Systolic Tensor Array: An Efficient Structured-Sparse GEMM Accelerator for Mobile CNN Inference

Zhi-Gang Liu, Paul N. Whatmough, Matthew Mattina

Arm ML Research Lab, Boston, MA, USA

Convolutional neural network (CNN) inference on mobile devices demands efficient hardware acceleration of low-precision (INT8) general matrix multiplication (GEMM). The systolic array (SA) is a pipelined 2D array of processing elements (PEs), with very efficient local data movement, well suited to accelerating GEMM, and widely deployed in industry. In this work, we describe two significant improvements to the traditional SA architecture, to specifically optimize for CNN inference. Firstly, we generalize the traditional scalar PE, into a *Tensor-PE*, which gives rise to a family of new *Systolic Tensor Array* (STA) microarchitectures. The STA family increases intra-PE operand reuse and datapath efficiency, resulting in circuit area and power dissipation reduction of as much as 2.08× and 1.36× respectively, compared to the conventional SA at iso-throughput with INT8 operands. Secondly, we extend this design to support a novel block-sparse data format called density-bound block (DBB). This variant (STA-DBB) achieves a 3.14× and 1.97× improvement over the SA baseline at iso-throughput in area and power respectively, when processing specially-trained DBB-sparse models, while remaining fully backwards compatible with dense models.

Index Terms—Systolic Array, Convolutional Neural Networks, Inference, Matrix Multiplication, Hardware Accelerators, GEMM.

I. INTRODUCTION

There is currently huge interest in accelerating Convolutional Neural Network (CNN) inference on mobile devices, for tasks such as image classification, detection and segmentation. CNN layers are typically implemented by lowering 2D convolution to general matrix multiply (GEMM) kernels, which are typically the runtime bottleneck when executed on CPUs, motivating hardware acceleration. The systolic array (SA) is a special-purpose processor for efficiently accelerating GEMM. The SA consists of an array of MAC processing elements (PEs), which communicate operands and results using local register-to-register communication only, which makes the array very efficient and easily scalable without timing degradation. These advantages have led to their deployment in commercial products, e.g. the Google Tensor Processing Unit (TPU) [1].

In this paper, we describe new SA microarchitectures specifically targeting mobile CNN inference with narrow INT8 operands, summarized below:

- The classical SA is generalized into a family of *Systolic Tensor Array* (STA) by replacing the traditional scalar PE with *Tensor-PEs*. STAs have twice the area efficiency of the baseline SA at iso-throughput.
- We introduce a novel *Density-Bound Block* (DBB) structured-sparse matrix, which vastly improves load balancing, with negligible accuracy loss on five CNNs.
- The STA is extended into STA-DBB, which supports DBB-sparse models. Results demonstrate 3× area efficiency and ∼ 2× power efficiency of the iso-throughput SA baseline. STA-DBB also outperforms SMT-SA [2] for INT8 operands, while remaining backward compatible with conventional (dense) CNNs.

II. BACKGROUND AND RELATED WORK

Systolic Arrays for CNN Inference. Although the GEMM kernel is obviously relevant to a wide range of workloads,
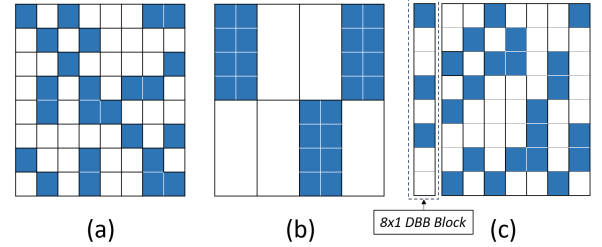

Fig. 1: Sparse matrices: (a) random sparse; (b) block sparse (4×2 block); (c) density-bound block (DBB) (8 × 1 block). Filled elements denote non-zero.

we focus here specifically on INT8 data type variants for CNN inference [1]. INT8 microarchitectures are especially challenging to optimize, as the datapath cost is relatively low compared to data movement, in contrast to floating-point data types which have a much higher relative datapath cost. Our starting point is a TPU-like baseline [1], modified with: (1) clock gating on zero operands, and (2) output-stationary dataflow, which keeps the larger INT32 accumulators in place and instead shifts the smaller INT8 operands.

