Special Session: When Dataflows Converge: Reconfigurable and Approximate Computing for Emerging Neural Networks

Di Wu and Joshua San Miguel
Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, Wisconsin, USA
di.wu@ece.wisc.edu, jsanmiguel@wisc.edu

Abstract—Deep Neural Networks (DNNs) have gained significant attention in both academia and industry due to the superior application-level accuracy. As DNNs rely on compute-or memory-intensive general matrix multiply (GEMM) operations, approximate computing has been widely explored across the computing stack to mitigate the hardware overheads. However, better-performing DNNs are emerging with growing complexity in their use of nonlinear operations, which incurs even more hardware cost. In this work, we address this challenge by proposing a reconfigurable systolic array to execute both GEMM and nonlinear operations via approximation with distinguished dataflows. Experiments demonstrate that such converging of dataflows significantly saves the hardware cost of emerging DNN inference.

I. INTRODUCTION

Research into deep neural networks (DNNs) has been prospering in both academia and industry since the championship of convolutional neural networks (CNNs) in image recognition [5], with broad applications in computer vision, natural language processing, speech recognition, etc. DNN models are intensive in general matrix multiply (GEMM) operations [2], e.g., more than 90% of total operations are GEMM. To mitigate the overhead in executing GEMMs, significant efforts have been paid to approximate computing, including approximate multipliers, either fixed units at design-time or reconfigurable units at run-time, and the high level synthesis of those approximate multipliers [13]. Those techniques reduce both the energy and power consumption for GEMMs at the cost of insignificant accuracy loss.

However, emerging DNNs with more sophisticated nonlinear operations are benefiting less than classical DNNs, like CNNs, from those approximation techniques, as more sophisticated nonlinear operations than ReLU or max pooling [7] can mitigate or even diminish the energy and power improvements. Those nonlinear operations, e.g., div, exp, log, tanh, sigmoid, softmax, etc., are of high complexity, high diversity and high cost [12]. Naively implementing those nonlinear operations alongside the approximate multipliers can increase the area and power to nearly 6.0× and 4.2×, respectively [12]. To address such inefficiencies, in [12] we initially propose to leverage existing multiply–accumulate (MAC) units to virtualize those nonlinear operations in a unified manner, so that the hardware overhead to execute those nonlinear operations can be minimized. We extend a standard MAC unit with extra logic, including least zero detection, shifters, lookup tables (LUTs), adders and multiplexers (MUX), to extensively support diverse nonlinear operations. Despite improved compatibility, those add-ons slabbased on a single processing element (PE) still introduce high overheads, costing more area and power than an individual MAC unit.

In this paper, to further mitigate the above area and power overheads, we propose to virtualize the nonlinear operations on eagerly optimized dataflow architectures. More specifically, we equip systolic arrays with the additional capability to approximate nonlinear operations. The proposed design is highlighted with three features compared with [12]. First, the dataflows for GEMM and approximate nonlinear operations converge inside one single architecture. A weight stationary dataflow is applied in Google TPU [3], a commercialized systolic array for DNNs. We also observe that another weight stationary dataflow can be applied to the Taylor approximations with Horner’s rule [12]. As such, we merge the two weight stationary dataflows inside one architecture. Second, our design disperses the extra logic in multiple PEs to increase the architectural-level area and power savings. Though the dataflows for GEMM and Taylor approximations are both weight stationary, their PE designs differ. In [12], each PE is extended with all required add-ons and able to perform a complete operation. In the dataflow architecture, each PE only performs a certain part of an operation. Our systolic array removes unnecessary parts in each PE to save more compared to arrays with all holistic PEs. Third, our design involves an architectural-level approximation, orthogonal to existing microarchitecture-level approximation techniques. Our design targets the architecture-level approximation of varying operations, i.e., we define the interaction among different PEs, beyond the microarchitecture-level approximations as in [1, 4, 9, 12, 13]. Such a different viewpoint allows the coexistence of both the architecture- and microarchitecture-level optimizations for further savings.

We list the contributions in this work as follows.

• We identify the deficiency of existing works in approximating nonlinear operations as the dedication to a single PE, raising the cost of the entire PE array.
• We propose a cross-level approximated systolic array to incorporate both GEMM and nonlinear operations and amortize the area and power overheads across multiple
PEs for hardware savings. Further, we evaluate the accuracy and hardware efficiency of our design on emerging DNNs with complex nonlinear operations, demonstrating its superior area, power and energy benefits over existing designs.

The rest content is organized as follows. Section II reviews the weight stationary systolic array [3] and unified nonlinear operations [12]. Then Section III articulates our proposal, which is later evaluated in Section IV. The final Section V concludes the paper.

