.31	30.69	8.81	91.19
E7, BS=128	HSA	0.56	43.1	80.04	19.96	4.93	95.07
E7, BS=128	RSA	0.7	52.76	79.63	20.37	4.93	95.07

(abbrev. ResNet). The results show that the runtime TOPS increases 1.25x-2.68x on three workloads for both small and large batch sizes when RSA is introduced. Moreover, the performance improves more on small batch size than large batch size. This implies that the workloads with lower operation intensity will gain more benefits from introducing RSA as the tensor unit. The performances of all three workloads are improved. This implies that RSA does not only benefit the operator of DConv2d but also helps the traditional Conv2d. This supports the observation in Sec. 4.3 that 128x128 systolic arrays are NOT always the best from another perspective. Table 5.4 further compares the runtime TOPS and the array utilization of DConv2d and Conv2d before and after replacing TPU-v2's HSA-based tensor units with the proposed RSAs. It gives a better illustration of the reduced proportion of DConv2d in the overall latency after applying the proposed RSAs.

5.6 Summary

This chapter proposes a reconfigurable systolic array (RSA) based CNN accelerator, which introduces quite small area and power overhead as well as minor changes on control logics while achieving both the high performance of the brawny designs and the high array utilization of the wimpy designs. The hardware modeling of the proposed RSA is built and integrated into NeuroMeter. The results show that the proposed RSA-based accelerator achieves 1.25x-2.68x performance gains on ResNet and EfficientNet compared with the TPU-v2 styled baseline.

Chapter 6

Cost-Aware Exploration for Chiplet-based Architecture

6.1 Introduction and Contribution Overview

This chapter is dedicated to build the analytical cost model for chiplet-based architecture and to conduct the cost-aware exploration to help the user to make the design decision on whether it will really make profit to adopt an advanced technology, especially the chiplet-related ones. The early-stage evaluation from the cost perspective is an essential supplement beyond the power, area, and timing discussed in Chp. 3-5.

Recently, the "chiplet"-based System-in-Package (SiP) becomes the potential replacement of the conventional System-on-Chip (SoC) which suffers from the increasing complexity and cost of new technology nodes. The SiP breaks a monolithic die based 2D SoC into multiple smaller pieces and keep them in the same package. These dies can be of different functionalities, hybrid technology nodes, and/or from multiple IP designers. Such a heterogeneous integration greatly reduces the design complexity of each die since more pre-built IP designs can be used.

Despite the advantage of simplifying the design for each individual die, SiP induces new design challenges on the inter-die connections. For example, Fig. 6.1 shows two major types of chiplet systems, i.e. (1) the interposer based 2.5D technology [24, 25] and (2) the organic substrate based multi-chip module (MCM) [27, 28, 100, 101]. In the first type, the interposer based 2.5D technology, an extra layer of interposer is introduced between the functional dies and the substrate for the inter-die connection; while the latter one, MCM, directly assembles the silicon dies onto the organic substrate, where the extra build-up layers are introduced for the die-to-die interconnections. The interposer can be implemented by different materials, including silicon [25], organic [24], etc. Different materials have different characteristics of performance and cost.

So far, the majority of the chiplet system research focuses on either demonstrating the prototypes to showcase the feasibility of the chiplet technologies [27, 28, 100, 101]; or studying the workload-aware dataflow [102], and the network-on-package [103, 104, 105] under the performance, energy, and thermal constraints. However, little attention has been given to the cost, which will become a critical factor when considering whether to adopt the chiplet and SiP as a mainstream design approach. All the potential advantages of the chiplet related technologies ultimately have to be translated into cost savings when evaluating a design strategy. For example, some interesting design questions for the chiplet-based SiP development at the early stage include:

(1) *How would the chiplet technology affect the cost terms?* For example, an extra process cost is incurred by a more complex placement-and-routing of the interposer or substrate for inter-chiplet connection; the bonding cost is also increased due to higher die count. However, the smaller silicon die size of each functional die results in a higher die yield than that of a larger 2D die and potentially reduces the cost.

(2) *How to break a monolithic system to a chiplet system?* If a monolithic system is re-designed in a chiplet way, it is not clear whether its scale is large enough to gain sufficient

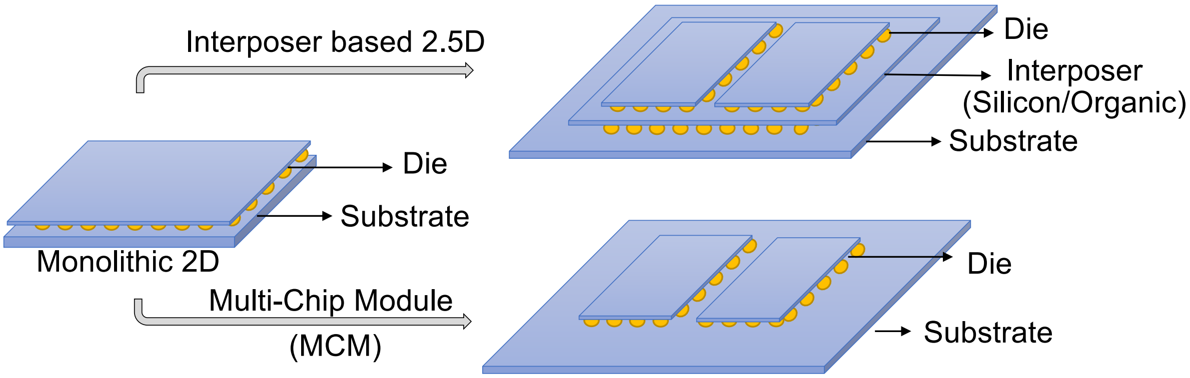


Figure 6.1: Illustration and comparison between different SiP interconnection technologies

die yield improvement, which will later be translated into cost benefits. Meanwhile, it is also unclear what is the optimal partitioning granularity and which packaging technology should be chosen from the cost perspective.

(3) *How to manage the heterogenous integration?* Different functional components have different paces of technology scaling. For example, AMD's EPYC2 CPU [27, 28] employs advanced technology nodes in computing logic and mature technology nodes in memory and I/O for cost efficiency. How to assign the technology nodes used in each die of the SiP system is worth of exploring for cost efficiency.

A straightforward approach to obtain the cost and answer these questions is to turn the chiplet design into product. However, such an approach may not be suitable for the emerging SiP chiplet systems. The cost of the design choices, including the inter-chiplet connection, the homogeneous/heterogeneous integration, makes up a large and complex design space that is not yet well explored. A prototype that is not well optimized towards the cost may mislead the design decision; and prototyping itself is costly. The ability to estimate the cost of the SiP at the early design stage is naturally invoked.

An analytical cost model can be the solution for cost estimation and guide the early-stage cost-aware design flow. Previously, cost model has been used in evaluating the TSV-based 3D architecture [56] or the silicon interposer based 2.5D integrated system [57, 58]. These works cannot be adopted in the chiplet-based SiP design because the heterogeneity, which is the key to the advantage of the chiplet-based SiP, never appears in previous technologies that only support homogeneous integration.

In this chapter, we make, to our best, the first attempt to build a cost model for chiplet-based SiP design space exploration. With the input of the system scale (e.g., the transistor count of compute die, memory cell count, and other statistics), the partition granularity (e.g., the die count), and the technology node, the proposed cost model will first translate the system scale into the area and the number of wiring layers. The cost breakdown of the die, the bonding, and the package is then calculated based on these data. With the collected data of transistor density as well as the wiring pitches, the proposed cost model is able to support the 2.5D Silicon interposer, the 2.5D organic interposer, and MCM under the technology nodes ranging from 28nm to 5nm to support the heterogeneous integration.

The proposed cost model induces several case studies; and several new observations are made on top of that. First, in most scenarios, the organic interposer and the MCM based SiP can obtain cost benefits while the manufacture cost of silicon interposer is as high as a silicon die of the same scale at a mature process. Second, a large number of chiplets and finer-grained partitions do not always lead to the cost efficiency, especially when the system scale is not large enough. Because the increased yield cannot amortize the growing bonding cost. Third, a hybrid system, which partitions the core logics and the I/O circuits into different dies, achieves prominent benefits on cost efficiency compared with the monolithic systems under both mature and advanced technology nodes. Because the extra bonding and package cost of the chiplet system is much smaller than the cost reduction from assigning the mature technology node to the I/O components. Note that the modeling results are subject to parameter changes because the chiplet-based

SiP technology keeps evolving and is still under development. Such changes will not affect the major contribution of the proposed cost model, which is firstly applicable to a heterogeneous chiplet-based SiP.

