SEA: Sign-Separated Accumulation Scheme for Resource-Efficient DNN Accelerators

Jing Gong*, Hassaan Saadat†, Haris Javaid‡, Hasindu Gamaarachchi*§, David Taubman†, Sri Parameswaran¶
* School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia
† School of Electrical Engineering and Telecommunications, The University of New South Wales, Sydney, Australia
‡ AMD, Singapore
§ Garvan Institute of Medical Research, Darlinghurst, Australia
¶ School of Electrical and Computer Engineering, The University of Sydney, Sydney, Australia

***Abstract*—Deep neural network (DNN) accelerators targeting training need to support the resource-hungry floating-point (FP) arithmetic. Typically, the additions (accumulations) are performed with higher precision compared to multiplications, resulting in a substantial proportion of resources consumed by the adders. In this paper, we aim to improve the resource efficiency of FP addition in DNN training accelerators and present a sign-separated accumulation scheme (SEA). The proposed SEA scheme accumulates the same-signed terms separately, followed by a final addition of the two oppositely-signed sub-accumulations. The separate sub-accumulations, when performed using our resource-efficient same-signed FP adders, result in significant improvement in overall resource efficiency. We present a SEA-based systolic array design, involving a novel processing element design, capable of performing FP matrix multiplications for DNN training. Our experimental results show substantial improvements in delay, area-delay product (ADP), and energy consumption, by achieving reductions of up to 19.8% in delay, 29.1% in ADP and 30.7% in energy, across various adder-multiplier combinations compared to the original designs. SEA introduces no approximations in accumulation or modifications in the rounding mechanism. We integrate the SEA-based systolic array into the open-source Gemmini [1] ecosystem for use by the broader community.**

***Index Terms*—DNN training, Gemmini, hardware accelerator, multiply-and-accumulate, systolic array.**

I. Introduction

Over the last decade, specialized accelerators to enhance the performance and resource efficiency of deep neural networks (DNN) training and inference have received substantial attention. Various designs have been proposed based on a variety of spatial array architectures (systolic arrays [1], [2] or vector [3], [4]), dataflows [1], [5], and on-chip storage hierarchies [6]. The DNN accelerators that target the training phase need to support FP operations [7], which are significantly more resource-hungry than their integer/fixed-point counterparts used in inference accelerators. Thus, *to build a high-performance and resource-efficient DNN accelerator for training, the resource efficiency of FP arithmetic operations is paramount to complement the various innovative spatial array architectures and dataflows.*

The core of DNN computations is the general matrix multiplication (GEMM), which fundamentally consists of a series of additions succeeding multiplications. Typically, the additions are performed with higher precision compared to multiplications, which results in a substantial proportion of resource-consumption by the addition operations. This is demonstrated in Fig. 1, which illustrates the proportion of resources consumed by adders within a typical systolic array, for various commonly encountered combinations of adder and multiplier datatypes (denoted by *a*ADD*m*MULT). Notably, the adders alone account for more than 50% of the ADP and energy in all combinations, emphasizing the importance of resource-efficient FP addition in DNN computations.

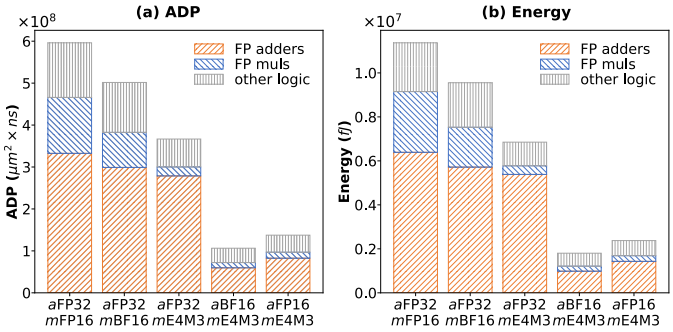

Fig. 1. The proportion of resources consumed by floating-point adders within a 6×6 weight-stationary systolic array for common combinations of *a*ADD*m*MULT datatypes: (a) area-delay product (ADP); (b) energy. These combinations of datatypes are commonly supported in many readily-available products from AMD [8], NVIDIA [7], [9], [10], Google [11], and Intel [12].