Sparse Matrices for CNNs. CNN layers typically have sparsity in both the weight data (known at compile time) and activation data (known only at runtime). These zero operands can be exploited by skipping operations with at least one zero. This is a very compelling optimization because GEMM kernels are typically compute bound ($O(N^3)$). Therefore, any computation on sparse data can potentially achieve an increase in throughput and/or power consumption. Naturally occurring sparsity is often referred to as *random sparsity* because there is no constraint on the locations of the zeros (Fig. 1(a)). Random sparsity can be very challenging to exploit, as (1) it requires indexes to be computed and communicated for each data item, and (2) load balancing challenges can lead to low utilization. In contrast, structured *block-sparsity* [3],

(a) Conventional Systolic Array (**SA**) (b) Systolic Tensor Array (**STA**) (c) Systolic Tensor Array for DBB (**STA-DBB**)
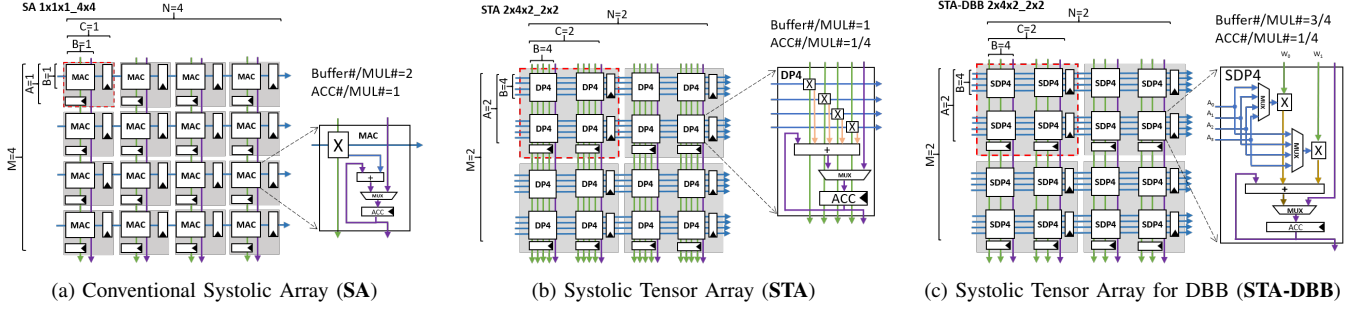
Fig. 2: Systolic array (SA) microarchitectures, generalized by extending each scalar PE to a tensor PE which performs a matrix multiplication on each clock cycle. Notation: A×B×C_M×N denotes a M×N 2-D array of A×B×C tensor PE (dashed box). Pipeline registers connect adjacent PEs, with only local data movement, which allows M×N to scale without timing degradation.

[4] (Fig. 1(b)) groups zero weights into coarse-grained blocks. Block sparse approaches are compelling, but impose a strict structure which results in poor accuracy on convolutional layers when the block size is sufficiently large to achieve an advantage in the hardware. Building on previous work, this paper proposes the density-bound block (DBB) matrix format of Fig. 1(c).

Sparse Matrix Multiplication Accelerators. An effective approach to exploit zero operands in a *scalar PE* is to clock-gate the input operand registers to reduce datapath toggling and therefore dynamic power (e.g. Eyeriss [5]). An alternative is traditional sparse linear algebra, which involves storing and computing on explicit indexes that accompany the non-zero data. For example, EIE [6] implements a sparse matrix-vector accelerator for fully-connected layers, and SCNN [7] implements indexed sparse CNN layers. However, index-based approaches have significant overhead for storing and computing on the indexes themselves (e.g. a 10-bit index for each 16-bit element in EIE). The overhead is especially severe for the common case for CNNs: "medium" sparsity INT8 weights. In contrast, fixed CNN feature extractors (FixyNN [8]) can very efficiently exploit random sparse weights.

