Systolic Building Block for Logic-on-Logic 3D-IC Implementations of Convolutional Neural Networks

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†, Douglas Yu†
*Harvard University †TSMC R&D

***Abstract*—We present a building block architecture for systolic array 3D-IC implementations of convolutional neural network (CNN) inference. The building block can be part of a library offered by a chip design service provider to support efficient CNN implementations. We describe how the building block can form systolic arrays for implementing low-latency, energy-efficient CNN inference for models of any size, while incorporating advanced packaging features such as "logic-on-logic" 3D-IC (micro-bump/TSV, monolithic 3D or other 3D technology). We present delay and power analysis for 2D and 3D implementations, and argue that as systolic arrays scale in size, 3D implementations based on, *e.g.,* micro-bump/TSV, lead to significant performance improvements over 2D implementations.**

I. Introduction

Convolutional Neural Network (CNN) inference can be viewed as a series of matrix multiplications between input data to each layer and the learned convolution filters of the layer. Systolic arrays can perform this matrix multiplication with high efficiency due to the high degree of parallelism, regular inter-processor communication, and minimal I/O costs associated with data-flow architectures [8], [5]. For these reasons, many recent processor implementations for deep learning (*e.g.,* [4], [1], [3], [13], [15], [14], and Google's Tensor Processing Unit (TPU) [7]) implement systolic arrays.

However, systolic arrays only achieve high efficiency if the underlying matrices to be multiplied can be effectively mapped onto systolic arrays. For instance, early layers in a CNN generally have smaller filter matrices, which may under-utilize the systolic arrays. Therefore, a fundamental challenge in using systolic arrays is to design an architecture which can achieve high performance for all layers, of varying sizes, in a CNN.

In this paper, we propose a versatile *systolic building block* architecture for CNN implementations. We show how multiple instances of this building block can be aggregated to efficiently implement the inference computation across CNN layers of different input data and filter matrix sizes. The building block must support the mapping of CNNs layers with a variable number of input channels and filters onto an array of building blocks, each containing fixed-size systolic arrays.

Additionally, it is desirable for the building block to support advanced implementation features, including:

- *3D-IC implementations* of systolic arrays for their compact physical embodiment, formed by the building block implemented with micro-bump/TSV stacking, monolithic 3D-IC or other 3D-IC technologies.
- *Cross-layer pipelining* where the inference computation of several CNN layers are pipelined over a series of systolic building blocks in order to minimize the I/O cost in accessing external memory systems.
- *Reduced data skews* for systolic array data synchronization in order to lower end-to-end CNN inference latency.
- *High-precision accumulation* in the multiplier-accumulators (MACs) of each systolic cell of a systolic array. In this work, we assume throughout bit-serial implementations of 32-bit accumulations for 8-bit weights and 8-bit input data. Our approach extends naturally to other precisions (see, *e.g.,* [11]).

In this paper we present a systolic building block which supports the mentioned features and analyze its resource requirements and performance gains of 3D-IC implementations when compared to 2D implementations. The main contributions of this paper are (1) the design of a versatile systolic array building block architecture for CNNs of any size and for their 3D-IC implementations, and (2) latency and power analysis demonstrating the benefits of using logic-on-logic 3D-IC in implementing systolic arrays at scale. To our knowledge, these results and the underlying design and analysis methodology presented are novel and are among the first ideas in this emerging field.

II. Background

In this section, we provide background on utilizing systolic arrays for CNN inference and how one or more CNN layers can be partitioned across multiple systolic arrays.

A. Systolic Matrix Multiplication for CNN Layer

In this work, we focus on pointwise convolution, where the kernel size is 1×1. While simplified, networks trained with *pointwise* convolution (*e.g.,* MobileNet [6]) have been shown to achieve similar performance to standard 3×3 convolutional networks for datasets such as ImageNet [2]. Additionally, pointwise convolution can be implemented efficiently in systolic arrays [9], [11]. For illustration, Figure 1 (a) shows a 3×3 systolic array matrix multiply unit that implements a pointwise convolution layer. The systolic array is made of *systolic cells*, nine cells in this example, each performing a multiplier-accumulator (MAC) computation. The weights (*e.g.,* $\mathsf{W}_{2,1}$) of the pointwise layer are stored in systolic cells, one weight per cell. The input data corresponding to an input channel (*i.e.,* d_1 for channel 1) enters into the bottom of the systolic array in a

