2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) | 979-8-3315-0294-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISPASS64960.2025.00026

SCALE-Sim v3: A modular cycle-accurate systolic accelerator simulator for end-to-end system analysis

Ritik Raj*, Sarbartha Banerjee†§, Nikhil Chandra*§, Zishen Wan*§, Jianming Tong*§,
Ananda Samajdhar‡, Tushar Krishna*
*Georgia Institute of Technology
†University of Texas Austin
‡IBM Research

***Abstract*—The rapid advancements in AI, scientific computing, and high-performance computing (HPC) have driven the need for versatile and efficient hardware accelerators. Existing tools like SCALE-Sim v2 provide valuable cycle-accurate simulations for systolic-array-based architectures but fall short in supporting key modern features such as sparsity, multi-core scalability, and comprehensive memory analysis. To address these limitations, we present SCALE-Sim v3 (GitHub Repository), a modular, cycle-accurate simulator that extends the capabilities of its predecessor. SCALE-Sim v3 introduces five significant enhancements: multi-core simulation with spatio-temporal partitioning and hierarchical memory structures, support for sparse matrix multiplications (SpMM) with layer-wise and row-wise sparsity, integration with Ramulator for detailed DRAM analysis, precise data layout modeling to minimize memory stalls, and energy and power estimation via Accelergy. These improvements enable deeper end-to-end system analysis for modern AI accelerators, accommodating a wide variety of systems and workloads and providing detailed full-system insights into latency, bandwidth, and power efficiency.**

A 128×128 array is 6.53× faster than a 32×32 array for ViT-base, using only latency as a metric. However, SCALE-Sim v3 finds that 32×32 is 2.86× more energy-efficient due to better utilization and lower leakage energy. For EdP, 64×64 outperforms both 128×128 and 32×32 for ViT-base. SCALE-Sim v2 shows a 21% reduction in compute cycles for six ResNet18 layers using weight-stationary (WS) dataflow compared to output-stationary (OS). However, when factoring in DRAM stalls, OS dataflow exhibits 30.1% lower execution cycles compared to WS, highlighting the critical role of detailed DRAM analysis.

***Index Terms*—cycle-accurate, sparsity, multi-core**

I. Introduction

The computation capacity has been increasing significantly over the years to drive algorithm advances in the field of Artificial Intelligence, Scientific computing, High-Performance Computing (HPC), AR/VR applications, and so on. As per [1], the training compute demands (FLOPS) have been doubling every nine months since 2015. There has been a plethora of accelerators targeting matrix multiplication in AI [2]–[6], Scientific computing [7]–[10], HPC [11]–[14], AR/VR [15]–[17], and robotic [18]–[21] applications. A fixed configuration for such accelerators is inefficient for all the varieties of workloads. These differences have led to an era of domain-specific AI accelerators, where it is necessary for accelerators to adapt to their specific workloads.

§The authors contributed equally to this work.

Designing an efficient AI accelerator is a challenging problem that requires a deeper analysis of computing hardware, domain-specific workloads, target constraints such as latency, power, and bandwidth as well as mapping strategies. To mitigate this problem, there exist several tools [22]–[26] for modeling performance and energy behavior of AI accelerators. SCALE-Sim (SystoliC AcceLErator SIMulator) v2 [24] is one of the popular open-source simulators for systolic array-based accelerators in use today. Fundamentally, it models the runtime for convolution and GEMM operators - which are the key building blocks within modern AI models. It also provides the flexibility to support systolic arrays of different shapes and sizes in addition to the sizes of double-buffered on-chip memories for input activations, weights, and output activations. In addition, it supports flexibility in mapping operators to the systolic array via three classic dataflows [27] - input stationary, weight stationary, and output stationary. SCALE-Sim v2 provides a detailed output report consisting of latency, bandwidth requirement, and cycle-accurate read/write traces for SRAM and main memory (DRAM or HBM).

SCALE-Sim v2 is a promising tool for exploring the design space of systolic-array-based AI accelerators and has been leveraged by several recent works [30], [35]–[38]. However, in this work, we identify limitations in SCALE-Sim v2 that restrict it from modeling recent innovations in AI accelerators and enabling full-system analysis and comparison of diverse designs, as we discuss next.

Limitation 1: Multiple Tensor Cores. Given a mix of both matrix and vector operations in modern AI workloads, recent accelerators leverage a mix of matrix multiplication engines (i.e., systolic arrays) and vector engines. Furthermore, to support more flexible partitioning and mapping, recent accelerators employ *multiple* tensor cores backed by a shared scratchpad memory. Examples include Google's TPUv4 [2] with two tensor cores, Meta's MTIA [4] with 64 PEs (each PE has two RISC-V cores) Cerebras WSE-2 [3] with 850k sparse tensor cores. SCALE-Sim v2 does not have support to model such chips, in part because when it was proposed that most accelerators just had single systolic arrays.

Limitation 2: Sparsity. In recent years, sparsity has become a key feature in state-of-the-art AI models [39]–[41], enabling a suite of software optimizations [42], [43], hardware accelerators [41], [44], [45], [45], [46] as well as hardware-

2766-0486/25/$31.00 ©2025 IEEE
DOI 10.1109/ISPASS64960.2025.00026

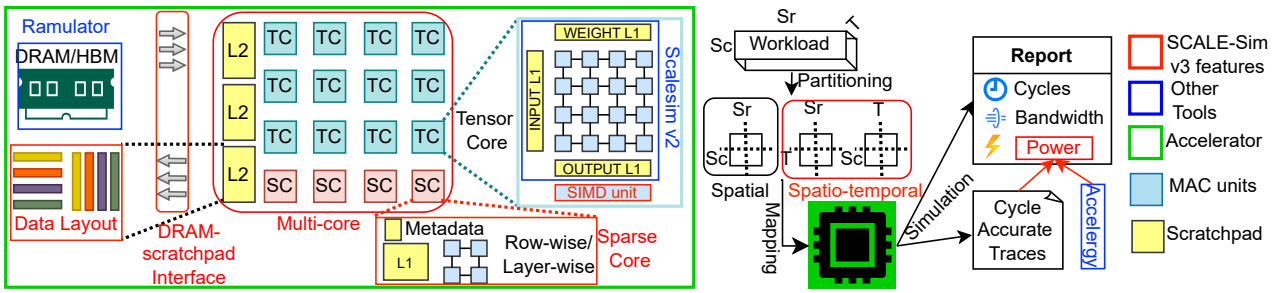


Fig. 1: Overview of SCALE-Sim v3 highlighting the new features over SCALE-Sim v2 marked in red

Framework	Compute	Cores	Partitioning	Sparse Core	Power	Energy Model	Data Layout	Main Memory Model
MAESTRO [25], [28]	analytical	single	N/A	N/A	N/A	Cacti-6.0 [29]	No	N/A
STONNE [23]	cycle-accurate	single	N/A	SIGMA-like [30] Unstructured	Average	Table-based model based on synthesis and place-and-route	No	N/A
Timeloop v4 [22]	analytical	many	Spatio-temporal	Distribution based	Average	Accelergy [31]	No *	CactiDRAM [32]
SCALE-Sim v2 [24]	cycle-accurate	many	Spatial	N/A	N/A	N/A	No	N/A
SCALE-Sim v3	cycle-accurate	many	Spatio-temporal	Layer and Row wise N:M	Instantaneous + Average	Accelergy [31]	yes	Ramulator [33]

TABLE I: Related Works

* FEATHER [34] paper adds data layout modeling support in Timeloop but is not part of Timeloop's public repo yet.

software co-design [47], [48] techniques to enhance computational efficiency. Furthermore, there is also an increasing interest in leveraging sparse matrix multiplication accelerators for domains beyond AI - such as scientific computing and graph analytics which are inherently hypersparse. Leveraging sparsity can reduce unnecessary computations and memory accesses, thereby enhancing computational efficiency and energy consumption. Unfortunately, SCALE-Sim v2 does not support sparse matrix multiplication units, which are now present in modern AI accelerators like TPU v5 [49].

Limitation 3: Main Memory Interface. The main memory technology implementation [50] and the workload's specific access pattern can lead to considerable variations in memory read/write latency, energy, and row conflicts. In fact, this behavior is exploited by attacks such as row hammer [51], [52]. Therefore, a deeper study of the DRAM/HBM and on-chip memory interface is needed to understand the memory behavior of different workloads running on an AI accelerator. Unfortunately, SCALE-Sim v2 models main memory as a monolithic entity with a fixed latency and bandwidth. The lack of a main memory interface in SCALE-Sim v2 limits it from providing the key statistics mentioned above.

Limitation 4: Data Layout. The organization of data within on-chip storage, such as SRAM or BRAM, plays a critical role in the performance of machine learning accelerators. The interaction between data layout and dataflow can significantly impact latency and compute efficiency [34], a ignorance of the actual data layout could lead to a magnitude of performance off. Optimal dataflows, which define how data is processed and moved through compute and memory resources, can be hindered by suboptimal data layouts due to issues like bank conflicts—when multiple data elements are accessed from the same memory bank simultaneously, causing stalls in computation. Therefore, a detailed modeling of data layout on the memory behavior is essential which is overlooked by SCALE-Sim v2.