II. BACKGROUND

A. Weight Stationary Systolic Array

The weight stationary dataflow for GEMM, i.e., \( o = x \cdot w \), is applied in systolic arrays like Google TPU [3], which contains multiple rows and columns of the PE in Figure 1a. First, the weight \( w \) is loaded into the weight register, i.e., WREG. Multiple cycles are needed to load a weight to the correct PE from top to bottom. Second, the input \( x \) is continuously streamed into the input register, i.e., IREG, and pipelined to the right PE in the next cycle. Finally, the output \( o \) is calculated by summing the partial result from the bottom PE and the product of the weight and input in the current PE. When all outputs corresponding to the current weights are streamed out, the above three steps are repeated for a complete GEMM operation. For the weight stationary dataflow, there have been a plethora of schedule algorithms, some of which are publicly available [8, 11].

B. Unified Nonlinear Operation

In [12], we propose to unify multiple nonlinear operations (UNO), e.g., \( y/x \), \( \exp(x) \) and \( \log(x) \), using Taylor approximation for minimized hardware overheads. Taylor approximation in Equation 1 is formulated as in Equation 2 and Equation 3 using Horner’s rule. In Equation 1, \( f^{(i)}(a) \) is the \( i \)-th derivative of \( f \) evaluated at input \( x = a \), and \( c_i \) denotes the coefficient of the \( i \)-th term. Then in Equation 2 and Equation 3, \( \text{mac}_n \) is the cascaded MAC result for degree-\( n \) Taylor approximation, \( \text{scale} \) and \( \text{offset} \) are used to adjust \( \text{mac}_n \) to the correct result, and \( \text{var} \) is an affine transformation of input. Equation 3 requires \( \text{var} \leq x \leq 1 \) for \( \exp \) and \( 0.5 \leq x \leq 1 \) for \( \text{div} \) and \( \text{log} \). Please refer to [12] for calculating \( \text{scale} \), \( \text{offset} \) and \( \text{var} \) in detail.

As such, every step in UNO is a MAC operation, whose PE is shown in Figure 1b. Note the logic for the coefficients \( c \), \( \text{scale} \), \( \text{offset} \) and \( \text{var} \) are not shown for simplicity. \( \text{scale} \), \( \text{offset} \) and \( \text{var} \) are input dependent, requiring extra hardware. In [12], the choice of building a SIMD architecture based on UNO leads to the duplication of those extra hardware in every PE, throttling the overall area and power savings.

In this work, by observing that the coefficients \( c \) are independent from the input, and can be statically stored, leading to a weight stationary dataflow, we propose to merge the two PEs in Figure 1 to a reconfigurable one, minimizing the overheads to support multiple dataflows at the architecture level.

III. THE PROPOSED DESIGN

A. Processing Element

The proposed PEs towards the convergence of GEMM and UNO dataflows are presented in Figure 2. The red components are identical to those in Figure 1a, serving the original GEMM dataflow. Unlike the PE in Figure 1a, our PEs are heterogeneous with different inputs and outputs. Then the blue components are added to route the data path to act as in Figure 1b, serving the UNO dataflow. But unlike the UNO PE in Figure 1b, the proposed PEs do not calculate the complete output in an individual PE, but rather circulate an intermediate result to the right PE, i.e., the approximation is now performed spatially instead of temporally. The double stroke means both \( x \) in IREG and \( \text{var} \) in VREG are pipelined to the right PE. The VREG is present together with IREG to minimize the overall area and power; otherwise, the more expensive \( \text{var} \) logic needs to exist in all PEs to generate \( \text{var} \), \( \text{scale} \) or \( \text{offset} \). For example, \( \text{var} \) is needed when \( 1 \leq i \leq n \) in Equation 3. For area and power saving, its calculation is done only in the leftmost PE, with its value pipelined to right PEs. Then \( \text{scale} \) and \( \text{offset} \) are accessed at the last step, so their logic can exist merely in the rightmost PE (or a few rightmost PEs with proper pipelining to ensure a reasonable critical path overhead, not shown in the figure). Note that \( y \) in the rightmost PE is only used in \( y/x \), not in \( \exp(x) \) and \( \log(x) \).
Fig. 2: The proposed PEs for hybrid dataflows. Corresponding PEs are located at the leftmost, middle and rightmost columns. The red and blue colors are for GEMM and UNO dataflows, respectively.

Fig. 3: The proposed systolic array for hybrid dataflows. The red and blue colors are for GEMM and UNO dataflows, respectively.

B. Systolic Array

With above heterogeneous PEs, we further present the entire systolic array as in Figure 3, which contains one column of leftmost, middle and rightmost PEs. We mark the essential inputs and outputs surrounding the systolic array, and emphasize the GEMM and UNO dataflows, i.e., how the input x flows in the systolic array to produce the output o and f. In both dataflows, the input x are pipelined from left to right PEs. The GEMM dataflow in red aggregates the partial results vertically, from bottom to top, in each column. The input x, weight w and output o are all stored in the on-chip SRAM, which needs to interact with the off-chip DRAM [3]. On the other hand, the UNO dataflow in blue collects the partial results horizontally, from left to right, in each row. The input x and output o also need to travel inside the memory hierarchy, including both the SRAM and DRAM. However, the coefficients c are stored with small LUTs, as they are predetermined and remain identical in each column. For example, to support \( \text{div}, \text{exp} \) and \( \text{log} \) simultaneously, 2 and 1 3-entry LUTs are needed for \( c \) in the leftmost and middle columns, respectively. Given the size of the systolic array as \( R \)-row-by-\( C \)-column, one set of \( \text{var, scale and offset} \) logic is shared by one row of \( C \) PEs, instead of per PE in [12], effectively reducing the area and power overheads. In terms of dataflow schedule, we only need to address the newly introduced UNO dataflow. Its schedule is even simpler than the GEMM dataflow, as it is purely elementwise, requiring minimal effort. Such an elementwise behavior nearly reduces the memory access conflicts to zero.