This paper targets improving the resource efficiency of the FP addition operations in DNN accelerators; primarily by exploiting the fact that DNN involves a series of additions (accumulations). We propose a novel accumulation scheme (SEA) that accumulates the same-signed terms separately, followed by a final addition of the two opposite-signed sub-accumulations. SEA performs the two same-signed sub-accumulations using our proposed *same-signed FP adders* resulting in superior performance and resource efficiency because our same-signed FP adders are much more resource-efficient than a generic FP adder. Importantly, SEA involves mere re-arrangement of addition operations without any approximation or modification in the IEEE-754 rounding mechanism, hence SEA introduces no errors (FP additions are not associative, which results in minuscule differences in results [13]). Further, we present a SEA-based systolic array design, for the weight stationary dataflow, involving the design of novel processing-elements (PE). Our design maximizes the

979-8-3503-4859-0/DATE24/© 2024 EDAA

utilization of same-signed adders while ensuring the design is transparent at the system level.

Our evaluations show that the proposed SEA results in reductions of up to 29.1% in ADP and 30.7% in energy for various sizes of the systolic array and aADDmMULT combinations. Further, to demonstrate the working of the SEA scheme in an end-to-end system, we have seamlessly integrated our SEA-based systolic array into Gemmini [1], an open-source full-stack DNN-accelerator generator spanning both the hardware and software stack to support different programming requirements, including a push-button and high-level software flow to boost programmers' productivity. We release the Gemmini-integrated SEA implementation as an open-source project [14].

Our key contributions are outlined below:

- We present SEA, a scheme of separately accumulating same-signed values with our proposed resource-efficient same-signed FP adders, followed by final accumulation using a generic FP adder.
- We present a SEA-based systolic array design, involving a novel PE design. Our PE design uses a dual-flow architecture, allowing separate accumulation of positive and negative values.
- We integrate the SEA-based systolic array into the Gemmini ecosystem, which is available at [14].

II. Related Work

Various DNN accelerators have been developed over the last decade, employing different spatial array architectures such as vector-like designs and systolic arrays that offer high parallelism improving performance, power, and memory usage during DNN training and inference. Vector-like accelerators, such as Brainwave [4] and NVDLA [3], leverage the single instruction, multiple data model to enable simultaneous operations on multiple data elements. Systolic arrays, such as Google's TPU [2] and Gemmini [1], are characterized by a rhythmic grid of PEs offering concurrent computations. Gemmini [1], an open-source DNN accelerator generator coupled with a RISC-V processor, leverages systolic arrays to produce efficient ASIC accelerators. Systolic arrays utilize three different dataflow strategies to effectively reduce memory transactions. Weight stationary minimizes weight transfers by re-using weights, output stationary reuses partial sums, and row stationary efficiently balances the movement between weights and activations [15]. The choice between these dataflows depends on specific computational needs and patterns of reuse [15].

Researchers have also focused on enhancing the efficiency of DNN accelerators through the reduction of arithmetic complexity of integer (inference) and FP (training) operations. In the integer domain, techniques include the utilization of approximate arithmetic units [16] and lower precision [17].

The study in [18] proposed a hybrid accumulation strategy within PEs, capitalizing on factoring and utilizing specialized adders for distinct bit categories, supplemented with a carry-propagation adder. While [18] optimizes arithmetic complexity, it focuses on integer arithmetic units within systolic arrays,

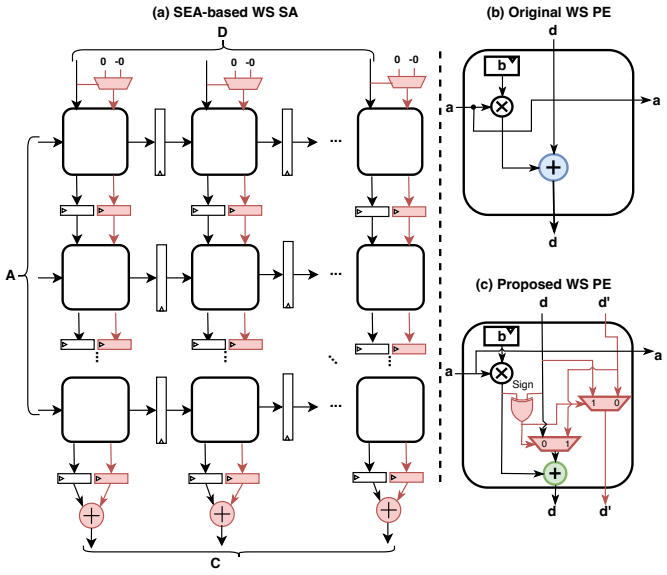

Fig. 2. Proposed SEA-based hardware designs: **(a)** SEA-based weight stationary (WS) systolic array (SA), where the original systolic array is depicted in black; **(b)** original weight stationary processing element (PE) shown for comparison; **(c)** proposed weight stationary processing element. Red and blue adders are generic FP adders, while green adder is same-signed FP adder.

and there has been no investigation or development regarding FP addition with separate accumulation.