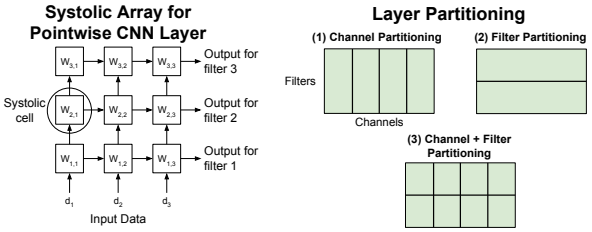

Fig. 1: (a) A 3×3 systolic array implementing a pointwise convolutional layer with 3 filters and 3 channels. (b) Partitioning a CNN layer across channels in (1), across filters in (2), and across both channels and filters in (3).

skewed fashion to permit data synchronization. At each cycle, the input data multiplied by the weight in a cell is added to the partial result computed by the previous cell from the left and forwarded to the next cell on the right. The complete results exit from the right side of the systolic array. In this paper, we assume the weight-stationary systolic array multiplication scheme depicted in Figure 1 (a), while noting that our methods can be extended to other schemes.

For CNN layers that are larger than a given fixed-size systolic array, a partitioning approach is used to break the layer into smaller tiles, where each tile can fit into a single copy of the systolic array. Figure 1 (b) shows how a layer can be partitioned into smaller tiles. As we discuss in Section III, by using this tiling approach, the flexible building block can be used to accommodate any number of filters and channels.

B. Cross-layer Pipelining

The run time of CNN inference can be dramatically reduced when multiple CNN layers are able to work in parallel on the same input sample, which we call *cross-layer pipelining*. Each layer in a CNN is partitioned into one or more tiles each implemented on a systolic array of the given size (as described in Section II-A). Figure 2 shows how a sample is pipelined across a three-layer CNN. The input data (in yellow) enters layer 1 (in red) and generates output which is immediately consumed by layer 2 (in blue). Layer 2 similarly generates output which is used as input to layer 3. In this manner, a single access to the external memory is sufficient for the operation of the systolic arrays of multiple layers. A barrier is in place at the output of layer 3 (in green), as the entire output from the final CNN layer is required to perform the CNN prediction.

III. Systolic Building Block for 3D-IC Implementations

In this section, we provide an overview of the proposed 3D systolic building block which can be deployed in 3D-IC implementations.

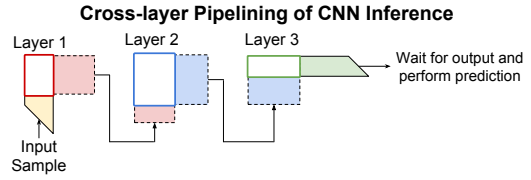

Fig. 2: Cross-layer pipelining for a CNN with three layers. Each layer is implemented using one or more systolic arrays of a given size. All three layers work on the same input sample in parallel. Data for the input (in yellow) and output (in green) are skewed for systolic array synchronization.

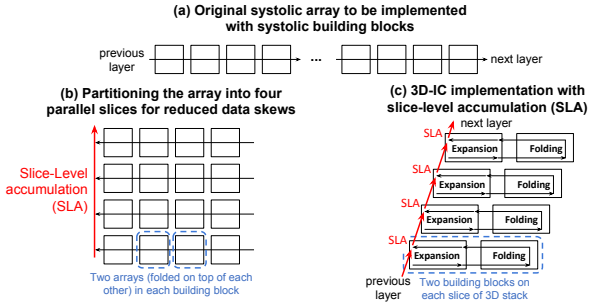

Fig. 3: Three approaches for partitioning a wide convolutional layer into multiple fixed-size systolic building blocks.

A. Partitioning Convolutional Layers onto 3D-IC Structures

In order to implement a large convolutional layer using a collection of fixed-size systolic arrays, we partition the layer as depicted in (1) of Figure 1 (b). Figure 3 illustrates an approach for applying channel partitioning to a wide convolutional layer that spans sixteen square systolic arrays (*i.e.,* it has sixteen times more channels than filters). In (a), this wide layer is implemented using sixteen square systolic *building blocks*, leading to a long data skew in order to maintain synchronization with the input data. In (b), the sixteen blocks are partitioned into four groups each with four blocks. Partial results from these groups are aggregated on the left-hand side, using slice-level accumulation (SLA) denoted by the red line, which reduces the data skew by a factor of 4. In (c), the reduced skew scheme presented in (b) is implemented in 3D-IC and aggregation of physical slices is performed using SLA. Each physical slice in (c) implements two building blocks with different roles, *Expansion* and *Folding*, as discussed in Section III-B.