6.2 Analytical Cost Model for Chiplet System

Sec. 6.2.1 introduces how to translate the number of gate modules (e.g., the logic gate count, the memory cell count, or the transistor count) into higher-level information, especially the number of metal layers and build-up layers. Sec. 6.2.2 show hows to estimate the individual die cost. Sec. 6.2.3 and Sec. 6.2.4 discuss the bonding cost and the package cost of the 2.5D chiplet system respectively.

6.2.1 The Estimation of Metal Layer and Build-Up Layer

The Estimation of Metal Layer

According to the Rent's rule [106], a theoretical upper bound of the average wire length given the number of gates and the routing efficiency can be expressed in Eq. 6.1:

$$\bar{L} = \frac{2}{9}(\frac{1-4^{p-1}}{1-N_g^{p-1}})(7\frac{N_g^{p-0.5}-1}{4^{p-0.5}-1} - \frac{N_g^{p-1.5}}{1-4^{p-1.5}}) \tag{6.1}$$

where N_g is the number of the gate modules; p is the Rent's exponent which represents the routing efficiency; and $\bar{L}$ is the average wire length in the unit of gate pitch.

Given the average wire length, the number of metal layers n_{metal} is estimated as in Eq. 6.2:

$$n_{metal} = \frac{f.o.\bar{L}w}{\eta}\sqrt{\frac{N_g}{A_{die}}} \tag{6.2}$$

where $f.o.$ is the gate fanout; $\bar{L}$ is the average wire length in Eq. 6.1; w is the wire pitch;

Table 6.1: Numbers of metal layers under different technology nodes for different chip scales. Parameters: The Rent's exponent $p = 0.6$, the number of gate modules N_g = (# of Transistors)/(4 Million), the fanout $f.o. = 4$, the wire pitch $w = 3.6\lambda$, the utilization efficiency of metal layers $\eta = 0.1$.

TechNode λ (nm)	Tx Density (MTx/mm^2)	# of Transistors					
		1M	10M	100M	1B	10B	100B
28	2.93	2	4	8	13	19	27
20	4.89	2	4	7	12	18	25
16	6.86	2	4	7	11	17	24
12	10.63	1	4	7	10	16	22
10	14.02	1	4	6	10	15	22
7	24.11	1	3	6	9	14	20
5	42.83	1	3	6	9	13	19

η is the utilization efficiency of the metal layers; N_g is the number of the gate modules; and A_{die} is the die area.

Leveraging Eq. 6.1 and Eq. 6.2, we estimate the numbers of metal layers under different technology nodes. Table 6.1 shows the estimated numbers of metal layers from 28nm to 5nm process. We collect the transistor density from WikiChip [107] and interpolate the missing technology nodes according to ITRS' scaling down ratio [108]. We use the number of the transistors as the indicator of the chip scale.

The Estimation of Build-Up Layer

In a monolithic-die based 2D system, all the signal interconnections are routed on the metal layers of the functional die. In the 2.5D chiplet system, the wiring on the metal layers of the Network-on-Chip (NoC) is partially offloaded to the build-up layers of the Network-on-Package (NoP) on the interposer or the substrate. This requires calculating the number of build-up layers when estimating the cost of the chiplet system. Considering that the pins of the inter-chiplet connections can only be placed at the edge of the chiplet, the number of the build-up layer is estimated in Eq. 6.3:

$$N_{build\_up\_layer} = \frac{pitch \cdot N_{pin} \cdot \bar{R}_{wire}}{\eta A_{route}} \tag{6.3}$$

where $pitch$ is the wire pitch of the build-up layers; N_{pin} and $\bar{R}_{wire}$ are the number of the microbumps and the average wire length, respectively, determined by the chiplet system; A_{route} here is the total routable area on the interposer or the substrate.

6.2.2 Manufacture Cost Model of An Individual Die/Interposer

The manufacture cost model of an individual die can either be used to estimate the manufacture cost of a monolithic 2D chip directly; or later be used as a component to calculate the overall manufacture cost of a 2.5D chiplet system. We start from the manufacture cost of a single functional die, then discuss the silicon interposer and the organic interposer.

Manufacture Cost of An Individual Die

Eq. 6.4 shows that the manufacture cost of a functional die (C_{die}) is determined by the wafer cost (C_{wafer}) and the number of dies per wafer (N_{die}). Here, the cost of the wafer (C_{wafer}) is determined by the wafer's technology node (TechNode) and the number of the wafer's metal layer (N_{metal}), i.e. $C_{wafer} = C_{wafer}(\text{TechNode}) + N_{metal} \cdot C_{metal}$, where C_{metal} is the cost per metal layer. The number of dies per wafer (N_{die}) calculates how many rectangular dies with the size A_{die} can be obtained from a round wafer with the diameter Φ_{wafer}.

$$C_{die} = \frac{C_{wafer}}{N_{die}} = \frac{C_{wafer}(\text{TechNode}, N_{metal})}{\frac{\pi \times (\Phi_{wafer}/2)^2}{A_{die}} - \frac{\pi \times \Phi_{wafer}}{\sqrt{2 \times A_{die}}}} \tag{6.4}$$

Besides the raw manufacture cost per die, the yield of the die should be considered as well when deploying the die into a monolithic 2D or interposer based 2.5D chiplet system. Using the negative binomial yield model [109], Eq. 6.5 shows the yield of a functional die (Y_{die}). Here Y_{wafer} is the yield of the wafer. A_{die} is the area of the die. α is a process dependent clustering parameter, frequently between 1 (high defect clustering) and 3 (moderate defect clustering). And D_0 is the defect density. The defect density of a new process is initially high, but it decreases generally by 2-5x for historical technologies as the process gets mature.

$$Y_{die} = Y_{wafer} \times (1 + \frac{A_{die} D_0}{\alpha})^{-\alpha} \tag{6.5}$$

Manufacture Cost of An Interposer

The silicon interposer can be regarded as silicon die at a relatively mature (or even out-of-date) technology node with TSVs and several layers of passive[1] metal interconnection on it. Without further clarification, the case studies in Sec. 6.3 assume that the silicon interposer is made up from the 300mm wafer with 1P/4Cu layers at the 65nm technology node. Eq. 6.6 shows the cost (C_{sil_int}) and yield (Y_{sil_int}) of a silicon interposer.

$$\begin{aligned} C_{sil\_int} &= \frac{C_{wafer}}{N_{sil\_int}} = \frac{C_{wafer}}{\frac{\pi \times (\Phi_{wafer}/2)^2}{A_{sil\_int}} - \frac{\pi \times \Phi_{wafer}}{\sqrt{2 \times A_{sil\_int}}}} \\ Y_{sil\_int} &= Y_{wafer} \times (1 + \frac{A_{sil\_int} D_0}{\alpha})^{-\alpha} \end{aligned} \tag{6.6}$$

Here, C_{wafer} and N_{sil_int} are the wafer cost and the number of partitioned silicon interposers from a wafer. Φ_{wafer} and A_{sil_int} are the wafer diameter and the area per silicon interposer. Y_{wafer}, D_0, α are the yield, defect density, and defect clustering parameter

---

[1] Here, only the passive interposer is considered. The active interposer, which allows to place logics on it, is out of discussion of this chapter.

of the interposer wafer. The area of the silicon interposer, A_{sil_int}, can be calculated as the sum of the area of all functional dies and overhead of die-to-die interconnection, i.e., $A_{sil\_int} = \sum A_{die} \times (1 + R_{overhead})$, where $R_{overhead}$ represents the area overhead ratio.

Differently, the organic interposer is built in a panel form (i.e., a large square or rectangle), rather than a wafer form (i.e., a round plate). Eq. 6.7. shows the cost (C_{org_int}) and yield (Y_{org_int}) of an organic interposer.

$$\begin{aligned} C_{org\_int} &= \frac{C_{panel}}{N_{org\_int}} = \frac{C_{panel}}{A_{panel}/A_{org\_int}} \\ Y_{org\_int} &= Y_{panel} \times (1 + \frac{A_{org\_int} D_0}{\alpha})^{-\alpha} \end{aligned} \tag{6.7}$$

Here, C_{panel} and N_{org_int} are the panel cost and the number of organic interposers obtained from a panel. A_{panel} and A_{org_int} are the area of the panel and the organic interposer, respectively. Y_{panel}, D_0, α are the yield, defect density, and defect clustering parameter of the interposer panel, respectively. They are collected from ICKnowledge [110] and are determined by the technology node.