Limitation 5. Energy. In recent years, hardware accelerators have become a key approach for improving energy efficiency by leveraging the unique characteristics of specific application domains. However, accurately predicting energy consumption often requires completing the physical design layout, which significantly hinders design space exploration due to high overhead and slow simulation times [31]. To enable faster and more efficient design exploration, energy estimation must be performed earlier in the design process, without requiring a fully detailed hardware description. Additionally, data movement energy constitutes a significant portion of an accelerator's total energy consumption, particularly in data-dependent scenarios. Unfortunately, SCALE-Sim v2 lacks support for energy and power modeling, limiting its ability to provide valuable insights for designing energy-efficient accelerators.

In this work, we address the aforementioned limitations of SCALE-Sim v2 and enhance it via five new features as shown in Figure 1. ❶ Modeling of multi-core features including spatio-temporal partitioning, L2 shared memory, heterogeneous tensor cores and non-uniform workload partitioning. ❷ Support for systolic-array based sparse accelerators targeting layer-wise and row-wise SpMM workloads. ❸ DRAM and on-chip memory interface modeled by Ramulator [33] integration. ❹ Modeling of data layout for a more accurate on-chip memory stall analysis. ❺ Energy and power analysis modeled by Accelergy [31] integration.

II. Background

A. Systolic arrays and multi-core accelerators

Systolic arrays are specialized hardware architectures designed to efficiently execute repetitive computations, particularly those involving linear algebra operations such as matrix multiplication and convolutions [53]–[59] commonly found in digital signal processing and machine learning workloads.

Research indicates that computational requirements for AI models have doubled every 3-4 months since 2012 [60]. These requirements are driven by the development of increasingly complex models and the expansion of AI applications across various sectors including natural language processing (NLP) [61]–[63], autonomous driving [64]–[66], healthcare [67]–[69] and so on. This exponential growth necessitates specialized hardware accelerators. A lot of AI accelerators, including Google TPU v5 [49], Meta MTIA [4], Cerebras WSE-2 [3], Nvidia DGX GH200 [70], and Tesla Dojo [71] have incorporated multi-chip and multi-core designs to meet the increasing demands.

B. Sparse Accelerators

Sparsity has been widely adopted in commercial accelerators due to its potential to enhance computational efficiency. For example, NVIDIA's Ampere architecture [45] introduced Sparse Tensor Cores, which leverage structured sparsity with a 2:4 sparsity ratio, ensuring that two out of every four tensor elements are non-zero. Similarly, Google TPUs [49] utilize sparse accelerators to efficiently process sparse tensors, incorporating mechanisms like sparse weight pruning and sparse matrix operations to scale effectively for large AI workloads. Accelerators such as VerSA [72] and VEGETA [46] propose a versatile systolic array capable of accelerating both dense and sparse matrix multiplications, addressing the diverse needs of modern deep neural network applications.

C. Memory Simulators

Memory simulators like Ramulator [33], [73] or DRAMSim3 [74] simulate the internal behavior of DDR and other memory technologies. Typically these simulators provide detailed metrics such as total memory requests, read/write operations, average latency, bandwidth utilization, row buffer hit rate, and power consumption estimates, aiding in comprehensive memory system performance analysis. Specifically, we integrate Ramulator [33] as the memory model for SCALE-Sim v3, which is a high-performance, cycle-accurate DRAM simulator facilitating the evaluation and design of memory systems. Its modular architecture allows for straightforward integration of various DRAM standards, including DDR3, DDR4, LPDDR4, GDDR5, WIO1, WIO2, and HBM, enabling researchers to assess and compare different memory technologies within a unified framework.

D. Accelergy

Accelergy [31] is an architecture-level energy estimation tool designed to provide accurate and flexible energy consumption analyses for accelerator designs. Developed to address the challenges of energy-efficient computing, Accelergy allows designers to model both primitive components and complex, user-defined compound components within an accelerator architecture. It systematically captures the energy consumption of various building blocks, facilitating rapid design space exploration without the need for detailed physical layouts, achieving up to 95% accuracy in estimating energy consumption for well-known deep neural network accelerators like Eyeriss [27].

E. SCALE-Sim v2

SCALE-Sim v2 is a cycle-accurate simulator designed to model systolic array-based accelerators for convolutions and GEMMs. The simulator accepts user-defined configurations, allowing for the specification of parameters including systolic array dimensions, on-chip double-buffered SRAM sizes, dataflow depending on reuse (input, weight or output stationary), and DRAM bandwidth. It processes neural network topologies provided in CSV format, detailing layer-wise specifications to accurately model workloads. Upon execution, SCALE-Sim v2 generates detailed reports, including compute cycles, bandwidth utilization, and cycle-accurate SRAM and DRAM traces, offering insights into the performance characteristics of the simulated architecture.

In the rest of the paper, we present the five enhancements we add to SCALE-Sim v2 - namely support for multiple tensor cores, sparsity, main memory, data layouts, and energy/power.

III. Multiple Tensor Cores

This section describes the multi tensor core simulation feature of SCALE-Sim v3 which is orthogonal to v2 in the following four ways: ❶ SCALE-Sim v3 expands the spatial workload partitioning mentioned in v2 to spatio-temporal partitioning (Section III-A) as shown in Figure 2. ❷ The memory on the chip is modeled in a hierarchical structure where multiple cores share an L2 scratchpad (Section III-B), as shown in Figure 4. ❸ Heterogeneous tensor cores (Section III-C) in terms of systolic array dimensions and SIMD length. ❹ Non-uniform workload partitioning (Section III-D) for cores having different latency profiles [75]

A. Spatio-temporal partitioning

Dataflow	Mapping		
	Sr	Sc	T
Input Stationary	K	N	M
Weight Stationary	K	M	N
Output Stationary	M	N	K

TABLE II: GEMM mapping for different dataflows

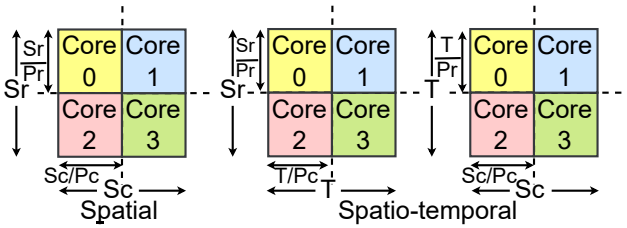


Fig. 2: Spatial and spatio-temporal partitioning

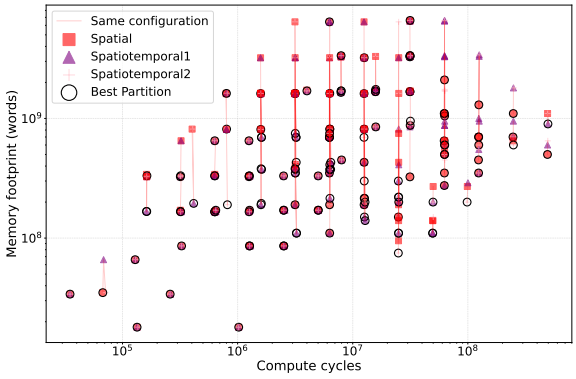
We introduce spatio-temporal partitioning as a new feature, enhancing spatial partitioning introduced in the SCALE-Sim v2 paper [24] as shown in Figure 2. It refers to partitioning along one of spatial (Sr/Sc) and temporal (T) dimensions (Table II) instead of just spatial dimensions. Assuming R and C as systolic array dimensions; Sr, Sc, and T as spatial and temporal mapping dimensions (Table II); Pr and Pc as the no. of row and column partitions (Figure 2) such as no. of cores = Pr×Pc, the following equations lists the runtime for different partitioning:

$$(2 * R + C + T - 2) * \lceil \frac{Sr}{\frac{Pr}{R}} \rceil * \lceil \frac{Sc}{\frac{Pc}{C}} \rceil \text{ [24]} \quad (1)$$

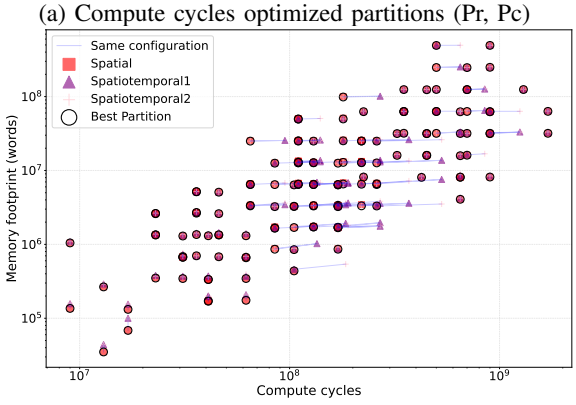
$$(2 * R + C + \lceil \frac{T}{Pc} \rceil - 2) * \lceil \frac{Sr}{\frac{Pr}{R}} \rceil * \lceil \frac{Sc}{Pc} \rceil \quad (2)$$

$$(2 * R + C + \lceil \frac{T}{Pr} \rceil - 2) * \lceil \frac{Sr}{R} \rceil * \lceil \frac{Sc}{\frac{Pc}{C}} \rceil \quad (3)$$

where Equation 1 represents spatial [24] while Equation 2 and Equation 3 represents spatio-temporal partitioning.