Relating more specifically to systolic arrays, Kung et al. [9] demonstrated column combining of sparse matrices, before processing on an SA architecture. While Shomrom et al. [2] describe a method to process multiple sparse matrices simultaneously in direct analogy to simultaneous multithreading (SMT) on CPUs. In common with these papers, we build on the basic SA architecture, but instead of random sparsity, we exploit a novel structured-sparse approach with a Tensor-PE.

III. DENSE GEMM ACCELERATORS

A. Conventional Systolic Array (SA)

Fig. 2a shows the classic SA, widely deployed in products. We develop our architecture on top of the SA, and use it to baseline our results. We target mobile vision applications, with INT8 operand registers (REG), and INT32 accumulator registers (ACC). M and N describe the height and width of the PE array, respectively. The top-to-bottom paths required to read out the spatially-distributed accumulators are not illustrated.

SAs are highly efficient due to very high operand reuse through local register-to-register communication. This is in contrast to repeated SRAM reads for a dot-product machine [6], or global configurable communication for a generalized spatial array [5]. However, there is significant room for improvement in the SA architecture by relaxing the ratio of operand registers (REG) to MACs. Each MAC operation traditionally requires two INT8 operand registers and one INT32 accumulator.

B. Systolic Tensor Array (STA)

We generalize the conventional SA (Fig. 2a) into the *Systolic Tensor Array* (STA, Fig. 2b), by fusing a block of scalar PEs into a single *tensor PE* . Each STA architecture contains $M \times N$ tensor PEs, with each tensor PE consisting of a sub-array of $A \times C$ MACs that each perform a dot-product operation on B operand pairs. This is denoted uniquely as A×B×C_M×N. Fig. 2b shows a 2×2 array of tensor PEs, each with a 2×2 datapath of 4 operand pair dot-product units ($DP4$). Note that the classic SA (Fig. 2a) is a special case of STA, with $A = B = C = 1$ (denoted 1×1×1_M×N).

Fig. 3 shows the data flow of a 4×4 by 4×4 matrix multiplication on a 2×2×2_2×2 STA. A significant efficiency improvement is achieved here by reducing the number of operand buffers per MAC by $2\times$, and the number of accumulator buffers per MAC by $4\times$. To our knowledge, tensor PEs have not been previously described in the SA context.

IV. SPARSE GEMM ACCELERATORS FOR CNNS

As described in Section II, both random and block sparse approaches introduce significant trade-offs – random sparsity is hard to exploit in hardware, and block sparsity significantly impacts accuracy in CNNs. In this section, we propose an alternative to the conventional block sparse matrix format.

A. Density-Bound Block (DBB)

Fig. 1(c) shows the proposed density-bound block (DBB) matrix, which simply places an upper limit on the number of non-zero (NNZ) elements in each block. For example, Fig. 1(c) consists of 8 blocks of 8×1, each with up to 3 non-zero values ($NNZ \leq 3$). This is in contrast to conventional sparse block (Fig. 1(b)), where each block is either entirely unconstrained or all zero. DBB is a middle-ground between random (Fig. 1(a)) and block (Fig. 1(b)) sparse formats, and
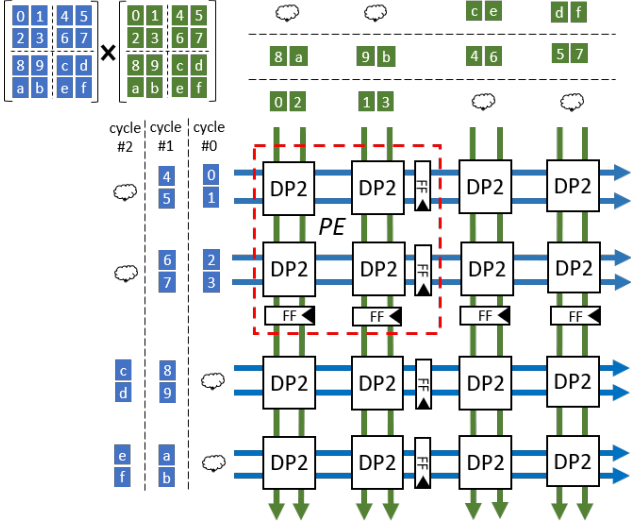
Fig. 3: Example 2×2_2_2×2 STA data flow. The tensor PE (red dashed box) computes a 4×4 by 4×4 matrix multiplication. Each input operand matrix is split into a 2×2 sub-matrix, with columns (rows) of data in blue (green) fed into the array from the left (top) edges over each clock cycle.