In the FP domain, researchers have proposed the use of approximate multipliers for training [19], [20], whereas several works have explored usage of various FP datatypes with reduced bit-width for cutting down computational complexity [9], [7], [11]. In contrast, our work focuses on reducing the arithmetic complexity of FP operations by performing the accumulation of same-signed quantities separately using resource-efficient same-signed adders with no approximation. Moreover, as demonstrated by our experiments, our SEA-based approach can be applied independently of the FP datatype.

III. Proposed SEA Scheme and Designs

The proposed SEA scheme fundamentally involves separate accumulations of the same-signed terms, followed by a final addition of the two opposite-signed sub-accumulations. The separate sub-accumulations, when performed using resource-efficient same-signed FP adders, result in overall computational resource efficiency. In the following subsections, we first describe the SEA-based systolic array design consisting of novel PEs with same-signed FP adders. Then, we discuss the resource-efficient same-signed FP adders.

A. Proposed SEA-based Systolic Array

Fig. 2(a) depicts the proposed SEA-based systolic array following the weight stationary dataflow to perform the multiplication of matrix A and matrix B, with elements a and b, respectively. Similar to a basic weight stationary systolic array[1], it consists of PEs organized in a grid structure, which

[1]Readers are referred to [21] for implementation details.

perform FP multiply-and-accumulate operations, and relay data to adjacent PEs through pipeline registers placed between the PEs. The weight matrix B is preloaded into the registers residing inside the PEs. Next, over several cycles, the input activation matrix A elements are introduced from the left and the partial sum matrix D elements are introduced from the top, allowing the matrix multiplication operation $C = A \times B + D$ in a rhythmic manner.

In the basic systolic array with weight stationary dataflow (as depicted by black-coloured components in Fig. 2(a) with its PE in Fig. 2(b)), the partial sum (accumulations) of each element in the matrix multiplication flow from top to bottom, while getting updated by the PEs. Each PE computes the product between a and b and adds it to the partial sum received from the above PE. It then relays the updated partial sum (accumulation) to the PE below it and the element a to the PE at its right, through the pipeline registers. The relayed entities are then used in the next cycle.

In our proposed SEA-based systolic array, we aim to perform the accumulation of same-signed quantities separately. This necessitates the flow of two distinct sub-accumulations (partial sums) from top to bottom. Specifically, as depicted in Fig. 2(a), each PE receives and relays two distinct partial sums (sub-accumulations) with opposing signs, d and d', which are stored in two separate pipeline registers. A generic FP adder is positioned at the bottom of the systolic array for the final addition of d and d'. Moreover, an initialization mechanism at the top is also required to ensure that the first row of PEs can receive zeros with distinct signs.[2]

The new PE design to support the dual-flow architecture is depicted in Fig. 2(c). As discussed above, it accepts two distinct partial sums (d and d') with opposing signs as inputs. It consists of a multiplier, register b, a same-signed FP adder, and importantly a swapping mechanism consisting of two multiplexers and an XOR gate. The mechanism ensures that the partial sum (d or d') supplied to the same-signed adder has the same sign as the product of the input activation and weight ($a \times b$), while the other partial sum remains unchanged. This results in updated d with the same-signed addition of either $d = a \times b + d$, or $d = a \times b + d'$, depending on the sign alignment performed using the same-signed FP adder. A bypass path consisting of the second multiplexer relays unused partial sum as updated d' to the PE below. In other words, for each PE, the relayed d' holds a sign different from that of $a \times b$ performed in the given cycle.

This dual-flow architecture, facilitated by the integration of a generic adder at the bottom of the systolic array, ensures that each PE can perform same-signed addition. without introducing any errors (except for the differences due to the non-associativity of FP additions[13]). Our design maximizes the utilization of same-signed adders (each adder is used in each cycle) while ensuring these modifications remain transparent to the rest of the system. Importantly, while it seems that the number of logic components and FP adders has increased compared to the basic systolic array, the proposed design results in smaller delay, ADP, and energy consumption, owing to the highly resource-efficient same-signed FP adders, as evidenced by results in Section V-D.

[2]FP format supports the concept of positive zero and negative zero [22].

