Scale-out Systolic Arrays

AHMET CANER YÜZÜGÜLER and CANBERK SÖNMEZ, EPFL, Switzerland
MARIO DRUMOND, CodeDepot, Switzerland
YUNHO OH, Korea University, South Korea
BABAK FALSAFI and PASCAL FROSSARD, EPFL, Switzerland

Multi-pod systolic arrays are emerging as the architecture of choice in DNN inference accelerators. Despite their potential, designing multi-pod systolic arrays to maximize effective throughput/Watt—i.e., throughput/Watt adjusted when accounting for array utilization—poses a unique set of challenges. In this work, we study three key pillars in multi-pod systolic array designs, namely array granularity, interconnect, and tiling. We identify optimal array granularity across workloads and show that state-of-the-art commercial accelerators use suboptimal array sizes for single-tenancy workloads. We, then evaluate the bandwidth/latency trade-offs in interconnects and show that Butterfly networks offer a scalable topology for accelerators with a large number of pods. Finally, we introduce a novel data tiling scheme with custom partition size to maximize utilization in optimally sized pods. We propose *Scale-out Systolic Arrays*, a multi-pod inference accelerator for both single- and multi-tenancy based on these three pillars. We show that SOSA exhibits scaling of up to 600 TeraOps/s in effective throughput for state-of-the-art DNN inference workloads, and outperforms state-of-the-art multi-pod accelerators by a factor of 1.5×.[1]

CCS Concepts: • **Computer systems organization → Systolic arrays;**

Additional Key Words and Phrases: DNN accelerators, scale-out architecture

ACM Reference format:
Ahmet Caner Yüzügüler, Canberk Sönmez, Mario Drumond, Yunho Oh, Babak Falsafi, and Pascal Frossard. 2023. Scale-out Systolic Arrays. *ACM Trans. Arch. Code Optim.* 20, 2, Article 27 (March 2023), 25 pages.
https://doi.org/10.1145/3572917

1 INTRODUCTION

Deep Neural Networks (DNN) are widely employed in various domains from computer vision to natural language processing. With the slowdown in silicon scaling and the increase in demand for throughput with a tight tail latency, designers have resorted to custom hardware accelerators for DNN workloads [11, 20, 21, 27, 40, 48, 57, 61]. While many prior works have adopted spatial architectures that implement certain dataflows (e.g., Eyeriss [8], MAERI [41], SIGMA [50]), systolic

[1]New Paper, Not an Extension of a Conference Paper.

Authors' addresses: A. C. Yüzügüler, C. Sönmez, B. Falsafi, and P. Frossard, EPFL, Rte Cantonal, Lausanne, VD, Switzerland, CH-1015; emails: {ahmet.yuzuguler, canberk.sonmez, babak.falsafi, pascal.frossard}@epfl.ch; M. Drumond, CodeDepot, Avenue d.Ouchy 4, Lausanne, VD, Switzerland, CH-1006; email: mario.drumond@pm.me; Y. Oh, Korea University, 145 Anam-ro, Seongbuk-gu, Seoul, South Korea, 02841; email: yunho_oh@korea.ac.kr.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1544-3566/2023/03-ART27 $15.00
https://doi.org/10.1145/3572917

arrays have emerged as the architecture of choice in commercial silicon products (e.g., Google's TPU [35], Tesla's FSD chip [5]) thanks to their exceptional power efficiency.

Since their inception, systolic arrays have dramatically evolved to achieve remarkable peak throughput with higher levels of silicon integration, power envelopes, and memory bandwidth. Unfortunately, translating peak throughput to higher utilization has remained challenging for designers because of mismatches between workload requirements and array size. Google's TPU v1 and v4 reported utilizations of about 22% [35] and 33% [33], respectively, while processing DNN models due their large systolic array-based matrix multiplication unit.

To improve the total cost of ownership and support multi-tenancy, recent accelerators—such as TPU v3 and v4—incorporate a few coarse-grain systolic-array *pods* connected through shared memory on a single die [4, 33]. While these *multi-pod* accelerators achieve much better utilization over their monolithic counterparts with multi-tenancy, variability in array size requirements in workloads remains a fundamental limitation to utilization in a few coarse-grain pods. In contrast, multi-pod designs with minimally sized arrays [38] target maximum utilization. Unfortunately, these designs compromise the inference accelerator's power efficiency by over-provisioning overall on-chip memory [18] (e.g., 8×8 arrays incur $5 - 10\times$ more memory accesses than 128×128 arrays). Therefore, even at high utilization, such multi-pod designs achieve inferior throughput/Watt relative to designs with coarse-grain pods [55].

In this paper, we make the observation that optimally sizing systolic arrays in multi-pod accelerators is key to achieving optimal *effective* throughput/Watt—i.e., the throughput/Watt adjusted when accounting for array utilization—in single workloads. Much like multi-threaded CPUs targeting high single-threaded performance, multi-pod accelerators with optimally sized pods seamlessly support efficient multi-tenancy in workloads.

A multi-pod inference accelerator with a large number of optimally sized pods requires a scalable interconnect with favorable bandwidth and latency characteristics. Much prior work has focused on the use of interconnects in inference accelerators. While many advocate Mesh [9, 10, 23, 56] or H-tree [38, 60], these topologies lack sufficient bisection bandwidth to support a large number of pods. Others advocate Benes [50], which requires a long round-trip latency on requests and may adversely impact the overall execution time. As such, interconnecting a large number of pods remains an open research problem. Similarly, an accelerator with a large number of optimally sized pods also creates an opportunity to customize partitioning for input activations. Prior work using multi-pod accelerators either does not partition the input [4] or uses sub-optimal partitions [12]. We make the observation that an input partition size that is optimal for a given array granularity exposes the available data parallelism within the DNN layers, thereby maximizing the available number of tiles that can be extracted from a single workload.

This paper proposes ***Scale-out Systolic Arrays (SOSA)***, a multi-pod architecture supporting both single- and multi-tenancy workloads. SOSA incorporates optimally sized systolic arrays for a spectrum of state-of-the-art inference workloads from computer vision to NLP. SOSA relies on a Butterfly network as a scalable fabric to interconnect systolic arrays and memory banks with a high bisection and low latency, and employs a novel data tiling scheme that maximizes the available number of tiles for a given array size. We use in-house simulators with timing models and synthesis tools for area and power to show that SOSA outperforms both coarse- and fine-grain multi-pod inference accelerators while achieving strong scalability.

In this paper, we make the following contributions:

- Our design space exploration across a wide spectrum of workloads indicates that SOSA with 32×32 pods achieves $1.5\times$ higher effective throughput/Watt than 128×128 pods used in state-of-art commercial accelerators (e.g., TPUs [25]) for single workloads.

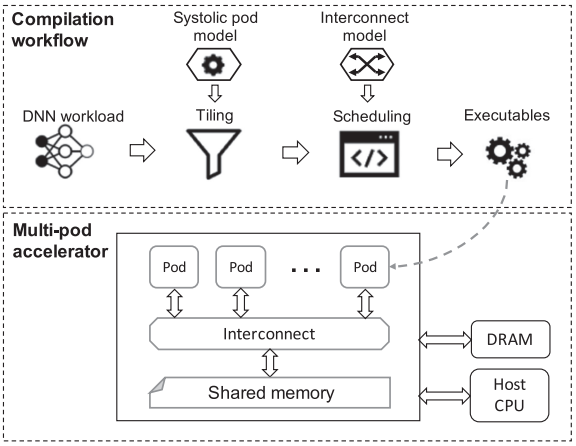


Fig. 1. General overview of a multi-pod systolic array.

- Our baseline SOSA with a Butterfly network achieves 2× higher effective throughput than one with a Benes network and 2.3× less power than one with a crossbar.
- Customizing the tiling strategy in multi-pod accelerators improves utilization by up to 5× over a system with no partitioning for activations.
- We show that the proposed design can achieve strong scaling up to 600 TeraOps/s at a TDP of 400 Watts for computationally-intensive DNN models such as Resnet.
- SOSA increases effective throughput by 1.44× with multi-tenancy over single-tenancy.

For reproducibility, we open-source our cycle-accurate DNN accelerator simulator.[2] To the best of our knowledge, this is the first open-source simulator for multi-pod systolic array accelerators.

2 WHY SCALE-OUT SYSTOLIC ARRAYS?

A myriad of prior works have proposed dataflow optimizations such as row-stationary scheduling [8], flexible interconnection [41, 46, 50], and novel data partitioning and scheduling schemes [23, 40]. These optimizations have two common microarchitectural requirements. First, each processing element must have a scratchpad memory to reuse data temporally. Second, interconnection between processing elements must be reconfigurable to support efficient data mapping and scheduling schemes. As a result of these two requirements, majority of power consumption per MAC operation is spent on memory access and data movement [8, 23, 40], which places an upper bound on the power efficiency of such accelerators.

Systolic arrays overcome the limitations of dataflow architectures by adopting simplistic processing element microarchitectures (i.e., without large register files or scratchpad memory) and implementing static interconnection between processing elements. As such, systolic arrays achieve higher power efficiency and peak throughput compared to dataflow architectures. Moreover, recent proposals to couple multiple systolic arrays in a single die (i.e., *multi-pod* designs [4, 33, 38]) allows benefiting data and task-level parallelism, further improving the gain from provisioned silicon.

Figure 1 depicts the overall diagram of a multi-pod inference accelerator [4, 25, 38], where each *pod* includes a systolic array and the necessary glue interface to an interconnect connecting the pods together to memory and the peripherals. Like recent custom inference accelerators

[2] https://github.com/yuezuegu/sosa-compiler.