B. Systolic Building Block Roles

For the 3D implementation of Figure 3 (c), each physical slice involves building blocks which have one of two different roles depending on the location of the block on the slice. Figure 4 depicts both roles for our systolic building block: an expansion block in (a) and a folding block in (b). Both roles implement fixed-size systolic arrays (called systolic *subarrays*) which are used to perform matrix multiplication between the stored filter weights and the input to a CNN layer.

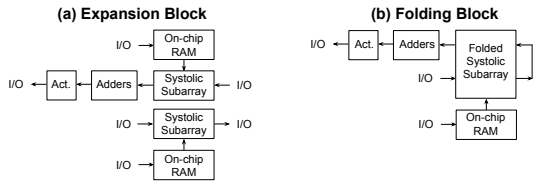

Fig. 4: The structure for the two roles supported by the systolic block: (a) an expansion block and (b) a folding block. Each systolic subarray stores CNN weights in the systolic cells which are loaded from on-chip RAM.

Each subarray takes input data from on-chip RAM which is placed adjacent to the subarray. For the leftmost block in a physical slice, SLA is performed using the Adders circuit. The Activation (Act.) circuit implements the ReLU activation function and is active only for the leftmost block in the final physical slice for a convolution layer. Batch normalization is folded into the weights after training and therefore does not require any additional computation at inference.

The systolic building block implements either the expansion or folding role reflecting the function deployed on its right end. For a folding building block, the right end implements folding connections that connect the two halves of the folded array. For an expansion building block, the right end provides same-slice I/O for the folded systolic array. It inputs from and outputs to a neighbor building block on the same slice. The left end of the systolic building block (folding or expansion) has a different implementation depending on the position of the block. Blocks on the leftmost edge of a physical layer add partial dot products computed by the systolic subarray to the values received from input TSVs using SLA and send the updated accumulation values to output TSVs. All other blocks on a layer take partial results from a neighboring building block and output results to another building block. In this way, multiple expansion building blocks can be placed next to each other on a physical slice, followed by a folding building block on the right-hand side of the slice as discussed further in Section III-D.

C. Folding Systolic Array

In order to change the direction of the partial accumulation data flow, the folding building block employs a folded systolic array. Figure 5 shows how an unfolded systolic array in (a) can be converted into a folded systolic array in (b). We assume that the unfolded systolic array inputs partial dot products arriving at the left and outputs updated partial dot products departing from the right. We fold the systolic array in the middle so that the left and right halves of the array will be on top of each other. This folded array now takes inputs from the left and sends outputs to the left. We then press the folded systolic array onto the 2D space so that the placements of left and right halves of the array are uniformly interleaved. With this folding scheme, we have achieved the goal that all wires connecting neighboring cells can be of a constant wire length independent of the systolic array size.

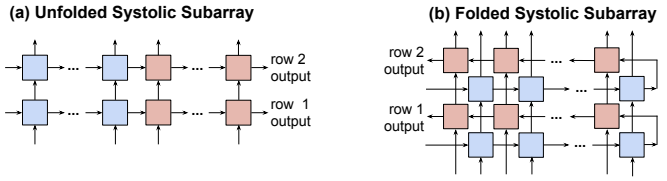

Fig. 5: An unfolded systolic array (a) is converted into a folded systolic array (b). Each square represents a systolic cell.

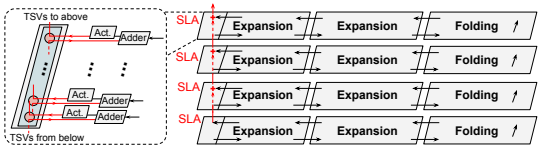

Fig. 6: Four physical slices, each implementing two expansion blocks and one folding block, are connected by TSVs which support SLA.

D. Multiple Expansion Blocks per Slice

The proposed systolic block design allows for slices to easily implement more systolic blocks, as the physical size of the slice increases, by simply increasing the number of expansion blocks on the slice. Figure 6 shows four physical slices that each implement two expansion blocks and one folding block. Each of the physical slices are connected by TSVs on the left-hand side, which facilitate SLA between each pair of slices. The partial results from the previous slice below are input from the TSV into the adder circuit on the current slice, which performs element-wise addition with the partial results output from the systolic array on the current slice.

IV. Performance Comparison Between 2D and 3D Implementations

In this section, we analyze the size, delay, and power of the systolic building block based on 28nm technology design rules and a 3D-IC technology using TSVs and micro-bumps [12]. The estimated chip area is 0.9mm^2 = 1.0mm (H) x 0.9mm (W) for a systolic array containing 50 rows by 50 columns. The height of the array depends on the number of rows, and varies from 0.46mm (0.36 mm for systolic array, 0.1mm for memory buffer) in a 20 rows by 125 columns array to 2.3mm (2.2 mm for systolic array, 0.1mm for memory buffer) in a 125 rows by 20 columns array.

For high-performance operation of the systolic array, we are interested in decreasing system latency and power consumption. We compare 3D-IC (micro-bumps) and 2D designs by analyzing wire delay (latency) and wire power consumption. Note that, as depicted in Figure 7, in a 2D layout, the worst wire delay occurs with the interconnect wires that span the entire height of the systolic array.

To obtain propagation delays and RC delays, we consider a 1mm 2D global line with 0.4μm line width and 0.4μm spacing between lines. 3D-IC interconnects using TSVs of 50μm depth and micro-bumps of 30μm height are used. The

TABLE I: Comparing block-to-block wire delay and power consumption for 2D global wire and 3D TSV with micro-bumps for 50 filter by 50 channel systolic building blocks.

Interconnect	Delay (ps)			Power Consumption[1]
	Propagation	RC	Wire	1GHz
2D	5.6	11.6	17.2	0.13
3D	0.82	0.8	1.6	0.08
2D/3D	7x	15x	11x	1.6x

extracted full EM models of the interconnects are imported into the Advanced Design System (ADS) and the wire delay is obtained by analyzing the response of a step signal in the time domain. The simulation results are summarized in Table I. The propagation delay with 3D interconnects is 7× smaller than with 2D interconnects due to the shorter wires. Similarly, the RC delay with 3D interconnects is 15x smaller compared to 2D interconnects due to the smaller resistance. Finally, the wire delay using 3D interconnects shows a similar improvement of 11x over 2D interconnects.

For wire power consumption, dynamic power P = CV^2f, where C, V, and f represent capacitance, voltage and frequency, respectively, is calculated for systolic arrays using 2D and 3D interconnects. The power consumption of the 3D-IC design is about 60% of that of the 2D design at 1GHz. We note that additional area from TSVs is relatively insignificant due to our design which only require a relatively small number of TSV connections on the building block edge (see Figure 6). With a TSV diameter of 6μm and a keep out zone of 3μm each side, each TSV takes up area of 113μm^2 and 50 TSVs for the 50×50 array uses about 0.6% of the systolic building block area.

We also study the systolic system scaling effect on delay and power consumption. In Figure 7, the 2D and 3D-IC structures for systolic systems with 50×50, 100×100 and 150×150 arrays are shown. Simulation results for wire delay and power consumption of a single interconnect in the systolic array are shown in Figure 8. For 2D, the interconnect length is increased linearly with the height of systolic array. Therefore, the corresponding wire delay is increased as the square of the array height. But for 3D-IC, the delay is the same for the three sizes of the systolic systems. The delay ratio for 2D and 3D-IC is 11, 32 and 166, respectively, for the three sizes, as shown in Figure 8 (a). The dynamic power consumption is proportional to the interconnect length. At current systolic system designs, the dynamic power for 2D is 1.6, 3.2 and 5.0 times larger than that for 3D-IC, as shown in Figure 8 (b). The results indicate 3D-IC brings more benefits when the systolic system is scaled up in size.

Note that we did not design the 50 × 50 systolic array building block in 28nm technology yet, but we have a validated Verilog design for the block. The chip area for the block is estimated based on the Verilog design and 28 nm technology parameters. The parameter values used in the simulation for Table I are extracted from measurement data

[1]Power Consumption = f×C×V^2, where $V = 0.9$ volts.

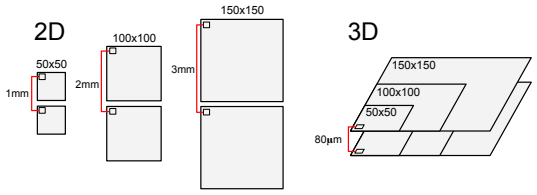

Fig. 7: Three systolic array sizes (50x50, 100x100, and 150x150) using 2D and 3D interconnects. In the 2D case, the wire length connecting arrays is proportional to the array height. In the 3D case, the wire length is a constant length regardless of array size.

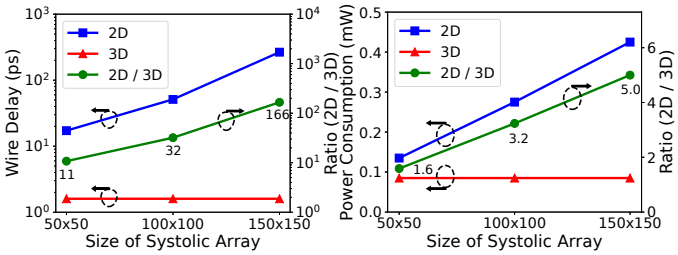

Fig. 8: (a) Wire delay and (b) power consumption of systolic arrays using 2D and 3D interconnects as a function of the size of the systolic array. The black dotted circles and corresponding black arrows for each line denotes the y-axis used.

based on production microbump/TSV technology. Thus, the interconnect simulation results (propagation delay and RC delay) are backed up by the experiment result, even though they are not directly measured from real 3D-IC system.

V. Conclusions

In this work, we present a versatile systolic building block, which can be used to implement CNN layers of any size. The building block is suited for logic-on-logic 3D-IC (micro-bump/TSV stacking, monolithic 3D or other 3D-stackings) implementations in that it allows for consistent wire length between blocks and mitigates corner turning issues [10] present in 2D implementations. Additionally, we present usage examples for a 3D-IC (micro-bump/TSV) implementation of this systolic block design which connects blocks on different slices of a 3D-IC stack using TSVs. These usage cases illustrate the benefits of the compact representation of logic-on-logic 3D-IC stacks as they can be extended across multiple physical circuit planes to scale to meet the energy efficiency and computational performance demands of CNN inference. The benefits are especially pronounced as systolic arrays scale up in size.

VI. Acknowledgement

This work is supported in part by the Air Force Research Laboratory under agreement number FA8750-18-1-0112, and a Joint Development Project between Harvard and TSMC. We thank Dr. Shih-Ya Huang of TSMC for her support with EM simulations.

REFERENCES

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[3] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[4] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109–116. IEEE, 2011.

[5] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[8] H. T. Kung. Memory requirements for balanced computer architectures. In *ACM SIGARCH Computer Architecture News*, volume 14, pages 49–54. IEEE Computer Society Press, 1986.

[9] H. T. Kung, B. McDanel, and S. Q. Zhang. Adaptive tiling: Applying fixed-size systolic arrays to sparse convolutional neural networks. *International Conference on Pattern Recognition*, 2018.

[10] H. T. Kung, B. McDanel, and S. Q. Zhang. Mapping systolic arrays onto 3d circuit structures: Accelerating convolutional neural network inference. *IEEE Workshop on Signal Processing Systems*, 2018.

[11] H. T. Kung, B. McDanel, and S. Q. Zhang. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. *The 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[12] T. Lo, M. Chen, S. Jan, W. Tsai, Y. Tseng, C. Lin, T. Chiu, W. Lu, H. Teng, S. Chen, et al. Thinning, stacking, and tsv proximity effects for poly and high-k/metal gate cmos devices in an advanced 3d integration process. In *Electron Devices Meeting (IEDM), 2012 IEEE International*, pages 33–4. IEEE, 2012.

[13] S. K. More. Bitcoins biggest tech player to release ai chips and computers. *IEEE Spectrum*, November 2017.

[14] S. K. More. Ceva neupro: A family of ai processors for deep learning at the edge. *IEEE Spectrum*, November 2017.

[15] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. Cambricon-x: An accelerator for sparse neural networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, page 20. IEEE Press, 2016.