Algorithm 1 Floating-Point Addition Operation

input: x, y, RM ▷ x and y are input FP operand with M mantissa bits, and RM determines rounding modes
output: z ▷ z is the addition result of x and y

1: **function** FLOATING-POINT ADDITION
2: $Mant_{align} \leftarrow Mant(y)$ ▷ Mantissa bits, $m+1$ bits with hidden bit
3: $Mant_{noalign} \leftarrow Mant(x)$ ▷ Mantissa bits, $m+1$ bits with hidden bit
4: $Exp(z) \leftarrow Exp(x)$
5: **if** $Exp(x) < Exp(y)$ **then**
6: $Mant_{align} \leftarrow Mant(x)$
7: $Mant_{noalign} \leftarrow Mant(y)$
8: $Exp(z) \leftarrow Exp(y)$
9: **end if**
10: $Exp_{diff} \leftarrow |Exp(x) - Exp(y)|$
11: $\{Mant_{align}, Mant_{GRS}\} \leftarrow Mant_{align} \gg Exp_{diff}$ ▷ $2M{+}3$ bits barrel shifter, $Mant_{align}$ has $M{+}1$ bits and $Mant_{GRS}$ has $M{+}2$ bits
12: $GRS \leftarrow GRS\_Unit(Mant_{GRS})$
13: $OP_m \leftarrow Sign_x \oplus Sign_y$
14: **if** OP_m **then**
15: $Mant_{sum} \leftarrow Mant_{align} - Mant_{noalign}$
16: **else**
17: $Mant_{sum} \leftarrow Mant_{align} + Mant_{noalign}$
18: **end if**
19: **if** OP_m && $is\_not\_normalized(Mant_{sum})$ **then**
20: *Shift $Mant_{sum}$ to the left until normalized*
21: *Subtract $Exp(z)$ by the number of bits shifted*
22: ▷ Above two steps require leading-zero detector and barrel shifter which are both resource-hungry.
23: **end if**
24: **if** $!OP_m$ && $is\_not\_normalized(Mant_{sum})$ **then**
25: $Mant_{sum} \leftarrow Mant_{sum} >> 1$
26: $Exp(z) \leftarrow Exp(z) + 1$
27: **end if**
28: *Round $Mant_{sum}$ based on RM and adjust $Exp(z)$ accordingly.*
29: $z \leftarrow \{Sign_z, Exp(z), Mant_{sum}\}$ ▷ The computation of the sign is omitted as it is beyond the primary focus of this algorithm.
30: **end function**

B. Same-Signed Floating-Point Adders

This section proposes the design of the same-signed FP adder, which is significantly more resource-efficient than a generic FP adder. This feature makes it the primary driver for the efficacy of the proposed SEA scheme.

The generic FP addition operation is described in Algorithm 1. The inputs to the algorithm are two signed FP operands x and y, and the rounding mode RM, while the output is the sum z. The lines 2–9 of Algorithm 1 involve determining the non-dominant FP input, characterized by a smaller magnitude. In lines 10–12, the mantissa of the non-dominant FP input is shifted to align with that of the dominant FP input. The $Mant_{\text{GRS}}$ and GRS_Unit components are used to compute the Guard, Round, and Sticky (GRS) bits, which are used by the rounding unit. Lines 13–18 of Algorithm 1 perform mantissa subtraction or addition according to the signs of x and y. After mantissa subtraction or addition, the radix point of the newer mantissa may not be normalized. Thus, lines 19–27 of Algorithm 1 involve the normalization of mantissa, followed by rounding in line 28 based on the rounding mode RM and outputting the FP result in line 29.

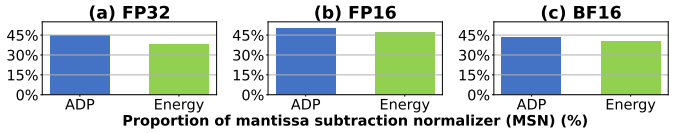

Fig. 3. ADP and energy consumption proportion of mantissa subtraction normalizers (in %) in FP32, FP16 and BF16 adders.

The normalization steps (in lines 19–27) need more attention. If the mantissa is not normalized, we perform either mantissa subtraction normalization or mantissa addition normalization, based on whether mantissa subtraction or addition was performed in lines 14–18. It is noteworthy that mantissa subtraction normalization (lines 19–23) requires a leading-zero detector and a barrel shifter, both of which are resource-hungry. Our analysis, depicted in Fig. 3, suggests that the mantissa subtraction normalizer (depicted in line 20 requiring a leading-zero detector and the barrel shifter), accounts for over 43% of the total ADP and exceeds 38% in terms of energy consumption. On the other hand, mantissa addition normalization in line 25 can be performed with a simple 2x1 multiplexer.