6.2.3 Bonding Cost Model

The unpackaged interposer-based chiplet system is made up of n functional dies on top and one interposer die at the bottom; while the MCM based chiplet system directly deploys n functional dies onto the organic substrate. The bonding cost of the unpackaged chiplet system is calculated by Eq. 6.8 and Eq. 6.9, which represent the interposer-based chiplet system ($C_{int\_2.5D}$) and the MCM-based chiplet system ($C_{MCM\_2.5D}$), respectively.

$$C_{int\_2.5D} = \frac{\frac{C_{int}}{Y_{int}} + \sum_{i=1}^{n} (\frac{C_{die}(i)}{Y_{die}(i)} + C_{bond}(i))}{\prod_{i=2}^{n} Y_{bond}(i)} \tag{6.8}$$

$$C_{MCM\_2.5D} = \frac{\sum_{i=1}^{n} (\frac{C_{die}(i)}{Y_{die}(i)} + C_{bond}(i))}{\prod_{i=2}^{n} Y_{bond}(i)} \tag{6.9}$$

Here n is the number of function dies. $C_{die}(i)$, and $Y_{die}(i)$ are the manufacturing cost and yield for the i^{th} functional die (Eq. 6.4 and Eq. 6.5). $C_{bond}(i)$ and $Y_{bond}(i)$ are the bonding cost and yield for the i^{th} die. In most cases each functional die is assumed to have the same bonding cost and yield. C_{int} and Y_{int} are the manufacturing cost and the yield for the interposer (Eq. 6.6 and Eq. 6.7). Comparing Eq. 6.8 and Eq. 6.9, the cost of the interposer based chiplet system has an extra term of $\frac{C_{int}}{Y_{int}}$, which depends on the material of the interposer.

6.2.4 Package Cost Model

The package cost depends on the type of package. Flip chip based organic substrate can be used in both interposer based and MCM based chiplet systems. For simplification, the flip chip based organic substrate is used to showcase the package cost model. Similar cost models can also be used in other package types. In addition to the package type, the package cost is also determined by three factors, i.e., the package area, the layer number of package (#core and #buildup), and the pin count.

1) *The area of organic substrate* (A_{sub}): A_{sub} is the interposer area or the sum of the chiplet area, together with microbump, C4 bump, and other overheads.

2) *The layer number of organic substrate*: The organic substrate is built up in a sandwich structure. The core layers in the structure provide mechanical strength in the middle and the build-up layers are for wire routing on the top and the bottom. In the interposer-based chiplet system, the inter-chiplet connections are routed on the interposer, leaving only the routing to the external I/O and power supply on the substrate. While in the MCM system, the inter-chiplet connections are routed on the substrate instead. This

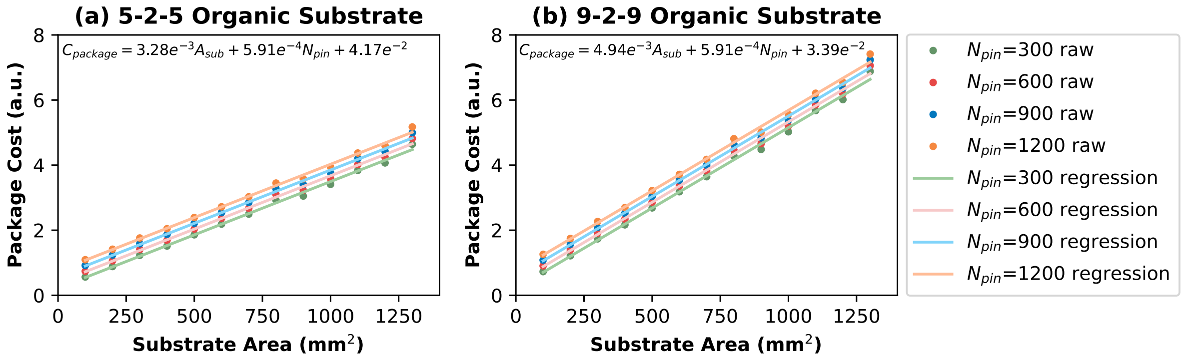


Figure 6.2: Package cost of organic substrates

makes the substrate routing of MCM much more complex than the former. And the routing complexity is further reflected on the number of build-up layers, as discussed in Sec.6.2.1. Referring to the published chips and prototypes [24, 26], the number of core layers is usually set as 1 or 2, and the number of the build-up layers is between 5 and 11.

3) *The pin count* (N_{pin}): N_{pin} can be estimated by the Rent's Rule [111] and it will affect the overall packaging cost. In this chapter, we directly take the pin count as an input of the package cost. Estimating the pin count according to the die characteristics is outside the scope of this chapter.

Given the numbers of core layers and the build-up layers, we collect the package cost under different combinations of substrate area and pin count from ICKnowledge [110]. With the collected data, we derive an empirical function of the package cost (C_P) with respect to the combination of the substrate area (A_{sub}) and the number of pins (N_{pin}) in Eq.6.10. μ_0, μ_1, μ_2 are the regression parameters determined by the numbers of core layers and the build-up layers.

$$C_P = \mu_0 A_{sub} + \mu_1 N_{pin} + \mu_2 \tag{6.10}$$

Fig. 6.2 shows the regressive package cost model of the organic substrates of two different configurations (a) 2 core layers and 5 build-up layers; and (b) 2 core layers and 9 build-up layers. The result shows a satisfying linearity of the developed regressive model. We will use this regressive package model in the follow-up case studies.

6.3 Case Study

In this section, we leverage the developed cost model to conduct a series of case studies and showcase the cost characteristics of the chiplet based architecture under different scenarios. We start from a homogeneous multi-chiplet system which evenly partitions the monolithic system into multiple chiplets (Sec. 6.3.1). We then study a heterogeneous multi-chiplet system with HBM stacks (Sec. 6.3.2). Finally, we explore a heterogeneous multi-chiplet system which partitions the core components and IO components onto different chiplets under different processes (Sec. 6.3.3).

6.3.1 Homogeneous Chiplet System

The homogeneous chiplet system partitions the monolithic 2D system into multi dies evenly. The homogeneous partition has a double-folded effect on cost. First, the partition leads to a higher yield due to a smaller area per die and potentially reduces the cost. Second, it introduces the extra cost overhead of chiplet bonding, silicon/organic interposer, and/or more complex packaging. The fold that finally wins out depends on the configuration of the target system. A quantitative analysis is required to determine the scale and the type of the chiplet system when it can achieve higher cost efficiency. The improvement of the cost efficiency is calculated in Eq.6.11 and used to measure how the cost changes after refactoring the monolithic 2D system into a functional-equivalent chiplet system. When $\Delta\text{CostEff} > 0$, it implies that the cost efficiency of the chiplet system is better than that of the 2D monolithic counterpart.

$$\Delta\text{CostEff} = \frac{\text{Cost(Monolithic)} - \text{Cost(Chiplet)}}{\text{Cost(Monolithic)}} \times 100\% \tag{6.11}$$

Fig. 6.3 shows the relative cost efficiency of the chiplet system under different scenarios. The four values of area (i.e., 200mm^2, 400mm^2, 800mm^2, 1600mm^2) exemplify the system scales of small, medium, large, and extremely large. Three different technology nodes (5nm, 7nm, and 16nm) are covered. We explore the three different package technologies, i.e., silicon interposer, organic interposer, and MCM, under the numbers of chiplets of 2, 4, and 8. The wafer cost and the defect ratio of different technology nodes are collected from ICKnowledge [110]. We observe that:

First, silicon interposer based chiplet system rarely gets cost benefits unless in the large scaled system under the most advanced process ($\geq$ 800mm^2, 5nm); while the organic interposer based chiplet system and MCM are able to get cost benefits in most scenarios. Moreover, the cost efficiency of organic interposer and MCM is quantitatively similar to each other in the explored design scenarios. This implies that although MCM has no interposer layer, it requires a complex substrate with higher cost to route the die-to-die interconnections. Further, we explore the switching points: the system scale that can get cost benefits from the chiplet design. As listed in Table 6.2, we find that the switching points of the MCM based chiplet system are always smaller than that of the organic interposer based chiplet system under the different technology nodes. Especially, the switching points of MCM are always smaller than 200mm^2. This implies that the MCM is more likely to be adopted in the small scaled systems.

Second, introducing a larger number of chiplets and finer-grained partition does not always lead to the cost efficiency, especially when the system scale is not very large. Take 7nm organic interposer-based SiP as an example. When the system scale is 200mm^2,

Table 6.2: Switching points of organic interposer/MCM based chiplet systems under different technology nodes

Tech Node	Chiplet Type	7nm	10nm	12nm	16nm	20nm	28nm
2D Area (mm^2)	Org 2.5D	178	191	264	279	313	479
	MCM	119	120	126	128	131	149
Tx Count (Billion)	Org2.5D	17.17	10.71	11.22	7.66	6.12	5.62
	MCM	11.48	6.73	5.36	3.51	2.56	1.75

the dual-chiplet system wins out in terms of cost efficiency compared with other finer-grained partitioned design options. When the system scale gets larger, the quad-chiplet system wins out. We further explored the relationship between the cost and the scale of the chiplet system under different bonding yields. The three subfigures of Fig. 6.4 respectively show the optimal numbers of chiplets of different scales under the bonding yields of 0.9, 0.95, and 0,99. The results show that: (1) with a higher bonding yield, the chiplet system starts to get a better cost efficiency than the monolithic counterpart at a smaller system scale (or smaller die area); (2) with a higher bonding yield, it is more likely that the SiP with a large number of small dies can get lower cost than the SiP with a small number of big dies.