(a) Compute cycles optimized partitions (Pr, Pc)



(b) Memory footprint optimized partitions (Pr, Pc)

Fig. 3: Compute cycles v/s memory footprint tradeoff for spatial and spatiotemporal partitioning for scale-out cases

Figure 3 shows the memory footprint and compute cycles trade-off between spatial, spatio-temporal 1 (Equation 2), and spatio-temporal 2 (Equation 3) partitioning schemes. GEMM dimensions M, N, and K are chosen from [1000, 5000, 10000], so a total of 27 workloads. Systolic array dimensions (row and column sizes) are selected from [8, 16, 32], while the number of scale-out cores is chosen from [16, 32, 64]. Pr and Pc are chosen based on optimizing compute cycles (Figure 3a) and memory footprint (Figure 3b). The line connections denote that the points have the same configuration of workload, array size, and no. of cores. In Figure 3a (compute-optimized), the best partition among the three connected points is the one with the least memory footprint. In Figure 3b (memory-optimized), the best partition is the one with the least compute cycles. There are multiple examples in Figure 3a where spatiotemporal partitioning schemes outperform spatial partitioning. However, in Figure 3b, spatial partitioning outperforms in most cases.

B. Hierarchical memory with shared L2

Due to spatial partitioning, each core works on input partition, Pr × T and weight partition, Pc × T as shown in Figure 4. Each core in the same row receives the same partition of the input matrix while each column receives the same partition of the weight matrix. If there is only L1 SRAM, there will be lots of duplication across multiple cores in the same row (duplication of input matrix) or the same column (duplication of weight matrix). To mitigate the data duplication, we use shared L2 SRAM as shown in Figure 4. To ensure no stalls, the size of L2 SRAM should be enough to accommodate the input/weight partitions. L1 SRAM size can be configured to get a good balance between L1 size and L1 misses. Similarly, the shared L2 will receive input/weight partitions with common dimension Sr and Sc in case of spatiotemporal partitioning.

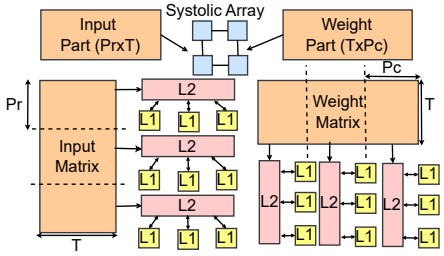


Fig. 4: Shared Input and Weight L2 SRAMs

C. Heterogeneous Tensor Cores

We add the support for tensor cores, following the naming convention from Google's TPU [2] where each TensorCore consists of one or more matrix-multiply units (MXUs) and a vector unit. The vector/SIMD unit is used for general computation such as activations and softmax. Similarly, SIMD unit in Meta's MTIA [4] handles quantization/de-quantization and nonlinear functions using lookup tables and floating-point units to approximate functions like exponential, sigmoid, and tanh. Therefore, SCALE-Sim supports arbitrary tensor cores with heterogeneity in terms of the length and type of SIMD units and/or systolic array dimensions as incorporated in [76]–[78]. In addition, the latency of SIMD units is customization as per the use case.

D. Non-uniform Workload Partitioning

Multi-Chip-Module [79]–[82] based accelerators including Simba [75] have a non-uniform latency profile based on the location of individual cores. The cores/chiplets which are farther away from the main memory requires more no. of hops or higher Network on Package (NoP) latency for communication of input or output matrices. On the other hand, the cores/chiplets which are closer to the main memory require lower NoP latency for input or output communication. This difference in NoP latency gives rise to non-uniform workload partitioning where farther cores/chiplets receive lower amount of workload (lower Pr and Pc) while nearer cores/chiplets receive higher amount of workload (higher Pr and Pc). SCALE-Sim supports non-uniform partitioning to effectively simulate cores/chiplets having different compute requirements.

IV. Sparsity

Sparsity is a fundamental characteristic observed in many deep learning models, where a significant proportion of tensor elements (such as weights or activations) are zero or negligibly small. Sparsity can be formally quantified as the ratio of non-zero valued elements in a block (N) to the total number of elements in the block (M) i.e. sparsity ratio is N:M.

A. Sparsity in SCALE-Sim v3

Building on this foundation, SCALE-Sim v3 extends its predecessor by incorporating support for layer-wise and row-wise sparsity in systolic array architectures.

1) Layer-Wise Sparsity: SCALE-Sim v3 allows the sparsity configurations to vary across layers in a neural network. This flexibility aligns with the observation that different layers exhibit varying degrees of sparsity. For example, initial layers in CNNs often have denser connections, while deeper layers are more amenable to higher sparsity levels.

2) Row-Wise Sparsity: Row-wise sparsity enforces a fixed number of nonzero elements at row granularity [46], thus offering more finer control over the sparsity of the workload. In SCALE-Sim v3, row-wise sparsity is implemented for various **N:M sparsity ratios**, where each group of M elements in a row contains exactly N nonzero values. This feature captures the realistic sparsity patterns present in real-world CNN models. SCALE-Sim v3 constraints sparsity ratios to $N \leq M/2$, ensuring that sparsity remains computationally advantageous. The density increases for $N > M/2$, negating the benefits of sparsity and approaching dense configurations.

B. SCALE-Sim v3 + Sparsity Integration

Step 1: Architectural and Workload Input: In topology files, a *SparsitySupport* column has been added representing sparsity ratios in the $N : M$ format. The configuration file has a new "sparsity" section, which contains the sparsity-related architectural knobs. For layer-wise sparsity, the *SparsitySupport* knob is set to *true* and the *OptimizedMapping* knob is set to *false*. For all the simulations presented in this paper, the *SparseRep* is set to *ellpack_block*. For row-wise sparsity, the *OptimizedMapping* knob is set to *true* and *BlockSize* holds the value of M in the $N : M$ ratio. The number of non-zero

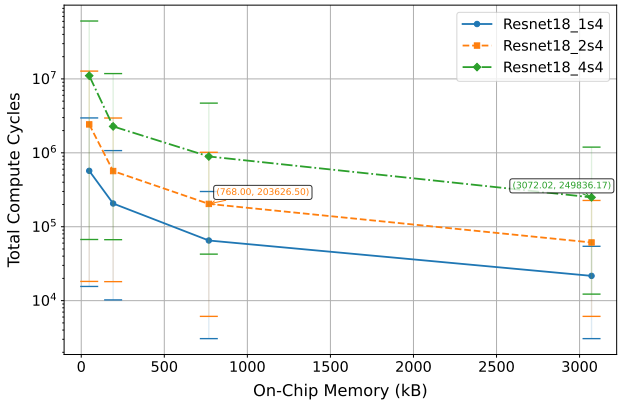


Fig. 5: Total compute cycles (including memory stalls) vs On-chip memory for ResNet-18 for 1:4, 2:4 and 4:4 sparsity ratios

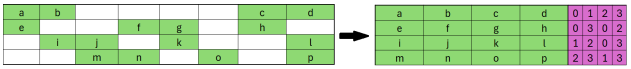


Fig. 6: (a) Original matrix (b) Blocked ELLPACK format

elements (N) is randomized for different rows and is kept $\leq M/2$. The dataflow is set to *weight-stationary* for all the sparsity simulations.

Step 2: Generating operand and demand matrices: Once the sparsity architectural knobs and details are read from the topology and the configuration files, the operand matrices shall be modified to reflect the sparsity of the model. For simplicity, we assume that the first N rows have non-zero elements and the remaining $N - M$ rows have zero elements. For row-wise sparsity, each row is assigned a random sparsity ratio such that $N \leq M/2$. Instead of streaming a single input element per row in the systolic array, we would need to stream a block of input elements. This is replicated in the *IFMAP* demand matrix generation by fetching corresponding addresses from *IFMAP SRAM*.

Step 3: Collecting statistics: The new SCALE-Sim v3 simulator built outputs a *SPARSE_REPORT.csv* that contains information and metrics such as Sparsity Representation, Original Filter Storage, and New Filter Storage (this consists of the compressed filter matrix and the metadata).

C. Sparsity Case Studies

Figure 5 demonstrates the relationship between total cycles including memory stalls and on-chip memory for the ResNet-18. This relationship is observed for different sparsity ratios viz., $1 : 4$, $2 : 4$ and $4 : 4$. As the size of on-chip memory increases, more data can be fetched into the SRAM in a single instance, which results in reduced stall cycles. For a given on-chip memory, sparse models require fewer compute cycles because of the increased sparsity.

SCALE-Sim v3 supports Compressed Sparse Row (CSR), Compressed Sparse Column (CSC) and Blocked ELLPACK formats. For all the sparsity related simulations, Blocked ELLPACK has been considered. For a matrix shown in Figure 6a and a block size equal to 4, the blocked ELLPACK

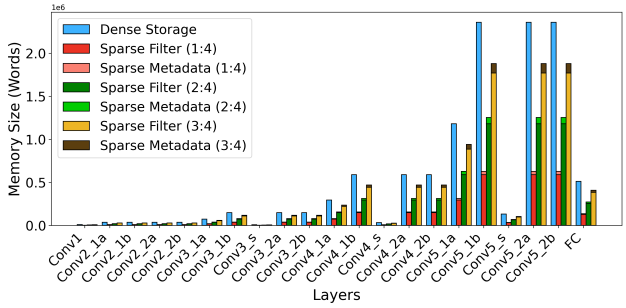


Fig. 7: The memory plot shows the memory storage comparison for dense, 1:4, 2:4 and 3:4 sparsity ratios for ResNet-18

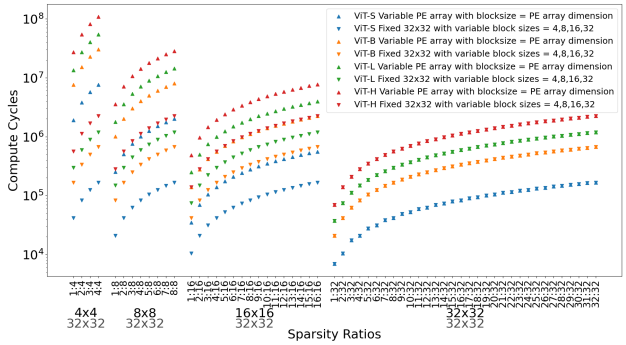


Fig. 8: Compute cycle variation shown for Feed Forward layers of ViTs for different array sizes, sparsity ratios and block sizes.

representation is depicted in Figure 6b; where the green cells represent the non-zero filter matrix values and the lavender cells represent the corresponding metadata. In the generic setting, the number of bits required for a single metadata entry is given by $\log_2(\text{Block Size})$.