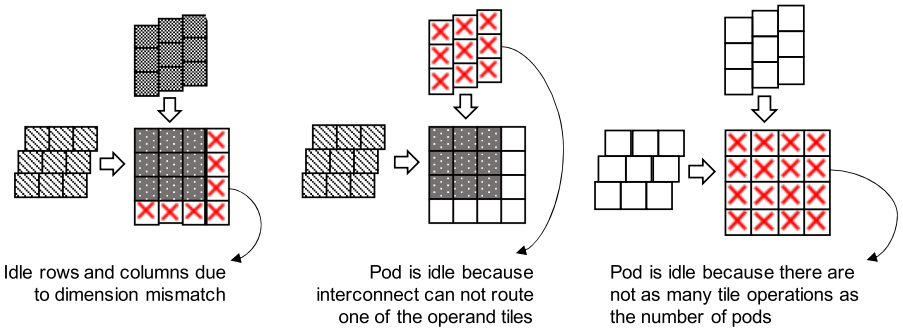


Fig. 2. Three main factors of underutilization in systolic arrays.

(e.g., Graphcore [1], Brainwave [21]), models and data are stored in on-chip memory with an interface to off-chip DRAM and a host CPU. To map workloads to multiple systolic arrays, DNN layers are first partitioned into tile operations of sizes that match a pod's array dimensions. Using information about connectivity and latency about the interconnect, a static scheduler optimizes the mapping of tile operations to pods to maximize parallelism and throughput.

A multi-pod accelerator's effective throughput is a function of the overall utilization of processing elements both within and across pods. Therefore, maintaining a high utilization not only maximizes the gain from provisioned silicon resources but also improves the overall performance of an accelerator. Figure 2 illustrates three main causes of the underutilization in a multi-pod accelerator: (1) dimension mismatch [55] between array and workload resulting in underutilization within a pod, (2) poor connectivity resulting in underutilization across pods, and (3) sub-optimal tiling resulting in underutilization both within and across pods.

Utilization within a pod highly depends on the pod's systolic array granularity and the layer dimensions of a DNN workload. Prior multi-pod accelerators [4, 25, 35] opt for larger array dimensions to reduce access to on-chip memory, provisioning higher power for processing elements. Unfortunately, larger arrays also increase the likelihood that the workload's layer dimensions are smaller than the number of array's rows or columns, resulting in idle processing elements and wasted throughput/Watt. In contrast, minimizing the array dimensions per pod reduces the mismatch between the workloads and the array, resulting in improved utilization. Smaller systolic arrays, however, increase the power required for on-chip memory access, undermining the overall throughput/Watt. In this paper, we show that the optimal array size for DNN workloads is an order of magnitude smaller than that widely adopted by academia and industry.

The interconnect also plays a key role in utilization and effective throughput in multi-pod accelerators [38]. To achieve a scalable multi-pod accelerator architecture with high utilization, the interconnect should allow transferring data tiles between systolic pods and memory banks with minimal contention at high efficiency. Therefore, the selected network topology should satisfy several design requirements such as high bisection bandwidth, high combinatorial power, low latency, and good scalability. As these requirements are often contradictory to each other, the network topologies should be carefully evaluated from the perspective of scale-out architectures to ensure high performance and scalability of the accelerators.

Finally, the tile size for each pod fundamentally impacts utilization in multi-pod accelerators and should be tuned with care. We observe that conventional approaches to tiling for systolic arrays [4, 12] fall well short of generating a sufficient number of tile operations to populate a large number of pods resulting in idle arrays during execution. On the one hand, choosing large tiles limits the overall number of tile operations and results in idle pods. On the other hand, choosing a small tiling size may introduce underutilization within pods due to internal buffering times.

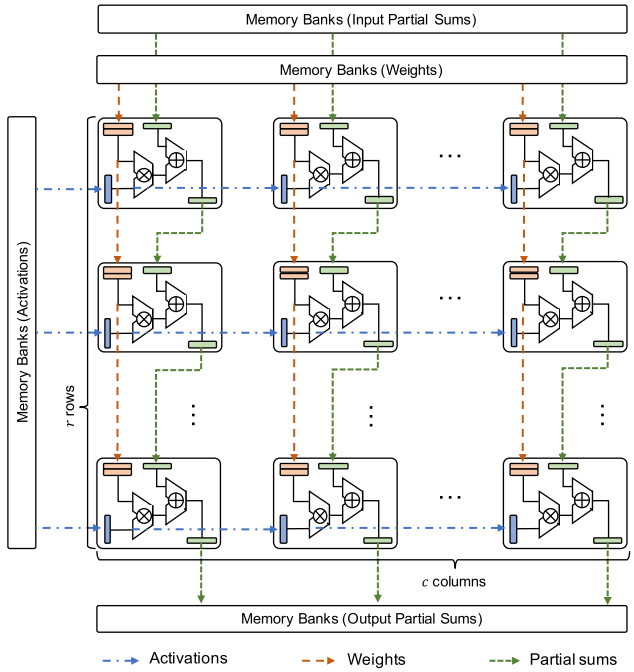


Fig. 3. A weight-stationary systolic array with r rows and c columns. Activations are assumed to traverse along the rows from left to right, weights and partial sums traverse along the columns from top to bottom.

3 KEY PILLARS OF MULTI-POD ACCELERATOR DESIGN

In Section 2, we argued that the optimal array size of systolic arrays is smaller than those in conventional DNN accelerators, which gives rise to multi-pod architectures. In this section, we introduce our approach regarding the key pillars of energy-efficient multi-pod DNN accelerators. First, we show that the optimal systolic array size is a function of data dimensions in target DNN workloads and perform a design space exploration of the optimal array sizes for various scenarios. Second, we introduce an analysis of various interconnection networks and our decision for the multi-pod DNN accelerators.

3.1 Optimal Systolic Array Size

DNN models typically comprise a number of layers of various types, such as convolutional, fully-connected, and attention, whose computation can be expressed as **general matrix multiplication (GEMM)** operations. Without loss of generality, we assume that GEMM operations are in the form of $XW + P_{in} = P_{out}$, where X and W denote DNN activations and weights; P_{in} and P_{out} denote input and output partial sums, respectively.

Among various versions of systolic arrays [36], we focus on the widely adopted weight-stationary design [35, 37, 38]. Figure 3 depicts a weight-stationary systolic array, in which the **processing elements (PEs)** are placed in a grid of r rows and c columns. The PEs are connected to their neighbors along rows and columns through uni-directional point-to-point links. To perform a GEMM operation, a systolic array first fetches the weight matrix row by row and stores them in dedicated registers in PEs. Then, in every cycle, the PEs in the left-most column fetch activations from the memory banks and pass them to the next column. Likewise, the PEs at the top row fetch the input partial sums from the memory banks, perform a **multiply-and-accumulate**

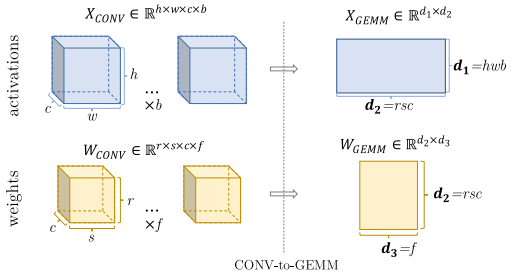
(MAC) operation, and pass the resulting partial sum to the row below. The operation continues with activations flowing from left to right, partial sums flowing from top to bottom, and weights staying stationary. The PEs at the bottom row produce the final results and write them to the memory banks.

In systolic arrays, only the PEs at the edges access the memory banks, whereas those in the middle fetch their operands from their neighbors. Therefore, the number of memory accesses increases linearly with the array dimensions, while the number of MAC operations grows quadratically. As such, large systolic arrays incur relatively lower memory overhead per MAC operation, which improves the overall power efficiency. However, as we increase r and c, the array dimensions exceed matrix dimensions in DNN workloads, resulting in idle rows and columns. If the weight matrix dimensions are smaller than the array dimensions, the excess rows and columns become idle, resulting in underutilization. Moreover, in systolic array designs with double buffering [53], the entire array stalls between operations if the matrix multiplication takes fewer cycles than the weight buffering time. Because the execution time of a GEMM operation is equal to the first dimension of the activation matrix d_1 (ignoring pipeline latencies), and the weight buffering time is proportional to the number of rows in the array r (assuming weights are fetched row by row), choosing $r > d_1$ also results in underutilization. In short, systolic arrays with large numbers of rows and columns are more likely to suffer from underutilization than small ones due to dimension mismatches.

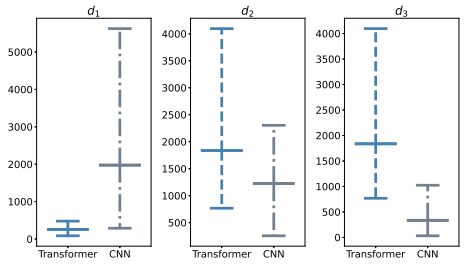
To gain insights into the distinguishing characteristics of different DNN types, we analyze the operand dimensions of the related GEMM operations. While fully-connected and self-attention layers in Transformer models can be directly represented as GEMM operations, convolutional layers must first be converted, which transforms the operand dimensions. Figure 4(a) illustrates the process of CONV-to-GEMM conversion [32], where $h \times w$ and $r \times s$ denote the window size of activations (X_{CON}) and weights (W_{CONV}), c, f, and b denote the number of input channels, filters, and batch size, respectively. The four-dimensional activation and weight tensors are rearranged to obtain two-dimensional X_{GEMM} and W_{GEMM} matrices. Their multiplication eventually produces the results of the convolutional operation. We denote the dimensions of the X_{GEMM} and W_{GEMM} matrices with d_1, d_2, and d_3, where $d_1 = hwb$, $d_2 = rsc$, and $d_3 = f$. For fully-connected and self-attention layers, d_1, d_2, and d_3 represent the batch size, number of features, and number of filters.

