FPGA Processor In Memory Architectures (PIMs): Overlay or Overhaul ?

MD Arafat Kabir*, Ehsan Kabir*, Joshua Hollis*, Eli Levy-Mackay*, Atiyehsadat Panahi†,
Jason Bakos‡, Miaoqing Huang* and David Andrews*
Department of Computer Science and Computer Engineering
*University of Arkansas,   ‡University of South Carolina,   †Cadence Design Systems
{makabir, ekabir, jrhollis, elevymac, apanahi, mqhuang, dandrews}@uark.edu, jbakos@cse.sc.edu

Abstract—The dominance of machine learning and the ending of Moore's law have renewed interests in Processor in Memory (PIM) architectures. This interest has produced several recent proposals to modify an FPGA's BRAM architecture to form a next-generation PIM reconfigurable fabric [1], [2]. PIM architectures can also be realized within today's FPGAs as overlays without the need to modify the underlying FPGA architecture. To date, there has been no study to understand the comparative advantages of the two approaches. In this paper, we present a study that explores the comparative advantages between two proposed custom architectures and a PIM overlay running on a commodity FPGA. We created PiCaSO, a Processor in/near Memory Scalable and Fast Overlay architecture as a representative PIM overlay. The results of this study show that the PiCaSO overlay achieves up to 80% of the peak throughput of the custom designs with 2.56× shorter latency and 25% – 43% better BRAM memory utilization efficiency. We then show how several key features of the PiCaSO overlay can be integrated into the custom PIM designs to further improve their throughput by 18%, latency by 19.5%, and memory efficiency by 6.2%.

Index Terms—Processing-in-Memory, Bit-serial, Overlay, FPGA, Machine Learning, SIMD

I. INTRODUCTION

Convolutional Neural Networks (CNNs), Multilayer Perceptrons (MLPs), and Recurrent Neural Networks (RNNs) have emerged as the dominant machine learning approaches for today's application domains. Each of the three networks have different computation to communication requirements, or operational intensities, that necessitate different types of architectural support [3].

CNNs exhibit high operational intensities where end-to-end inference latencies are dominated by arithmetic compute times. Conversely, MLPs and RNNs exhibit much lower operational intensities where the end-to-end inference latencies are dominated by bus bandwidth and memory swapping times. Processor in/near memory (PIM) architectures [4]–[7] are making a resurgence to address these types of network requirements. PIM architectures break the sequential von Neumann bottleneck by integrating bit-serial processors within memory.

PIM architectures can leverage the continued trend in machine learning arithmetic towards lower precision. Less than full precision operands can result in better utilization of limited memory, and the bit-serial processing elements (PEs) can provide better energy efficiency compared to full precision PEs. PIM systems offer a theoretical peak performance limited only by the memory bandwidth.

The trend towards PIM architectures is inspiring new reconfigurable fabrics that integrate bit-serial arithmetic units into BRAM IP to form PIM tiles [1], [2], [8]–[15]. These architectures may represent the future but are not currently available. To fill the void PIM architectures can be created as overlays in existing FPGAs. The fundamental question we explore in our work is, how close in performance can an overlay come to the performance being reported for next-generation PIM reconfigurable compute fabrics?

To explore this question we created PiCaSO, a Processor in/near Memory Scalable and Fast Overlay as an open-source PIM overlay architecture [16]. We present performance comparisons that show PiCaSO achieves 80% of the peak throughput of these emerging proposed custom designs while delivering 2.56× shorter latency and 25% – 43% better BRAM memory utilization efficiency. This validates PiCaSO's ability to bring enhanced designer productivity to the design of FPGAs without the traditional performance sacrifices of an overlay.

Finally, we apply several PiCaSO design optimizations to the custom PIM designs to further improve their throughput by 18%, latency by 19.5%, and memory efficiency by 6.2%. The specific contributions of this work are:

- A PIM overlay architecture that scales linearly with the BRAM capacity of a device, without sacrificing the clock frequency.
- A comparative study with a state-of-the-art PIM overlay showing improvements of clock speed by 2×, resource utilization by 2×, and accumulation latency by 17×.
- An improved version of an existing custom PIM design incorporating the novel features of the proposed overlay architecture.
- A comparative study between the proposed overlay and custom PIM designs analyzing the trade-offs and use cases of the overlay and custom designs.

PiCaSO is open-source and freely available at [16] for use, modification, and distribution without restriction.