6.3.2 Heterogeneous Chiplet System with HBM Stacks

The heterogeneous chiplet system with high bandwidth requirement integrates the core dies and the HBM stacks onto the silicon interposer or the organic interposer. Each HBM stack is made up of a base die at the bottom and several layers of memory dies atop [112]. For the base die, the area of a 1024-bit signal interface is mainly determined by the pitch width of the microbumps. The pitch width is set as 45μm for the silicon interposer and 110μm for the organic interposer. MCM is excluded in this case study because the C4 bump pitch is too large to place the whole signal and I/O interface under

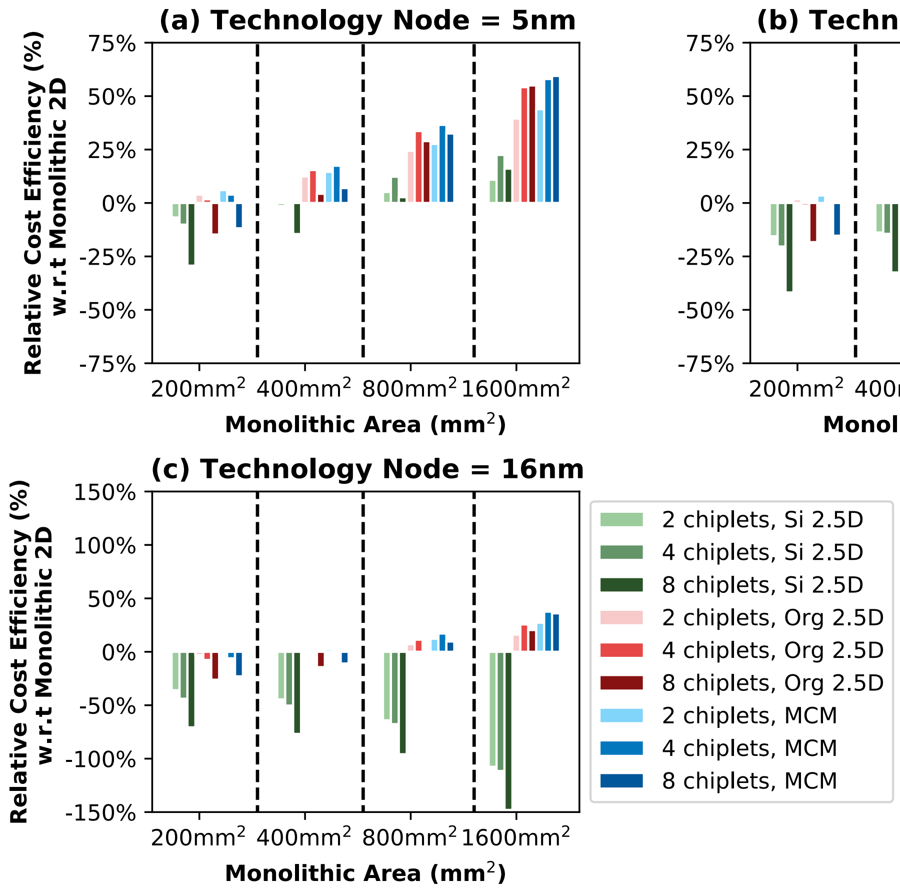


Figure 6.3: Cost efficiency improvement of chiplet systems under different technology nodes

the area constraint of existing HBMs[2].

This case study aims to study the extra cost introduced by HBM stacks. Fig. 6.5 visualizes the relative cost breakdown of interposer[3] and bonding, where the manufacture cost of core dies as the 100% base unit. The three subfigures show the system scales of 200mm^2, 400mm^2, and 800mm^2 under 7nm process respectively. We observe that the organic interposer based chiplet system introduces less than 50% overhead for HBM stacks and the bonding yield takes the majority of the overhead. While for the silicon

---

[2] According to public data [112], HBM1 takes the area of 5.48mm × 7.29mm. This implies that when the bump pitch width is larger than 197μm, the 1024-bit I/O interface will use up the overall area budget of HBM stack.

[3] The cost of interposer considers the area of HBM stacks. Yet the cost of memory stacks is not calculated due to the lack of data.

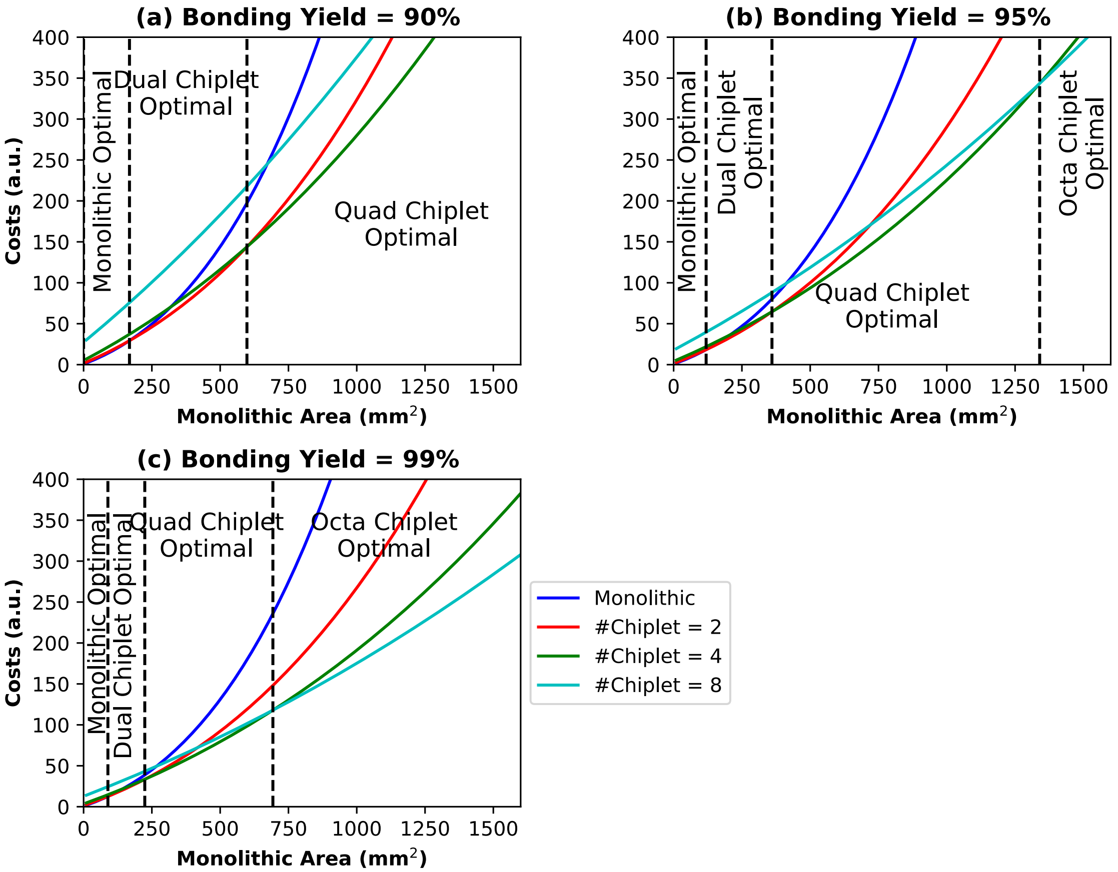


Figure 6.4: Cost of organic interposer based homogeneous chiplet vs area of monolithic chip under of corresponding bonding yield and 7nm process

interposer based chiplet system, the relative cost overhead is much larger.

6.3.3 Heterogeneous Chiplet System with Hybrid Processes

Considering the technology scaling of I/O and other peripheral circuits are much slower than that of the compute logic and the on-chip memory [113], the cost efficiency may potentially increase if different components are assigned to different dies and are implemented in different technology nodes. Inspired by AMD's EYPC2 CPU [27, 28], this case study explores the heterogeneous architecture where the cores and the IO peripheral circuits are split into different dies and implemented in different technology nodes. As shown in Fig. 6.6, we study the MCM based chiplet system under different system scales and different I/O proportion. To quantitatively explore the cost efficiency of the heterogeneous chiplet system, we compare the cost of a 7nm monolithic system, a 12nm monolithic system, and three hybrid systems with different numbers of core dies where the core dies and the I/O die are respectively assumed at 7nm and 12nm technology nodes. The three subfigures of Fig. 6.6 respectively show the circumstances when the numbers of transistors on the core die are 5 billion, 10 billion, and 50 billion. And each subfigure includes the proportions of I/O circuits in the range of {30%, 40%, 50%}.