Figure 7 compares the memory storage requirements of ResNet-18 layers for dense, 1:4, 2:4, and 3:4 sparsity ratios. For each sparsity ratio, the storage consists of sparse filter data and associated metadata. As sparsity increases, the number of filter elements stored in the memory and their corresponding metadata reduce highlighting the effect of sparsity on memory requirements.

To study the effects of varying block sizes on performance, a study has been performed as shown in Figure 8. The first set of runs has varying systolic array sizes (4x4, 8x8, 16x16, 32x32) and the block size considered is always equal to the systolic array dimension. Hence, sparsity ratio ranges only from $1 : M$ to $M : M$. This is compared against the second set of runs that has a fixed array size (32x32) and the block size considered is varied (M = 4, 8, 16 and 32). For each M, we shall have M different values of N, thus resulting in N different sparsity ratios. By increasing the block size, we get more finer granular control over the model, and utilizing the lower range/spectrum of $N : M$ values leads to better performances.

V. Main Memory Integration

One of the key limitations of SCALE-Sim v2 was the absence of a memory model with the systolic computation unit. The earlier version operated under the assumption that the data would somehow be readily available in the on-chip scratchpad. This assumption overlooked the estimation of memory latency, stalls caused by DRAM bank conflicts and other pipeline bubbles in the datapath, critical in memory intensive workloads.

SCALE-Sim v3 addresses this limitation by integrating Ramulator [33] memory model with the systolic array. It enables memory datapath modeling by configuring the load/store queue size, the DRAM channels and the memory technology (i.e, DDR4, HBM, LPDDR4 etc.). It also adds the memory latency, along with pipeline stalls to the ML inference latency.

We first introduce the primitives required for main memory integration in Section V-A, followed by the steps to simulate the ML inference latency using Ramulator in Section V-B, and finally reporting some results including memory throughput and the overall execution latency in Section V-C.

A. Memory Integration Primitives

1) Main Memory Model: A detailed memory model simulates the round-trip latency for each load and store memory transaction. Existing main memory models like Ramulator [33], [73], DRAMSim3 [74] etc. simulate different DRAM technologies while allowing the user to configure the memory controller. The model takes a memory demand request trace as an input. Each trace entry comprises of the request cycle, memory address, and transaction type (read or write). The main memory model provides the round-trip latency for each transaction, along with overall runtime statistics including the memory throughput, row buffer hits/misses, bank conflicts etc. This statistics can be used to study the execution bottlenecks of each workload dataflows, memory controller configurations and main memory technologies.

2) Finite Memory Request Queues: SCALE-Sim v3 adds configurable memory request queues to hold pending memory transactions. Demand requests logged in the request queues are cleared upon transaction completion. Read requests are cleared once the data value is received, whereas write requests are cleared once it is logged in the memory controller. The finite size of these request queues stalls the accelerator when the pending queue is full. These queues help model memory stalls that impact execution runtime of memory-intensive workloads.

3) Memory delay modeling: A systolic array operates on multiple data bits, which are fed from the input and the weight scratchpad SRAM buffers. SCALE-Sim v3 accounts for the memory load delay and triggers the systolic array only when each data is available in the scratchpad. This adds runtime latency and pipeline bubbles during data streaming degrading the compute performance for certain dataflows. Exact modeling of data-plane components enables the user to evaluate dataflow, sparsity, and other workload optimizations.

B. Memory Simulation Runtime

The main memory simulation workflow is as follows:

Step 1: Accelerator memory request generation: The systolic array simulator is used to generate the memory demand

transactions based on the sequence of data required to execute the workload. The *ifmap, filter and ofmap* addresses are generated along with a cycle timestamp, which are sequenced to form a single memory trace. Each entry contains a request cycle, a memory address and a transaction type.

Step 2: Evaluating the memory round-trip latency: We feed the generated memory trace to a memory simulator. We use Ramulator [33] in our evaluation. The memory parameters like technology, number of ranks and channels, frequency etc. are set in a configuration file. The memory simulator reports the round-trip latency for each individual request. The memory model also captures memory statistics like throughput, row-buffer hits and misses. The round-trip latency includes delays from memory bank conflicts and controller delays.

Step 3: Simulating SCALE-Sim v3 with memory delays: The systolic array simulator is run again, but with a finite request queue and with the actual memory load/store latency. The data from the memory simulator is sent as input to SCALE-Sim v3. The execution stalls are calculated by enabling the finite memory request queues and the memory delay modeling described in Section V-A. Overall, the above workflow provides a realistic systolic array efficiency based on the memory block simulated for an ML accelerator.

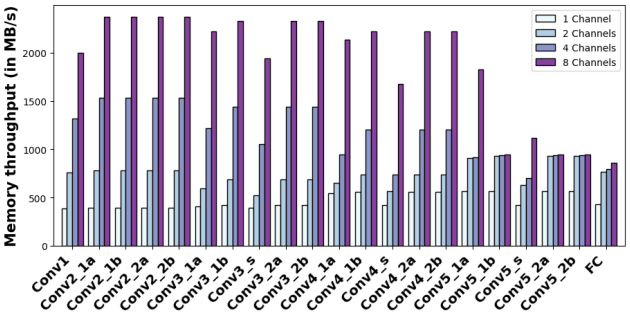


Fig. 9: Impact of DRAM channels on memory throughput in Resnet18

C. Memory Model Evaluation

1) SCALE-Sim v3 and Memory Configuration: SCALE-Sim v3 is run with the Google TPU configuration. The read and write request queues are individually configured to a size of 128 entries. The Ramulator simulates a DDR4 memory with 4Gb capacity for each channel with a 2400 MHz speed.

2) Impact of DRAM channels on memory throughput: ML inference execution is known to benefit from multiple DRAM channels, especially for memory-intensive layers. We evaluate the impact of memory throughput for Resnet18 layers by tuning the number of DDR channels from 1 to 8 as shown in Figure 9. We observe a proportional increase in the memory throughput with the addition of more memory channels for initial layers, the performance of the last layers saturates at a channel size of 2. The larger ifmap and filter sizes of the initial layers benefit from a larger number of channels, with some layers reaching a bandwidth of more than 2000 MB/s. The 1×1 filters and smaller ifmaps reduce the memory throughput for later convolution and fully connected layers. While memory-intensive layers and streaming patterns benefit from multiple channels, compute-intensive or smaller layers fail to leverage multiple channels efficiently. Each memory channel also comes at an additional area cost for the memory controller and a power cost for parallel data loads.

3) Stalls created by memory modeling : Prior ML inference simulators did not model stalls emanating from the memory transactions. We define the number of *stall cycles* as execution cycles when the systolic array is waiting on data in the scratchpad. Figure 10 illustrates the number of stall cycles for different workloads as a fraction of the total execution cycles. The three bars represent the demand request queue size of 32, 128 and 512 entries. The 32 entry request queue certainly adds a lot of stall cycles as the systolic array easily fills a small queue with demand requests. However, we see the percentage of stall cycles and the total execution cycles reduce with increasing entries. The average total cycles reduce by $3.76\times$ when the request queue size changes from 32 to 128 with a further improvement of 38% with 512 entries. Overall, the stall cycles due to memory simulation can delay the overall inference latency from 10 thousand cycles to a few million cycles depending on the workload and is important to consider while estimating the overall inference latency. The stall cycles in Figure 10 show the percentage of unaccounted execution inefficiency in SCALE-Sim-v2 (which assumed a zero latency data transfer). DRAM modeling enhances the precision of total execution cycle estimates and optimizes overall systolic array utilization metrics.

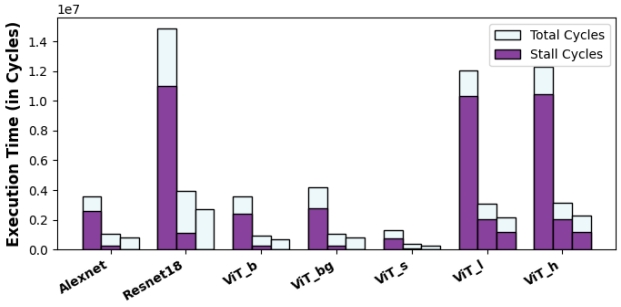


Fig. 10: Impact of memory stalls on the overall ML inference latency. The three bars configure a read/write request queue of size 32, 128 and 512 respectively.