TABLE I: CNNs trained with 8-bit DBB-sparse weights.

Model	Dataset	Baseline Acc.(%)	Pruned Model Result	
			Acc.(%)	NNZ (%)[1]
LeNet-5 (DBB)	MNIST	99.1	98.7	1.05K (25)
ConvNet (DBB)	CIFAR10	86.0	85.3	26.8K (25)
MobileNetV1 (DBB)	ImageNet	70.9	69.8	1.6M (50)
ResNet-50V1 (DBB)	ImageNet	75.2	74.2	8.79M (37.5)
VGG-16 (DBB)	ImageNet	71.5	71.4	5.39M (37.5)
AlexNet [10]	ImageNet	57.4	57.5	2.81M (75)
VGG-19[2] [9]	ImageNet	–[3]	71.8	4.1M (12.6)

[1]Convolution layers. [2]Modified VGG-19, ∼31M conv param. [3]Not reported.

TABLE II: Throughput-normalized area and power efficiency with 50% sparse activation at 1GHz. Normalized to SA 1x1x1 (baseline).

Design	Model Sparsity	Array	Area Eff.[1]	Power Eff.[1]
SA-NCG[2] [1]	Dense	1×1×1	0.95	0.65
SA[3]	Dense	1×1×1	1.00	1.00
STA	Dense	4×8×4	2.08	1.36
SMT-SA [2]	Random (62.5%)	T2Q4	1.21	0.80
STA-DBB	DBB (50%)	4×8×4	3.14	1.97

[1] Throughput normalized. [2] SA no clock gating. [3] Baseline clock-gated SA.

results in higher CNN accuracy with the same NNZ, because the distribution of non-zero elements is significantly less constrained. At the same time, the compute required from the hardware is known a-priori for each block and high utilization is guaranteed. A simple bitmask compression is used to encode each block of 8 elements (one byte overhead per block), along with the four bytes of non-zero data. This yields a 37.5% reduction in weight memory footprint.

B. Systolic Tensor Array for DBB (STA-DBB)

Next, we add support to the STA architecture for DBB-sparse weight matrices. Weight sparsity is easier to exploit than activations, as the values are known ahead of time. The DBB weight matrix upper bounds the number of non-zero operands. Therefore, the number of physical MAC units required is no greater than the DBB sparsity bound. For instance, if the DBB block size is 8 with $NNZ \leq 4$, each 8-input Dot Product unit ($DP8$) only requires 4 MAC units instead of 8, which represents a 50% reduction in MAC hardware at the same throughput. This approach requires a multiplexer in font of MAC to select the corresponding input feature map (activation) based on the index of the non-zero weight.

Fig. 2c illustrates a 2×4×2_2×2 Sparse STA for DBB (STA-DBB) with 4-input Sparse Dot Product ($SDP4$) unit, which has 4-value vector activation inputs $[A_0, A_1, A_2, A_3]$ from left and a 50% DBB compressed weight inputs $[W_0, W_1]$ and associate 2-bit non-zero indices from top and accumulates the dot product into accumulator (ACC). In each SDP4, we trade two 8-bit multipliers for two 8-bit 4:1 multiplexers (MUX), where a MUX costs significantly less than an 8-bit multiplier. The array performs 16 effective MACs per clock cycle with only 8 physical multipliers. The final results in the stationary accumulators are read out from bottom of the array through shift chains in four clock cycles, after the computation is

finished. The data flow is similar to Fig.3, except that the matrix input from the top is 50% DBB-sparse. Notably, this architecture still supports conventional dense GEMM at half throughput, which will probably remain important to support the widest range of workloads.