We find that the hybrid system which partitions the core logics and the I/O circuits into different dies achieves salient benefits on cost efficiency compared with the monolithic systems under both the mature and advanced technology nodes. Moreover, as the scale of the chiplet system gets larger, both the cost efficiency of the hybrid system and the optimal number of core dies increase. For the three different system scales in Fig. 6.6, the optimal chiplet system respectively achieves 34%, 48%, and 77% cost efficiency improvement compared with the 7nm monolithic counterpart, and the optimal numbers of core dies are respectively 2, 4, and 8.

6.4 Summary

This chapter builds an analytical cost model for the 2.5D chiplet system under various interconnection options and technology nodes. A series of case studies are conducted to explore the cost characteristics under both homogeneous and heterogeneous scenarios. By analyzing the case study results, several observations are made on the interposer selection, design partition granularity, and hybrid technology node adoption for the cost-efficient chiplet-based SiP design.

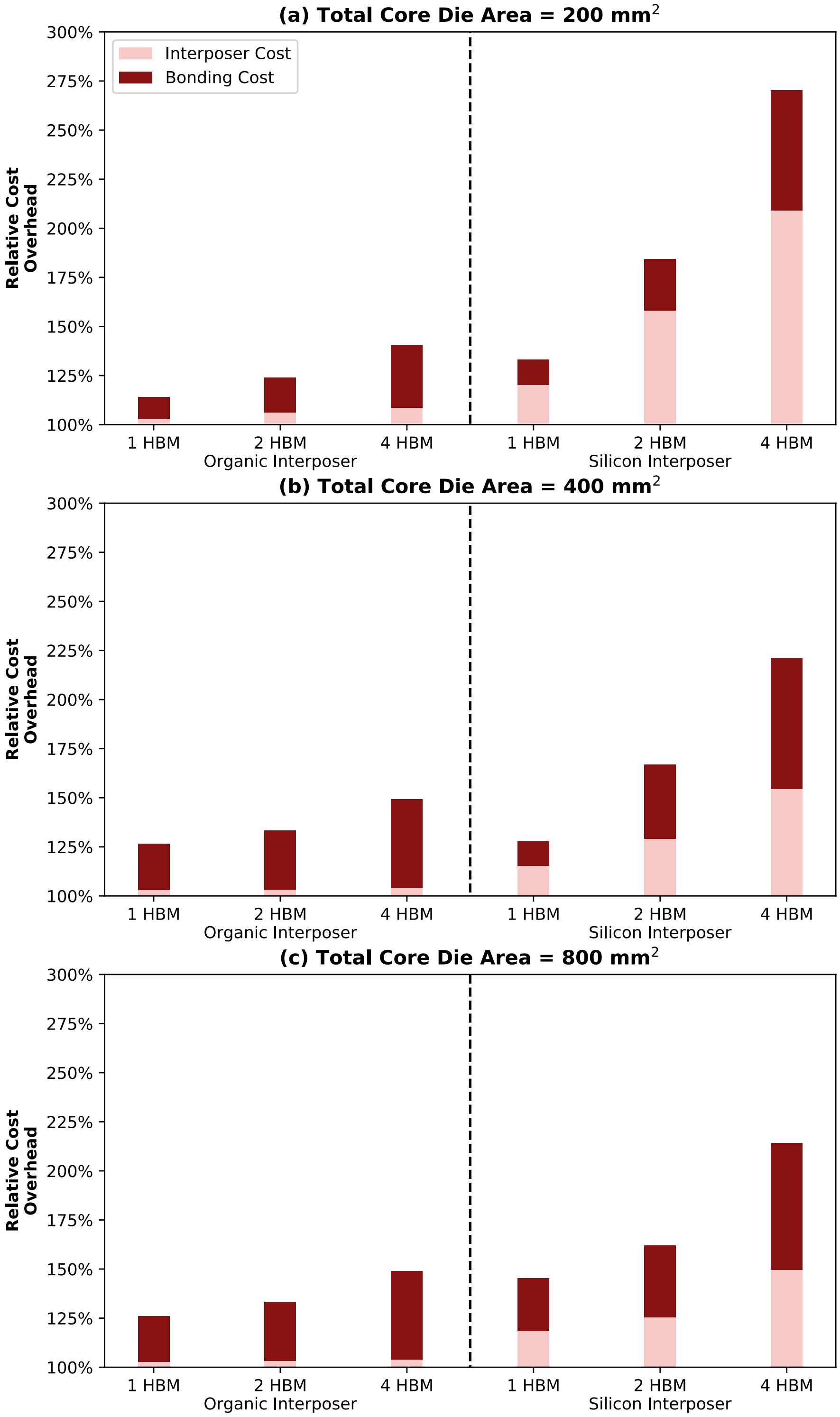


Figure 6.5: Relative cost overhead of heterogeneous chiplet system with HBM stacks of different scales under 7nm process

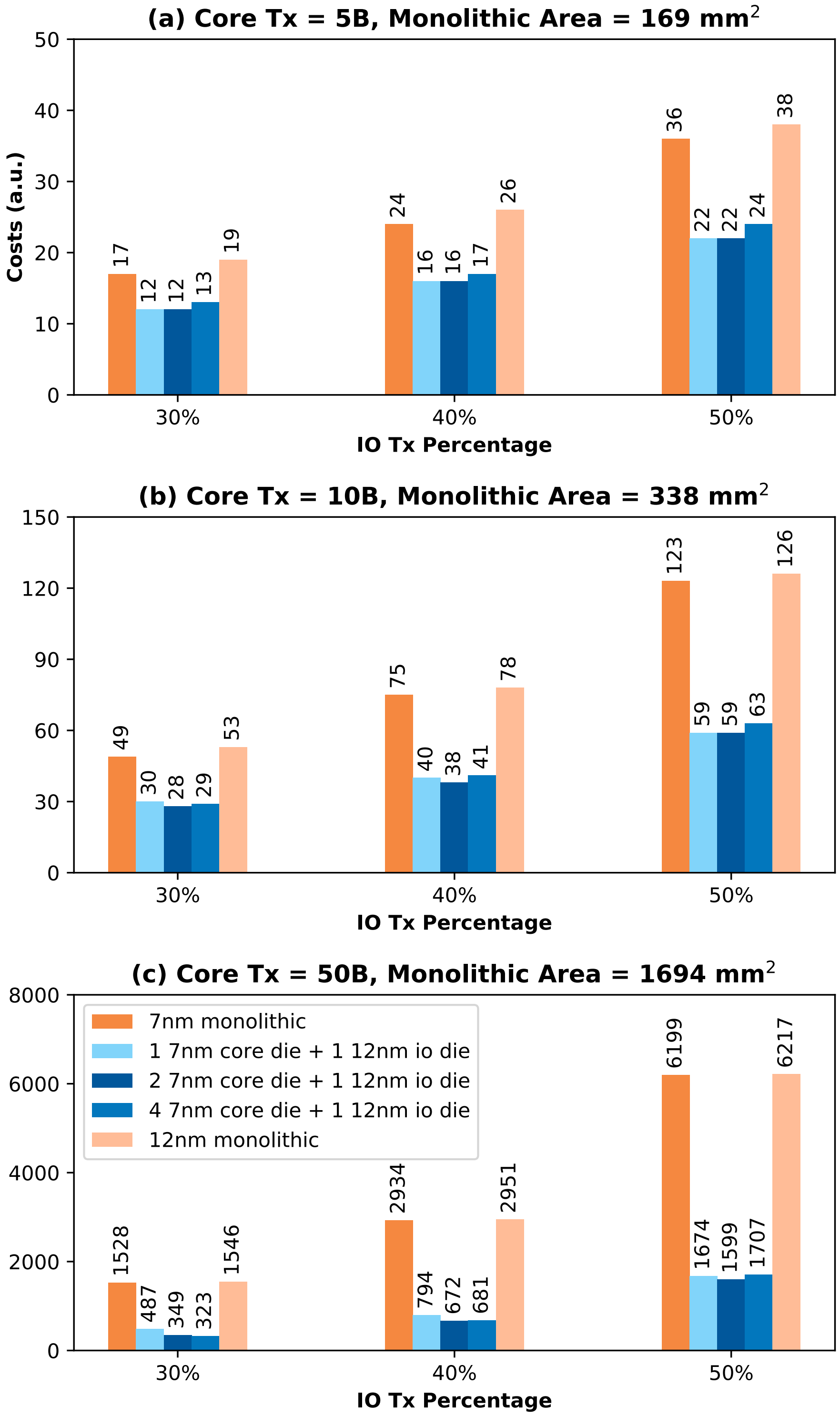


Figure 6.6: Cost of MCM based heterogeneous chiplet system with hybrid processes under different system scales

Chapter 7

Summary

7.1 Thesis Contribution

This thesis research addresses the urge for the early-staged hardware modeling for domain-specific accelerator design; and showcases examples of hardware-and-software co-design space for machine learning (ML) accelerators to address a broad spectrum of workload characteristics. The thesis research helps explore the co-design workflow and automation toolset to increase the productivity on domain-specific accelerator design. The contributions are listed as follows:

1. **Analytical Hardware Modeling** This thesis develops NeuroMeter, an integrated power, area, and timing modeling framework for ML accelerators. With the integrated electrical characteristics models (e.g., resistance-capacitance path model and Elmore delay model) of typical circuit structures in it, NeuroMeter, the proposed modeling tool, allows the user to input the high-level architectural specifications and technology enablers alone. The tool itself formulates the possible circuit representations of ML accelerators, explores the design space internally and transparently, and generates a fast and accurate estimation on power, area, and chip timing. It enables the runtime analysis

of system-level performance and efficiency when the runtime activity factors are provided at the pre-RTL design stage.