VI. Data Layout in Memory

In multi-core accelerators, multiple compute units share the common on-chip memory and process different workloads in parallel. When data is requested by multiple compute units located at different rows of the same bank within the on-chip SRAM, multiple cycles latency is required to bring required data out of the off-chip, termed as bank conflict slowdown. This slowdown can significantly degrade practical performance [34]. To model this, we integrate precise layout modeling into SCALE-Sim v3.

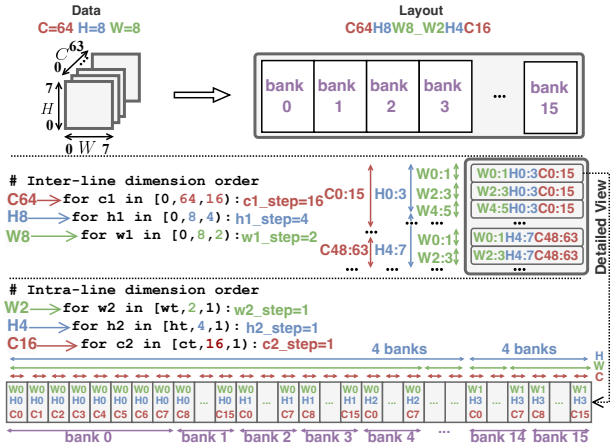


Fig. 11: Overview of data layout modeling in multi-bank on-chip memory. We use nested loop to model data layout, where each line is an aggregation of the line with the same index from all banks. Each bank is 2 dimensional array offering limited number of ports for concurrent read/write access.

In general, layout modeling includes two components:

Layout Modeling: Supports various data organizations in the multi-bank on-chip buffer.

Memory Latency Evaluation with Layout Consideration: Evaluates actual bank conflicts for accessing given data.

A. Data Layout Modeling

In modern multi-bank on-chip memory systems, global bandwidth is evenly distributed across memory banks. We model the multi-bank on-chip memory as a 2D array, where each row aggregates the row with the same index from all banks. This abstraction ensures the bandwidth of a single row matches the total on-chip bandwidth. Each bank has its own individual data accessing ports, enabling fine-grained data access and reducing bank conflicts.

We define the data layout as the organization of data within the 2D array derived from the multi-bank memory. This includes determining the row and column positions of each data element. We term the order of row and column positions as inter-line dimension order and intra-line dimension order, respectively. Each dimension order is represented as nested loops, as shown in the Figure 11.

B. Memory Latency Evaluation

SCALE-Sim v3 analyzes the latency for data requests from the compute array at the granularity of cycles. Taking a $C \times H \times W$ tensor from Figure 11 as an example, when element (c, h, w) is requested, the row-, column- and bank-index could be derived from the following equations.

$$\begin{aligned} line_{id} = &\lfloor c/c1\_step \rfloor \times \lfloor H/h1\_step \rfloor \times \lfloor W/w1\_step \rfloor \\ &+ \lfloor h/h1\_step \rfloor \times \lfloor W/w1\_step \rfloor \\ &+ \lfloor w/w1\_step \rfloor \end{aligned}$$

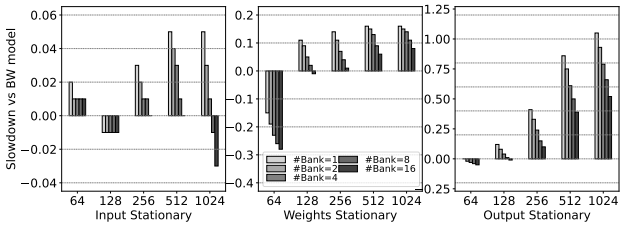


Fig. 12: Slowdown of different dataflows under different (on-chip bandwidth, bank numbers). Area size: 128×128. Workload: ResNet18.

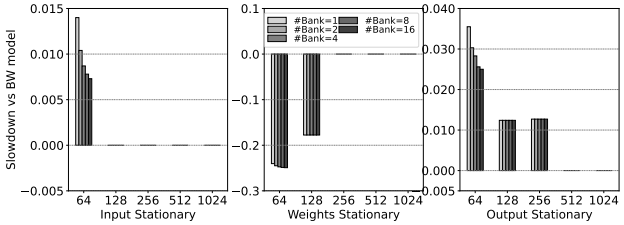


Fig. 13: Slowdown of different dataflows under different (on-chip bandwidth, bank numbers). Area size: 128×128. Workload: ViT.

$$\begin{aligned} col_{id} = &w\%w1\_step \times h1\_step \times c1\_step \\ &+ h\%h1\_step \times c2\_step \\ &+ c\%c1\_step \end{aligned}$$

$$bank_{id} = \lfloor \frac{col_{id}}{bandwidth\_per\_bank} \rfloor$$

where $bandwidth\_per\_bank$ represents the maximal number of elements within a single line of a bank.

Given the $(line_{id}, col_{id}, bank_{id})$ for all data a certain dataflow might require, we could calculate the total number of lines in need of being accessed concurrently per cycle for each individual bank, yielding the following overall data accessing latency, assuming the total number of ports available for concurrent reading from each bank as num_ports.

$$slowdown = \max_{i=0}^{num\_bank} \lceil total\_rows\_bank_i / num\_ports\_bank_i \rceil$$

Among all banks in on-chip memory, the bank with the maximal bank conflicts determines the critical latency of providing requested data from on-chip memory.

The performance differences between realistic on-chip bank models and the original ideal bandwidth modeling in SCALE-Sim v2 are illustrated in Figure 12 and Figure 13. The figure normalizes latency under the realistic layout model against the pure bandwidth modeling in SCALE-Sim v2, highlighting the actual slowdown. Notably, given the same on-chip bandwidth, an increased number of banks consistently improves performance, as evidenced by the reduced slowdown with a higher bank count. This improvement is attributed to the finer-grained on-chip data access flexibility provided by additional banks, which effectively mitigates bank conflicts.

Counts Name	Action Name	Argument	
		Data delta	Address delta
Repeated read	Read	0	0
Random read		1	1
Repeated write	Write	0	0
Random write		1	1
Idle		-	-

Table: Action types in Accelergy

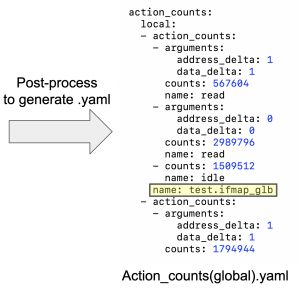


Fig. 14: Translation table for action types in Accelergy with a snippet of the generated YAML file for memory action counts.

VII. Energy and Power Modeling

This section presents the energy and power modeling of SCALE-Sim v3, achieved through the integration of Accelergy [31]. It covers the framework overview (Section VII-A) and details the integration methodology (Section VII-B– Section VII-E).

A. SCALE-Sim v3 Energy Modeling Framework Overview

The SCALE-Sim v3 energy modeling framework comprises three key components: (1) unified user input, (2) trace files and action counts, and (3) performance and energy reports. Below is a detailed breakdown:

Step 1: Architectural and Workload Input: The neural network description remains unchanged across SCALE-Sim v3 and Accerlegy, serving as an independent input to generate trace files and cycle performance data. SCALE-Sim v3 automatically generates a YAML file for architectural input, combining user inputs with pre-defined systolic array component descriptions and energy estimation plugins to create the Energy Reference Table (ERT). Users can also customize component descriptions for greater flexibility.

Step 2: Trace File and Action Counts: Accelergy distinguishes energy consumption for different action types (e.g., repeated vs. random memory accesses). To leverage this, SCALE-Sim v3 includes a feature to count action types in addition to generating trace files. The action counts file, along with performance metrics (latency, array utilization, and memory access counts), is produced by SCALE-Sim and used as input for Accelergy's energy analysis.

Step 3: Performance and Energy Output: After the Accelergy simulation completes, it generates a unified output that integrates energy estimations with SCALE-Sim v3's cycle performance summary. This combined output provides insights into both timing predictability and energy efficiency, enabling users to optimize their accelerator design for a superior energy-delay product.

B. Input Integration of SCALE-Sim v3 and Accelergy

The input files for SCALE-Sim v3 and Accelergy differ in abstraction levels. SCALE-Sim v3's configuration input file specifies high-level configurations such as array shape, SRAM sizes, and dataflow, while Accelergy's architecture YAML file describes lower-level details like hardware component types, memory bank counts, and data widths.

To ensure consistency between the two frameworks, we introduce a YAML file generator that extracts high-level systolic array configurations and extrapolates detailed architectural descriptions using a baseline YAML template, and outputs the final architecture.yaml file for energy modeling. The baseline template includes default components such as three register files and an integer MAC unit for each PE, along with three smart buffer SRAMs for input activations, weights, and partial sums. Users can customize component types by providing corresponding Accelergy-compatible descriptions.

C. Repeated Access Lookup

One of the key features enabling Accelergy's accurate energy estimates is its ability to distinguish between different action types, such as repeated and random memory accesses, which can differ in energy consumption by more than double. To support this, SCALE-Sim v3 generates an action count file as input for Accelergy.

SCALE-Sim v3 introduces two tunable parameters: 'row size' and 'bank size.' The 'row size' parameter defines the block of data retrieved during each memory access, allowing for energy-efficient repeated reads when accessing consecutive addresses within the same block. The 'bank size' parameter accounts for multiple row buffers in a memory bank, enabling data reuse across cycles without reloading.