Noting the fact that mantissa subtraction normalization is performed only when the input operands x and y hold different signs, *we propose a same-signed adder by excluding the mantissa subtraction normalizer resulting in highly resource-efficient design.* The single-cycle logic design of the proposed same-signed FP adder based on Algorithm 1 is illustrated in Fig. 4. The Stage 1 of Fig. 4, corresponding to lines 2–10 of Algorithm 1, determines the non-dominant FP input using subtractor for exponent comparison and multiplexers for selection of non-dominant inputs. Next, in Stage 2, corresponding to lines 11–12 of the Algorithm 1, the variable right shifter (VRSH) shifts the non-dominant FP input by an amount equal to the difference of exponent, and the shifted output is paired with the GRS generation unit. Stage 3 of Fig. 4 performs the mantissa addition (line 17) unconditionally, as highlighted by the green adder, since our design focuses solely on same-signed addition and no subtraction is required. Furthermore, in Stage 4, the normalization process is simplified to a basic one-bit shifter, realized by an $(M+4)$-bit 2x1 multiplexer, depicted in green, aligning with lines 24–27 of Algorithm 1. In other words, the resource-hungry mantissa subtraction normalizer (lines 19–23) is excluded. The final Stage 5 of Fig. 4, corresponding to line 28 of Algorithm 1, performs rounding according to the supplied rounding mode RM while adjusting the exponent accordingly. The untouched sign-bit, the adjusted exponent, and the rounded mantissa form the output.

C. Integration into Gemmini

We integrate our proposed SEA-based architecture into Gemmini, and also we provide a standalone Scala FP arithmetic library, which includes a generic FP adder, same-signed adder, and rounding unit that can be configured during design time, allowing researchers to create tailored designs with self-defined datatypes, systolic arrays, and arithmetic units. The integration offers two generation options for PEs and systolic arrays: the standard Gemmini configuration or our SEA-based

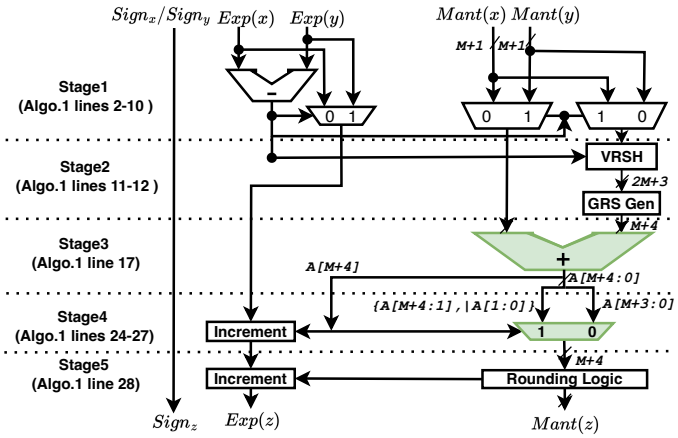

Fig. 4. Proposed single-cycle same-signed FP adder design.

version. This flexible, modular approach aligns with Gemmini's existing ecosystem and facilitates ease of use. It allows users to utilize Gemmini's standard configurations or our proposed approaches without requiring in-depth knowledge of the underlying mechanics.

IV. Experimental Setup and Evaluations

A. Datatypes for FP Adders and FP Multipliers

The tick-marks in Table I depict the combinations of FP adder (first-column) and FP multiplier (top-row) datatypes used in our experiments. The FP32 adders are paired with FP16 [7], bfloat16 (BF16) [11], and hybrid FP8 (e.g., E4M3 and E5M2) [9] multiplier datatypes, since they are supported by major vendors (AMD, NVIDIA, Google and Intel). FP16 and BF16 adders are paired with multipliers with the same or lower precision. We evaluate these different combinations to demonstrate the efficacy of our proposed designs.

B. Hardware Implementation, Verification, and Synthesis

All the adders and multipliers are implemented in Scala using the Chisel3 library and verified using chisel-tester. The SEA-based weight stationary systolic arrays with different FP adders and FP multipliers are integrated into Gemmini source code. The functional correctness of the systolic array is verified through C++ Verilator. The steps followed are: (1) compiling Scala implementation into Verilog code using FIRRTL compiler; (2) feeding the Verilog code into the Verilator to generate cycle-accurate models; and (3) loading bare-metal testbenches written in C into generated models and comparing their output with the original design to verify the correctness. Our implementation and hardware generation scripts are available at [14].