Figure 4(b) shows the range of the values for d_1, d_2, and d_3 in state-of-the-art CNN and Transformer models. Although the dimensions of the DNN layers vary both across and within models, we make the following observations. First, GEMM operations in CNN models have significantly larger values for d_1 (15× on average) than Transformer models. This large difference is mainly due to the convolutional reuse, i.e., filters in convolutional layers stride across input images. As a result, CNN models run at higher utilization on systolic arrays with a larger number of rows than columns. Our second observation is that GEMM operations in Transformer models have significantly larger values for d_3 (about 6×) than convolutional layers, which indicates that Transformer models are better suited than CNNs for systolic arrays with more columns than rows.

Although our analysis on data dimensions gives us insights into the contrasting requirements of different DNN types, finding the optimal array size requires a more complex hardware modeling including both utilization and power efficiency. To that end, we devised an optimization metric called effective throughput/Watt, that takes these two variables into account. Due to the utilization component of the effective throughput/Watt metric, the optimal array size is sensitive to the selection of target workloads. Therefore, we conduct a design space exploration for three cases: we first study the CNN and Transformer models separately, and then a mixture of both types. To compare our proposed design choice, we pick four baseline array sizes that represent the majority of designs in industry and academia: 512×512, which represents the monolithic systolic array; 256×256, which represents Google's TPU v1 [35], 128×128, which represents TPU v2 [25] and



(a) Illustration of CONV-to-GEMM conversion. Four-dimensional activation and weight tensors of a convolution operation are reshaped into two-dimensional matrices.



(b) Range of the matrix dimensions for the GEMM operations in BERT and CNN models. d_1 denotes the first dimension of activation matrix (X_{GEMM}) and d_2 and d_3 denote the first and second dimension of weight matrix (W_{GEMM}). Horizontal lines show 10th percentile, average, and 90th percentiles.

Fig. 4. Operand dimensions of GEMM operations in DNN workloads.

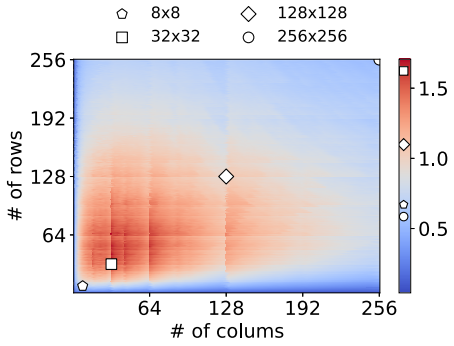
AI-MT [4], and 8×8, which represents the Maestro [38]. For this design space exploration, we use our systolic hardware model to obtain the power efficiency and utilization values, and then calculate the effective throughput as peak throughput multiplied by utilization. The correctness of the systolic array model used in this section is validated against the functional simulations of our RTL design. Our design space exploration is isopower because DNN inference accelerators are typically bound by power consumption because of their dense arithmetic formats [61].

Figure 5(a) shows the design space exploration for CNN models. Because the number of filters in CNN models is typically limited compared to the number of filter reuse and the number of features as shown in Figure 4(b), we observe that optimal design points have a large number of rows and a small number of columns. In fact, the design point with the highest effective throughput for CNN models is 66×32, which is about $1.3\times$ better than any other baselines. In contrast, Figure 5(b) shows the design space for Transformer models. Because the number of filter reuse in Transformer models is typically limited compared to the number of filters and number of features as shown in Figure 4(b), we observe that the optimal design points have a large number of columns and a small number of rows. More specifically, the design point with the highest effective throughput for Transformer models is 20×128, which is also about $1.7\times$ better than any other baselines. These two scenarios show that two widely used DNN types, namely CNNs and Transformers, benefit from non-square array shapes, which contradicts the common design patterns in industry or academia.

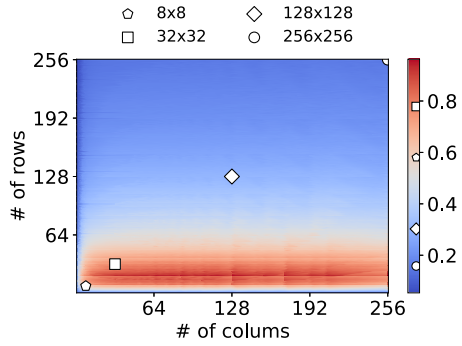
Figure 5(c) shows the design space for a scenario that targets both CNN and Transformer models. In this case, data dimensions of both model types have an impact on the shape of the design space: the areas with large array dimensions exhibit low effective throughput/Watt due to underutilization. Likewise, areas with very small dimensions also have low effective throughput/Watt due to poor power efficiency. We identify that only a small part of the design exhibits high effective throughput/Watt, where the number of rows and columns are in the order of ten to hundreds. In fact, the highest effective throughput/Watt is obtained with array dimensions of 20×32. Without loss of generality, we choose 32×32 as the array size in our design to facilitate implementation and connectivity to memory (e.g., alignment to cache block size).

3.2 Interconnection Network

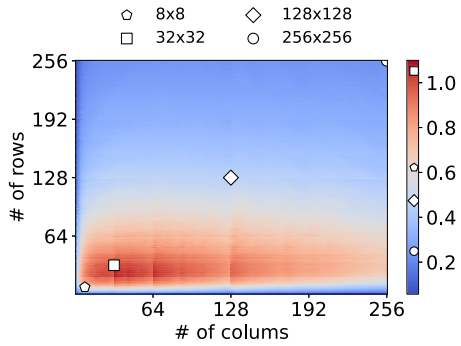
For the interconnect between systolic arrays and memory banks, we prioritize four design requirements. First, the interconnect should provide sufficient bisection bandwidth to allow continuous data read and write for all systolic pods simultaneously. Second, the interconnect should facilitate



(a) Effective throughput (TeraOps/s) per Watt for CNN models.



(b) Effective throughput (TeraOps/s) per Watt for Transformer models.



(c) Effective throughput (TeraOps/s) per Watt for both CNN and Transformer models.

Fig. 5. Design space exploration for systolic arrays. x- and y-axes are the number of columns and rows in a systolic array and the colormap represents the effective throughput (TeraOps/s) per Watt. CNN models are Inception-v3, ResNet50, ResNet101, ResNet152, DenseNet121, DenseNet169, and DenseNet201 with input image sizes of 224×224, 256×256, and 299×299. Transformer models are BERT-mini, small, medium, base, and large with sequence lengths of 10, 20, 40, 60, 80, 100, 200, 300, 400, 500 as taken from [63]. Ripples and discrete lines in the images occur due to the discretization during data tiling.

high combinatorial power (i.e., the ratio of realizable input-output permutations [6]) to connect large numbers of pods with memory banks freely. Third, the interconnect should have a latency shorter than the execution of tile operations so that the interconnect latency can be hidden by computation. Finally, the interconnect should scale well up to hundreds of systolic pods in terms of power consumption and silicon area.

2D mesh [9, 10, 23, 56] and H-trees [38, 60] are popularly used in many DNN accelerators thanks to their relatively low hardware cost. However, neither of them can provide enough bisection bandwidth for large numbers of systolic pods. To improve their bisection bandwidth, one can replicate an H-tree interconnect N-times (scaled-up H-tree [38]), which results in an unfeasible hardware cost with a complexity of N^2. Although Crossbar interconnect [69] offers high bisection bandwidth, it also has a quadratically increasing hardware cost with respect to the number of pods. As such, we conclude that H-tree and Crossbar are not suitable for multi-pod DNN accelerators due to their excessive hardware cost.

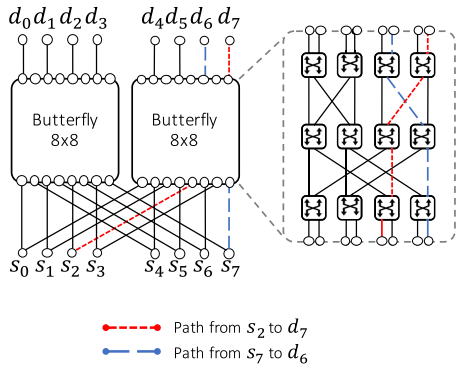


Fig. 6. An 8×8 Butterfly network with an expansion factor of 2. Routings from s_2 to d_7 and from s_7 to d_6 are shown with blue and red lines, respectively.

Multistage interconnect networks emerge as a promising solution for the energy-efficient multi-pod DNN accelerators as they exhibit sufficient bisection bandwidth, relatively low hardware cost with a complexity of $N \log N$, and low latency. Prior DNN accelerators [50] have proposed to use Benes network [6], which is a non-blocking multistage interconnection that consists of $(2 \log N - 1)$ stages. Compared to the H-tree and Crossbar, the Benes network has a feasible hardware cost of $N \log N$ while providing a sufficient bisection bandwidth of N. However, the Benes network suffers from a considerable latency that is proportional to its large number of stages, $2 \log N - 1$. While the Benes network can route all possible input-output permutations without contention, it offers only a limited multi-casting capability. Due to this limitation, we consider the augmented version of the Benes network with a copy network [43] instead of its standard implementation, which enables the full multi-casting capability at the expense of longer latency. Consequently, none of these interconnects mentioned above satisfy the design requirements of an accelerator with a large number of pods.

The Butterfly network, which is also a multistage interconnect, offers high bisection bandwidth with low latency and scalable hardware cost. A standard Butterfly network provides only limited multicasting and combinatorial power; however, this limitation can be alleviated by employing multiple of them in parallel [43], where the number of parallel butterflys is referred to as *expansion factor*. Figure 6 depicts an example Butterfly network with eight source and destination ports, and with an expansion factor of two. Thanks to the redundant switches and links between source and destinations facilitated by the expansion, we achieve higher combinatorial power than the standard Butterfly network; i.e., more permutations are feasible without network contention. For instance, the paths from s_3 to d_2 and s_6 to d_3 can be routed simultaneously as shown in Figure 6, whereas this permutation would not be possible in a standard Butterfly implementation. Moreover, because the network is expanded vertically rather than horizontally, its latency remains low, allowing to overlap data movement with computation for a larger number of pods. In the rest of this paper, we will refer to the expanded Butterfly network as Butterfly-k, where k is the expansion factor.