2. Cost-Aware Analysis on Chiplet and Heterogeneous Integration This thesis develops an analytical cost model that can estimate the cost of the 2.5D chiplet-based System-in-Package (SiP) systems under various interconnection options and technology nodes. The cost modeling is used as a good supplement beyond the analytical modeling of power, area, and timing. It helps the user figure out whether SiP's design flexibility enabled by various inter-chiplet connection and heterogeneous integration can be translated into cost efficiency at the early design stage. Leveraging the proposed analytical cost model, a series of case studies is conducted to explore the cost characteristics of the 2.5D chiplet-based SiP system in different scenarios. By analyzing the case study results, several observations are made on the interposer selection, design partition granularity, and technology node adoption for cost-efficient chiplet-based SiP design.

3. Efficient Architecture Exploration By combining the power, area, and timing results of NeuroMeter with performance simulation, this thesis explores the manycore ML accelerator design in different scenarios; and proposes new architecture, i.e.

First, the brawny vs wimpy study shows that brawny designs with 64x64 systolic arrays are the most performant and efficient for inference tasks in the 28nm datacenter architectural space with a 500mm^2 die area budget. The exploration also reveals important tradeoffs between performance and efficiency. For datacenter accelerators with low batch inference, a small ($\sim$16%) sacrifice of performance can lead to more than a 2x efficiency improvement (in achieved TOPS/TCO).

Second, this thesis also conducts a case study on energy efficiency (TOPS/Watt) implications of sparsity on different ML accelerators to showcase NeuroMeter's capability to model a wide range of accelerator architectures. The results show that despite its relatively low energy efficiency, it is easier for wimpier accelerator architectures to benefit from sparsity processing.

Third, this thesis proposes a reconfigurable systolic array (RSA) based CNN accelerator, which introduces quite small area and power overhead as well as minor changes on control logics. The internal reconfigurability of the proposed RSA enables both the high performance of small numbers of brawny systolic arrays and the high utilization of large numbers of wimpy systolic arrays. Especially, it increases the utilization especially for the operators of depthwise convolution with low operational intensity. The result show that the performance of ResNet and EfficientNet achieves 1.25x-2.68x gain on the proposed RSA based accelerator compared with the TPU-v2 styled baseline.

7.2 Future Work

The experiences from my thesis research naturally lead to a wide range of future directions. We listed them as below from the following two aspects:

1. Analytical Hardware Modeling With the technology node scaling down, the circuit-level electrical characteristics models (e.g., resistance-capacitance path model and Elmore's model) for FinFET and gate-all-around (GAA) nanosheet are required if the advanced technology nodes ($\leq$7nm) are planned to support in the future hardware modeling tools. With the slowdown of Moore's law, the 2.5D chiplet integration and fine-grained 3D integration have attracted more attention from both industry and academia. Beyond the single-chip level power, area, and timing (PAT) modeling tool proposed in this thesis, future work may extend it to a chiplet system level PAT modeling framework which supports 2.5D (or 3D) integration with the inter-chiplet (or inter-tier) connections with resistance-inductance-capacitance (RLC) path modeling as well as the interposers and the substrates.

On top of PAT analysis, future work will explore the integration cost as well to evaluate whether the advantages of 2.5D or 3D integration ultimately can be translated into cost feasibility at the early design stage. Besides the manufacture cost and the package cost discussed in the analytical cost model in this thesis, the testing cost and the cooling cost also play an important role in the early-stage design decision. With the consideration of 2.5D chiplet system or 3D integration, the thermal issue may become critical compared with the 2D monolithic system. Besides power, area, timing, and cost, future work may consider the thermal issue collaboratively at the early design stage.

2. Architectural Exploration and Hardware/Software Co-Design Looking forward, the borderline between the architectural innovation and the software optimization is becoming blurred. There is a large space to explore for the cross-stack optimization between hardware, software, and algorithm. An automatic and ML-guided workflow is in need for more efficient hardware/software co-optimization and exploration. A high-performance compiler framework and a unified intermediate representation (IR) are in need to serve as a bridge between the continuously evolving algorithms and the "Cambrian explosion" [61] of domain-specific hardware platforms. The innovations inspired from reconfigurability at the architectural level and beyond may be further applied on even broader ML workloads and software optimizations, including but not limited to multi-precision [114], structured or unstructured pruning [115, 116], dynamic sparsity [117, 118, 119], and runtime optimizations [120, 121].

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[3] M. Tan and Q. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, *Advances in neural information processing systems* **25** (2012).

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, *arXiv preprint arXiv:2004.10934* (2020).

[6] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, *Vectornet: Encoding hd maps and agent dynamics from vectorized representation*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11525–11533, 2020.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, *Advances in neural information processing systems* **30** (2017).

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, *arXiv preprint arXiv:1810.04805* (2018).

[9] H. Gao, Z. Wang, and S. Ji, *Large-scale learnable graph convolutional networks*, in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1416–1424, 2018.

[10] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, *Graph convolutional neural networks for web-scale recommender systems*, in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

[11] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, *Highly accurate protein structure prediction with alphafold*, *Nature* **596** (2021), no. 7873 583–589.

[12] O. Lange and L. Perez, "Traffic prediction with advanced graph neural networks." `https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks`. Deepmind.

[13] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, *In-datacenter performance analysis of a tensor processing unit*, in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017.

[14] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson, *A domain-specific supercomputer for training deep neural networks*, *Communications of the ACM* **63** (2020), no. 7 67–78.

[15] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, *The design process for google's training chips: Tpuv2 and tpuv3*, *IEEE Micro* **41** (2021), no. 2 56–63.

[16] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, *Ten lessons from three generations shaped google's tpuv4i: Industrial product*, in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, IEEE, 2021.

[17] M. Anderson, B. Chen, S. Chen, S. Deng, J. Fix, M. Gschwind, A. Kalaiah, C. Kim, J. Lee, J. Liang, H. Liu, Y. Lu, J. Montgomery, A. Moorthy, S. Nadathur, S. Naghshineh, A. Nayak, J. Park, C. Petersen, M. Schatz, N. Sundaram, B. Tang, P. Tang, A. Yang, J. Yu, H. Yuen, Y. Zhang, A. Anbudurai, V. Balan, H. Bojja, J. Boyd, M. Breitbach, C. Caldato, A. Calvo, G. Catron, S. Chandwani, P. Christeas, B. Cottel, B. Coutinho, A. Dalli, A. Dhanotia, O. Duncan, R. Dzhabarov, S. Elmir, C. Fu, W. Fu, M. Fulthorp, A. Gangidi, N. Gibson, S. Gordon, B. P. Hernandez, D. Ho, Y.-C. Huang, O. Johansson, S. Juluri, S. Kanaujia, M. Kesarkar, J. Killinger, B. Kim, R. Kulkarni, M. Lele, H. Li, H. Li, Y. Li, C. Liu, J. Liu, B. Maher, C. Mallipedi, S. Mangla, K. K. Matam, J. Mehta, S. Mehta, C. Mitchell, B. Muthiah, N. Nagarkatte, A. Narasimha, B. Nguyen, T. Ortiz, S. Padmanabha, D. Pan, A. Poojary, Y. C. Qi, O. Raginel, D. Rajagopal, T. Rice, C. Ross, N. Rotem, S. Russ, K. Shah, B. Shan, H. Shen, P. Shetty, K. Skandakumaran, K. Srinivasan, R. Sumbaly, M. Tauberg, M. Tzur, S. Verma, H. Wang, M. Wang, B. Wei, A. Xia, C. Xu, M. Yang, K. Zhang, R. Zhang, M. Zhao, W. Zhao, R. Zhu, A. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, *First-generation inference accelerator deployment at facebook*, *arXiv preprint arXiv:2107.04140* (2021).