D. Memory Action Counts

As discussed in Section VII-C, the action-count summary file enables Accelergy to provide detailed architecture-level energy estimates. The dataflow between SCALE-Sim v3 and Accelergy occurs in two stages.

First, SCALE-Sim v3 generates memory transaction counts (idle, random-read/write, and repeated-read/write) for the ifmap, ofmap, and filter data using the implemented repeat count function. For example:

$$ifmap_{SRAM-idle} = cycles \times arraysize - counts$$

$$ifmap_{SRAM-random} = counts - repeat - counts$$

These counts are exported into a summary file.

In the second stage, this data is post-processed to create a YAML file for Accelergy's energy estimation. Additional arguments, such as $data\Delta$ and $address\Delta$, indicate whether the corresponding wire switches are activated. Detailed parameter configurations are shown in Figure 14. The resulting YAML file is used as the primary input for Accelergy after the SCALE-Sim v3 simulation completes.

E. Processing Element Action Counts

MAC action counts. In SCALE-Sim v3, we define two basic MAC actions: *MAC_random* (normal operations with new data) and *MAC_constant* (no data change or computation). These are calculated as follows:

$$MAC\_random = \#PEs \times cycles \times utilization$$

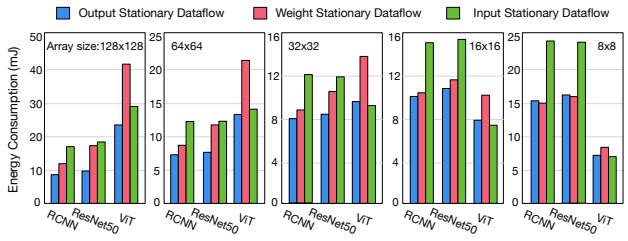


Fig. 15: Energy consumption under different dataflow types and systolic array dimensions (128×128, 64×64, 32×32, 16×16, and 8×8) across three workloads.

$$MAC\_constant = \#PEs \times cycles \times (1 - utilization)$$

Unused MAC units can leverage clock gating, modeled in Accelergy as *MAC_gated*, which consumes only static energy, reducing dynamic clock energy consumption. This optimization is incorporated in our energy evaluation.

Scratchpad action counts. There are three scratchpad memory inside each PE, namely ifmap_scratch_pad, weights_scratch_pad, and psum_scratch_pad. Based on Accelergy, each scratchpad has two action types: *write* and *read*. We generate action counts for each scratchpad. For *weight_spad*, we model *write* counts as the number of SRAM filter reads, *read* counts as the number of MAC operations. For *ifmap_spad*, we model *write* counts as the number of SRAM IFMAP reads, *read* counts as the number of MAC operations. For *psum_spad*, we model both *write* and *read* as the number of MAC operations. The number of writes and reads of each scratchpad relates to dataflow. Input stationary will result in less number of *imap_spad write* action. Output stationary makes each MAC unit responsible for all the computations required for an OFMAP pixel. Weight stationary will result in less number of *weight_spad write* action.

Putting all together, Figure 15 presents an example of the energy consumption analysis for SCALE-Sim v3 when arrays of varying sizes execute the workload using different dataflow. OS outperforms the other two dataflows in almost every case. Compared with WS and IS, WS is preferable under smaller array sizes, while IS is preferable under larger array sizes. This analysis offers valuable insights into energy efficiency for designing diverse accelerators. Additionally, it can be seamlessly integrated with other features (e.g., Ramulator, sparsity, layout, etc.) to derive deeper design insights.

VIII. Validation

SCALE-Sim v3 builds on top of SCALE-Sim v2, Accelergy, and Ramulator, which are already validated against RTL of systolic array (cycle-accurate), Eyeriss chip (within 5% error), and RTL of Micron DDR4 [83] (cycle-accurate) respectively. We provide some additional validation information below:

Sparsity. We support two types of sparse accelerators:

Ampere fixed 2:4 sparsity. SCALE-Sim v3 achieves 100% accuracy compared to Sparse Tensor Core's [84] report. While Sparseloop [85] claims compute cycles are deterministic, memory stalls are not. SCALE-Sim v3 models cycle-accurate memory stalls, validated against RTL, making it more hardware-accurate.

Vegeta [46] flexible N:M sparsity. We validated 2:4 and 1:4 against Vegeta RTL with $\leq 5\%$ error.

Accelergy integration: Using CIFAR-10 input data and 16-bit quantized weights from AlexNet and ResNet-18, we compare total energy consumption and relative energy breakdown (GLB, NoC, PE array) between PnR results (65nm) and SCALE-Sim v3, observing deviations of 4.6% for Eyeriss and 4.8% for a general systolic array (8x8 array, OS dataflow). Accelergy provides precise energy estimations for various system states (e.g., *idle*, *read_rand*, *read_repeat*, *write_rand*, *write_repeat*, *write_cst_data*) using unit energy values validated through RTL, synthesis, and PnR. SCALE-Sim v3 generates a cycle-accurate workload trace, which Accelergy processes to derive action counts for each state, enabling accurate energy and power calculations. This integration ensures a precise representation of system behavior, as shown in Table III:

System State	PnR Energy	SCALE-Sim v3+Accelergy	Error
Idle (clk gating)	12.3	12.6	+2.4%
Active	315.8	308.5	-2.3%
Power gating	4.7	4.9	+4.3%

TABLE III: Validation of Accelergy integration across system states (idle, active, power-gated)

Workload	Multi core	Sparsity 2:4	Sparsity 1:4	Acce-lergy	Ramu-lator	Layout
Alexnet	1.83x	0.29x	0.14x	1.44x	1.19x	22.62x
Resnet-18	2.06x	0.30x	0.30x	1.09x	1.74x	3.6x
ViT-L	2.59x	0.47x	0.32x	1.06x	2.03x	18.88x
ViT-S	2.67x	0.62x	0.39x	1.17x	3.57x	19.01x
Mean	2.29x	0.42x	0.29x	1.19x	2.13x	16.03x

TABLE IV: Simulation time overhead on a TPU-v2 like configuration

Ramulator integration: The read and write request queues are modeled to mimic the number of in-flight transactions in the AXI bus in the Micron DDR4 Verilog model [83].

IX. Comparison with SCALE-Sim v2

A. Simulation time comparison

Simulation time overhead of SCALE-Sim v3 as compared to v2 is given in Table IV. The mean overhead of multi-core, 2:4 sparsity, 1:4 sparsity, Accelergy, Ramulator and Layout features are 2.29×, 0.42×, 0.29×, 1.19×, 2.13× and 16.03× respectively. Layout simulation overhead is large due to the detailed and accurate modeling of data layout in SRAM banks and bank conflicts.

B. Different designs found by SCALE-Sim v3 as compared to v2

This section demonstrates the necessity for enhancements in SCALE-Sim v3 where it leads to a different design as compared to v2.

Energy. SCALE-Sim v2 shows a 128×128 array is 6.53× faster than 32×32 for ViT-base, using only latency as a metric.

Metric	Resnet-50			RCNN			ViT-base		
	32x32	64x64	128x128	32x32	64x64	128x128	32x32	64x64	128x128
Latency (cycles/layer)	98721	35838	19501	126830	52243	29581	444970	130601	68160
Energy (mJ)	6.91	10.6	15.2	12.61	28.36	29.25	11.02	16.31	31.49
EdP (cycles x mJ/layer)	682162.11	379882.8	296415.2	159932	148161	865244.25	490356.9	2130102.31	2146358.4

TABLE V: Latency, Energy and EdP Comparison for 32x32, 64x64 and 128x128 arrays

Dataflow	Single core 128x128		16 cores having 32x32 PEs	
	Latency	Energy	Latency	Energy
Ratio ws/is	1.87	0.71	1.14	0.70

TABLE VI: Latency and energy overhead of multi-core

However, v3 finds that 32×32 is 2.86× more energy-efficient due to lower leakage from better utilization. For EdP, 64×64 outperforms both 128×128 and 32×32 for ViT-base as shown in Table V.

DRAM. SCALE-Sim v2 shows a 21% reduction in compute cycles for six ResNet18 layers using weight-stationary (WS) dataflow compared to output-stationary (OS). However, when factoring in DRAM stalls, OS dataflow exhibits 30.1% lower execution cycles compared to WS, highlighting the critical role of detailed DRAM analysis. We see similar results in memory-intensive layers and for designs with smaller request queue sizes. We will add an analysis in the revision.

Sparsity. In Figure 5, a latency-constrained (250,000 cycles) design requires 3.00 MB on-chip memory for a dense core. Assuming 2:4 sparsity, SCALE-Sim v3 enables a sparse core with 768 kB, significantly reducing area.

Multi-core. Comparing iso-compute designs (128×128 single-core vs. 16×32×32 multi-cores) for ViT-base, weight stationary outperforms input stationary by 1.87× (single-core) and 1.14× (multi-core) in latency. SCALE-Sim v2 favors weight stationary, but v3 allows further analysis before ruling out input stationary due to a small difference in latency, which performs 1.31× better in EdP for multi-core as shown in Table VI.

X. Related Work

Cycle accurate simulators. Cycle-accurate simulators are crucial tools for evaluating the performance and efficiency of AI accelerators, particularly those based on systolic arrays. Several existing simulators address various aspects of accelerator design as shown in Table I. uSystolic-SIM [26] is built on top of SCALE-Sim [24] which introduces a unique approach to cycle-level simulation by leveraging unary encoding to simplify computations, though it primarily targets byte-crawling architectures and does not extend to multi-core or sparse accelerators.