To evaluate various interconnect types from the perspective of the design requirements, we identified three performance metrics, which are listed in Table 1. First, the percentage of busy pods is defined as the average ratio of busy pods to the total number of pods. We expect that the interconnects with higher combinatorial power achieve a higher percentage of busy pods due to reduced contention. Second, the number of cycles per tile operation describes how long it takes for a systolic pod to complete a tile operation. As we overlap the interconnect accesses with the systolic pod computation, the interconnect latency is typically hidden by the execution time of

Table 1. Interconnect Performance Metrics Generated by our Cycle-accurate Simulator Averaged Across all the Workloads

Type	Busy Pods [%]	Cycles per Tile Op.	mW/byte
Butterfly-1	66.81	19.72	0.23
Butterfly-2	72.41	20.17	0.52
Butterfly-4	72.26	20.27	1.15
Butterfly-8	72.43	20.48	2.53
Crossbar	72.38	19.73	7.36
Benes	72.38	30.00	0.92

tile operations. However, if the interconnect latency is too long, it may become exposed, thereby increasing the number of cycles per tile operation metric. Finally, Watt/byte of the interconnect characterizes the power consumption. We seek a smaller Watt/byte value for reducing the overall power consumption.

Table 1 presents the previously mentioned performance metrics for various types of interconnects. First, we observe that the standard Butterfly network has a reduced percentage of busy pods by 6% due to its insufficient combinatorial power. However, expanding it with a factor of two is sufficient to increase its percentage of busy pods to that of interconnects with full combinatorial power, such as Crossbar and Benes. Second, the interconnect types with low latencies (i.e., Butterfly and Crossbar) results in the minimum number of cycles per tile operation because their latencies are hidden by the computation. However, Benes network, which has a substantially longer latency, incurs a significant overhead (around 50% more) in the number of cycles per tile operation. Finally, Crossbar, whose power consumption increases quadratically with the number of arrays, requires 8× and 32× more watts per byte than Benes or standard Butterfly networks for 256 pods. To conclude, the standard Butterfly, Benes, and Crossbar interconnects are not suitable for multi-pod accelerators due to their insufficient combinatorial power, long latency, and high watts per bytes metrics, whereas Butterfly network with an expansion of two is the optimal design choice thanks to its sufficiently high combinatorial power, relatively short latency, and low watts per bytes.

3.3 Tiling & Scheduling

Tiling and scheduling play an essential role in maintaining high utilization across large numbers of pods. In this subsection, we first propose a fixed-length tiling strategy with an emphasis on optimal tiling dimensions that maximize utilization across a large number of pods. Then, we explain our scheduler implementation, which maps tile operations onto systolic pods while handling the interconnect constraints, tile dependencies, and bank conflicts.

Performing a GEMM operation on systolic arrays often requires partitioning data into tiles due to dimension mismatches. The resulting tile operations can be performed on a single array sequentially, or they can be distributed among multiple arrays and performed in parallel. In weight stationary systolic arrays, the weight matrix W is spatially laid out onto the systolic arrays; thus, W must be partitioned into tiles of $r \times c$ to match the array dimensions, where r and c denote the number of array rows and columns, respectively. Because the first dimension of W must match the second dimension of the activation matrix X in a matrix multiplication, X's second dimension is also required to be partitioned with a size of r. Moreover, we can further partition X's first dimension to obtain more tile operations. Because the execution time of a matrix multiplication is approximately equal to X's first dimension, the resulting tile operations have shorter execution times.

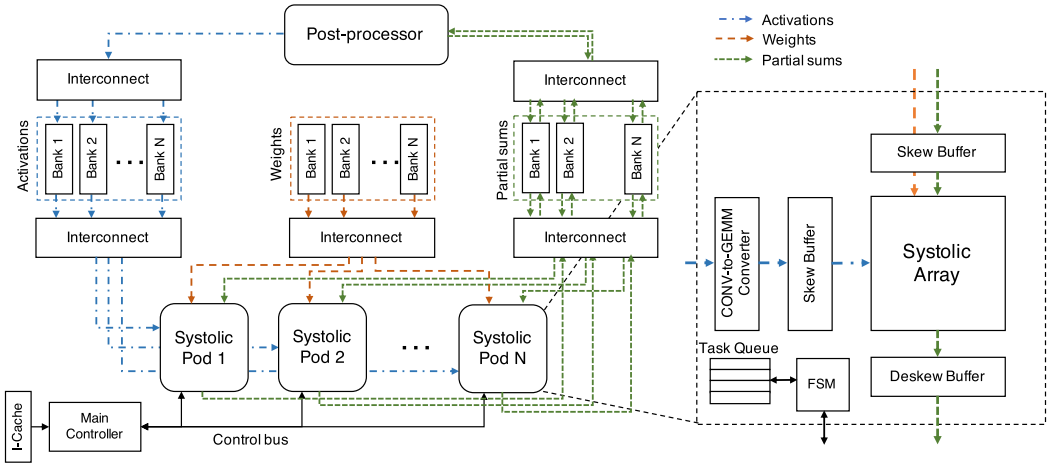


Fig. 7. Overview of the proposed architecture, with the internals of the Systolic Pod shown on the right-hand side.

We observe that the data tiling strategies proposed by prior work either do not partition X's first dimension [4], or choose a partition size that is much larger than array dimensions [12]. We argue that not partitioning X's first dimension or choosing a large partition size does not exploit the available data-level parallelism in matrix multiplication operations to the fullest extent, limiting the number of pods that can run in parallel. Partitioning the X matrix into smaller tiles produces more tile operations that can run in parallel. However, decreasing the partition size below a certain threshold value results in underutilization within pods. This threshold is the number of rows in an array (r) because the execution time for tile operations becomes shorter than r cycles, which exposes the weight buffering time. Therefore, we propose to partition the activation matrix X into tiles of $r \times r$, which produces as many parallel tile operations as possible without undermining utilization within pods.

Our compiler first performs a tiling stage during the compile time, which determines the start and end of the tiles addresses from activation and weight matrices. During the runtime, the proposed architecture loads the tiles onto on-chip memory banks in the proposed tiling format and performs GEMM operations using the loaded tiles. The resulting partial sum tiles are either consumed directly from on-chip memory by the subsequent GEMM operations, or they are written back to the off-chip memory.

4 SCALE-OUT SYSTOLIC ARRAYS

Our baseline SOSA accelerator employs systolic arrays with dimensions (32×32) that maximize effective throughput/Watt. In today's server form factors, which allow up to several hundreds of Watts, we can allocate hundreds of optimally sized systolic arrays in parallel to accelerate DNN inference workloads. Unfortunately, employing such large numbers of parallel systolic arrays poses a unique set of challenges to maintain high utilization across arrays. In this section, we also show how the proposed architecture overcomes these challenges.

Figure 7 shows the overall diagram of a SOSA accelerator. Each systolic pod encapsulates a systolic array with the peripherals required to perform GEMM operations. The main controller fetches instructions from a dedicated cache, issues them to the corresponding pods, and synchronizes the pods to perform their operations in lockstep. Activation, weight, and partial sum tiles

are stored in dedicated on-chip memory banks to reduce the interconnect width between memory banks and systolic pods. A SIMD post-processor performs element-wise operations on the partial sums and writes its results back to the activation or partial sum banks.

4.1 Systolic Pod Microarchitecture

Our systolic pod design brings CONV-to-GEMM converter and skew/deskew buffers near systolic arrays to minimize interconnect traffic. As shown on the right-hand side of Figure 7, a systolic pod consists of a systolic array, a CONV-to-GEMM converter, skew/deskew buffers, and a local controller (FSM) with a task queue that stores instructions. The CONV-to-GEMM converter [44] is a hardware block that converts activation data from a four-dimensional convolutional to two-dimensional matrix format to prevent redundant on-chip memory accesses. Likewise, skew and deskew buffers apply a skew to activation and input partial sums, and remove the skew from output partial sums to improve the memory efficiency. In this design, we encapsulate the systolic arrays with the CONV-to-GEMM converters and skew/deskew buffers to improve memory efficiency and reduce network traffic.

In standard systolic array implementations, activations and partial sums propagate at a rate of one column and one row per cycle. These slow propagation rates incur long pipeline latencies and require large skew/deskew buffers. To mitigate these problems, prior work [36, 45] proposed to multicast activations across multiple rows at each cycle and use adder trees to accumulate multiple partial sums at each cycle. In our systolic pod design, we also use activation multicasting and partial sum fan-in methods to reduce pipeline bubbles and buffer sizes. In each cycle, we multicast activation values to U consecutive processing elements along the rows. Likewise, we propagate partial sums with an offset of V processing elements along the columns and use adder trees accumulate V partial sums. The selection of the design parameters U and V is critical for the performance of the systolic pods. On the one hand, setting the parameters U and V as one (corresponds to the standard systolic arrays) leads to the best timing thanks to the short paths between registers. However, it incurs large pipeline latencies that result in idle processing elements between tasks. On the other hand, choosing large U and V parameters hinders timing characteristics due to longer paths between registers, but it improves utilization thanks to reduced pipeline latencies. For the optimal array size that we found in our design space exploration (32×32), we choose the parameters U and V as 16 and 16, respectively.