[18] K. Lee, V. Rao, and W. Arnold, "Accelerating facebook's infrastructure with application-specific hardware." `https://engineering.fb.com/2019/03/14/data-center-engineering/accelerating-infrastructure`. Facebook Engineering Blog.

[19] Y. Jiao, L. Han, and X. Long, *Hanguang 800 npu–the ultimate ai inference solution for data centers*, in *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1–29, IEEE Computer Society, 2020.

[20] "Edge tpu: Google's purpose-built asic designed to run inference at the edge." `https://cloud.google.com/edge-tpu`.

[21] "Advanced ai embedded systems – nvidia jetson: The ai platform for autonomous machines." `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/`.

[22] J. Choquette, O. Giroux, and D. Foley, *Volta: Performance and programmability*, *Ieee Micro* **38** (2018), no. 2 42–52.

[23] J. Choquette, E. Lee, R. Krashinsky, V. Balan, and B. Khailany, *3.2 the a100 datacenter gpu and ampere architecture*, in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 48–50, IEEE, 2021.

[24] M. Ishida, *Apx (advanced package x)-advanced organic technology for 2.5 d interposer*, in *2014 CPMT Seminar, Latest Advances in Organic Interposers*, pp. 27–30, 2014.

[25] “Chip-on-wafer-on-substrate (cowos) - tsmc.” `https://en.wikichip.org/wiki/tsmc/cowos`. Accessed: August 23, 2020.

[26] L. Li, P. Chia, P. Ton, M. Nagar, S. Patil, J. Xue, J. Delacruz, M. Voicu, J. Hellings, B. Isaacson, M. Coor, and R. Havens, *3d sip with organic interposer for asic and memory integration*, in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, pp. 1445–1450, IEEE, 2016.

[27] S. Naffziger, K. Lepak, M. Paraschou, and M. Subramony, *2.2 amd chiplet architecture for high-performance server and desktop products*, in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 44–45, IEEE, 2020.

[28] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, *Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families: Industrial product*, in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 57–70, IEEE, 2021.

[29] T. Tang, S. Li, L. Nai, N. Jouppi, and Y. Xie, *Neurometer: An integrated power, area, and timing modeling framework for machine learning accelerators industry track paper*, in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 841–853, IEEE, 2021.

[30] W. C. Elmore, *The transient response of damped linear networks with particular regard to wideband amplifiers*, *Journal of applied physics* **19** (1948), no. 1 55–63.

[31] T. Tang and Y. Xie, *Cost-aware exploration for chiplet-based architec- ture with advanced packaging technologies*, in *The 1st International Workshop on High Performance Chiplet and Interconnect Architectures (HiPChips), in conjunction with the 49th International Symposium on Computer Architecture (ISCA)*, 2022.

[32] D. Hisamoto, W.-C. Lee, J. Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, T.-J. King, J. Bokor, and C. Hu, *Finfet-a self-aligned double-gate mosfet scalable to 20 nm*, *IEEE transactions on electron devices* **47** (2000), no. 12 2320–2325.

[33] S. Barraud, B. Previtali, C. Vizioz, J. Hartmann, J. Sturm, J. Lassarre, C. Perrot, P. Rodriguez, V. Loup, A. Magalhaes-Lucas, R. Kies, G. Romano, M. Cassé, N. Bernier, A. Jannaud, A. Grenier, and F. Andrieu., *7-levels-stacked nanosheet gaa transistors for high performance computing*, in *2020 IEEE Symposium on VLSI Technology*, pp. 1–2, IEEE, 2020.

[34] T. Austin, E. Larson, and D. Ernst, *Simplescalar: An infrastructure for computer system modeling*, *Computer* **35** (2002), no. 2 59–67.

[35] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and F. Zulian, *The gem5 simulator: Version 20.0+*, *arXiv preprint arXiv:2007.03152* (2020).

[36] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, *Analyzing cuda workloads using a detailed gpu simulator*, in *2009 IEEE international symposium on performance analysis of systems and software*, pp. 163–174, IEEE, 2009.

[37] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, *Accel-sim: An extensible simulation framework for validated gpu modeling*, in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 473–486, IEEE, 2020.

[38] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao, H. Barclay, A. K. Ziabari, Z. Chen, R. Ubal, J. L. Abellán, J. Kim, A. Joshi, and D. Kaeli, *Mgpusim: enabling multi-gpu performance modeling and optimization*, in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 197–209, 2019.

[39] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, *Cacti 6.0: A tool to model large caches*, *HP laboratories* **27** (2009) 28.

[40] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, *Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures*, in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, 2009.

[41] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, *Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **31** (2012), no. 7 994–1007.

[42] Y.-H. Chen, J. Emer, and V. Sze, *Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks*, *ACM SIGARCH Computer Architecture News* **44** (2016), no. 3 367–379.

[43] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, *Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices*, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **9** (2019), no. 2 292–308.

[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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 754–768, 2019.

[45] L. Ke, X. He, and X. Zhang, *Nnest: Early-stage design space exploration tool for neural network inference accelerators*, in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 1–6, 2018.

[46] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, *Scale-sim: Systolic cnn accelerator simulator*, *arXiv preprint arXiv:1811.02883* (2018).

[47] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, *Interstellar: Using halide's scheduling language to analyze dnn accelerators*, in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 369–383, 2020.

[48] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, *Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines*, *Acm Sigplan Notices* **48** (2013), no. 6 519–530.

[49] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, *Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures*, in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 97–108, IEEE, 2014.

[50] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, *Minerva: Enabling low-power, highly-accurate deep neural network accelerators*, in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 267–278, IEEE, 2016.

[51] J. Cong and J. Wang, *Polysa: polyhedral-based systolic array auto-compilation*, in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[52] L. Jia, L. Lu, X. Wei, and Y. Liang, *Generating systolic array accelerators with reusable blocks*, *IEEE Micro* **40** (2020), no. 4 85–92.

[53] Y. N. Wu, J. S. Emer, and V. Sze, *Accelergy: An architecture-level energy estimation methodology for accelerator designs*, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019.

[54] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, *Timeloop: A systematic approach to dnn accelerator evaluation*, in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 304–315, IEEE, 2019.

[55] "Nvdla deep learning accelerator." `http://nvdla.org`, 2007.

[56] J. Zhao, X. Dong, and Y. Xie, *Cost-aware three-dimensional (3d) many-core multiprocessor design*, in *Design Automation Conference*, pp. 126–131, IEEE, 2010.

[57] D. Stow, I. Akgun, R. Barnes, P. Gu, and Y. Xie, *Cost analysis and cost-driven ip reuse methodology for soc design based on 2.5 d/3d integration*, in *Proceedings of the 35th International Conference on Computer-Aided Design*, pp. 1–6, 2016.

[58] D. Stow, Y. Xie, T. Siddiqua, and G. H. Loh, *Cost-effective design of scalable high-performance systems using active and passive interposers*, in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 728–735, IEEE, 2017.

[59] H. Kung and C. E. Leiserson, *Systolic arrays (for vlsi)*, in *Sparse Matrix Proceedings 1978*, vol. 1, pp. 256–282, Society for industrial and applied mathematics, 1979.

[60] H.-T. Kung, *Why systolic architectures?*, *Computer* **15** (1982), no. 01 37–46.

[61] J. L. Hennessy and D. A. Patterson, *A new golden age for computer architecture*, *Communications of the ACM* **62** (2019), no. 2 48–60.

[62] L. A. Barroso and U. Hölzle, *The datacenter as a computer: An introduction to the design of warehouse-scale machines*, *Synthesis lectures on computer architecture* **4** (2009), no. 1 1–108.

[63] C. Delimitrou and C. Kozyrakis, *Amdahl's law for tail latency*, *Communications of the ACM* **61** (2018), no. 8 65–72.

[64] X. Liang, M. Nguyen, and H. Che, *Wimpy or brawny cores: A throughput perspective*, *Journal of Parallel and Distributed Computing* **73** (2013), no. 10 1351–1361.

[65] D. Meisner and T. F. Wenisch, *Does low-power design imply energy efficiency for data centers?*, in *IEEE/ACM International Symposium on Low Power Electronics and Design*, pp. 109–114, IEEE, 2011.

[66] R. Kumar, V. Zyuban, and D. M. Tullsen, *Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling*, in *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 408–419, IEEE, 2005.

[67] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, *Heterogeneous chip multiprocessors*, *Computer* **38** (2005), no. 11 32–38.

[68] W. J. Song, A. Buyuktosunoglu, C.-Y. Cher, and P. Bose, *Measurement-driven methodology for evaluating processor heterogeneity options for power-performance efficiency*, in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 284–289, 2016.