The Verilog designs are synthesized using the Cadence RTL compiler with the TSMC $45nm$ standard-cell library to obtain metrics for delay, area, and power consumption. For the power analysis, the inputs were annotated with a switching activity, applying a 25% toggle rate and a 50% probability of switching activity. A medium-level synthesis effort setting was used in the Cadence RTL compiler. The reductions in delay, ADP, and energy consumption are computed as $\frac{M_{original}-M_{proposed}}{M_{original}} \times 100\%$, where M represents the metric of interest.

V. Results and Discussion

A. Numerical Behavior of SEA-based Systolic Array

Our SEA-based systolic array simply rearranges the addition operations without introducing any approximation in the accumulation or modification of the IEEE-754 rounding mechanism. However, since the FP addition operation is not associative [13], the results of matrix multiplication performed using our proposed systolic array are expected to differ slightly from the original systolic array used in the Gemmini framework. This section investigates these slight differences.

We perform 100 matrix multiplications of $N\times N$-sized randomly generated matrices (for $N{=}8, 16, 32$) using original Gemmini and SEA-based systolic arrays. To compare the results, we compute the mean absolute percentage deviation (MAPD) as $\frac{1}{K}\sum_{i=1}^{K}|\frac{x_i-y_i}{x_i}|\times 100\%$, where X and Y are multiplication results of original and SEA-based systolic arrays, respectively, and $K{=}N^2$. To put this analysis of multiplication results into perspective, we also compare the original Gemmini results with PyTorch [23], which is a widely-used DNN framework that incorporates matrix multiplication operation.

The results are depicted in Table II. We observe that the MAPDs of the SEA-based systolic array (2nd row) are comparable to those of PyTorch (3rd row). These results demonstrate that our proposed SEA scheme introduces no errors except for the minuscule differences due to the non-associative property of FP additions. Further, such differences are inherent to different implementations of matrix multiplication, as evident by non-zero MAPDs for Pytorch *vs.* original Gemmini.

B. Same-Signed FP Adders Results

The resource efficiency improvements of our proposed same-signed FP adders compared to generic FP adders are depicted in Fig. 5 for various datatypes. The improvements are depicted in the form of reduction percentages using bar charts, with grey numbers atop bars representing the results of generic FP adders considered as baseline. From Fig. 5, we observe a significant reduction in all design metrics for all datatypes of the same-signed FP adders (FP32, BF16, and FP16). Specifically, a geometric mean reduction of 17.2% in delay, 45.9% in ADP, and 41.5% in energy consumption is observed for the three datatypes. These results demonstrate that the same-signed adder is significantly more resource-efficient (smaller, faster, and energy-efficient) than the generic FP adder for common FP datatypes, making it the primary driver for the efficacy of the proposed SEA scheme.

C. Processing Elements (PE) Results

Fig. 6 demonstrates the percentage reductions in delay, ADP, and energy achieved by our proposed SEA-based PE compared to the original PE. The notation (aADDmMULT) in the x-axis of Fig. 6 represents the datatype of FP adders and FP multipliers deployed in the PEs. The values in Fig. 6 above the bars denote synthesis results for the original PE which are considered as the baseline for computing the percentage reductions.

In Fig. 6, when taking all SEA-based PE designs across all datatype combinations into account, we observe a geometric mean reduction of 15.6%, 35.0%, and 33.7% in delay, ADP, and energy consumption, respectively. Specifically, the aFP32mE4M3 and aFP16mE4M3 combinations (based on hybrid FP8 format, which has recently appeared in training setups [9] and H100 GPU [10]) exhibit the most significant improvements across all three metrics, with reductions of at least 18.4% in delay, 43.4% in ADP, and 41.6% in energy consumption. The aFP32mBF16 combination (which is one of the most commonly used configurations for training [11]) exhibits the third largest improvement, with reductions of 15.3% in delay, 37.3% in ADP, and 38.1% in energy consumption. This is because the proportion of resources used by FP adders in these combinations is much larger compared to other datatype combinations.

D. Systolic Array Results

Fig. 7 demonstrates the percentage reductions in delay, ADP, and energy achieved by our proposed SEA-based systolic array compared to the original systolic array. For this experiment, we evaluated systolic arrays of size 6×6 (i.e., containing six rows and six columns of PEs). The notation (aADDmMULT) in the x-axis of Fig. 7 represents the datatype of FP adders and FP multipliers deployed in the systolic array. The grey numbers above the bars denote synthesis results for the original systolic array which are considered as baseline for computing the percentage reductions.

We can see a similar percentage reduction trend across all combinations as we did for the PE. Overall, our proposed SEA-based systolic array achieves a reduction between 12.9%–19.8% in delay, 15.4%–29.1% in ADP, and 15.1%–30.7% in energy consumption, for the various datatype combinations. Similarly, the aFP32mE4M3 and aFP16mE4M3 combinations demonstrate the most significant enhancements

TABLE I: Datatypes for FP adders and FP multipliers.

FP adders *(1, exponent, mantissa)*	FP Multipliers *(1, exponent, mantissa)*			
	FP16 $(1,5,10)$	BF16 $(1,8,7)$	E4M3 $(1,4,3)$	E5M2 $(1,5,2)$
FP32 $(1,8,23)$	✓	✓	✓	✓
FP16 $(1,5,10)$	✓	-	✓	✓
BF16 $(1,8,7)$	-	✓	✓	✓

TABLE II: Comparison of matrix multiplication results from original Gemmini, SEA-based systolic array and PyTorch.

	Matrix Size ($N\times N$)	8×8	16×16	32×32
MAPD (%)	SEA *vs.* orig. Gemmini	1.7×10^{-7}	4.1×10^{-7}	1.3×10^{-6}
	PyTorch [23] *vs.* orig. Gemmini	3.1×10^{-7}	1.6×10^{-6}	6.7×10^{-6}

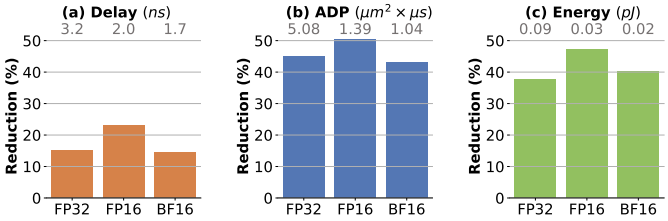

Fig. 5. Improvements in same-signed FP adders compared with generic FP adders. The grey numbers atop the bars represent results for generic FP adders (baseline), in the corresponding units.

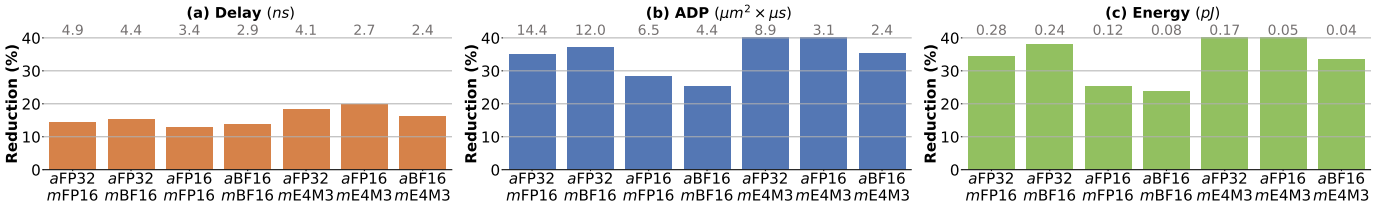

Fig. 6. Improvements from proposed SEA-based PEs compared with original PEs. The grey numbers atop the bars represent results for the original PE (baseline), in the corresponding units.

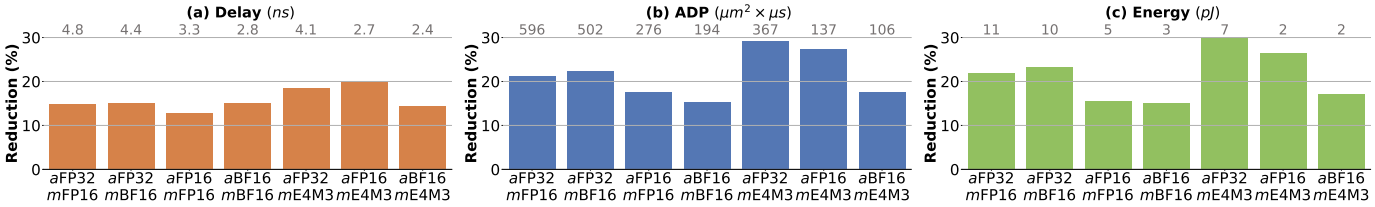

Fig. 7. Improvements from proposed SEA-based systolic arrays compared with original systolic arrays. The grey numbers atop the bars represent results for the original systolic array (baseline), in the corresponding units.

