NeuroMeter: An Integrated Power, Area, and Timing Modeling Framework for Machine Learning Accelerators

Industry Track Paper

Tianqi Tang
UC Santa Barbara
Santa Barbara, CA, US
tianqi_tang@ucsb.edu

Sheng Li
Google
Mountain View, CA, US
lsheng@google.com

Lifeng Nai
Google
Mountain View, CA, US
lnai@google.com

Norm Jouppi
Google
Mountain View, CA, US
jouppi@google.com

Yuan Xie
UC Santa Barbara
Santa Barbara, CA, US
yuanxie@ece.ucsb.edu

***Abstract*—As Machine Learning (ML) becomes pervasive in the era of artificial intelligence, ML specific tools and frameworks are required for architectural research. This paper introduces 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. NeuroMeter'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). NeuroMeter has accurate modeling results, with the average power and area estimation errors below 10% and 17% respectively when validated against TPU-v1, TPU-v2, and Eyeriss.**

Leveraging the NeuroMeter's new capabilities on architecting manycore ML accelerators, this paper presents the first in-depth study on the design space and tradeoffs of "Brawny and Wimpy" inference accelerators in datacenter scenarios with the insights that are otherwise difficult to discover without NeuroMeter. Our 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. Our study also reveals important tradeoffs between performance and efficiency. For datacenter accelerators with low batch inference, a small ($\sim$16%) sacrifice of system performance (in achieved Tera OPerations per Second, aka TOPS) can lead to more than a 2x efficiency improvement (in achieved TOPS/TCO). To showcase NeuroMeter's capability to model a wide range of diverse ML accelerator architectures, we also conduct a follow-on mini-case study on implications of sparsity on different ML accelerators, demonstrating wimpier accelerator architectures benefit more readily from sparsity processing despite their lower achievable raw energy efficiency.

***Index Terms*—accelerator, hardware modeling, deep learning**

I. Introduction

As Machine learning (ML) becomes pervasive in the era of artificial intelligence, we have witnessed a "Cambrian explosion" of ML accelerators with a plethora of different accelerators being proposed and/or implemented from both academia and industry [47]. These ML accelerators are designed for a wide range of use cases, ranging from cloud to edge devices; they are designed as either standalone accelerators or near-memory processors [37]. With significantly different scenarios, performance and efficiency targets, the design space of ML accelerators is very large and complex, which in turn drives the clear need for a fast and accurate modeling of power, area, and chip timing of ML accelerator architectures. On the other hand, architecture level analytical modeling frameworks, such as CACTI [43], McPAT [39], Wattch [15], and GPUWattch [38], have been proven to be very useful for the architecture community. These modeling frameworks provide fast, accurate, and easy-to-understand modeling results of power, area, and timing on cache, memory, CPU, and GPU. However, with the recent boom of new ML accelerators, the community lacks an architectural analytical modeling framework for ML accelerators.

This paper introduces NeuroMeter, an integrated power, area, and timing modeling framework for ML accelerators. NeuroMeter advances the state-of-the-art from at least three aspects. Firstly, unlike prior ML accelerator modeling frameworks that either model power, area, or timing in isolation or require EDA tools, NeuroMeter is the first framework to simultaneously model power, area, and timing analytically at the architecture level. Secondly, NeuroMeter supports detailed modeling of critical architectural components of ML accelerators, including 2D systolic arrays, reduction trees, 1D vector units, vector register files, and beyond. Thirdly, compared to previous modeling frameworks such as McPAT, 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. To ensure accuracy, we have conducted a rigorous validation on NeuroMeter results 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

2378-203X/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00075

when validated against TPU-v1 [30], TPU-v2 [29], and Eyeriss [17]. 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). With its new capabilities, NeuroMeter empowers architects with a fast yet accurate modeling framework for exploring emerging manycore ML accelerators in a large architectural design space.

The first contribution of this paper is that NeuroMeter significantly advances the state-of-the-art, enhancing the architectural modeling ecosystem of ML accelerators for the community. Recent work such as Accelergy [55] and Timeloop [44] provide an ecosystem for architecture level ML accelerator modeling. Timeloop [44] 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 [55], 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 [31] [36] [51], 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 second contribution of this paper is the in-depth and comprehensive design space exploration on ML accelerators. With the recent "Cambrian explosion" 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 [30]; dual cores, each with one 128x128 systolic array in TPU-v2 [21]), 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 [19]) and Ampere (512 TensorCores with 1024 FMAs per clock per TensorCore and hardware supports for structural sparsity [3]). 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, we conduct comprehensive and consistent studies on the design space and tradeoffs of "Brawny and Wimpy" for datacenter inference accelerators. Our 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, our 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, we also conduct a follow-on mini-case study 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. Our results show that despite their relatively lower energy efficiency, it is easier for wimpier accelerator architectures to benefit from sparsity processing.