[69] S. Chen, S. GalOn, C. Delimitrou, S. Manne, and J. F. Martinez, *Workload characterization of interactive cloud services on big and small server platforms*, in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 125–134, IEEE, 2017.

[70] H. Kung, B. McDanel, S. Q. Zhang, X. Dong, and C. C. Chen, *Maestro: A memory-on-logic architecture for coordinated parallel use of many systolic arrays*, in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160, pp. 42–50, IEEE, 2019.

[71] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, C. Young, and H. Esmaeilzadeh, *Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks*, in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 681–697, IEEE, 2020.

[72] A. Caner Yüzügüler, C. Sönmez, M. Drumond, Y. Oh, B. Falsafi, and P. Frossard, *Scale-out systolic arrays*, *arXiv e-prints* (2022) arXiv–2203.

[73] H. Kwon, A. Samajdar, and T. Krishna, *Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects*, *ACM SIGPLAN Notices* **53** (2018), no. 2 461–475.

[74] "Tf-sim: A tensorflow performance simulator for machine learning accelerator architectural exploration."

[75] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, *Sigma: A sparse and irregular gemm accelerator with flexible*

interconnects for dnn training, in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 58–70, IEEE, 2020.

[76] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, *Cambricon-x: An accelerator for sparse neural networks*, in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, 2016.

[77] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, *Eie: efficient inference engine on compressed deep neural network*, *ACM SIGARCH Computer Architecture News* **44** (2016), no. 3 243–254.

[78] "Berkey hardware floating point unit library." `https://github.com/ucb-bar/berkeley-hardfloat`.

[79] K. Bhanushali and W. R. Davis, *Freepdk15: An open-source predictive process design kit for 15nm finfet technology*, in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, pp. 165–170, 2015.

[80] T. Miyashita, K. Ikeda, Y. Kim, T. Yamamoto, Y. Sambonsugi, H. Ochimizu, T. Sakoda, M. Okuno, H. Minakata, H. Ohta, Y. Hayami, K. Ookoshi, Y. Shimamune, M. Fukuda, A. Hatada, K. Okabe, T. Kubo, M. Tajima, T. Yamamoto, E. Motoh, T. Owada, M. Nakamura, H. Kudo, T. Sawada, J. Nagayama, A. Satoh, T. Mori, A. Hasegawa, K. H, K. Sukegawa, A. Tsukune, S. Yamaguchi, K. Ikeda, M. Kase, T. Futatsugi, S. Satoh, and T. Sugii, *High-performance and low-power bulk logic platform utilizing fet specific multiple-stressors with highly enhanced strain and full-porous low-k interconnects for 45-nm cmos technology*, in *2007 IEEE International Electron Devices Meeting*, pp. 251–254, IEEE, 2007.

[81] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, *Chisel: constructing hardware in a scala embedded language*, in *DAC Design Automation Conference 2012*, pp. 1212–1221, IEEE, 2012.

[82] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, *Parameter variations and impact on circuits and microarchitecture*, in *Proceedings of the 40th annual Design Automation Conference*, pp. 338–342, 2003.

[83] J. Dean, *1.1 the deep learning revolution and its implications for computer architecture and chip design*, in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 8–14, IEEE, 2020.

[84] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[85] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[86] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, *Learning transferable architectures for scalable image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[87] "Tf-graph." `https://www.tensorflow.org/api_docs/python/tf/Graph`.

[88] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[89] "Accelerated linear algebra (xla): Optimizing compiler for machine learning."

[90] "Space-to-batch operation." `https://www.tensorflow.org/api_docs/python/tf/nn/space_to_batch`.

[91] "Space-to-depth operation." `https://www.tensorflow.org/api_docs/python/tf/nn/space_to_depth`.

[92] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, *The architectural implications of facebook's dnn-based personalized recommendation*, in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 488–501, IEEE, 2020.

[93] J. L. Greathouse and M. Daga, *Efficient sparse matrix-vector multiplication on gpus using the csr storage format*, in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 769–780, IEEE, 2014.

[94] A. Azad and A. Buluç, *A work-efficient parallel sparse matrix-sparse vector multiplication algorithm*, in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 688–697, IEEE, 2017.

[95] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey, *Faster cnns with direct sparse convolutions and guided pruning*, *arXiv preprint arXiv:1608.01409* (2016).

[96] "Performance per watt." `https://en.wikipedia.org/wiki/Performance_per_watt`. Wikipedia.

[97] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, *Mnasnet: Platform-aware neural architecture search for mobile*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

[98] M. Tan and Q. Le, *Efficientnetv2: Smaller models and faster training*, in *International Conference on Machine Learning*, pp. 10096–10106, PMLR, 2021.

[99] S. Li, M. Tan, R. Pang, A. Li, L. Cheng, Q. V. Le, and N. P. Jouppi, *Searching for fast model families on datacenter accelerators*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8085–8095, 2021.

[100] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, *A 0.32–128 tops, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm*, *IEEE Journal of Solid-State Circuits* **55** (2020), no. 4 920–932.

[101] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, *Mcm-gpu: Multi-chip-module gpus for continued performance scalability*, *ACM SIGARCH Computer Architecture News* **45** (2017), no. 2 320–332.

[102] Z. Tan, H. Cai, R. Dong, and K. Ma, *Nn-baton: Dnn workload orchestration and chiplet granularity exploration for multichip accelerators*, in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1013–1026, IEEE, 2021.

[103] V. Goyal, X. Wang, V. Bertacco, and R. Das, *Neksus: An interconnect for heterogeneous system-in-package architectures*, in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 12–21, IEEE, 2020.

[104] A. Coskun, F. Eris, A. Joshi, A. B. Kahng, Y. Ma, A. Narayan, and V. Srinivas, *Cross-layer co-optimization of network design and chiplet placement in 2.5-d systems*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **39** (2020), no. 12 5183–5196.

[105] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. S. B. Altaf, N. E. Jerger, and G. H. Loh, *Modular routing design for chiplet-based systems*, in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 726–738, IEEE, 2018.

[106] W. Donath, *Placement and average interconnection lengths of computer logic*, *IEEE Transactions on Circuits and Systems* **26** (1979), no. 4 272–277.

[107] "Transistor density." `https://en.wikichip.org/wiki/mtr-mm%C2%B2`.

[108] A. Stillmaker and B. Baas, *Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm*, *Integration* **58** (2017) 74–81.

[109] C. H. Stapper, *Defect density distribution for lsi yield calculations*, *IEEE Transactions on Electron Devices* **20** (1973), no. 7 655–657.

[110] "Ic cost and price model, icknowledge llc." `https://www.icknowledge.com/products/icmodel.html`.

[111] X. Dong, J. Zhao, and Y. Xie, *Fabrication cost analysis and cost-aware design space exploration for 3-d ics*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **29** (2010), no. 12 1959–1972.

[112] J. Kim and Y. Kim, *Hbm: Memory solution for bandwidth-hungry processors*, in *2014 IEEE Hot Chips 26 Symposium (HCS)*, pp. 1–24, IEEE, 2014.

[113] H.-S. P. Wong, K. Akarvardar, D. Antoniadis, J. Bokor, C. Hu, T.-J. King-Liu, S. Mitra, J. D. Plummer, and S. Salahuddin, *A density metric for semiconductor technology [point of view]*, *Proceedings of the IEEE* **108** (2020), no. 4 478–482.

[114] B. Feng, Y. Wang, G. Chen, W. Zhang, Y. Xie, and Y. Ding, *Egemm-tc: accelerating scientific computing on tensor cores with extended precision*, in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 278–291, 2021.

[115] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, *Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus*, in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 359–371, 2019.

[116] Y. Wang, B. Feng, and Y. Ding, *Dsxplore: Optimizing convolutional neural networks via sliding-channel convolutions*, in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 619–628, IEEE, 2021.

[117] L. Liu, L. Deng, X. Hu, M. Zhu, G. Li, Y. Ding, and Y. Xie, *Dynamic sparse graph for efficient deep learning*, *arXiv preprint arXiv:1810.00859* (2018).

[118] L. Liu, Z. Qu, Z. Chen, Y. Ding, and Y. Xie, *Transformer acceleration with dynamic sparse attention*, *arXiv preprint arXiv:2110.11299* (2021).

[119] Z. Chen, Y. Quan, Z. Qu, L. Liu, Y. Ding, and Y. Xie, *Dynamic n: M fine-grained structured sparse attention mechanism*, *arXiv preprint arXiv:2203.00091* (2022).

[120] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, *Tvm: An automated end-to-end optimizing compiler for deep learning*, in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, 2018.

[121] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, *Gnnadvisor: An adaptive and efficient runtime system for gnn acceleration on gpus*, in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pp. 515–531, 2021.