V. EVALUATION RESULTS

A. DBB-Sparse CNN Training Results

To demonstrate the feasibility of DBB models, we trained five CNNs, applying conventional INT8 quantization and amplitude-based pruning for VGG-16, MobileNetV1, ResNet-50V1, 5-layer ConvNet and LeNet-5 on ImageNet, CIFAR10 and MNIST datasets. The validation results are given in Table I. The accuracy loss is in the range of 0.1% to 1.1% across all five DBB models, which include both relatively big examples (ResNet-50V1) and parameter-efficient models (MobileNetV1).

B. Hardware Accelerator Results

In this section, we evaluate four architectures: (1) conventional TPU-like SA baseline (1×1×1) with a dense model; (2) optimized STA with dense model; (3) STA-DBB with a 50% DBB-sparse model, and (4) SMT-SA with 62.5%-sparse model [2]. Each design was implemented in Verilog using a parameterized RTL generator, and synthesized in a TSMC 16nm FinFET process using Synopsys Design Compiler with a 1GHz clock. For power simulation we use Synopsys PrimeTimePX with estimated clock tree and fully-annotated switching activity.

Fig. 5 shows area and power results for the baseline dense SA (with and without clock gating for zero operands), and improved STA designs, across the design space of tensor-PE dimensions that meet a 1GHz clock frequency. Results confirm that the traditional SA (1×1×1) has 36% area and 54.3% power attributed to registers alone. The optimized STA designs
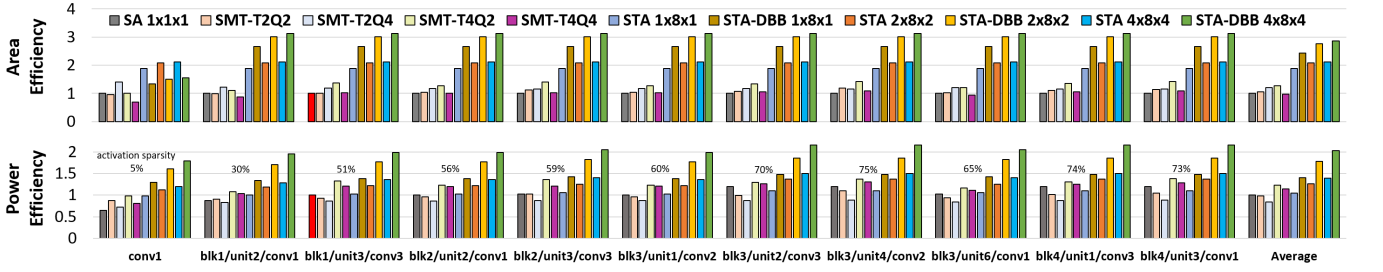
Fig. 4: Normalized area and power efficiency (iso-throughput, higher is better) for typical layers of ResNet50_V1 model, trained at 8-bits, 62.5% sparse weight, 1×8 DBB (conv1 remains dense). Area and power efficiency normalized to SA 1x1x1 with 50% average activation sparsity, closest to the *blk1/unit3/conv3* layer in this ResNet50 example.
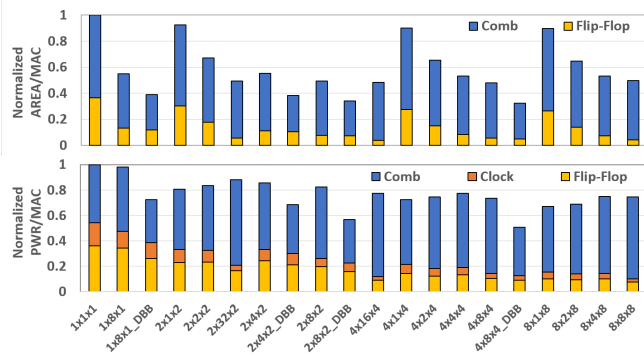

Fig. 5: Area and power at iso-throughput (lower is better) for STA and STA-DBB (50% sparse), with cell breakdown.

show improvements of as much as 47% area and 73% power at the design space sweet spot (4×8×4). The flip-flop reduction shown by the blue portions in Fig. 5, comes from reduced operand pipeline buffers and accumulator flip-flops sharing. The reduction in combinational logic (yellow), is due to efficiencies of the adder tree in the tensor MAC (dot-product).