4.2 Offline Scheduling Algorithm

Based on the fixed partition size that we described in Section 3.3, we propose an offline scheduling algorithm for multi-pod systems. Compared to online (dynamic) scheduling algorithms, the proposed algorithm enables performing more aggressive code optimizations (e.g., reordering and remapping the operations), leading to higher resource utilization and faster execution. Because our data tiling scheme produces tile operations with identical execution times, we design a scheduler with fixed time slices of r cycles. The scheduler takes a list of tile operations generated by the tiling algorithm and attempts to map and schedule them on the slots of available systolic pods at the earliest available time slice. The scheduler has three constraints while checking a tile operation's availability for a slot. First, there must be no read-after-write data dependence between scheduled tile operations. Second, a memory bank cannot be accessed by another pod as we assume single-ported banks. Third, the interconnect must be able to route all pod-bank permutations for a given time slice. Starting from the first tile operation, the scheduler searches for available slots in time slices that satisfy all three conditions.

The scheduler first finds the earliest possible time slice (l) by checking the dependencies among tile operations. Then, it finds the idle systolic pods and memory banks in the time slice found in

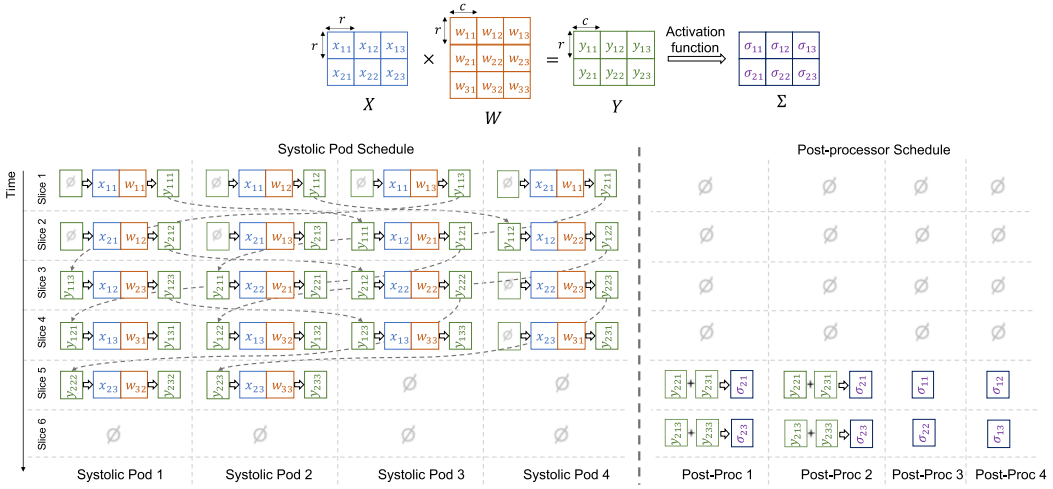


Fig. 8. Tiling and scheduling example of a matrix multiplication of $X \times W \rightarrow Y$, followed by an activation function $Y \rightarrow \Sigma$. The example shows the scheduling for four systolic pods with array sizes of $r \times c$, and four post-processors.

the previous step. Next, it exhaustively searches all combinations of available pods and memory banks and checks whether a routing between pods and banks is possible. If it finds a valid routing for all X, W, and P interconnects, it schedules the tile operation in the time slice l. If it fails to find any valid routing after all combinations are exhausted, it repeats the same steps for the next time slice, $l + 1$. The scheduler repeats this process until all tile operations are scheduled.

Figure 8 shows the result of our tiling and scheduling algorithm on a small-scale example. Each column in the figure represents a systolic pod or post-processor and each row represents a time slice. In each time slice, a systolic pod performs a tile operation $x_{ij} \times w_{jk} + y_{imk} = y_{ijk}$, where x_{ij} and w_{jk} are tiles from X and W, respectively, y_{imk} is an optional input partial sum, and y_{ijk} is the output partial sum. The output partial sums (y_{ijk}) are then aggregated to obtain the final output tiles: $y_{ik} = \sum_j y_{ijk}$. Finally, post-processors applies an activation function on final output tiles (y_{ik}) to obtain output activations, σ_{ik}.

Due to the aggregation operations between partial output tiles, there are data dependencies between the tile operations. There are two ways of performing these tile aggregations. The output of a tile multiplication can be mapped onto a later multiplication as the input partial sum (shown with dashed lines in Figure 8), or the aggregation of two tiles can be performed in the idle slots of post-processors. Post-processors work in pairs to perform tile aggregations to match the throughput of systolic pods. We use an exhaustive search to map aggregation operations on systolic arrays and post processors.

5 METHODOLOGY

We synthesized the proposed systolic pods using the TSMC 28nm process technology and Synopsis Design Compiler, then measured that the energy consumption per MAC operation is 0.4 pJ at a clock frequency of 1GHz. We encoded the weight and activation values as 8-bit and partial sums as 16-bit integers, the same as prior work [4, 19, 35]. We modeled the on-chip memory banks using Cacti-P [42]. As we use an N-to-N interconnect, we employ the same number of SRAM banks as the number of systolic pods. We chose the SRAM bank size as 256 KB, which is the smallest bank size that can store the working set of all of the benchmarks. We calculated that the energy per

byte for accessing memory banks is 2.7 pJ/Byte using Cacti-P [42]. For off-chip memory access, we assume 32GB HBM3 memory with a bandwidth of 1.2 TB/s.

To evaluate the proposed method, we selected a number of benchmarks from the two most widely used application domains of DNNs, namely computer vision and natural language processing. Because the majority of DNN models (all top ten models in the Imagenet competition) for computer vision tasks are convolutional, we choose a number of widely used, state-of-the-art CNN models, namely Inception-v3 [62], ResNet50, ResNet101, ResNet152 [29], DenseNet121, DenseNet169, and DenseNet201 [30]. We use the pre-trained models provided by Keras [13] for CNNs with an input image size of $299 \times 299 \times 3$. Transformer models are ubiquitous in the NLP domain (eight out of the top ten in the WMT-14 English-German dataset use a form of Transformer models). Therefore, we select three BERT models [16], namely BERT-medium, BERT-base, and BERT-large [26]. For BERT models, we select the median value of the sequence lengths from the benchmark [63], which is equal to 100.

6 RESULTS

In this section, we first show that the proposed array size (32×32) offers the highest effective throughput among all other design points. Then, we show and discuss the impact of multi-batching and multi-tenancy in the effective throughput and its scalability. After that, we evaluate various interconnect types presented in Section 4 and demonstrate that the Butterfly network is the most optimal choice to connect large numbers of systolic pods. Next, we evaluate the proposed tiling scheme and quantitatively show the improvement achieved by choosing the optimal tiling size. Later, we analyze the SRAM bank size's impact on the off-chip DRAM usage and effective throughput, and identify the optimal SRAM bank size for the proposed design. Finally, we share the details of our RTL implementation and synthesis results to show the validity of our insights and assumptions. For all design points, we consider the **thermal design power (TDP)** of 400 Watts, which is taken from [15], and calculate the number of parallel systolic arrays as the largest power-of-two number that results in a peak power consumption smaller than the TDP.

6.1 Array Granularity

Systolic array designs in academia and industry covers a wide range of array sizes (e.g., 8×8 in Maestro [38], 128×128 in TPUv2, v3, v4 [25] and AI-MT [4], and 256×256 in TPUv1 [35]. To cover the entire design space, we run our experiments with array sizes from 16×16 to 512×512 with steps of power-of-two numbers. Moreover, to compare SOSA against the monolithic designs (e.g., TPUv1 [35]), we have the Monolithic baseline, in which available processing elements are organized in the form of a single systolic array.

Table 2 summarizes the performance results of SOSA with varying array granularities. Because large systolic arrays have higher power efficiency than smaller ones, Monolithic baseline achieves the highest theoretical peak throughput (1.85 PetaOps/s) among all other array granularities. However, due to underutilization in large systolic arrays, Monolithic baseline exhibits only 10.3 % of its peak throughput, which corresponds to an effective throughput of 191.3 TeraOps/s. In contrast, array granularity of 16×16 has the lowest peak throughput due to its low power efficiency. As a result, even though it achieves the highest utilization, its effective throughput is only 198.9 TeraOps/s because of its limited peak throughput.

As discussed in the design space exploration of Section 3.1, the array size of 20×32 finds a balance between power efficiency and utilization and achieves the maximum effective throughput, which is 344.5 TeraOps/s. However, the number of rows in this specific array size is not a power of two, which introduces the following implementation difficulties and inefficiencies: First, memory bus and SRAM banks typically have a width value that is power-of-two; thus, aligning data size

Table 2. Performance of SOSA with Various Array Sizes

Systolic Array Size	# of Pods	Peak Power [Watts]	Peak Throughput @400W [TeraOps/s]	Util. [%]	Effective Throughput @400W [TeraOps/s]
512 × 512	1	113.2	1853	10.3	191.3
256 × 256	8	245.0	1712	14.0	183.0
128 × 128	32	283.1	1481	13.8	205.0
64 × 64	128	362.2	1158	17.4	200.9
16 × 16	512	210.6	498.0	55.5	198.9
20 × 32	256	211.1	620.8	40.0	344.5
32 × 32	**256**	**260.2**	**806.0**	**39.4**	**317.4**

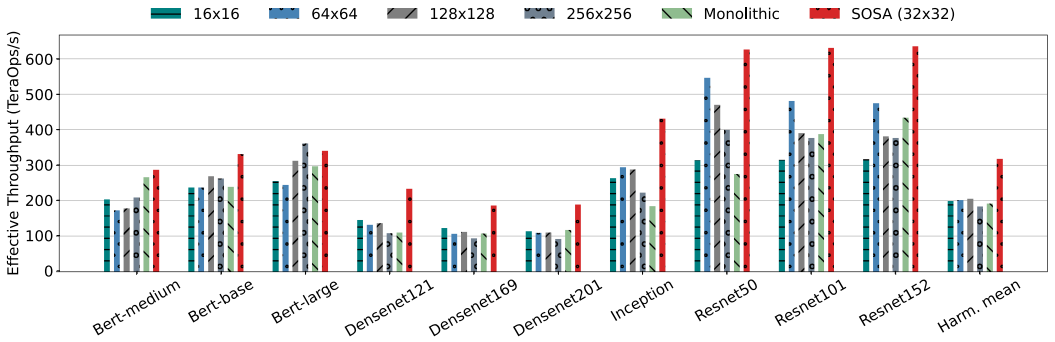


Fig. 9. Effective throughput of SOSA with various array sizes and Monolithic baseline for various DNN benchmarks. All values are normalized to 400 Watts.

with the bus and cache block width by choosing a number of rows that is a power of two in an array improves memory efficiency. Second, systolic arrays typically utilise adder trees to reduce pipeline latencies: their efficiency is maximised when the array dimensions are powers of two. Thus, we round up the number of rows from 20 to 32 and use the array size of 32×32.