Multi-core simulators. Multi-core systolic accelerators have gained traction for their ability to enhance performance in data-parallel workloads. Tools like Timeloop [22] offer analytical modeling for multi-core accelerators with basic energy estimation through Accelergy [31], but they lack cycle-accurate simulation and data layout support. MuchiSim [86] is another notable tool that simulates large-scale multi-core and multi-chip accelerators with detailed modeling of inter-core communication and memory hierarchies. However, it lacks compute modeling and relies on third-party tools.

Sparse accelerator simulators. Sparse accelerators have gained prominence in recent times and various simulators have been developed to explore the design and performance of such accelerators. Sparseloop [85] extends Timeloop's capabilities by adding explicit support for sparse dataflows and mappings, enabling detailed exploration of sparsity patterns and their effects on accelerator performance. It models sparsity-related hardware optimizations including gating, skipping, and compression. However, it is analytical (modeling sparsity as a distribution) and lacks the support for cycle-accurate insights. STONNE [23] offers a cycle-accurate simulation of a specific unstructured sparse accelerator - SIGMA [30]. However, it does not support multi-core architectures, data layout, or detailed memory modeling.

XI. Conclusion

In this work, we introduced SCALE-Sim v3, a modular and cycle-accurate simulator designed to address the evolving needs of modern AI accelerator design and analysis. Our enhancements over SCALE-Sim v2 provide substantial improvements, including support for multi-core features, spatio-temporal partitioning, sparse matrix operations, memory interface modeling via Ramulator, energy/power estimation through Accelergy and on-chip data layout modeling. These features collectively enable comprehensive end-to-end system analysis, facilitating deeper insights into the performance, efficiency, and architectural trade-offs. We believe SCALE-Sim v3 will be a valuable tool for researchers and developers aiming to explore and optimize the design space of domain-specific AI accelerators.

Acknowledgements

This work was supported in part by ACE, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2022.

[2] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, 2023.

[3] Cerebras, "Wafer Scale Engine (WSE-2) Dataset." https://f.hubspotusercontent30.net/hubfs/8968533/WSE-2%20Datasheet.pdf.

[4] A. Firoozshahian, J. Coburn, R. Levenstein, R. Nattoji, A. Kamath, O. Wu, G. Grewal, H. Aepala, B. Jakka, B. Dreyer, *et al.*, "Mtia: First generation silicon targeting meta's recommendation systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–13, 2023.

[5] C. Lichtenau, A. Buyuktosunoglu, R. Bertran, P. Figuli, C. Jacobi, N. Papandreou, H. Pozidis, A. Saporito, A. Sica, and E. Tzortzatos, "Ai accelerator on ibm telum processor: Industrial product," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 1012–1028, 2022.

[6] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *2020 IEEE high performance extreme computing conference (HPEC)*, pp. 1–12, IEEE, 2020.

[7] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–382, IEEE, 2018.

[8] G. Sun, S. Kang, and S.-W. Jun, "Burstz+: Eliminating the communication bottleneck of scientific computing accelerators via accelerated compression," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 2, pp. 1–34, 2022.

[9] A. Bocco, Y. Durand, and F. De Dinechin, "Smurf: Scalar multiple-precision unum risc-v floating-point accelerator for scientific computing," in *Proceedings of the Conference for Next Generation Arithmetic 2019*, pp. 1–8, 2019.

[10] R. Weber, A. Gothandaraman, R. J. Hinde, and G. D. Peterson, "Comparing hardware accelerators in scientific applications: A case study," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 58–68, 2010.

[11] M. Ujaldón, "Hpc accelerators with 3d memory," in *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pp. 320–328, IEEE, 2016.

[12] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 356–367, 2012.

[13] K. A. Britt, F. A. Mohiyaddin, and T. S. Humble, "Quantum accelerators for high-performance computing systems," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–7, IEEE, 2017.

[14] K. Li, W. Mao, J. Zhou, B. Li, Z. Yang, S. Yang, L. Du, S. Huang, and H. Yu, "A vector systolic accelerator for multi-precision floating-point high-performance computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 10, pp. 4123–4127, 2022.

[15] L. Yang, R. M. Radway, Y.-H. Chen, T. F. Wu, H. Liu, E. Ansari, V. Chandra, S. Mitra, and E. Beigné, "Three-dimensional stacked neural network accelerator architectures for ar/vr applications," *IEEE Micro*, vol. 42, no. 6, pp. 116–124, 2022.

[16] H. E. Sumbul, J.-s. Seo, D. H. Morris, and E. Beigne, "A fully-digital and row-pipelined compute-in-memory neural network accelerator with soc-level benchmarking for ar/vr applications," *IEEE Micro*, 2023.

[17] S. Li, Y. Zhao, C. Li, B. Guo, J. Zhang, W. Zhu, Z. Ye, C. Wan, and Y. C. Lin, "Fusion-3d: Integrated acceleration for instant 3d reconstruction and real-time rendering," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 78–91, IEEE, 2024.

[18] S. Krishnan, Z. Wan, K. Bhardwaj, P. Whatmough, A. Faust, S. Neuman, G.-Y. Wei, D. Brooks, and V. J. Reddi, "Automatic domain-specific soc design for autonomous unmanned aerial vehicles," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 300–317, IEEE, 2022.

[19] Y. Hao, Y. Gan, B. Yu, Q. Liu, Y. Han, Z. Wan, and S. Liu, "Orianna: An accelerator generation framework for optimization-based robotic applications," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 813–829, 2024.

[20] Q. Liu, Z. Wan, B. Yu, W. Liu, S. Liu, and A. Raychowdhury, "An energy-efficient and runtime-reconfigurable fpga-based accelerator for robotic localization systems," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 01–02, IEEE, 2022.

[21] S. Liu, Z. Wan, B. Yu, and Y. Wang, *Robotic computing on fpgas*. Springer, 2021.

[22] 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.

[23] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "Stonne: Enabling cycle-level microarchitectural simulation for dnn inference accelerators," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 201–213, IEEE, 2021.

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

[25] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[26] D. Wu and J. San Miguel, "usystolic: Byte-crawling unary systolic array," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 12–24, IEEE, 2022.

[27] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[28] 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.

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

[30] 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.

[31] 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.

[32] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[33] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[34] J. Tong, A. Itagi, P. Chatarasi, and T. Krishna, "Feather: A reconfigurable accelerator with data reordering support for low-cost on-chip dataflow switching," 2024.

[35] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1237–1254, IEEE, 2022.

[36] T. Ma, Y. Feng, X. Zhang, and Y. Zhu, "Camj: Enabling system-level energy modeling and architectural exploration for in-sensor visual computing," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–14, 2023.

[37] H. Guo, Y. Zhao, Z. Li, Y. Hao, C. Liu, X. Song, X. Li, Z. Du, R. Zhang, Q. Guo, *et al.*, "Cambricon-u: A systolic random increment memory architecture for unary computing," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 424–437, 2023.

[38] A. Samajdar, E. Qin, M. Pellauer, and T. Krishna, "Self adaptive reconfigurable arrays (sara) learning flexible gemm accelerator configuration and mapping-space using ml," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 583–588, 2022.

[39] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re, *et al.*, "Deja vu: Contextual sparsity for efficient llms at inference time," in *International Conference on Machine Learning*, pp. 22137–22176, PMLR, 2023.

[40] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[41] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 806–814, 2015.

[42] T. Gale, D. Narayanan, C. Young, and M. Zaharia, "Megablocks: Efficient sparse training with mixture-of-experts," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 288–304, 2023.

[43] H. Shen, H. Meng, B. Dong, Z. Wang, O. Zafrir, Y. Ding, Y. Luo, H. Chang, Q. Gao, Z. Wang, *et al.*, "An efficient sparse inference software accelerator for transformer-based language models on cpus," *arXiv preprint arXiv:2306.16601*, 2023.

[44] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures," *ACM SIGMETRICS Performance Evaluation Review*, vol. 50, no. 1, pp. 33–34, 2022.

[45] "NVIDIA Ampere Architecture." https://www.nvidia.com/en-in/data-center/ampere-architecture/, 2020.

[46] G. Jeong, S. Damani, A. R. Bambhaniya, E. Qin, C. J. Hughes, S. Subramoney, H. Kim, and T. Krishna, "Vegeta: Vertically-integrated extensions for sparse/dense gemm tile acceleration on cpus," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 259–272, IEEE, 2023.

[47] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, "Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pp. 600–614, 2019.

[48] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 15–28, IEEE, 2018.

[49] Google, "TPUv5e." https://cloud.google.com/tpu/docs/v5e.

[50] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," in *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4, IEEE, 2017.

[51] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 1116–1121, IEEE, 2017.

[52] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , vol. 39, no. 8, pp. 1555–1571, 2019.

[53] H. T. 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 Philadelphia, PA, USA, 1979.

[54] H.-T. Kung, *Why systolic architecture?* Design Research Center, Carnegie-Mellon University, 1982.

[55] C. Mead and L. Conway, "Introduction to vlsi systems," 1980.

[56] B. K. Saptalakar, D. Kale, M. Rachannavar, and M. Pavankumar, "Design and implementation of vlsi systolic array multiplier for dsp applications," *International Journal of Scientific Engineering and Technology*, vol. 2, no. 3, pp. 156–159, 2013.