The rest of the paper is organized as follows: Sec. II gives the overview, modeling methodology, and validation of NeuroMeter; Sec. III leverages NeuroMeter to conduct the case study on brawny and wimpy manycore ML accelerators in the datacenter inference scenarios; Sec. IV conducts a sparse-oriented mini-case study to showcase NeuroMeter's functionality to model diverse architectures and support various workloads; Sec. V discusses the related work; and Sec. VI concludes the paper with a summary on NeuroMeter and the insights discovered from our two case studies.

II. NeuroMeter: Overview, Modeling Methodology, and Validation

NeuroMeter is an integrated power, area, and timing modeling framework for ML accelerators. Fig. 1 gives an overview of NeuroMeter, and highlights its input/output interface. There are two types of inputs to NeuroMeter: 1) the accelerator hardware configuration (mandatory) for NeuroMeter to construct and optimize the target accelerator; 2) the runtime statistics (optional) for NeuroMeter to conduct runtime analysis. NeuroMeter by default outputs the power, area, and timing of target ML accelerators based on their specified hardware configuration. With the help of an external application-level performance simulator, NeuroMeter enables system performance and efficiency analysis as well.

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. 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

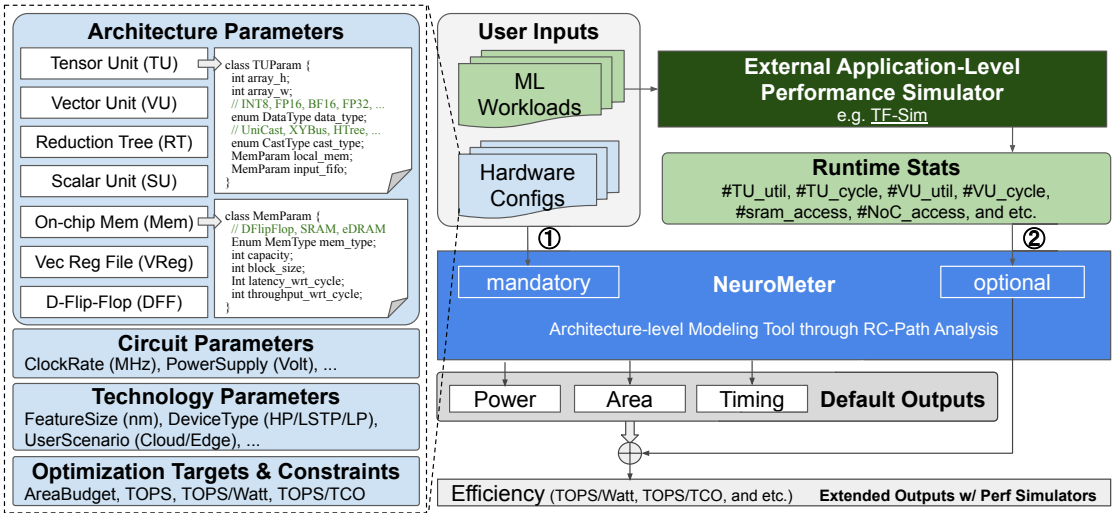

Fig. 1. Overview of NeuroMeter Framework

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 [23]. 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[1], 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.

A. Architecture-Level Modeling

NeuroMeter follows a top-down modeling methodology. As shown in Fig. 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 backend technology device and wiring parameters. At the chip architecture level, NeuroMeter models a multi-core ML accelerator. Fig. 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,