The results given in Table 2 also show the effectiveness of the proposed design space exploration method introduced in Section 3.1. Although the computational model used in the design space exploration is a first-order approximation and orders of magnitude faster than the cycle-accurate simulations, it can identify the optimal array size for the given DNN workloads. Therefore, we conclude that the proposed design space exploration method can be employed to analyze the performance of multi-pod systolic arrays.

Figure 9 shows the breakdown of effective throughputs for individual DNN models. We observe, in nine out of ten benchmarks, SOSA with the array size of 32×32 outperforms other designs by up to a factor of ~ 1.6. The only benchmark that 32×32 does not exhibit the highest throughput is BERT-large, for which the array size of 256×256 outperforms 32×32 by a factor of 1.06. We argue that the reason why 256×256 outperforms other designs for BERT-large is because its array dimensions are well-aligned with the data dimensions in BERT-large. Nevertheless, these results show that SOSA with the array size of 32×32 is the optimal design point for a wide range of state-of-the-art DNN workloads, with an average effective throughput higher than prior designs by a factor of 1.55×.

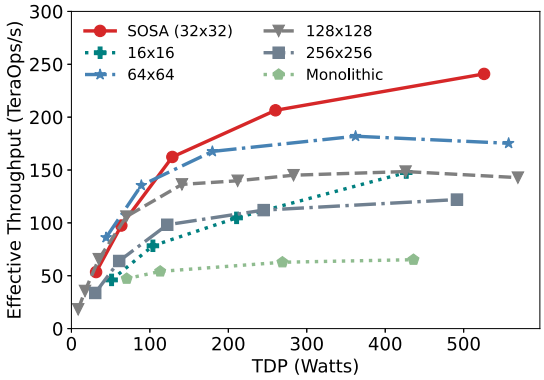


Fig. 10. Effective throughput of SOSA and Monolithic baseline for various TDP values. For Monolithic baseline, we assume a single systolic array and vary its dimensions between 400×400 and 1024×1024; whereas for SOSA designs, we use keep the array size constant and vary the number of parallel pods.

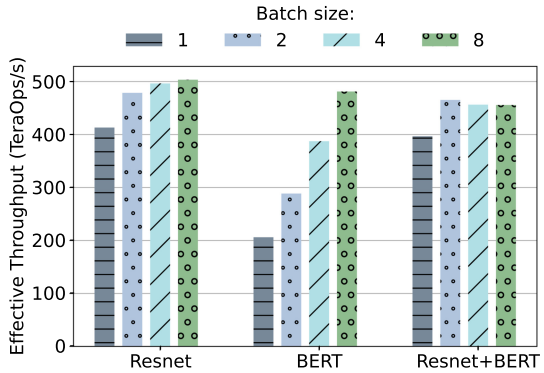


Fig. 11. Effective throughput of SOSA for varying batch sizes for Resnet only, BERT only, and both Resnet and BERT in parallel.

In the previous evaluation, we assumed the number of systolic pods as 256. However, AI accelerators' power and area budgets may vary depending on the system requirements; thus, we evaluate the proposed architecture's sensitivity to the number of pods. Figure 10 shows the effective throughput for various numbers of pods. We observe that, for TDP values larger than 90 Watts, SOSA with the array size of 32×32 outperforms all other designs by a factor up to 1.5×, when the number of arrays is scaled up to 512. We also observe that the increase in the effective throughput starts to saturate as we increase the number of arrays more than 128. This is because of the fact that we target a batch size of one for all workloads to mimic an online setting, which easily falls short of tile operations that run in parallel in large numbers of arrays. This limitation can be easily avoided by increasing the data size, either by increasing the batch size or running multiple workloads in parallel.

To show the impact of choosing larger batch sizes and running multiple workloads in parallel, we measured the effective throughput of a Resnet-152 and BERT-medium models with varying batch sizes, which is shown in Figure 11. We observe that, because the proposed design already achieves an effective throughput close to its peak throughput for the Resnet model, increasing the batch size does not lead to a significant improvement in its effective throughput. In contrast, because the

Table 3. Optimal Array Size for Varying Batch Size and Number of Parallel Workloads

		No. of workloads		
		1	2	4
Batch size	1	20×32	32×64	40×32
	2	40×32	40×32	64×32
	4	40×32	64×32	80×32
	8	80×32	80×32	80×32
	16	80×32	96×64	128×64

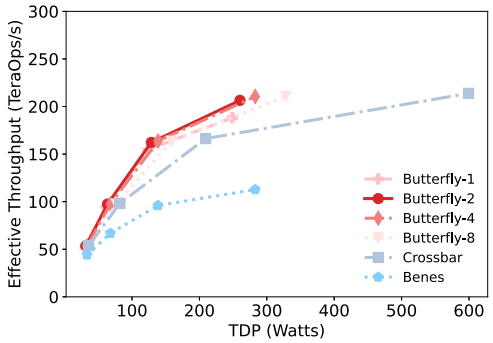


Fig. 12. Effective throughput versus TDP for various interconnect types. Points represent the number of pods, which are equal to 32, 64, 128, and 256.

proposed design is underutilized while running the BERT model due to insufficient number of tile operations, increasing the batch size results in a significant improvement in effective throughput. Likewise, running multiple workloads in parallel also increases the number of tile operations. As such, running the Resnet and BERT models in parallel with a batch size of one achieves an effective throughput of 397 TeraOps/s, which is 1.44× higher than running them sequentially.

To understand how the optimal array size changes in multi-workload and multi-batch scenarios, we perform an ablation study where we vary the batch size and the number of parallel workloads, and calculate the optimal array size using our design space exploration. Table 3 shows the optimal array sizes for various batch sizes and numbers of workloads. We observe that increasing the number of parallel workloads or the batch size results in an increase in the optimal array size due to larger model dimensions and higher degree of available parallelism. Therefore, we conclude that the designs that target single-batch and single-workload settings (e.g., real-time applications) should pick a small array size (i.e., 20×32), whereas the designs that target multi-batch and multi-workload settings (e.g., datacenter applications) benefit from picking larger array sizes (e.g., TPU).

6.2 Interconnect

To demonstrate the interconnect's impact on a multi-pod system, we measured the effective throughput and calculated the TDP for various numbers of pods and for various types of interconnects, which is shown in Figure 12. Crossbar achieves the highest effective throughput (213.8 TeraOps/s) thanks to its high connectivity and low latency but it requires around 2.3× more peak power than any other interconnect due to its quadratically increasing hardware cost. Although Benes network offers high connectivity, it performs poorly for the increasing number of pods mainly due to the fact that their long latency becomes exposed, reducing its effective throughput.

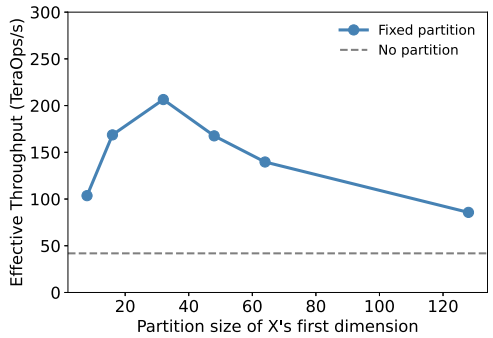


Fig. 13. Effective throughput versus data partition size for activation matrices.

This experiment confirms that Butterfly is the optimal interconnect for multi-pod systems, which achieves an effective throughput only 4% less than Crossbar but at a much lower TDP.

Figure 12 also evaluates the impact of the expansion factor for the Butterfly network. For increasing expansion factors, Butterfly networks are capable of routing more input and output permutations, reducing network contention and improving effective throughput. However, the hardware cost of the interconnect increases with the expansion factor, which reduces its effectiveness. Our experiment shows that the expansion factors larger than two achieve only a marginal increment in effective throughput (less than 2%) while the hardware cost of the interconnect doubles with every expansion. Thus, we conclude that Butterfly network with an expansion factor of two is the optimal choice of interconnect, with an effective throughput of 206.5 TeraOps/s at a TDP of 260 Watts.

6.3 Tiling

To demonstrate the effect of our proposed tiling strategy quantitatively, we measured the effective throughput for CNN and BERT benchmarks using various partition sizes for activation matrix's first dimension. Figure 13 shows the sensitivity in effective throughput with respect to partition size. When the activation matrix's first dimension is not partitioned or the partition size is larger than the number of rows in the systolic array, the effective throughput is suboptimal because the number of tile operation that can run in parallel is not sufficient to keep large numbers of arrays active. When the partition size is smaller than the number of rows, the effective throughput also decreases because the interconnect and buffering latencies become exposed. Therefore, we identify the optimal partition size for the activation matrix as $r \times r$, which maximizes the overall effective throughput in a multi-pod system.