[57] D.-F. Chiper, M. Swamy, M. O. Ahmad, and T. Stouraitis, "A systolic array architecture for the discrete sine transform," *IEEE transactions on signal processing*, vol. 50, no. 9, pp. 2347–2354, 2002.

[58] P. Quinton, *The systematic design of systolic arrays*. PhD thesis, INRIA, 1983.

[59] K. T. Johnson, A. R. Hurson, and B. Shirazi, "General-purpose systolic arrays," *Computer*, vol. 26, no. 11, pp. 20–31, 1993.

[60] OpenAI, "AI and compute." https://openai.com/index/ai-and-compute/.

[61] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[62] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805* , 2023.

[63] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[64] W. Zheng, R. Song, X. Guo, C. Zhang, and L. Chen, "Genad: Generative end-to-end autonomous driving," in *European Conference on Computer Vision*, pp. 87–104, Springer, 2025.

[65] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, *et al.*, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17853–17862, 2023.

[66] S. Atakishiyev, M. Salameh, H. Yao, and R. Goebel, "Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions," *IEEE Access*, 2024.

[67] A. Rahman, J. M. J. Valanarasu, I. Hacihaliloglu, and V. M. Patel, "Ambiguous medical image segmentation using diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11536–11546, 2023.

[68] Y. Zhang, H. Jiang, Y. Miura, C. D. Manning, and C. P. Langlotz, "Contrastive learning of medical visual representations from paired images and text," in *Machine Learning for Healthcare Conference*, pp. 2–25, PMLR, 2022.

[69] Y. Gao, X. Xiang, N. Xiong, B. Huang, H. J. Lee, R. Alrifai, X. Jiang, and Z. Fang, "Human action monitoring for healthcare based on deep learning," *Ieee Access*, vol. 6, pp. 52277–52285, 2018.

[70] Nvidia, "DGX GH200 for Large Memory AI Supercomputer." https://www.nvidia.com/en-in/data-center/dgx-gh200/.

[71] E. Talpes, D. Williams, and D. D. Sarma, "Dojo: The microarchitecture of tesla's exa-scale computer," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, pp. 1–28, IEEE Computer Society, 2022.

[72] J. Seo and J. Kong, "Versa: Versatile systolic array architecture for sparse and dense matrix multiplications," *Electronics*, vol. 13, no. 8, p. 1500, 2024.

[73] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A modern, modular, and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 23, no. 1, pp. 112–116, 2023.

[74] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[75] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 14–27, 2019.

[76] A. Symons, L. Mei, S. Colleman, P. Houshmand, S. Karl, and M. Verhelst, "Towards heterogeneous multi-core accelerators exploiting fine-grained scheduling of layer-fused deep neural networks," *arXiv preprint arXiv:2212.10612*, 2022.

[77] A. Nandakumar, S. Shao, and B. Nikolic, "Accelerating deep learning on heterogenous architectures," 2022.

[78] O. Spantidi, G. Zervakis, S. Alsalamin, I. Roman-Ballesteros, J. Henkel, H. Amrouch, and I. Anagnostopoulos, "Targeting dnn inference via efficient utilization of heterogeneous precision dnn accelerators," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 112–125, 2022.

[79] J. H. Lau, *Chip on board: technology for multichip modules*. Springer Science & Business Media, 1994.

[80] D. A. Doane and P. Franzon, *Multichip module technologies and alternatives: the basics*. Springer Science & Business Media, 2013.

[81] 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*, vol. 45, no. 2, pp. 320–332, 2017.

[82] V. N. Sekhar, M. D. Kumar, S. K. Tippabhotla, B. C. Rao, I. C. Daniel, S. C. Chong, and V. S. Rao, "Multi-chip stacked memory module development using chip to wafer (c2w) hybrid bonding for heterogeneous integration applications," in *2024 IEEE 74th Electronic Components and Technology Conference (ECTC)*, IEEE, 2024.

[83] Micron Technology, "Micron DDR4 Verilog Model," 2018.

[84] "Nvidia a100 tensor core gpu architecture." https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[85] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical approach to sparse tensor accelerator modeling," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1377–1395, IEEE, 2022.

[86] M. Orenes-Vera, E. Tureci, M. Martonosi, and D. Wentzlaff, "Muchisim: A simulation framework for design exploration of multi-chip manycore systems," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 48–60, IEEE, 2024.

APPENDIX

A. Abstract

The artifact appendix section describes how to access the artifacts of the SCALE-Sim v3 simulator introduced in Figure 1. The artifact also describes how to reproduce the experiments in Section III (Figure 3), Section IV (Figure 5, Figure 7, and Figure 8), Section V (Figure 9 and Figure 10), Section VI (Figure 12 and Figure 13) conducted through the SCALE-Sim v3 framework.

B. Artifact check-list (meta-information)

- **Program: Python3 and C++11 compiler (e.g., clang++, g++-5)**
- **Hardware: Linux-based**
- **Metrics: Latency, Memory, Throughput and Energy**
- **Output: Compute and Bandwidth csv reports in addition to sparsity report, Ramulator report and Accelergy report**
- **How much disk space required (approximately)?: 10 GB**
- **How much time is needed to prepare workflow (approximately)?: 5 minutes**
- **How much time is needed to complete experiments (approximately)?: 15-20 hours**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: MIT License (LICENCE file in the GitHub repository)**

C. Description

1) How to access: The artifact is uploaded to Zenodo (DOI: 10.5281/zenodo.15170910)

2) Hardware dependencies: The artifact has been tested on Linux-based systems

3) Software dependencies: Python3 and C++11 compiler (e.g., clang++, g++-5)

D. Installation

For artifact evaluation, please follow the following steps:

```
$ git clone
https://github.com/scalesim-project/scale-sim-v2.git
$ cd scale-sim-v2
$ git checkout v3_artifact
$ pip install -r requirements.txt
$ pip install -e .
$ export PYTHONPATH=.:$PYTHONPATH
```

E. Experiment workflow

Now that our environment is set up, we will run the SCALE-Sim v3 artifact and evaluate the result. The first step is to write config and topology files. The config and topology files for the experiments are provided under the folder *configs* and *topologies* respectively. In addition, the layout files are provided under the folder *layouts*.

For config file and topology file customization for sparsity, see the Experiment customization section.

To evaluate the experiment more conveniently, we provide python and shell scripts for each part, which will be introduced in the next section.

F. Evaluation and expected results

1) Multi-core (Figure 3):

```
$ python3 multi-core/analytical.py
$ python3 multi-core/plot_scatter.py
```

Figures will be produced in *multi-core* directory named *compute_optimized_tradeoff.pdf* and *memory_optimized_tradeoff.pdf* reproducing figures 3a and 3b respectively.

2) Sparsity (Figure 5):

```
$ python3 sparsity/compute_cycles_plot_128x128.py
```

A figure will be produced in *sparsity* directory named *resnet18_compute_cycles.pdf* reproducing Figure 5.

Please note that the python file plots based on experimental data collected in *sparsity/sparsity_results_128x128* folder. To reproduce the experimental data, please follow the *README.md* file in the *sparsity folder*. The expected time to finish all the runs is around 10-12 hours. The same experimental data has been used for the next subsection as well.

3) Sparsity (Figure 7:

```
$ python3 sparsity/memoryplot.py
```

A figure will be produced in *sparsity* directory named *memoryplot.pdf* reproducing Figure 7.

4) Sparsity (Figure 8):

```
$ bash sparsity/run_vit_compute.sh
```

A figure will be produced in *sparsity* directory named *vit_compute_cycles_scatterplot.pdf* reproducing Figure 8.

5) Layout (Figure 12, Figure 13):

```
$ bash layout_plots/run_layout.sh
```

Figures will be produced in *layout_plots* directory named *multi_bank_vs_bandwidth_model_resnet18.pdf* and *multi_bank_vs_bandwidth_model_vit.pdf* reproducing Figure 12 and Figure 13 respectively.

6) Ramulator : Please follow the given steps:

```
$ git checkout dev-ramulator-merge
$ pip install -r requirements.txt
$ git submodule update --init --recursive
$ cd submodules
$ cp -r ../scripts/ramulator_patch/* ./ramulator/
$ cd ramulator
$ make -j <num_jobs>
```

To reproduce Figure 9, use the following command:

```
$ cd ../../
$ source generate_fig9_ramulator_mem_bw_plot.sh
```

To reproduce Figure 10, use the following command:

```
$ source generate_fig10_ramulator_stall_plot.sh
```

G. Experiment customization

To run customized experiments, users can write their own config and topology files and feed it to the program. Let's take sparsity as an example. In a config file, the full list of sparsity parameters include:

```
SparsitySupport : enable or disable sparsity support
SparseRep: sparsity representation (csr, csc,
ellpack_block)
OptimizedMapping : enable row-wise sparsity,
the default is layer-wise sparsity
BlockSize : M in N:M ratio for row-wise sparsity
```

The sparsity topology files look as follows:

```
Layer Name, M, N, K, Sparsity,
GEMM_1, 3, 5, 16, 3:4,
GEMM_2, 1, 5, 16, 1:4,
```

where, M, N, and K are regular GEMM dimensions. The entries in *Sparsity* are in the N:M format corresponding to N:M row-wise sparsity.

H. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae