A Word-Level Multi-Precision Systolic Array Architecture for Accelerating Deep Neural Networks

Xinjun Zhou, Fen Ge, Hui Chen and Weiqiang Liu
Nanjing University of Aeronautics and Astronautics, Nanjing, China
{zhouxinjun, gefen, huichen, liuweiqiang}@nuaa.edu.cn

***Abstract*—Systolic arrays are widely adopted in deep neural network (DNN) accelerators due to their regular structure and energy-efficient data movement, but conventional designs suffer from long operand pre-filling latency and inefficient arithmetic unit utilization when supporting mixed-precision computation. This paper presents a multi-precision systolic array architecture that jointly optimizes processing element (PE) arithmetic design and dataflow. A word-parallel PE is proposed to support dynamic reconfiguration among 4-bit, 8-bit, and 16-bit precisions while maintaining high arithmetic utilization, and a ring-based diagonal dataflow is introduced to reduce operand pre-filling latency through bidirectional propagation, accelerating arithmetic operation scheduling. Implemented in a 28 nm CMOS technology, the proposed architecture achieves up to 36% higher MAC utilization, 6.31× higher throughput and 4.44× higher energy efficiency compared with existing mixed-precision architectures under 4-bit precision, while reducing execution time by up to 39.0% over conventional systolic arrays. When applied to DNN accelerators, the proposed architecture achieves a 1.59× performance improvement and a 1.61× reduction in energy consumption compared with existing architectures.**

***Index Terms*—Mixed-precision, systolic array, ring-based dataflow, word-level.**

I. Introduction

Deep Neural Networks (DNNs) have been widely adopted in a broad range of applications, including computer vision [1], natural language processing [2], and recommendation systems [3]. The rapid advancement of DNN models has led to significant improvements in accuracy and generalization capability. However, their rapidly growing computational cost poses severe challenges to performance scalability and energy efficiency, making hardware accelerators indispensable for practical deployment.

To address these challenges, systolic array architectures have been explored for DNN acceleration. Systolic arrays adopt a deterministic and spatially regular dataflow, in which arithmetic operands are propagated locally between neighboring processing elements (PEs) with minimal control logic. This structured data movement pattern significantly reduces global data transfers, DRAM access frequency, and control overhead, resulting in lower energy consumption and improved energy efficiency [4], [5]. Despite these advantages, a drawback of conventional systolic arrays is their relatively long pre-filling time, caused by the unidirectional propagation of data through the array.

To mitigate this issue, Axon [6] introduces a diagonal data-feeding strategy that enables bidirectional propagation of arithmetic operands within the systolic array, reducing pre-filling latency compared with conventional systolic arrays and improving runtime efficiency. In addition, Axon demonstrates favorable energy efficiency and improved utilization for memory-bound operations. Nevertheless, the diagonal dataflow in Axon constrained by boundary-based operand injection, which fundamentally limits further reduction of the pre-filling latency. More importantly, these dataflow optimizations are typically designed under fixed precision assumptions, making them inefficient for supporting flexible or fine-grained precision switching within the array.

While optimizing dataflow improves data movement efficiency, maximizing the computation efficiency within each PE is equally critical, especially for mixed-precision DNN models [7], [8]. However, the majority of existing systolic array architectures are designed to support only a single or fixed precision [9], [10], which fundamentally limits their ability to efficiently exploit mixed-precision DNN workloads. The systolic arrays support for mixed-precision computation has primarily focused on precision-scalable multiply–accumulate (MAC) units, where arithmetic units can be dynamically reconfigured to operate at different bit-widths. Prior work [11]–[14] has demonstrated that precision-scalable MAC units can provide proportional throughput gains as precision is reduced. However, many of these architectures suffer from insufficient MAC utilization at low precision [11], [12]. To alleviate this issue, FlexBlock [15] reorganizes precision-scalable MAC units into hierarchical processing structures, enabling partial sums to be selectively accumulated or bypassed depending on the precision mode. While this approach substantially improves MAC utilization across different precisions and layer types compared to earlier fusion-based architectures [11], insufficient MAC utilization can still be observed in DNN models such as MobileNetV1 [16], where depthwise and pointwise convolution layers expose limited accumulation depth and operand-

This work was supported in part by the NSFC Distinguished Young Scholars Fund under Grant 62425404, in part by the National Nature Science Foundation of China under Grant 62404102 and in part by the Jiangsu Province Major Scientific Project under Grant BG2025012.

level parallelism. Moreover, FlexBlock is fundamentally built upon conventional MAC array organizations. Compared to systolic arrays, such organizations provide less efficient on-chip data reuse and therefore tend to incur more frequent off-chip memory accesses, leading to non-negligible latency and energy overhead [17].

Motivated by these observations, this paper proposes a word-level multi-precision systolic array architecture that jointly optimizes both the PE arithmetic unit and the systolic dataflow organization. The proposed architecture achieves high MAC utilization across multiple precision modes while greatly reducing execution latency. The main contributions of this work are summarized as follows:

1) A multi-precision systolic array architecture is proposed, supporting dynamic switching among 4-bit, 8-bit, and 16-bit precisions.
2) A word-level parallelism within each PE is proposed, which achieves up to 36% higher MAC utilization and 4.44× higher energy efficiency compared with existing architectures under the 4-bit precision mode.
3) A novel ring-based diagonal dataflow for the systolic array is proposed, achieving up to 39.0% runtime reduction compared with conventional systolic arrays.
4) The proposed architecture is evaluated across diverse DNN workloads. Experimental results demonstrate an average 1.59× inference performance improvement and a 1.61× reduction in energy consumption compared with existing mixed-precision architectures.

II. Background and Motivation

A. Systolic Array

Systolic arrays are widely adopted in DNN accelerators due to their regular structure, scalable parallelism, and energy-efficient data movement [4], [5]. A conventional systolic array is composed of a two-dimensional grid of homogeneous PEs, where each PE integrates a MAC unit and local buffers for temporary data storage, and communicates only with its adjacent PEs, as illustrated in Fig. 1(b). Computation is organized in a boundary manner in which input feature maps (ifmaps) and weights are injected from the edges of the array and propagate across the PEs in fixed directions, while partial sums are accumulated within the array as data advances. This highly structured organization enables predictable dataflow and efficient on-chip data reuse, making systolic arrays well suited for large-scale matrix computations commonly found in DNN workloads.

B. Dataflow and Runtime Modeling

Systolic arrays typically employ three representative dataflow patterns: output-stationary (OS), weight-stationary (WS), and input-stationary (IS). In the OS dataflow, partial sums remain stationary within each PE, while ifmaps and weights are streamed through the array. In the WS dataflow, weights are preloaded and kept stationary in the PEs, whereas ifmaps and partial sums propagate across the array. The IS dataflow follows a similar principle, except that ifmaps are

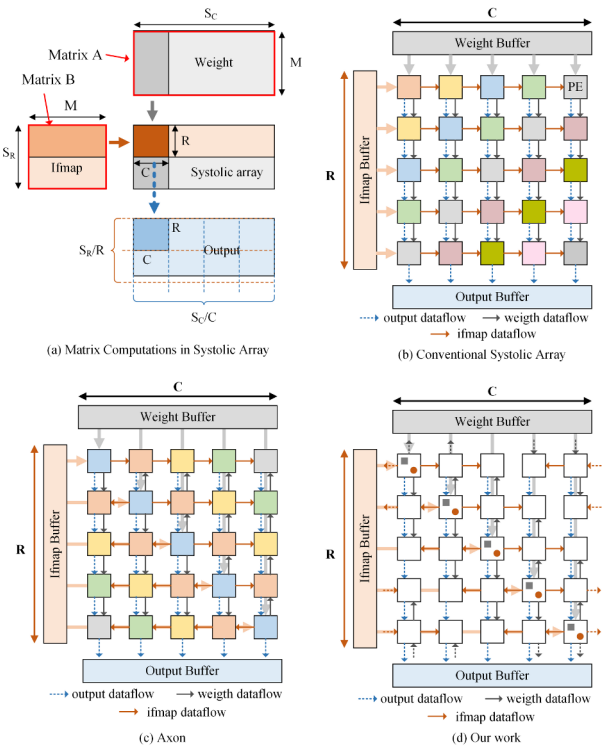

Fig. 1. Conventional systolic array, Axon [6] and our work.

held stationary and weights are streamed through the systolic array. [18] developed an analytical model for systolic array runtime, which is applicable for all the three dataflows. The OS mode is taken as an example. The matrix computation in Fig. 1(a) and the conventional systolic array shown in Fig. 1(b) are used to derive the runtime as:

$$T = (2R + C + M - 2) \cdot \lceil \frac{S_R}{R} \rceil \cdot \lceil \frac{S_C}{C} \rceil \tag{1}$$

where R and C are the rows and columns of the systolic array and M is the temporal dimensions, S_R and S_C are matrics rows and columns.

Specifically, the pre-filling latency corresponds to the time required to inject operands from the array boundaries until all PEs become fully active, which equals $(R+C-1)\cdot\lceil S_R/R\rceil\cdot\lceil S_C/C\rceil$. The term $(M-1)\cdot\lceil S_R/R\rceil\cdot\lceil S_C/C\rceil$ represents the effective computation time, during which each PE performs M MAC operations for each tiled sub-matrix. Finally, the output readout phase incurs a latency of $R\cdot\lceil S_R/R\rceil\cdot\lceil S_C/C\rceil$, corresponding to draining the accumulated results from the systolic array.

From the runtime decomposition, it can be observed that when the target matrices are small and need to be processed in a large number of tiles, the pre-filling latency constitutes a large portion of the total execution time, thereby limiting overall runtime efficiency. To mitigate this issue, Axon [6] introduces a diagonal input data orchestration scheme, as illustrated in Fig. 1(c), in which operands are injected through

PEs located on the principal diagonal and then propagated bi-directionally within the array. By reducing the maximum distance that operands need to travel, this approach effectively shortens the pre-filling time and improves overall runtime efficiency. Under this diagonal feeding strategy, the pre-filling latency is reduced to $\max(R, C) \cdot \lceil S_R/R \rceil \cdot \lceil S_C/C \rceil$, and the total runtime can be reformulated as:

$$T = (R + M + \max(R, C) - 1) \cdot \lceil \frac{S_R}{R} \rceil \cdot \lceil \frac{S_C}{C} \rceil \quad (2)$$

However, this form of dataflow remains constrained by boundary-based propagation, which fundamentally limits further reduction of the pre-filling latency.

C. Precision-Scalable MAC units

MAC units are the fundamental building blocks of systolic array-based DNN accelerators. A conventional fixed-precision MAC typically operates on high-precision operands (e.g., 16-bit or 32-bit), which guarantees numerical accuracy but leads to substantial area, power, and bandwidth overhead. As a result, recent accelerators increasingly adopt precision-scalable MAC units, which can dynamically adjust computation precision to match algorithmic requirements. Fig. 2(a) illustrates the principle of a standard 16-bit×16-bit multiplication. Given two 16-bit operands X and W, each can be decomposed into four 4-bit sub-words as:

$$X = \sum_{i=0}^{3} x_i 2^{4i}, \quad W = \sum_{j=0}^{3} w_j 2^{4j} \quad (3)$$

where x_i and w_j denote the corresponding 4-bit segments. The full-precision multiplication can then be expressed as:

$$X \cdot W = \sum_{i=0}^{3} \sum_{j=0}^{3} (x_i \cdot w_j) 2^{4(i+j)} \quad (4)$$

which consists of sixteen 4-bit×4-bit partial products followed by shift-and-accumulate operations.

Early precision-scalable MAC designs adopt bit-serial computation schemes [19], [20], but they require varying numbers of clock cycles under different precision modes and rely on more complex control logic, which limits their achievable throughput. Several recent works [11], [15], [21] employ bit-parallel designs to achieve higher computational throughput. Among them, BitFusion [11] is a representative precision-scalable architecture, whose MAC organization is illustrated in Fig. 2(b). BitFusion decomposes high-precision multiplications into multiple low-precision operations using an array of small multipliers, enabling support for 4-bit, 8-bit, and 16-bit computations through dynamic fusion of compute units. However, BitFusion suffers from insufficient MAC utilization at low precision, particularly when the input parallelism is limited. As shown in Fig. 3(a), under the 4-bit precision mode, when only two input operands are available, merely a small fraction of the available MAC resources can be activated, resulting in a MAC utilization of only 12.5%.

Building upon BitFusion, FlexBlock [15] further improves the MAC organization by transforming the original 2D parallelism into a 1D parallelism structure, as shown in Fig. 2(c). FlexBlock allows multiple outputs to be produced in parallel under low-precision modes, thereby alleviating partial-sum accumulation bottlenecks and improving overall throughput. Nevertheless, FlexBlock still exhibits underutilized MAC resources under low-precision and low input parallelism scenarios. As illustrated in Fig. 3(b), when operating at 4-bit precision with only two input operands, the MAC utilization reaches 50%. Such low parallelism is common in practical DNN workloads that exhibit limited channel-wise or accumulation depth, such as depthwise convolutions and certain pointwise or reduction dominated layers. A representative example is MobileNetV1 [16], where depthwise convolution layers inherently expose limited channel-wise parallelism, often resulting in insufficient operand availability and low MAC utilization when mapped onto systolic arrays.

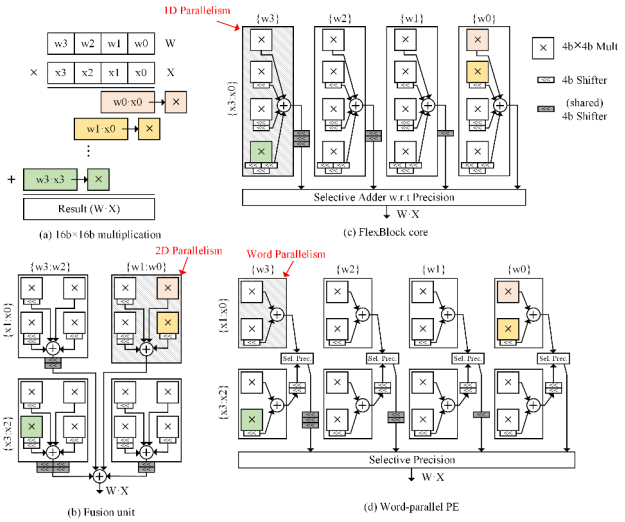

Fig. 2. Organizations of precision-scalable MAC units. (c) Fusion unit in BitFusion [11] and (d) Flexblock core in [15].

III. Hardware Architecture

A. Overall Architecture

We adopt a two-dimensional systolic array composed of homogeneous PEs as the overall architecture, as illustrated in Fig. 4. Each PE integrates local computation and storage resources and shares a unified control logic across the entire array. The systolic array is equipped with dedicated control mechanisms for weights and ifmaps, which control the injection and propagation of operands within the array. In addition, a global precision control signal is broadcast to all PEs to configure their computation precision. Input buffers (IBUFs) and weight buffers (WBUFs) are placed at the boundaries of the array, where IBUFs distribute ifmaps along rows and WBUFs supply weights along columns. Output

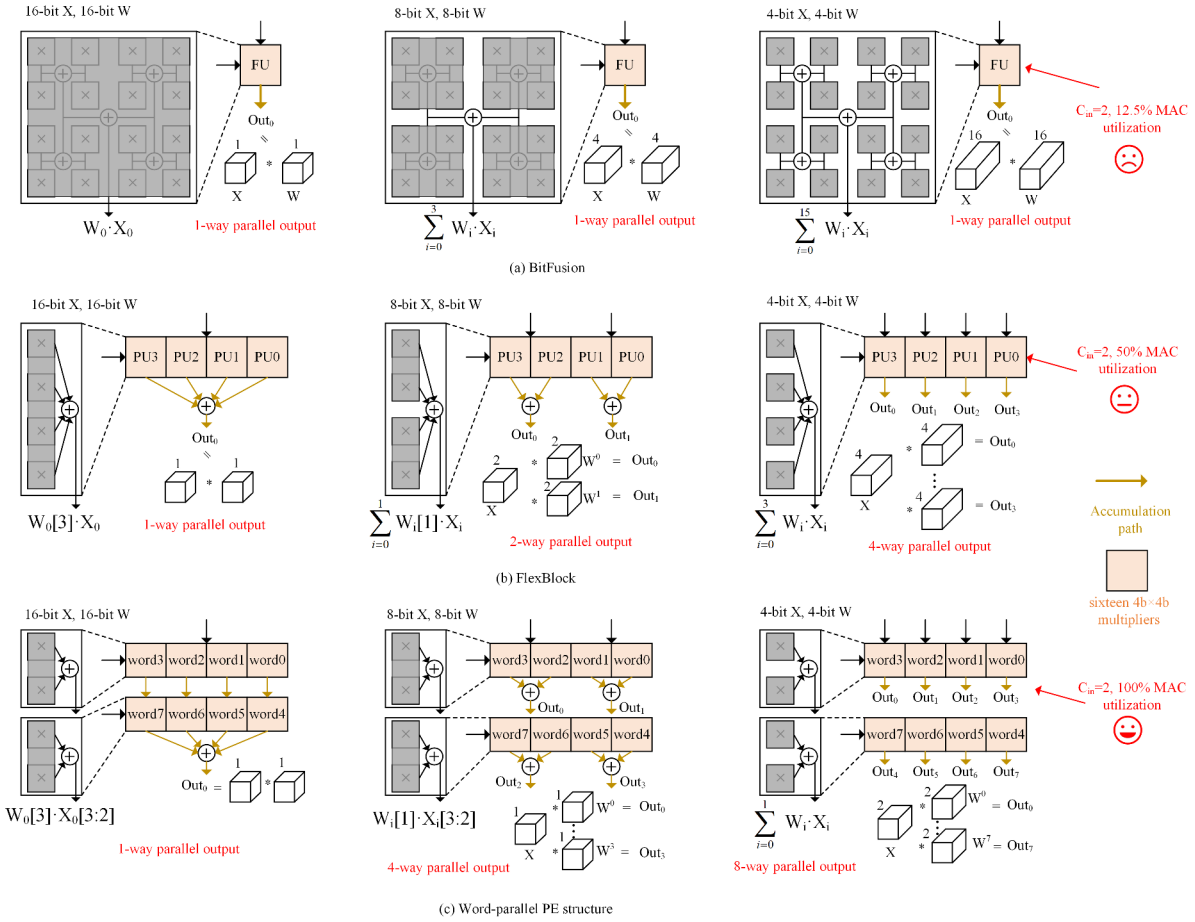

Fig. 3. PE structures under different precision modes. (a) BitFusion in [11] and (b) Flexblock in [15]

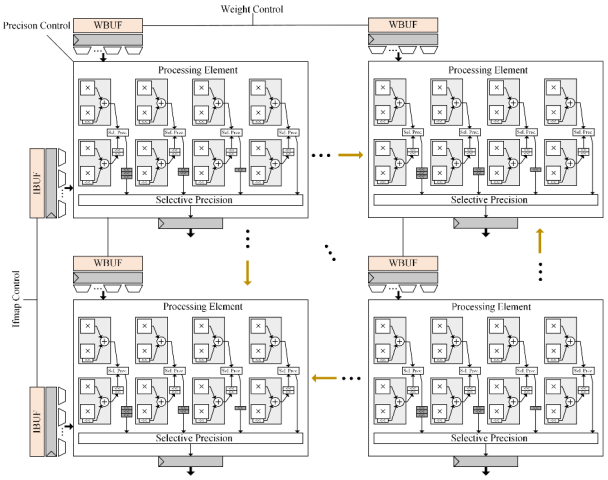

Fig. 4. Architecture of the multi-precision systolic array.

buffers are located at the bottom of the array to collect the results accumulated by column-wise accumulators.

Based on this organization, the systolic array operates as a single computational unit capable of executing matrix–vector multiplications under different precision modes. The structured sharing of ifmaps and weights across rows and columns enables efficient data reuse and accumulation, thereby reducing accesses to on-chip memory. By employing homogeneous PEs with shared control logic, the proposed architecture minimizes control overhead and simplifies hardware implementation. Moreover, using the proposed PE as the fundamental building block allows the systolic array to achieve a balanced match between computation throughput and supported precision levels, which directly determines the achievable parallelism at the array level.

B. PE Design

To support efficient multi-precision computation while maintaining high hardware utilization, this work proposes a word-parallel PE structure, as illustrated in Fig. 2(d). The design groups two adjacent multipliers along the X dimension into a word, enabling word-level parallelism. A total of eight such words are integrated within one PE. Both W and X operands exploit word parallelism, while the accumulation of

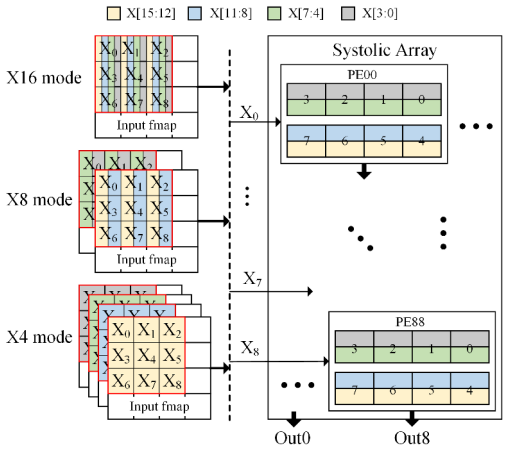

Fig. 5. Reconfigurable mapping of ifmaps under different precision modes.

partial sums (psums) is dynamically configured according to the selected precision mode. Specifically, in the 16-bit mode, psums from all eight words are accumulated to generate a single output. In the 8-bit mode, the psums are grouped and accumulated to produce four independent outputs. In the 4-bit mode, the psums from the eight words bypass the adder tree and are directly forwarded as independent outputs. This flexible accumulation strategy allows the PE to adapt its output parallelism to the computation precision without introducing additional control complexity.

The advantages of the proposed word-parallel PE is more clear when compared with prior designs, as shown in Fig. 3. While all designs exhibit identical behavior under the 16-bit mode, the proposed PE achieves higher output parallelism at lower precisions. In the 8-bit mode, the output parallelism is increased by $4\times$ over BitFusion and $2\times$ over FlexBlock. In the 4-bit mode, the gains further increase to $8\times$ and $2\times$, respectively. Importantly, the proposed PE improves MAC utilization under low input parallelism. For example, in the 4-bit mode with only two input operands available, BitFusion and FlexBlock achieve MAC utilizations of only 12.5% and 50%, respectively, whereas the proposed PE fully activates all MAC units, achieving 100% utilization. As a result, this PE design effectively enhances both MAC efficiency and overall throughput across different precision modes.

C. Reconfigurable Mapping

The proposed architecture supports fine-grained reconfigurability by dynamically adapting the mapping of ifmaps and weights to the word-parallel PEs under different precision modes. Fig. 5 and Fig. 6 illustrate the data distribution mechanisms for ifmaps and weights, respectively.

Fig. 5 illustrates the reconfigurable mapping of ifmaps. Each ifmap element is broadcast along the rows of the systolic array and mapped to the word-parallel PEs according to the selected precision mode. In the 16-bit mode (X16), each ifmap element is decomposed into four 4-bit units, which are spatially aligned

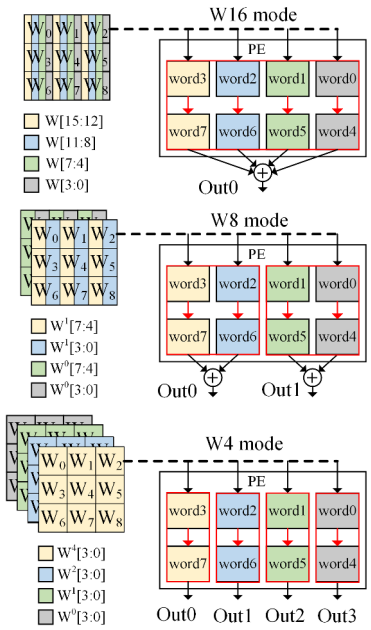

Fig. 6. Reconfigurable mapping of weights under different precision modes.

with the corresponding 4-bit×4-bit multipliers within a PE. The resulting partial sums are fully accumulated through the adder tree to produce a single output. In the 8-bit mode (X8), each ifmap element is decomposed into two 4-bit units, enabling two independent ifmap groups to be processed in parallel, each mapped to a set of 4-bit×4-bit multipliers. In the 4-bit mode (X4), each ifmap element corresponds to a single 4-bit unit, allowing four independent ifmap groups to be processed concurrently.

Fig. 6 shows the corresponding weight distribution strategy. In the 16-bit mode (W16), each weight is decomposed into four 4-bit units that are aligned with the ifmap units and broadcast along the columns of the systolic array, enabling full accumulation within each PE. In the 8-bit mode (W8), each weight is decomposed into two 4-bit units and mapped to 4-bit×4-bit multipliers. When ifmaps are 16-bit, two outputs can be generated in parallel; when ifmaps operate at lower precision, up to four outputs can be produced concurrently. In the 4-bit mode (W4), each weight corresponds to a single 4-bit unit mapped to a 4-bit×4-bit multiplier. In this case, four outputs can be generated in parallel with 16-bit ifmaps, and up to eight outputs can be produced when ifmaps operate at lower precision. This word-level channel partitioning enables multiple output channels to be computed concurrently while maintaining full MAC utilization, even under limited input or output dimensionality.

D. Systolic Array Dataflow

To further reduce the pre-filling latency inherent in conventional systolic arrays, we employ a diagonal input strategy combined with a ring-based bidirectional propagation scheme.

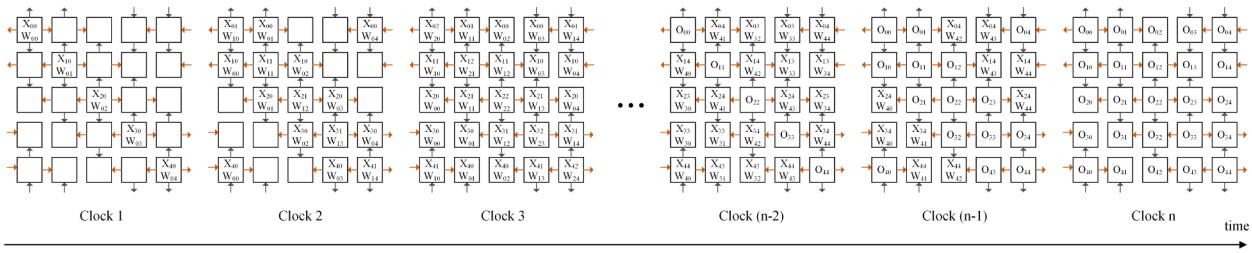

Fig. 7. Temporal evolution of the ring-based diagonal dataflow.

As illustrated in Fig. 1(d), input operands are injected through PEs located along the principal diagonal of the array and then propagate in two opposite directions along each dimension. When reaching the array boundary, operands are wrapped around to the opposite side, forming a logical ring that enables faster spatial propagation of data across the array. Compared with the diagonal bidirectional dataflow in Axon, the ring-based propagation further shortens the maximum travel distance of operands, allowing data to reach all PEs with fewer pipeline stages while preserving a regular and deterministic routing structure.

Under this ring-based diagonal OS dataflow, the effective pre-filling latency is reduced to approximately half of that in Axon, and can be expressed as $\lceil \max(R, C)/2 \rceil \cdot \lceil S_R/R \rceil \cdot \lceil S_C/C \rceil$. The total execution time of a matrix multiplication can be expressed as

$$T = (R + M + \lceil \max(R, C)/2 \rceil - 1) \cdot \left\lceil \frac{S_R}{R} \right\rceil \cdot \left\lceil \frac{S_C}{C} \right\rceil \quad (5)$$

Fig. 7 further visualizes the temporal evolution of data within the systolic array, illustrating how operands are rapidly distributed across the array and how partial sums remain stationary within PEs during computation. For example, in a 5×5 systolic array ($R = C = 5$) executing a matrix multiplication with $S_R = S_C = 5$, both the array pre-filling and the output draining phases complete within only three clock cycles each under the proposed dataflow, clearly demonstrating the effectiveness of the ring-based diagonal propagation. By combining output-stationary accumulation with ring-based diagonal input, the proposed dataflow achieves significantly reduced pre-filling latency while compatibility with multi-precision execution.

IV. Evaluation

A. MAC utilization for PE

We compare the MAC utilization of the proposed architecture with BitFusion and FlexBlock using the MobileNetV1 [16] benchmark, which is representative of practical DNN workloads with limited channel-wise parallelism due to the extensive use of depthwise convolution layers. The evaluation covers three major phases of training and inference, including forward propagation (FW), backward propagation (BW), and weight update (WU), under different fixed precisions. The experiments results are illustrated in Fig. 8.

Under the 16-bit precision mode, the proposed design achieves an average MAC utilization improvement of 16% compared to BitFusion, while exhibiting comparable utilization to FlexBlock. In the 8-bit mode, the proposed architecture improves MAC utilization by 21% and 4% over BitFusion and FlexBlock, respectively. When operating at the 4-bit precision, the improvements are 36% over BitFusion and 11% over FlexBlock. Overall, the proposed architecture achieves high MAC utilization across all precision modes, with noticeable improvements at low precision.

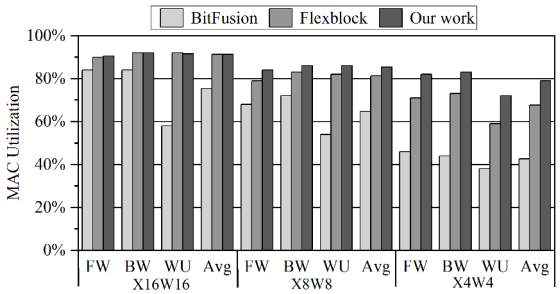

Fig. 8. Comparison with the MAC utilization during training of MobileNetV1 with various precision-scalable MAC arrays.

B. Runtime Analysis of Systolic Array Dataflow

We evaluate the runtime advantages of the proposed systolic array dataflow by comparing it with Axon and the conventional systolic array. The evaluation considers matrix multiplication between a 49×16 matrix and a 16×49 matrix, while varying the systolic array size from 9×9 to 64×64. For each configuration, the execution time is measured and normalized against the conventional systolic array baseline.

As shown in Fig. 9, both Axon and the proposed design reduce execution time compared to the conventional systolic array by alleviating operand pre-filling latency. Axon achieves runtime reductions ranging from 7.3% to 24.2%, with more pronounced benefits observed at smaller array sizes. The proposed ring-based diagonal dataflow delivers larger speedups, achieving runtime improvements of 22.8% to 39.0% across all evaluated configurations.

C. Architecture Comparison

To evaluate the effectiveness of the proposed hardware architecture, we compare it against three representative mixed-precision architectures: BitFusion Cores, FlexBlock Cores, and BitSys Cores. For a fair comparison, all designs are

TABLE I
ARCHITECTURAL COMPARISONS BETWEEN BITFUSION-BASED CORES, FLEXBLOCK CORE, BITSYS CORES, AND PROPOSED ARCHITECTURE IN TERMS OF AREA, POWER CONSUMPTION, AND ENERGY EFFICIENCY.

Hardware Architecture	BitFusion Cores [11]	FlexBlock Cores [15]	BitSys Cores [14]	Our work
Technology	28nm	28nm	28nm	28nm
Supported Precision	4/8/16	4/8/16	4/8/16	4/8/16
Array Size	16×16 (for FIX16)	16×16 (for FIX16)	16×16 (for FIX16)	16×16 (for FIX16)
Area [μm^2]	332733	350449	342142	371864
Power Consumption [mW]	67.21 / 66.35 / 65.63	97.12 / 96.45 / 95.82	59.23 / 58.05 / 57.89	83.42 / 81.21 / 80.37
Clock Frequency	333MHz	333MHz	333MHz	333MHz
Throughput [GFLOPS]	176.26 / 110.63 / 41.25	548.75 / 209.38 / 45.22	151.34 / 78.25 / 35.21	**954.35** / **420.32** / **46.45**
Efficiency [GFLOPS/W]	2622.53 / 1667.37 / 628.52	5650.23 / 2170.87 / 471.93	2555.12 / 1347.98 / 608.22	**11332.41** / **5157.72** / 577.95

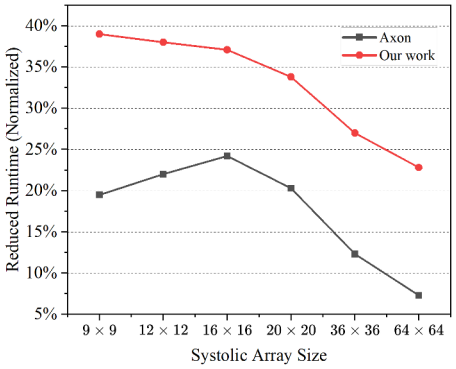

Fig. 9. Runtime improvements over conventional systolic array and Axon.

synthesized using Synopsys Design Compiler in a 28 nm CMOS technology node, operating at a clock frequency of 333 MHz. A systolic array configuration equivalent to a 16-bit $\times$ 16-bit MAC array is adopted for all architectures. We evaluate and compare power consumption, area, throughput, and energy efficiency under different fixed-point precision modes, including 4-bit, 8-bit, and 16-bit configurations.

The experimental results are summarized in Table I. As shown, when supporting identical 4/8/16-bit fixed-point precisions, the proposed architecture achieves the highest throughput among all compared designs. In particular, under the 4-bit precision mode, the proposed architecture outperforms BitFusion, FlexBlock, and BitSys by $5.41\times$, $1.74\times$, and $6.31\times$, respectively. Correspondingly, the energy efficiency under 4-bit precision is improved by $4.32\times$, $2.01\times$, and $4.44\times$ compared to these three architectures. Although the proposed design incurs a modest increase in area and power consumption, this overhead primarily stems from additional control logic required to support fine-grained precision selection and dynamic reconfiguration. Nevertheless, the substantial gains in throughput and energy efficiency demonstrate that the proposed architecture achieves a superior performance and efficiency trade-off, particularly under low-precision operating modes.

TABLE II
EVALUATED DNN BENCHMARKS.

DNN model	Type	Domain	Precision
MobileNetV1	CNN	Image Classification	4/8/16
ResNet-18	CNN	Image Classification	4/8/16
LSTM	RNN	Language Modeling	4/8/16
DQN	DRL	Control Policy	4/8/16
SAC	DRL	Control Policy	4/8/16

D. Performance and Energy Evaluation on DNN workloads

Table II summarizes five representative benchmarks covering Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and neural networks used in Deep Reinforcement Learning (DRL) workloads from diverse application domains, including image classification, language modeling, and control policy learning. The selected benchmarks encompass model weights and computational characteristics, enabling a evaluation of the proposed architecture in comparison with FlexBlock. Specifically, we evaluate MobileNetV1 [16], ResNet-18 [22], LSTM [23], and the neural network backbones of DQN [24] and SAC [25], all of which are popular and widely used DNN models.

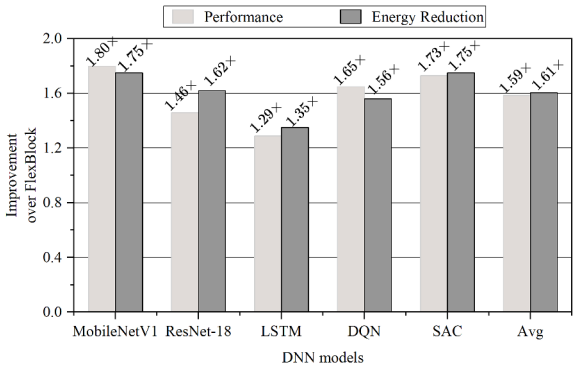

Fig. 10. Performance and energy improvements over Flexblock.

To evaluate the performance and energy efficiency advantages of the proposed architecture, we conduct a comparison

with FlexBlock under identical experimental conditions. We adopt the quantized models provided by FlexBlock and ensure that both designs use the same precision bit-width. Both architectures are configured with an equivalent 16×16 array, synthesized in a 28 nm CMOS process and operated at a clock frequency of 333 MHz. Inference performance and energy consumption are evaluated across all selected benchmarks.

As shown in Fig. 10, the proposed architecture outperforms FlexBlock in both performance and energy efficiency across all evaluated models. On average, the inference performance is improved by $1.59\times$, while the energy consumption is reduced by $1.61\times$. The performance gains are primarily attributed to the higher degree of parallelism and improved MAC utilization enabled by the proposed word-parallel PE design. Meanwhile, the energy efficiency improvements mainly stem from the systolic array organization, which reduces memory access frequency compared to conventional fixed MAC array architectures.

V. Conclusion

This paper proposes a word-level multi-precision systolic array architecture for efficient DNN acceleration, which jointly optimizes PE parallelism and systolic dataflow. By employing a word-parallel PE design and a ring-based diagonal dataflow, the proposed architecture achieves high MAC utilization across 4-bit, 8-bit, and 16-bit precision modes while significantly reducing pre-filling latency. Implemented in a 28 nm CMOS technology, the proposed design demonstrates up to 36% higher MAC utilization and $4.44\times$ higher energy efficiency under 4-bit precision, achieves up to 39.0% runtime reduction compared with conventional systolic arrays, and delivers an average $1.59\times$ inference performance improvement with $1.61\times$ lower energy consumption across diverse DNN workloads, indicating its effectiveness for mixed-precision DNN acceleration.

References

[1] S. R. Reddy, G. S. Varma, and R. L. Davuluri, "Deep neural network (dnn) mechanism for identification of diseased and healthy plant leaf images using computer vision," *Annals of Data Science*, vol. 11, no. 1, pp. 243–272, 2024.

[2] B. Rogers, N. Noman, S. Chalup, and P. Moscato, "A comparative analysis of deep neural network architectures for sentence classification using genetic algorithm," *Evolutionary Intelligence*, vol. 17, no. 3, pp. 1933–1952, 2024.

[3] B. B. Sinha and R. Dhanalakshmi, "Dnn-mf: Deep neural network matrix factorization approach for filtering information in multi-criteria recommender systems," *Neural Computing and Applications*, vol. 34, no. 13, pp. 10 807–10 821, 2022.

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

[5] Z.-G. Liu, P. N. Whatmough, and M. Mattina, "Systolic tensor array: An efficient structured-sparse gemm accelerator for mobile cnn inference," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 34–37, 2020.

[6] M. M. R. Nayan, R. Raj, G. B. Shaik, T. Krishna, and A. J. Naeemi, "Axon: A novel systolic array architecture for improved run time and energy efficient gemm and conv operation with on-chip im2col," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.

[7] J. Xu, J. Yu, X. Liu, and H. Meng, "Mixed precision dnn quantization for overlapped speech separation and recognition," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 7297–7301.

[8] S. W. Pun, B. Bao, S.-I. Filip, G. Lemieux, J. V. Kim, N. Misyats, N. Pande, V. Ravain, and R. Sherrick, "Range extension with super-normals for mixed-precision 8-bit dnn training," in *2025 IEEE 32nd Symposium on Computer Arithmetic (ARITH)*. IEEE, 2025, pp. 1–4.

[9] H. Sun, J. Shen, T. Zhang, Z. Tang, C. Zhang, Y. Li, Y. Shi, and H. Liu, "Fams: A framework of memory-centric mapping for dnns on systolic array accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2025.

[10] C. Sestito, S. Agwa, and T. Prodromakis, "Trim, triangular input movement systolic array for convolutional neural networks: Dataflow and analytical modelling," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2025.

[11] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[12] S. Ryu, H. Kim, W. Yi, and J.-J. Kim, "Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[13] X. Liu, X. Wu, H. Shao, and Z. Wang, "A flexible fpga-based accelerator for efficient inference of multi-precision cnns," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2024, pp. 1–5.

[14] Y. Liu, S. Ullah, and A. Kumar, "Bitsys: Bitwise systolic array architecture for multi-precision quantized hardware accelerators," in *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2024, pp. 220–220.

[15] S.-H. Noh, J. Koo, S. Lee, J. Park, and J. Kung, "Flexblock: A flexible dnn training accelerator with multi-mode block floating point support," *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2522–2535, 2023.

[16] 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.

[17] Y. Zheng, H. Yang, Y. Shu, Y. Jia, and Z. Huang, "Optimizing off-chip memory access for deep neural network accelerator," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 4, pp. 2316–2320, 2022.

[18] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 58–68.

[19] X. Cheng, Y. Wang, W. Ding, H. Lou, and P. Li, "Leveraging bit-serial architectures for hardware-oriented deep learning accelerators with column-buffering dataflow," *Electronics*, vol. 13, no. 7, p. 1217, 2024.

[20] S. Li and P. Gupta, "Bit-serial weight pools: Compression and arbitrary precision execution of neural networks on resource constrained processors," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 238–250, 2022.

[21] E. Reggiani, A. Pappalardo, M. Doblas, M. Moreto, M. Olivieri, O. S. Unsal, and A. Cristal, "Mix-gemm: An efficient hw-sw architecture for mixed-precision quantized deep neural networks inference on edge devices," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1085–1098.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. Pmlr, 2018, pp. 1861–1870.