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

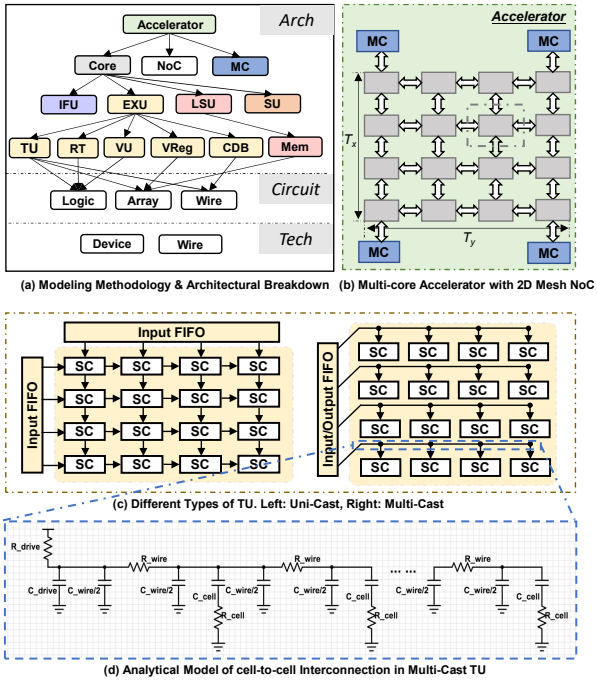

Fig. 2. NeuroMeter's Top-Down Modeling Methodology

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 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 and 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. 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. 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 [36] [48] [57] 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 [26] 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.

B. Circuit and Technology-Level Modeling

NeuroMeter models the power, area, and timing of the hardware components analytically and simultaneously. Similar to McPAT, 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 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 synthesis results from Design Compiler using the RTL models from Berkeley Hardware Floating Point Unit Library [2] with the technology backend of FreePDK [13] [42] libraries.

C. 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, we conduct rigorous validations at both the *component* level and the whole *chip* level. *At the component level*, we validate NeuroMeter's power, area, and timing results against the synthesis results from Chisel [11] with the FreePDK45 library. The validation against EDA tools shows that NeuroMeter's prediction is within a 15% area error margin, which provides strong confidence for our

component level modeling accuracy. As power highly depends on the block activity factors, we rigorously validate it at the chip level assuming average power traces. *At chip level*, we validate against TPU-v1 [30], TPU-v2 [29], and Eyeriss [17]. 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 [14] is comparable to the magnitude of the power validation errors of NeuroMeter.

Fig. 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 ($<331\text{mm}^2$). 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(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 by $\sim$10%, which may be due to the lack of knowledge of optimized placement-and-routing for the interconnect between systolic array and unified buffer in TPU-v1. We also model the peripheral interfaces including DRAM port (6.0% v.s. 2.8%) and PCIe interface (3.0% v.s. 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 $\sim$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(b), where the systolic array is the biggest power consumer with 56% of the total chip power.

Fig. 4 shows the area validation of TPU-v2 at an assumed 16nm technology node[2] 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 ($< 611\text{mm}^2$); and the modeling results of TDP (255W) have $\sim$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)

[2]According to the published information [29], TPU-v2's technology node is greater than 12nm.

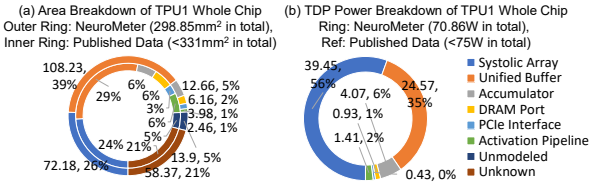

Fig. 3. Area and Power Break Down of TPU-v1 Published Data [30] v.s. NeuroMeter Simulation Results. TPU-v1 @ 700MHz with 0.86V power supply is fabricated at 28nm. Architecture parameters used in the model are: Systolic Array Size: 256x256; Accumulator: 256 int32 adders; Unified Buffer: 24MB, dual banks, 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.

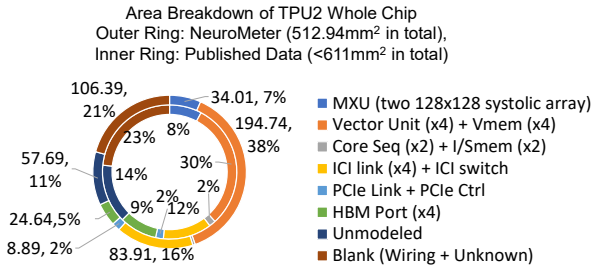

Fig. 4. Area Break Down of TPU-v2 Published Data [29] v.s. NeuroMeter Simulation Results. TPU-v2 @ 700MHz with 0.75V power supply assumes to be fabricated at 16nm. Architecture parameters used in the model are: MXU: two 128x128 systolic arrays with BF16 multiplier and FP32 adder; VMem: 8MB, quad-banks, 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.

and NoC given the bisectional bandwidth at 496Gb/s per direction. Other peripheral components, including HBM ports (9% v.s 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 $\sim$21% of the chip area, and we use the same percentage as white space in our overall area estimation.