STA-DBB further enhances efficiency by taking advantage of DBB-sparse models. Fig. 5 shows the STA-DBB designs with DBB-sparse weight compression reduce the area and power per MAC by up to 30%, over and above corresponding FM-SA configuration. The reduction in combinational logic (blue), comes from the 50% reduction in physical multipliers, while the reduction in registers (yellow) and clock load (orange), is from pipeline buffers saving.

We also evaluated Simultaneous Multi-Threading Systolic Arrays (SMT-SA) [2], which process random-sparse weights, and use a FIFO to overcome the load imbalance challenge. We implemented the T2Q2 configuration, which denotes 2 threads with a 2-deep FIFO queue, as well as the T2Q4, T4Q2 and T4Q4, all with INT8 operands and INT32 accumulation. Note that for INT8, SMT-SA which exploits random sparsity is actually less efficient than STA, which doesn't even exploit sparsity. This is due to the overhead that the FIFOs introduce, relative to the size of the INT8 datapath logic.

Fig. 4 shows the efficiency of some typical layers from the ResNet50_v1 model. SMT-SA area and power efficiencies are 0.8×-1.21× the SA baseline and around 2× lower than the corresponding STA and STA-DBB architectures with 62.5% sparse weights and 39-75% sparse input feature maps. The

high hardware cost of the FIFO in SMT-SA designs cancels out the benefit of sparsity for INT8 operands.

Table II summarizes the area and power efficiency results of the best configurations of the four architectures studied.

VI. CONCLUSION

The systolic array (SA) is an efficient, highly parallel architecture widely used to compute matrix multiply. In this paper, we improved on the classic SA architecture by specifically optimizing for efficient CNN inference. Firstly, we generalize the traditional SA by introducing a tensor PE with multiple parallel operations. This systolic tensor array (STA) architecture significantly reduces circuit area and power dissipation by as much as 2.08× and 1.36× respectively compared a conventional clock gated SA, due an increase in intra-PE operand reuse and datapath efficiency, reducing operand register count and clock tree load. Secondly, we further extend the STA architecture to efficiently accelerate models trained with density-bound block (DBB). This new sparse architecture (STA-DBB) shows 3.14× and 1.97× improvement in area and power respectively when processing specially trained DBB-sparse models, while remaining fully backward compatible with traditional dense models.

REFERENCES

[1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Int. Symp. on Computer Architecture (ISCA)*, 2017.
[2] G. Shomron, T. Horowitz, and U. Weiser, "SMT-SA: Simultaneous multithreading in systolic arrays," *IEEE Comput. Archit. Letters*, 2019.
[3] R. Sredojevic *et al.*, "Structured Deep Neural Network Pruning via Matrix Pivoting," *arXiv e-prints*, p. arXiv:1712.01084, Nov 2017.
[4] S. Narang, E. Undersander, and G. Diamos, "Block-Sparse Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1711.02782, Nov 2017.
[5] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *Int. Solid-State Cir. Conf. (ISSCC)*, 2016.
[6] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Int. Symp. on Computer Architecture (ISCA)*, 2016.
[7] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural nets," in *Int. Symp. on Comp. Arch. (ISCA)*, 2017.
[8] P. N. Whatmough *et al.*, "FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning," in *2nd Conference on Machine Learning and Systems (SysML)*, 2019.
[9] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Int. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.
[10] W. Wen *et al.*, "Learning Structured Sparsity in Deep Neural Networks," in *Adv. in Neural Information Processing Systems (NIPS 2006)*, 2016.