6.4 Memory

The effective throughput of the proposed architecture is also sensitive to the on-chip SRAM capacity because a SRAM capacity that is smaller than the workload's active working set may lead to latency and energy overhead due to frequent off-chip DRAM accesses, whereas an unnecessarily large SRAM capacity increases energy consumption and silicon area cost. To show the effective throughput's sensitivity to on-chip SRAM capacity, we vary the bank size from 64kB to 1MB, and measured the effective throughput and DRAM bandwidth usage, which are shown in Figure 14. In this experiment, we used the workload with the largest active working set among all benchmarks, namely Resnet152, with a batch size of eight. We observe that, choosing an SRAM bank size smaller than 256kB results in tile eviction and SRAM misses, which leads to both increased DRAM bandwidth usage, which consequently reduces the effective throughput. As a result, we pick a bank size of 256kB in our proposed design.

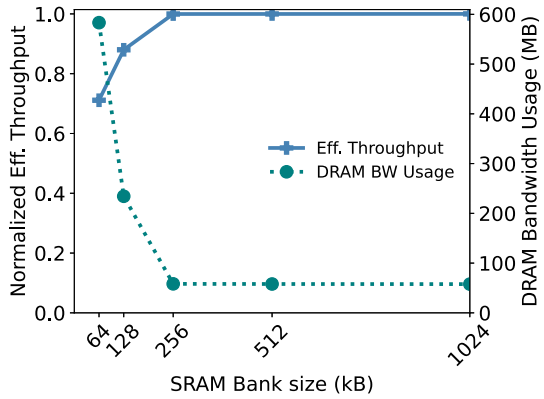


Fig. 14. Effective throughput (normalized to maximum value) and off-chip DRAM usage for varying on-chip SRAM bank sizes.

Table 4. Power and Area Breakdown of the Proposed Architecture for 256 Systolic Pods

		Power [%]	Area [%]
SRAM		45.81	75.37
Post-processor		0.56	0.25
Interconnect		15.06	4.18
Systolic Pod	Systolic Array	37.64	19.76
	Job Queue	0.30	0.18
	Act. Buffer	0.07	0.01
	Conv. Buffer	0.19	0.06
	Input Psum Buffer	0.09	0.03
	Output Psum Buffer	0.09	0.03
	Others	0.19	0.13

The design is synthesized in Synopsys Design Compiler using the TSMC 28nm library for up to 16 systolic pods and the results are extrapolated for 256 systolic pods.

6.5 RTL Synthesis

To verify our assumptions on hardware parameters, we synthesized the proposed design in Synopsys Design Compiler using TSMC 28nm library. Table 4 summarizes our synthesis results. We observe that on-chip SRAM memory constitutes the majority of the power consumption (45.81%) and silicon area (75.37%). The results also demonstrate that the proposed interconnect, namely the Butterfly network with an expansion factor of two, is only 15.06% of the total power consumption and 4.18% of the total silicon area. Unsurprisingly, systolic arrays constitute a large portion of a pod's total power consumption (97.58%) and silicon area (97.82%), whereas the rest is taken by the control logic and buffers. Leakage power constitutes only 1.5% of the total power of systolic pods.

7 RELATED WORK

With the increase in the popularity of DNNs in the last decade, many hardware and software techniques have been proposed to accelerate DNN workloads. In this section, we first describe software optimization that targets DNN workloads, then we explain custom silicon solutions, and finally, we discuss the proposals to improve the performance of systolic arrays.

Due to the low cost of commodity hardware, many services employ general-purpose CPUs [28] and GPUs [47, 52] to process DNN workloads. Prior work has proposed memory management [52, 64] and multithreading [12, 68] to improve execution on CPU and GPUs. Despite such software optimizations, CPUs and GPUs often fall short of the required throughput and efficiency due to their complex control flow and long execution pipeline. Recent **General-purpose GPUs (GPGPU)** employ tensor cores [14] to accelerate HPC and DNN workloads. While tensor cores are also spatial architectures that consist of a grid of processing elements, they differ from systolic arrays in terms of design requirements and trade-offs. Therefore, prior work on tensor cores is not necessarily applicable to systolic arrays.

Others have proposed to use FPGAs as hardware accelerators for DNNs [3, 17, 20, 21, 49, 54, 67]. Although FPGAs may benefit from reconfiguration to accelerate DNN workloads compared to general-purpose CPU and GPUs, the overhead of reconfigurability inherent to FPGAs places an upper bound on their arithmetic density and efficiency. Therefore, researchers have focused on custom silicon solutions [2, 7, 8, 27, 45, 51] that exploit low precision arithmetic and regular memory access patterns of DNNs. Many of these ASIC solutions implement certain types of dataflows optimized to improve the power efficiency [22, 23, 41, 46, 56]. However, these dataflow architectures require large register files or scratchpad memories distributed among the processing elements [8], which reduces their efficiency and density. While prior work also studies sizing [8], tiling [23], and interconnection [11] in the context of such dataflow architectures, the proposed methods and findings are not applicable to systolic architectures as the latter presents a different set of challenges and trade-offs.

Many DNN accelerator architectures are based on the variants of systolic arrays [4, 12, 18, 31, 36, 37, 44, 45, 59, 65, 66]. Google has adopted an ASIC solution and developed **Tensor Processing Units (TPU)** for its cloud services. TPUv1 is a monolithic systolic array, and it is an inference accelerator [35]. TPUv2/v3 target both inference and training, and they consist of two systolic arrays sharing a common vector memory [34]. Although TPUs achieve high throughput, the large systolic arrays used in their designs suffer from underutilization while executing a wide range of DNN workloads, as we explained in Section 3.1. Furthermore, they cannot share a task among a large number of systolic arrays efficiently due to off-chip communication requirements. Maestro addressed the underutilization problem of the DNN accelerators and proposed a DNN accelerator architecture based on small-sized systolic arrays on 3D interconnect fabric [38, 39]. However, the proposed architecture suffers from low power efficiency due to reduced computation to memory access ratio.

Prior work has proposed solutions to address various types of underutilization in systolic array-based accelerators. AI-MultiTasking [4] and PREMA [12] resort to multi-tenancy to mitigate the underutilization problem that stems from memory stalls and inter-layer dependencies. While these papers use different tiling schemes that we show nonoptimal for scale-out architectures, the techniques used to mitigate memory stalls and dependencies can also be used in the proposed architecture. A number of prior works exploit the sparsity in DNN models to improve utilization in systolic arrays. SMT-SA [59] proposes to execute multiple DNNs simultaneously on a systolic array. Likewise, Kung et al. [37] propose to combine multiple sparse DNN columns to improve ALU utilization. While we assume dense DNN models in this paper, these techniques can also be used in the SOSA architecture to improve its utilization while processing sparse DNN models. Prior work also investigated dynamically configurable array architectures [24, 58]. While these designs can mitigate the underutilization in systolic arrays, dynamic configurability incurs a significant overhead especially for an increasing number of available configurations in large arrays. In this work, we adopt a fixed array structure that attains high utilization while avoiding the overhead of dynamic configurability.

8 CONCLUSION

In this work, we studied multi-pod systolic architectures and their three key design aspects, namely array granularity, interconnect, and tiling. We first analyzed the DNN workloads to reveal their computational requirements and then performed a design space exploration to find the optimal systolic array size. For state-of-the-art DNN workloads from various application domains, we identified that the array size of 32×32 exhibits the highest effective throughput/Watt among all prior designs.

Then, we introduced SOSA, which consists of optimally sized systolic arrays. We identified the design requirements for the interconnection between systolic arrays in multi-pod architectures and evaluated various topologies. We showed that Butterfly, which is a multi-stage interconnect, outperforms is the ideal topology for multi-pod accelerators thanks to their scalable hardware cost and short round-trip latency. Then, we proposed a novel tiling scheme to maximize the utilization across multiple pods. We showed that choosing a partitioning size that matches the number of rows in an array improves utilization in a multi-pod system up to 5× compared to tiling schemes with no partitioning. We demonstrated that the proposed design scales well with the increasing number of pods and achieves up to 600 TeraOps/s at a TDP of 400 Watts with a single-batch DNN workload.

REFERENCES

[1] 2017. Introduction to the IPU architecture. https://www.graphcore.ai/nips2017_presentations. Accessed: 2019-08-06.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA'16)*. IEEE Press, 1–13. https://doi.org/10.1109/ISCA.2016.11

[3] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 22, 12 pages.

[4] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. 2020. A multi-neural network acceleration architecture. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020*. IEEE, 940–953. https://doi.org/10.1109/ISCA45697.2020.00081

[5] Pete Bannon, Ganesh Venkataramanan, Debjit Das Sarma, and Emil Talpes. 2019. Computer and redundancy solution for the full self-driving computer. In *2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, August 18–20, 2019*. IEEE, 1–22.

[6] V. E. Beneš. 1964. Optimal rearrangeable multistage connecting networks. *The Bell System Technical Journal* 43, 4 (1964), 1641–1656.

[7] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS'14)*. Association for Computing Machinery, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967

[8] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18–22, 2016*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[9] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid State Circuits* 52, 1 (2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[10] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A machine-learning supercomputer. In *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13–17, 2014*. IEEE Computer Society, 609–622.

[11] Yu-Hsin Chen, Tien-Ju Yang, Joel S. Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[12] Yujeong Choi and Minsoo Rhu. 2020. PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22–26, 2020*. IEEE, 220–233. https://doi.org/10.1109/HPCA47549.2020.00027

[13] François Chollet. 2015. Keras. https://keras.io.

[14] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 tensor core GPU: Performance and innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[15] NVIDIA Corporation. 2020. *NVIDIA A100 40GB PCIe GPU Accelerator Product Brief*. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2–7, 2019, Volume 1 (Long and Short Papers)*. 4171–4186. https://doi.org/10.18653/v1/n19-1423

[17] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi. 2016. Caffeinated FPGAs: FPGA framework for convolutional neural networks. In *2016 International Conference on Field-Programmable Technology (FPT)*. 265–268.

[18] Mario Drumond, Louis Coulon, Arash Pourhabibi, Ahmet Caner Yüzügüler, Babak Falsafi, and Martin Jaggi. 2021. Equinox: Training (for free) on a custom inference accelerator. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO'21)*. Association for Computing Machinery, New York, NY, USA, 421–433. https://doi.org/10.1145/3466752.3480057

[19] Mario Paulo Drumond. 2020. ColTraIn: Co-located DNN training and inference. (2020), 115. https://doi.org/10.5075/epfl-thesis-10265

[20] Clément Farabet, Berin Martini, B. Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2011, Colorado Springs, CO, USA, 20–25 June, 2011*. 109–116. https://doi.org/10.1109/CVPRW.2011.5981829

[21] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A configurable cloud-scale DNN processor for real-time AI. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1–6, 2018*. 1–14. https://doi.org/10.1109/ISCA.2018.00012

[22] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8–12, 2017*. 751–764. https://doi.org/10.1145/3037697.3037702

[23] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13–17, 2019*. 807–820. https://doi.org/10.1145/3297858.3304014

[24] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, Cliff Young, and Hadi Esmaeilzadeh. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17–21, 2020*. IEEE, 681–697.

[25] Google. 2017. Cloud TPU. https://cloud.google.com/tpu. Accessed: 2018-01-31.

[26] Google. 2020. BERT. https://github.com/google-research/bert. Accessed: 2020-10-5.

[27] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA'16)*. IEEE Press, 243–254. https://doi.org/10.1109/ISCA.2016.30

[28] Kim M. Hazelwood, Sarah Bird, David M. Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied machine learning at Facebook: A datacenter infrastructure perspective. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24–28, 2018*. IEEE Computer Society, 620–629. https://doi.org/10.1109/HPCA.2018.00059

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[30] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21—26, 2017*. IEEE Computer Society, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[31] Kashif Inayat and Jaeyong Chung. 2022. Hybrid accumulator factored systolic array for machine learning acceleration. *IEEE Trans. Very Large Scale Integr. Syst.* 30, 7 (2022), 881–892. https://doi.org/10.1109/TVLSI.2022.3170233

[32] Marc Jordà, Pedro Valero-Lara, and Antonio J. Peña. 2019. Performance evaluation of cuDNN convolution algorithms on NVIDIA volta GPUs. *IEEE Access* 7 (2019), 70461–70473.

[33] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. 2021. Ten lessons from three generations shaped Google's TPUv4i : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[34] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David A. Patterson. 2020. A domain-specific supercomputer for training deep neural networks. *Commun. ACM* 63, 7 (2020), 67–78.

[35] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24–28, 2017*. 1–12. https://doi.org/10.1145/3079856.3080246

[36] H. T. Kung. 1982. Why systolic architectures? *IEEE Computer* 15, 1 (1982), 37–46. https://doi.org/10.1109/MC.1982.1653825

[37] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13–17, 2019*. 821–834. https://doi.org/10.1145/3297858.3304028

[38] H. T. Kung, Bradley McDanel, Sai Qian Zhang, Xin Dong, and Chih-Chiang Chen. 2019. Maestro: A memory-on-logic architecture for coordinated parallel use of many systolic arrays. In *30th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2019, New York, NY, USA, July 15–17, 2019*. 42–50. https://doi.org/10.1109/ASAP.2019.00-31

[39] H. T. Kung, Bradley McDanel, Sai Qian Zhang, C. T. Wang, Jin Cai, C. Y. Chen, Victor C. Y. Chang, M. F. Chen, Jack Y. C. Sun, and Douglas Yu. 2019. Systolic building block for logic-on-logic 3D-IC implementations of convolutional neural networks. In *IEEE International Symposium on Circuits and Systems, ISCAS 2019, Sapporo, Japan, May 26–29, 2019*. 1–5. https://doi.org/10.1109/ISCAS.2019.8702753

[40] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. 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, MICRO 2019, Columbus, OH, USA, October 12–16, 2019*. ACM, 754–768. https://doi.org/10.1145/3352460.3358252

[41] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24–28, 2018*. 461–475. https://doi.org/10.1145/3173162.3173176

[42] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. 2011. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *2011 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2011, San Jose, California, USA, November 7–10, 2011*. 694–701. https://doi.org/10.1109/ICCAD.2011.6105405

[43] Soung C. Liew and Tony T. Lee. 2010. *Principles of Broadband Switching and Networking*. Vol. 32. John Wiley & Sons.

[44] Zhi Gang Liu, Paul N. Whatmough, and Matthew Mattina. 2020. Sparse systolic tensor array for efficient CNN hardware acceleration. *CoRR* abs/2009.02381 (2020). arXiv:2009.02381. https://arxiv.org/abs/2009.02381.

[45] Zhi Gang Liu, Paul N. Whatmough, and Matthew Mattina. 2020. Systolic tensor array: An efficient structured-sparse GEMM accelerator for mobile CNN inference. *IEEE Comput. Archit. Lett.* 19, 1 (2020), 34–37. https://doi.org/10.1109/LCA.2020.2979965

[46] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4–8, 2017*. 553–564. https://doi.org/10.1109/HPCA.2017.29

[47] NVIDIA. 2019. NVIDIA Deep Learning SDK Documentation, 52 pages. [Online], Available: https://docs.nvidia.com/deeplearning/sdk.

[48] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S. Chung. 2015. Toward accelerating deep learning at scale using specialized hardware in the datacenter. In *2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, August 22–25, 2015*. 1–38. https://doi.org/10.1109/HOTCHIPS.2015.7477459

[49] Maurice Peemen, Arnaud A. A. Setio, Bart Mesman, and Henk Corporaal. 2013. Memory-centric accelerator design for convolutional neural networks. In *2013 IEEE 31st International Conference on Computer Design, ICCD 2013, Asheville, NC, USA, October 6–9, 2013*. 13–19. https://doi.org/10.1109/ICCD.2013.6657019

[50] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22–26, 2020*. IEEE, 58–70. https://doi.org/10.1109/HPCA47549.2020.00015

[51] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA'16)*. IEEE Press, 267–278. https://doi.org/10.1109/ISCA.2016.32

[52] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. 2016. VDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 18, 13 pages.

[53] Jonathan Ross. 2017. Prefetching weights for use in a neural network processor. US 2017/0103314 A1.

[54] M. Roukhami, M. T. Lazarescu, F. Gregoretti, Y. Lahbib, and A. Mami. 2019. Very low power neural network FPGA accelerators for tag-less remote person identification using capacitive sensors. *IEEE Access* 7 (2019), 102217–102231.

[55] Ananda Samajdar, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN accelerator. *CoRR* abs/1811.02883 (2018). arXiv:1811.02883 http://arxiv.org/abs/1811.02883.

[56] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Ross Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel S. Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12–16, 2019*. 14–27. https://doi.org/10.1145/3352460.3358302

[57] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[58] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1–6, 2018*, Murali Annavaram, Timothy Mark Pinkston, and Babak Falsafi (Eds.). IEEE Computer Society, 764–775.

[59] Gil Shomron, Tal Horowitz, and Uri C. Weiser. 2019. SMT-SA: Simultaneous multithreading in systolic arrays. *Computer Architecture Letters* 18, 2 (2019), 99–102. https://doi.org/10.1109/LCA.2019.2924007

[60] Jacob R. Stevens, Ashish Ranjan, Dipankar Das, Bharat Kaul, and Anand Raghunathan. 2019. Manna: An accelerator for memory-augmented neural networks. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12–16, 2019*. ACM, 794–806. https://doi.org/10.1145/3352460.3358304

[61] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740

[62] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016*. 2818–2826. https://doi.org/10.1109/CVPR.2016.308

[63] Tencent. 2020. Turbo Transformers. https://github.com/Tencent/TurboTransformers. Accessed: 2020-09-26.

[64] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. 2018. Superneurons: Dynamic GPU memory management for training deep neural networks. *SIGPLAN Not.* 53, 1 (Feb. 2018), 41–53. https://doi.org/10.1145/3200691.3178491

[65] Di Wu and Joshua San Miguel. 2022. uSystolic: Byte-crawling unary systolic array. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022, Seoul, South Korea, April 2–6, 2022*. IEEE, 12–24. https://doi.org/10.1109/HPCA53966.2022.00010

[66] Peng Xue, Lunshuai Pan, Litao Sun, and Mingqiang Huang. 2022. Dual-line-systolic array for high performance CNN accelerator. In *30th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2022, New York City, NY, USA, May 15–18, 2022*. IEEE, 1. https://doi.org/10.1109/FCCM53951.2022.9786215

[67] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and J. Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[68] Amir Yazdanbakhsh, Jongse Park, Hardik Sharma, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. 2015. Neural acceleration for GPU throughput processors. In *Proceedings of the 48th International Symposium on Microarchitecture* (Waikiki, Hawaii) *(MICRO-48)*. Association for Computing Machinery, New York, NY, USA, 482–493. https://doi.org/10.1145/2830772.2830810

[69] Haozhe Zhu, Yu Wang, and C.-J. Richard Shi. 2020. Tanji: A general-purpose neural network accelerator with unified crossbar architecture. *IEEE Des. Test* 37, 1 (2020), 56–63. https://doi.org/10.1109/MDAT.2019.2952329

Received 6 April 2022; revised 10 November 2022; accepted 13 November 2022