Fig. 5 shows Eyeriss's validation results of power and area, at a 65nm technology node with a 200MHz target clock rate. As shown in Fig. 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. II-A. Other chip-level components, including Run-Length Code & ReLU, Config Scan Chain, and Top-Level Ctrl, are also modeled. As shown in Fig. 5(b), the relative area breakdown of PE array is overestimated by $\sim$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 $\sim$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

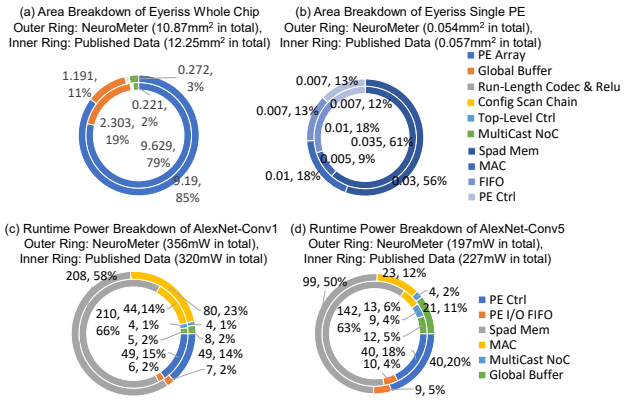

Fig. 5. Area and Power Break Down of Eyeriss-v1 Published Data [17] v.s. NeuroMeter Simulation Results.Eyeriss @ 200MHz with 1.0V power supply is fabricated at 65nm. Architecture parameters used in the model are: PE Array Size: 14x12; Local Buffer per PE: 448byte SRAM, 72byte registers, PE I/O FIFO transferring 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.

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[3] against the report from Eyeriss when running publicly available ML models. As shown in Fig. 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. 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.

III. Case Study on Brawny and Wimpy Manycore Machine Learning 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 [30] in TPU-v1 or dual cores with one 128x128 systolic array per core in TPU-v2 [21]). 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 [19]) and Ampere architecture (512 TensorCores with 1024 FMAs per clock per TensorCore and hardware supports for structural sparsity [3]). 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, we conduct detailed analyses to compare brawny and wimpy manycore ML accelerator designs. Interestingly, the brawny v.s. wimpy design tradeoffs have been a critical topic in CPU design and date back to decades ago as summarized in previous work [12]. We hope our work can 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, we focus on the inference accelerators in this paper and leave the study of training accelerators to future work.

A. Experiment Methodology and Setup

In our study, we follow the general architecture of manycore ML accelerators shown in Fig. 6. 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 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.

1) 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

[3]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.

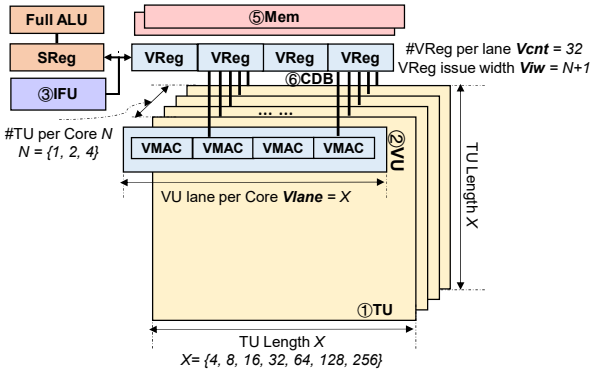

Fig. 6. Single Core Architecture of the Datacenter Inference Chips

TABLE I
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).

lanes, the VReg issue width, and VReg port count accordingly as shown in Fig. 6.

To some extent, chip architecting can be considered as an optimization problem, where 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 I, for datacenter inference accelerators, we constrain the die area to 500mm^2 and TDP to 300W based on recent data center ML accelerators [21] [30]. 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 [46]. Note that TPU-v2/v3 are designed for both training and inference [21]. 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. 6 and Table I.

Before setting the ranges of the implicit hardware parameters in Fig. 6, 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.

TABLE II
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

2) Machine Learning Models: Our datacenter case study uses three widely adopted CNN models, including ResNet-50 (abbrev. ResNet) [27], Inception-v3 (abbrev. Inception) [54], and NasNet-A-Large (abbrev. NasNet) [58]. Table II 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).

3) Performance Simulation and Efficiency Modeling: We use TF-Sim [9] to simulate the performance of the ML models running on the target accelerators. TF-Sim first takes the computational graph (e.g., tfGraph [7]) 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 [28].

TF-Sim supports advanced runtime graph scheduling and optimization, following the best practices in modern ML compiler/runtime such as XLA [1]. 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

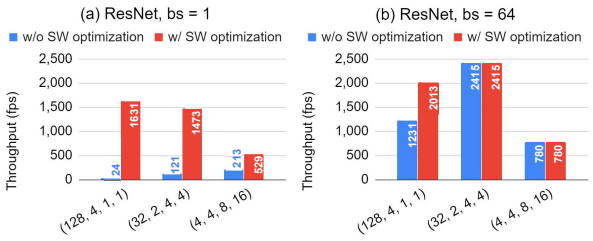

Fig. 7. Throughput Before and After Software Optimization

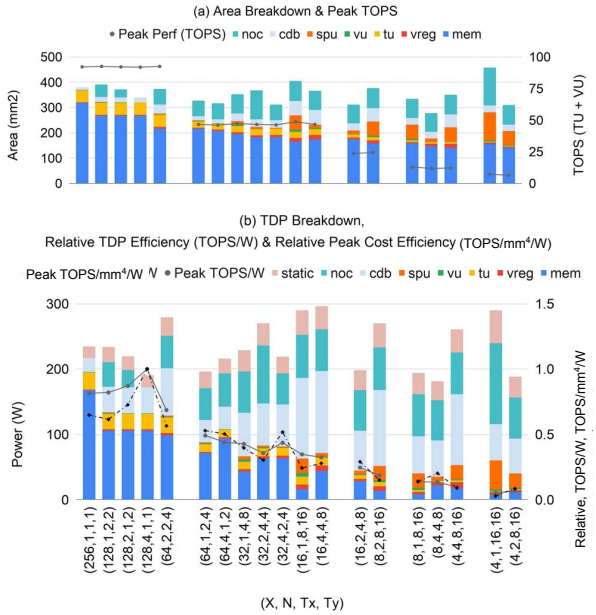

Fig. 8. Area, TDP Breakdown, Peak TOPS, and Relative Power, Cost Efficiency in DataCenter Inference Chips. Figure (a) and (b) share the same x-axis that indicates the design point defined in Table I. The subclusters are bins of peak TOPS.

without tiling. Moreover, TF-Sim also supports optimizations to improve parallelism, such as Space-to-Batch [5], Space-to-Depth [6], and double memory buffering. Fig. 7 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.

B. Results: Datacenter Inference Accelerator

In this subsection, we first explore the design space using the chip area and TDP, then analyze the runtime performance and efficiency by using NeuroMeter in conjunction with TF-Sim [9], our performance simulator. Our study reveals important insights for ML inference accelerator designs, which otherwise cannot be discovered in a fast-yet-accurate way without NeuroMeter.

1) Chip Thermal Design Power and Area: Fig. 8 shows the die area and chip thermal design power (TDP) for the

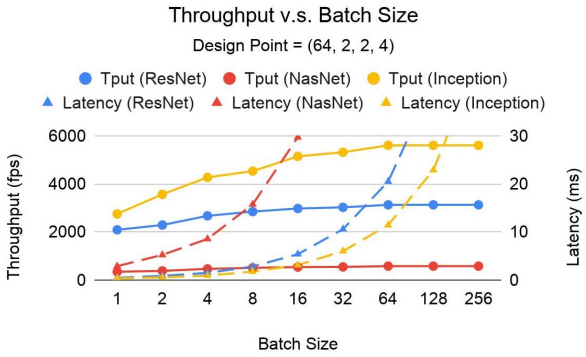

Fig. 9. Performance on Different Batch Size. Throughput is measured as frame per sec (fps) that essentially is TOPS as operation per frame is constant.

representative design points in the design space as defined in Table I. As shown in Fig. 8(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 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. 8(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. 8(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 to runtime performance and efficiency.

2) 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. 9 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.

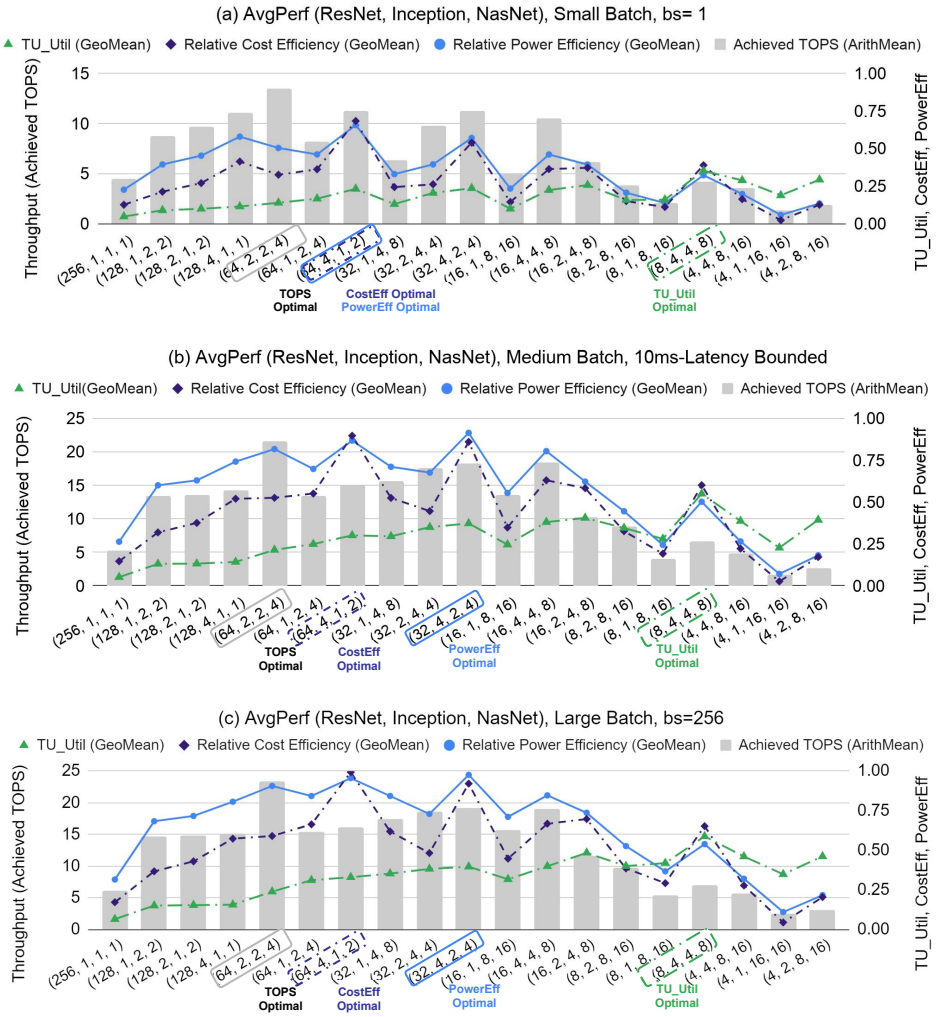

Fig. 10. Average Runtime Performance of ResNet, Inception, and NasNet of DataCenter Inference Chips. (a) bs=1, (b) bs = bounded by 10ms latency, (c) bs=256. The relative cost/power efficiency in right y-axes in three subfigures are normalized against the largest absolute values in Subfigure (c). The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

Fig. 9 also gives us the boundary on batch size for real-time tasks. Concretely, we assume real-time online inference has a latency constraint of 10ms based on production requirements from Google [30] and Facebook [25]. Therefore, the upper-bound batch sizes 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, we use the same approach to determine the maximum batch sizes that maximize the throughput while meeting the latency requirements. We call such batch size as latency limited batch size (or medium batch size). Our 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. 10 shows the average performance and efficiency of the three datacenter workloads. Fig. 10(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. 10 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. 10(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. 10(b) and (c). These tradeoffs provide important design guidance for architecting ML inference accelerators with different design priorities.

3) Summary of the Key Observations and Insights: We summarize the key observations and insights of our study on brawny and wimpy manycore ML accelerators 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.

IV. Mini-Case Study on Sparsity Implications on Different ML Accelerator Architectures

To showcase NeuroMeter's capability in modeling a wide range of diverse ML accelerator architectures, we conduct a small 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. 10(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 reduction tree 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 [24], including the non-zero elements and the row/column indices. The batched vectors are assumed dense in this case study; and SpMSpV [10] (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. III, does not support sparse operations, we develop a simple roofline model similar to that in [45] 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:

$$t_{\text{d}} = \max(t_{\text{d\_comp}}, t_{\text{d\_bw}}) = \max(\frac{C}{F}, \frac{S_V + S_W}{B});$$

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

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

where t_{d} the runtime for dense MV; and $t_{\text{d\_comp}}$ and $t_{\text{d\_bw}}$ 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 $t_{\text{s\_comp}}$ and $t_{\text{s\_bw}}$ 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

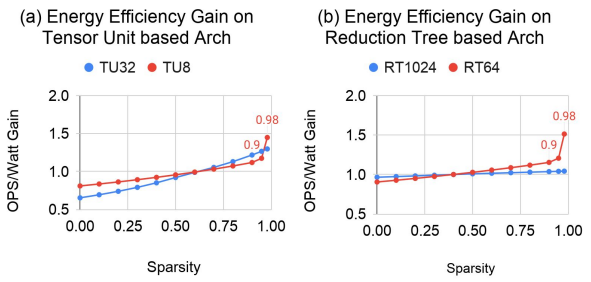

Fig. 11. The Energy Efficiency Gain of Sparse over Dense computation at Different Sparsity Levels on (a) Tensor Unit and (b) Reduction Tree based Architectures. For each architecture, its energy efficiency at different sparsities are normalized against that of the baseline dense processing on the same architecture. Thus, the energy efficiency gain larger than one indicates improvements.

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 [4]) 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.

Fig. 11 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. 11, 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. 10, a wimpier architecture with fine-grained computing units can benefit from element-wise sparsity more than a brawnier coarse-grained architecture.

V. Related Work

CACTI [43] is the first analytical modeling framework for cache and memory arrays. McPAT [39] uses the same analytical modeling methodology and builds up the modeling framework for manycore general-purpose processors. NeuroMeter leverages the same methodology and techniques used in CACTI and McPAT.

Eyeriss [17], Eyeriss-v2 [18], and MAESTRO [35] provide dataflow analysis and modeling framework for ML accelerators. NNest [31] provides a generalized spatial architecture framework for exploring the design space of ASIC-based ML inference accelerators. Scale-Sim [51] provides a cycle-accurate performance simulator for systolic CNN accelerators through on-chip and off-chip memory access traces. Interstellar [56] uses Halide's algorithm and scheduling primitives [49] to express different ML accelerator architectures. Aladdin [52], Minerva [50], and PolySA [20] provide different frameworks with (semi) HLS-level capabilities. Timeloop [44] and Accelergy [55] together provide an ecosystem to model ML accelerators. Besides providing modeling tools, previous work like NVDLA [8] open-source the RTL codes of typical ML accelerator designs; and this kind of work boosts the modeling ecosystem from another perspective. With simultaneously and analytically modeling power, area, and timing of key ML accelerator micro-architectures, NeuroMeter advances the state-of-the-art and provides foundational support for the modeling ecosystem.

In the era of artificial intelligence, a plethora of ML accelerator architectures are proposed. It has been an open question of how one can make the design choices among the architectures based on systolic arrays [17] [21] [30], reduction trees [48] [57], or SIMD vectors [19] for various scenarios and different workloads (e.g., training v.s. inference, dense v.s. sparse, datacenter v.s. edge, and beyond). NeuroMeter is able to model these popular emerging ML accelerator micro-architectures, which can help foster an even more comprehensive study on the ML accelerator frontier.

Brawny v.s. wimpy study [12] has been conducted extensively, with aspects including latency [22], throughput [40], energy efficiency [41], interconnect [33], heterogeneity [32] [53], and workload characteristics [16] in the CPU design space. With the growing ML workloads and the increasing deployment of ML inference accelerators in the datacenter, a similar brawny v.s. wimpy question has been raised in domain specific hardware. Kung et. al [34] have studied the latency of accelerators with different systolic array sizes.

VI. Conclusion

NeuroMeter is an architectural analytical framework for simultaneously modeling power, area, and 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.

By combining the power, area, and timing results of NeuroMeter with performance simulation, we explore the manycore ML accelerator design, including wimpy and brawny cores. Our 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. Our study 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). To showcase NeuroMeter's capability to model a wide range of accelerator architectures, we also conduct a mini-case study on energy efficiency (TOPS/Watt) implications of sparsity on different ML accelerators. Our results show that despite its relatively low energy efficiency, it is easier for wimpier accelerator architectures to benefit from sparsity processing.

Acknowledgement

We thank all anonymous reviewers for their valuable comments. This work was done when Tianqi Tang worked as an intern at Google. The work is also supported in part by NSF 1725447 and 1817037.

References

[1] "Accelerated linear algebra (xla): Optimizing compiler for machine learning," *https://www.tensorflow.org/xla/*.

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

[3] "Nvidia a100 tensor core gpu architecture - unprecedented acceleration at every scale," *https://coral.withgoogle.com/products/som/*.

[4] "Performance per watt," *Wikipedia*.

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

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

[7] "Tf-graph," *https://www.tensorflow.org/api_docs/python/tf/Graph*.

[8] "Nvdla deep learning accelerator," *http://nvdla.org*, 2007.

[9] Anonymous, "Tf-sim: A tensorflow performance simulator for machine learning accelerator architectural exploration," In submission.

[10] 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)*. IEEE, 2017, pp. 688–697.

[11] 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*. IEEE, 2012, pp. 1212–1221.

[12] 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*, vol. 4, no. 1, pp. 1–108, 2009.

[13] 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*, 2015, pp. 165–170.

[14] 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*, 2003, pp. 338–342.

[15] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 83–94, 2000.

[16] 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)*. IEEE, 2017, pp. 125–134.

[17] 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*, vol. 44, no. 3, pp. 367–379, 2016.

[18] 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*, vol. 9, no. 2, pp. 292–308, 2019.

[19] J. Choquette, O. Giroux, and D. Foley, "Volta: Performance and programmability," *Ieee Micro*, vol. 38, no. 2, pp. 42–52, 2018.

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

[21] 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)*. IEEE, 2020, pp. 8–14.

[22] C. Delimitrou and C. Kozyrakis, "Amdahl's law for tail latency," *Communications of the ACM*, vol. 61, no. 8, pp. 65–72, 2018.

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

[24] 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*. IEEE, 2014, pp. 769–780.

[25] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia *et al.*, "The architectural implications of facebook's dnn-based personalized recommendation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 488–501.

[26] 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*, vol. 44, no. 3, pp. 243–254, 2016.

[27] 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*, 2016, pp. 770–778.

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

[29] 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*, vol. 63, no. 7, pp. 67–78, 2020.

[30] 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*, 2017, pp. 1–12.

[31] 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*, 2018, pp. 1–6.

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

[33] 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)*. IEEE, 2005, pp. 408–419.

[34] 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. IEEE, 2019, pp. 42–50.

[35] 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*, 2019, pp. 754–768.

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

[37] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 740–753.

[38] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: enabling energy optimizations in gpgpus," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 487–498, 2013.

[39] 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*, 2009, pp. 469–480.

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

[41] 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*. IEEE, 2011, pp. 109–114.

[42] T. Miyashita, K. Ikeda, Y. Kim, T. Yamamoto, Y. Sambonsugi, H. Ochimizu, T. Sakoda, M. Okuno, H. Minakata, H. Ohta *et al.*, "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*. IEEE, 2007, pp. 251–254.

[43] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[44] 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)*. IEEE, 2019, pp. 304–315.

[45] 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.

[46] D. Patterson, ""key tpuv1/v2/v3 features vs volta gpu" in the talk of "domain specific architectures for deep neural networks: Three generations of tensor processing units" (26'33")," *Allen School Distinguished Lecture, 2019*.

[47] ——, ""the launch of 1000 chips" in the talk of "domain specific architectures for deep neural networks: Three generations of tensor processing units" (53'09")," *Allen School Distinguished Lecture, 2019*.

[48] 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)*. IEEE, 2020, pp. 58–70.

[49] 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*, vol. 48, no. 6, pp. 519–530, 2013.

[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)*. IEEE, 2016, pp. 267–278.

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

[52] 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)*. IEEE, 2014, pp. 97–108.

[53] 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*, 2016, pp. 284–289.

[54] 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*, 2016, pp. 2818–2826.

[55] Y. N. Wu and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," 2019.

[56] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "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*, 2020, pp. 369–383.

[57] 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)*. IEEE, 2016, pp. 1–12.

[58] 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*, 2018, pp. 8697–8710.