C. Cross-Level Approximation

The above systolic array approximates nonlinear operations at the architecture level. Applying more eager microarchitecture-level approximations [1, 4, 9] leads to a cross-level approximation to further reduce the area and power. We look at a simple \( \text{heuristic} \) approximation, instead of the accuracy-guaranteed approximation [6, 9]. Observing that UNO dataflow proceeds from high to low order, the lower-order Taylor terms are multiplied fewer times and can be more eagerly approximated. Thus, we can apply less bits to the PEs on the right, e.g., each PE has one less fraction bit than its left PE. For example, a \( N \)-bit design of size \( R \)-by-\( C \) has \( N \)-bit inputs for MAC in the leftmost PE, and \( (N-C+1) \)-bit inputs for MAC in the rightmost PE.

IV. Evaluation

A. Experimental Setup

This work aims to improve the efficiency of emerging DNNs with 1) a PE-enhanced systolic array and 2) a cross-level approximation, therefore both accuracy and performance are evaluated. The accuracy is simulated with customized PyTorch operators [10], covering both the operation- and application-level results. Note that no retraining is involved to minimize accuracy loss. The evaluated DNN is the CapsNet model in [12], containing \( \text{div, exp, log, sigmoid and softmax} \). On the other hand, the performance evaluation, in terms of area, power and energy efficiency, is done with 32nm TSMC technology under 400MHz. We set the array size of our design and the SIMD UNO array (of size \( R \cdot C \)) to \( R = C = 4 \) and \( R = C = 8 \) and bit width to \( N = 16 \).
B. Accuracy

1) Operation-Level

The operation-level absolute root mean square error of our design is presented in Table I (Other values can be obtained using the scale logic without accuracy drop). We observe that our design exhibits slightly higher error compared to UNO with an identical cycle $C$. Moreover, with a longer cycle count $C$, our design incurs higher errors due to less-bit MACs at the end, in contrast to UNO’s increasing accuracy.

<table>
<thead>
<tr>
<th>Operation (Input range)</th>
<th>Ours (R, C, N)</th>
<th>UNO (R, C, N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{div } (y = 1, 0.5 \leq x \leq 1)$</td>
<td>0.022</td>
<td>0.039</td>
</tr>
<tr>
<td>$\text{exp } (0 \leq x \leq 1)$</td>
<td>0.080</td>
<td>0.032</td>
</tr>
<tr>
<td>$\text{log } (0.5 \leq x \leq 1)$</td>
<td>0.005</td>
<td>0.026</td>
</tr>
</tbody>
</table>

2) Application-Level

The DNN accuracy is shown in Table II. The floating-point model has an accuracy of 98.78%. Our design has similar accuracy loss compared to $C$-cycle UNO, even with the cross-level approximation in most cases. For $R = C = 8$ and $N = 16$, our design has an obvious accuracy drop due to the cross-layer approximation aggressively reduces the fraction bits.

C. Performance

The performance results are in Table III, with a focus on the computing kernel. The total 16 bits are allocated as 1-bit sign, 5-bit integer and 10-bit fraction.

<table>
<thead>
<tr>
<th>Sign-Integer-Fraction (bit)</th>
<th>Ours (R, C, N)</th>
<th>UNO (R, C, N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,4,16</td>
<td>8,8,16</td>
<td>4,4,16</td>
</tr>
<tr>
<td>1-7-8</td>
<td>98.70</td>
<td>86.27</td>
</tr>
<tr>
<td>1-6-9</td>
<td>98.87</td>
<td>97.75</td>
</tr>
<tr>
<td>1-6-10</td>
<td>98.75</td>
<td>98.36</td>
</tr>
<tr>
<td>1-4-11</td>
<td>94.05</td>
<td>93.55</td>
</tr>
</tbody>
</table>

V. Conclusion

In this work, we identify the overheads of our initial work in executing nonlinearity-intensive emerging deep neural networks. We propose a reconfigurable systolic array to flexibly execute the dataflow for either GEMM or nonlinear operations and boost the efficiency with the cross-level approximation, including both the architecture and microarchitecture levels. The area, power and energy efficiency of our design is up to 43.3%, 38.2% and 61.7% higher than our initial work. The accuracy simulation codes are available online [10].

References