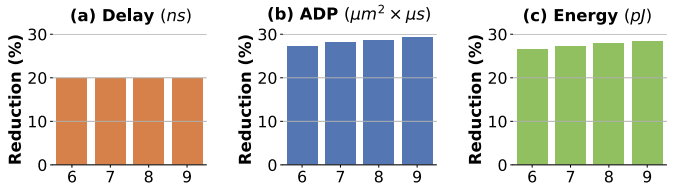

Fig. 8. Scalability of $N \times N$ SEA-based systolic arrays, where $N = 6, 7, 8, 9$, for *a*FP16*m*E4M3 datatype compared with the baseline.

across all three metrics. Note that the improvements here are lower than those reported for PEs (Section V-C) since the proposed systolic array contains additional pipeline registers between each pair of PEs and generic FP adders at the bottom of the array (Fig. 2). Nonetheless, the improvements are quite significant, especially when considering no approximation is introduced.

Scalabilty of SEA-Based Systolic Array: In addition to the 6×6 systolic array configuration, we also conducted experiments with 7×7, 8×8, and 9×9 configurations to examine the scalability of the proposed SEA. As depicted in Fig. 8, the synthesis results revealed a consistent trend of percentage reductions across delay, ADP, and energy consumption for all sizes.

VI. Conclusions

This paper focused on improving the efficiency of FP addition operations in DNN accelerators supporting training. We proposed SEA, a novel scheme that performs separate accumulations of same-signed terms using our proposed resource-efficient same-signed FP adders, followed by a final addition using generic FP adders. Evaluations of our proposed SEA-based designs demonstrated significant improvements in delay, ADP, and energy consumption. Furthermore, our experiments with various datatypes demonstrated that the proposed scheme is orthogonal to, and can be combined with existing arithmetic-complexity reduction schemes.

References

[1] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. DAC*, 2021.

[2] N. Jouppi *et al.*, "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. ISCAS*, 2023.

[3] "NVIDIA Deep Learning Accelerator," http://nvdla.org/index.html, accessed: 2023-09-11.

[4] E. Chung *et al.*, "Serving DNNs in real time at datacenter scale with project Brainwave," *IEEE Micro*, vol. 38, pp. 8–20, 2018.

[5] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 52, no. 1, pp. 127–138, 2017.

[6] S. Angizi *et al.*, "Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?" in *ISVLSI*, 2019.

[7] O. Kuchaiev et al., "Mixed-precision training for NLP and speech recognition with OpenSeq2Seq," *arXiv*, Nov. 2018.

[8] "AMD Instinct MI200 Series Accelerators," https://www.amd.com/system/files/documents/amd-instinct-mi200-datasheet.pdf, accessed: 2023-08-15.

[9] P. Micikevicius *et al.*, "FP8 formats for deep learning," *arXiv*, 2022.

[10] "NVIDIA H100 Tensor Core GPU Datasheet," https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet, accessed: 2023-08-15.

[11] S. Wang and P. Kanwar, "Bfloat16: The secret to high performance on cloud TPUs," https://cloud.google.com/blog/products/ai-machine-learning/, Aug. 2019, accessed: 2020-05-06.

[12] "Intel® Deep Learning Boost New Deep Learning Instruction bfloat16," https://www.intel.com/content/www/us/en/developer/articles/technical/intel-deep-learning-boost-new-instruction-bfloat16.html, accessed: 2023-08-15.

[13] V. Eijkhout, *Introduction to High Performance Scientific Computing*. Lulu.com, 2012.

[14] J. Gong, "Gemmini-sea," https://github.com/AaronJing/Gemmini-SEA, 2022.

[15] V. Sze *et al.*, *Efficient Processing of Deep Neural Networks*. San Rafael: Morgan & Claypool Publishers, Jun. 2020.

[16] V. Mrazek *et al.*, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. DATE*, 2017.

[17] "Int4 Precision for AI Inference," https://developer.nvidia.com/blog/int4-for-ai-inference/, accessed: 2023-09-11.

[18] K. Inayat and J. Chung, "Hybrid accumulator factored systolic array for machine learning acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 7, pp. 881–892, 2022.

[19] J. Gong *et al.*, "Approxtrain: Fast simulation of approximate multipliers for dnn training and inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2023.

[20] T. Cheng *et al.*, "Minimizing power for neural network training with logarithm-approximate FP multiplier," in *Proc. PATMOS*, 2019.

[21] "Gemmini," https://github.com/ucb-bar/gemmini, 2023, accessed: 2020-05-06.

[22] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[23] "PyTorch," https://pytorch.org/docs/stable/generated/torch.matmul.html, accessed: 2023-09-11.