(a) Processing-in-Memory architecture
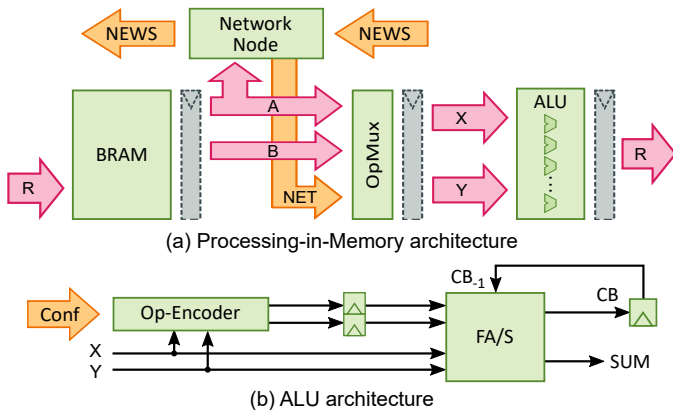
(b) ALU architecture

Fig. 1. Proposed overlay architecture for processing in/near memory, PiCaSO

Op-Code	Output (SUM)	Description
ADD	X + Y	Acts as a Full-Adder (FA)
SUB	X - Y	Acts as an FA with borrow logic
CPX	X	Copies operand X unmodified
CPY	Y	Copies operand Y unmodified

Conf	YX	ALU Op-Code	Description
000	xx	ADD	Request ADD
001	xx	CPX	Select X operand
010	xx	CPY	Select Y operand
011	xx	SUB	Request SUB
1xx	00	CPX	NOP
1xx	01	ADD	+Y
1xx	10	SUB	-Y
1xx	11	CPX	NOP

II. RELATED WORK

PIM architectures are a growing area of research [8]–[15]. Building on earlier work such as Logic-In-Memory [6], Terasys [5], Shamrock [4], and Computational RAM [7], PIM architectures seek to break the classic von Neumann bottleneck by moving the processing closer to the data residing in memory in a Single Instruction Multiple Data (SIMD) architectural organization [17]–[24].

Recently, the reconfigurable computing community has been exploring modifying the internal circuity of the on-chip BRAMs within a modern FPGA with bit-serial arithmetic and logic operations to form a PIM tile [1], [2]. Examples include RIMA (Reconfigurable In-Memory Accelerator) [2], which is built upon Neural Cache [11]. Their compute capable BRAMs (CCB) enhanced the peak MAC throughput by factors of $1.6\times$ and $2.3\times$ for 8-bit integer and block floating point precision at a cost of a 7.4% increase in BRAM tile area [2]. Reported clock frequencies range from 250 MHz to 455 MHz on a Straix 10 device. CCB requires simultaneous activation of multiple wordlines on a port and modifications to the voltage source for robustness.

CoMeFa [1] builds upon CCB taking advantage of the dual-port nature of BRAMs. Two versions of PIM blocks were proposed. Optimized for the delay, CoMeFa-D showed 25.4% tile area increase due to the inclusion of 160 PEs, 120 sense amplifiers (SA), and write drivers. Optimized for the area, CoMeFa-A showed an 8.1% increase in the BRAM tile area mainly attributed to the addition of 40 PEs. The maximum clock frequency (735 MHz) dropped $1.25\times$ to 588 MHz for CoMeFa-D. The clock frequency for CoMeFa-A dropped $2.5\times$ to 294 MHz, to perform 4 reads and 2 writes in a single cycle.

PiCaSO is very synergistic with these efforts. We show how design optimizations developed for PiCaSO can be applied to these BRAM tile designs and potentially reclaim the clock frequency difference with the BRAM's supported maximum.

III. PICASO ARCHITECTURE

Fig. 1 shows the processor in-memory architecture of Pi-CaSO. PiCaSO builds on the SPAR-2 PIM processor array reported in [25]–[27] but with the key modifications discussed below. Custom bit-serial PIM designs including those reported in [1], [2], [26] stream operands between memory and ALUs across dedicated bitlines. Such an organization does not provide support for fast reduction operations (summation of product terms) between the PEs and instead requires explicit buffered transfer or copying of the product terms (for multiply-accumulate) between BRAM columns. PiCaSO enables zero-copy reduction operations with the operand-multiplexer (Op-Mux) shown in Fig. 1. The operand-multiplexer allows pass-through of bitlines from BRAMs to ALUs for multiplication but then supports zero-copy reduction summation of the product terms. The Network Node in Fig. 1 provides a streaming interface between PIM blocks enabling the streaming of partial products into the ALU of the destination PE for summation, without intermediate copying. Section III-C presents how the reduction operation can be optimized by inserting pipeline stages that overlap data transfers with ALU operations, hiding the transfer latency.

A. Parallel to Serial Corner Turning

PiCaSO is a bit-serial array processor designed to work with standard processors. Parallel data read/written from DRAM and external I/O devices is corner-turned into bit-serial data and stored as a striped column within the BRAMs. This is a standard storage scheme for bit serial ALUs similar to [1], [2], [26]. PiCaSO configures a BRAM to be 16-bits wide to concurrently feed bit-serial data to 16 ALUs [26]. In SPAR-2 [26], the benchmark overlay, the 16 PEs form a logical 4×4 PE Block. PiCaSO structurally organizes the PE-Block as a 1×16 linear array to optimize layout in the columnar architecture of Virtex FPGAs. This reduces routing complexity and wire delay, allowing a greater number of PEs to be synthesized into the FPGA and improving system clock speed.

B. Bit-Serial ALUs

Fig. 1(b) shows the architecture of the bit-serial ALU consisting of a Full-ADD/SUB module (FA/S) and an op-code encoder. The FA/S implements the four operations in Table I. CPX and CPY support min/max pooling and other filter operations that require the selection of one of the two input operands. Op-Encoder provides an abstract interface for
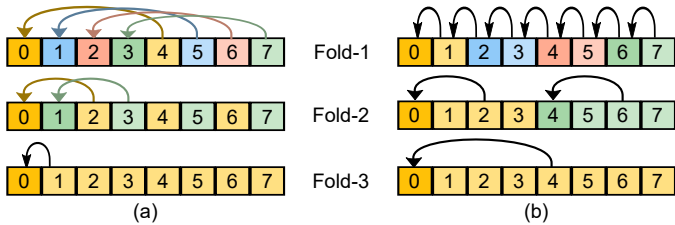
Fig. 2. Folding patterns in Operand Multiplexer (OpMux)

(a) Network Architecture   (b) Jump over PE-Blocks

(c) Network node (N) architecture for hopping

Fig. 3. Data network for fast accumulation and reduction operations

TABLE III
CONFIGURATIONS OF OPERAND MULTIPLEXER

Config Code	X	Y	Description
A-OP-B	A	B	Used in standard operations
A-FOLD-1	A	$\{0, A[H2]\}$	A[H2]: second half of A
A-FOLD-2	A	$\{0, A[Q2]\}$	A[Q2]: second quarter of A
A-FOLD-3	A	$\{0, A[HQ2]\}$	A[HQ2]: second half-quarter of A
A-FOLD-4	A	$\{0, A[HHQ2]\}$	A[HHQ2]: second half of A[HQ1][1]
A-OP-NET	A	NET	Operates on network stream
0-OP-B	0	B	Used in the first iteration of MULT

[1] A[HQ1] : first half-quarter of A

the FA/S module. Table II shows the encoding for Booth's Radix-2 multiplication algorithm.

C. Supporting Reduction Operations

The operand-multiplexer (OpMux) provides a data path for reduction operations between the PEs in a PE-Block without having to copy the operands between bitlines. This is achieved using a folding technique. Fig. 2 shows two types of folding patterns for a PE row with 8 columns enabled by the OpMux module. In pattern (a), after adding an operand with its fold-1 pattern, PE 0, 1, 2, and 3 contain the summation of 0 & 4, 1 & 5, 2 & 6, and 3 & 7 respectively. In pattern (b), after adding an operand with its fold-1 pattern, PE 0, 2, 4, and 6 contain the summation of 0 & 1, 2 & 3, 4 & 5, and 6 & 7. In both cases, after applying fold-1, fold-2, and fold-3 in that order, the accumulation result will be stored in PE-0. Fold-1 of the pattern (b) can be especially useful in CNN models, where each PE needs access to its adjacent PEs. A similar type of folding scheme can be realized using multiplexers at the output of SAs in custom PIM blocks. Results presented in Section V show the potential reduction in accumulation latency for the custom designs provided with this optimization.

Table III shows configurations currently supported by the OpMux module. Configuration A-OP-B connects ports A to X and B to Y and is used in element-wise operations. A-FOLD-x implements folding patterns similar to Fig. 2(a). A-OP-NET directly feeds the network stream into the ALU. 0-OP-B is used as the initialization step in Booth's multiplication.

D. Network Architecture

Fig. 3(c) expands the Network Node shown in Fig. 1. Fig. 3(a) shows the PE-Blocks (PB) connected to the data network through the network module (N). Fig. 3(b) illustrates data reduction patterns between 8 nodes. Each node can be configured as a transmitter (T), receiver (R), or pass-through (P) based on a level (L) parameter and its position in the array. Fig. 3(b), shows that level 0 logically connects even nodes as receivers with their right neighbor as transmitters
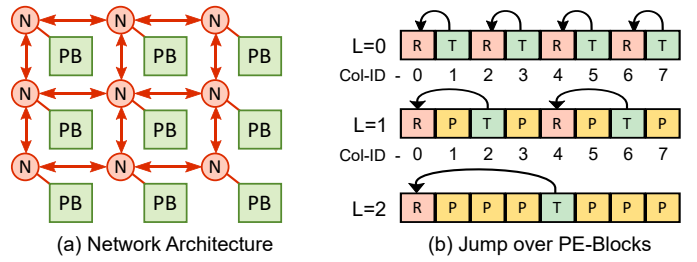
between columns. For level 1, the middle node of every 3 consecutive nodes acts as a pass-through, effectively connecting its neighbors. Similarly, level 2 connects node-4 to node-0. During accumulation, bits of the operand in the transmitter hop through P-nodes to reach the receiver ALU where they are added (serially) to the operand in the receiver. After levels 0, 1, and 2, PE 0 contains the accumulation result of an entire row in the array.

E. Pipelining Options for PIM-Blocks

The dashed registers in Fig. 1(a) show three potential points for pipelining the PIM Block: register file output, OpMux output, and ALU output. The *Single-Cycle* configuration has no pipeline stages and is equivalent to the custom BRAM designs [1], [2] and the benchmark overlay [26]. PiCaSO can be configured in different pipeline configurations based on network requirements and choice of FPGA. *RF-Pipe* inserts a pipeline stage at the register file output to hide the read latencies of the BRAM. *Op-Pipe* inserts a pipeline stage at the OpMux output to hide long wire delays through the network. *Full-Pipe*, referred to as PiCaSO-F, enables all three pipeline stages as shown in Fig. 1 (a).

IV. ANALYSIS

A. Performance and Utilization

Table IV compares the pipeline configurations outlined in subsection III-E against SPAR-2, the benchmark overlay from [26]. All designs were implemented and run on Virtex-7 (xc7vx485) and Alveo U55 FPGAs. Utilization numbers follow the tile definition in SPAR-2 consisting of 256 PEs organized in a 4×4 array of PE blocks, with 16 PEs in each block. The total utilization per tile and the average utilization per block are shown. The Full-Pipe configuration achieved a 2.25× and a 1.67× increase in clock frequency compared to the benchmark design on Virtex-7 and U55 devices, respectively. In both devices, Full-Pipe provided a 2× improvement in resource utilization over SPAR-2.

TABLE IV
COMPARISON BETWEEN TILES OF 4×4 PE-BLOCKS OF DIFFERENT OVERLAY CONFIGURATIONS

	Benchmark [26]				Full-Pipe				Single-Cycle				RF-Pipe				Op-Pipe			
	Virtex-7		U55		Virtex-7		U55		Virtex-7		U55		Virtex-7		U55		Virtex-7		U55	
	Tile	Block	Tile	Block	Tile	Block	Tile	Block	Tile	Block	Tile	Block	Tile	Block	Tile	Block	Tile	Block	Tile	Block
LUT	3023	189	2449	153	835	52	774	48	895	56	1068	67	1017	64	1064	67	836	52	774	48
FF	1024	64	768	48	1799	112	1799	112	1031	64	1031	64	1543	96	1527	95	1543	96	1543	96
Slice	1056	66	556	35	522	33	243	15	395	25	223	14	451	28	243	15	472	30	295	18
Max-Freq	240 MHz		445 MHz		540 MHz		737 MHz		245 MHz		487 MHz		360 MHz		600 MHz		370 MHz		620 MHz	

TABLE V
CYCLE LATENCY OF DIFFERENT OPERATIONS

Operation	Benchmark [26]	PiCaSO-F
ADD/SUB	$2N$	$2N$
MULT[1]	$2N^2 + 2N$	$2N^2 + 2N$
Accumulation[2]	$(q - 1 + 2\log_2 q)N$	$15 + \frac{q}{16} + 4N + (N+4)J$
$q = 128$, $N = 32$	4512	259

[1] Booth's Radix-2 multiplication
[2] q : Number of columns to be accumulated
N : Operand width
J : Number of network jumps needed $= \log_2(q/16)$

TABLE VI
COMPARISON OF LARGEST OVERLAY ARRAYS IN VIRTEX DEVICES

	Virtex-7		Alveo U55	
	Benchmark [26]	PiCaSO-F	Benchmark [26]	PiCaSO-F
Max-Size	24K	33K	63K	64K
LUT	74.6%	32.5%	41.6%	14.8%
FF	16.0%	38.0%	9.7%	17.3%
BRAM	73.8%	99.9%	98.4%	100.0%
Uniq. Ctrl. Set	32.1%	2.1%	19.5%	0.8%
Slice	86.0%	76.4%	63.4%	32.0%

The Single-Cycle configuration achieved similar performance on the Virtex-7 and better performance on the U55 compared to the benchmark system, with $2.6\times$ and $2.5\times$ utilization improvements, respectively. It had a smaller flip-flop count and slice utilization compared to the Full-Pipe due to the absence of the pipeline registers. Both RF-Pipe and Op-Pipe achieved better clock speeds but with an increase in slice utilization compared to Single-Cycle, due to the addition of the pipeline stages. As argued in Subsection III-E, Op-Pipe had better performance compared with RF-Pipe by minimizing the clock latency contributed by the network. All configurations offered at least $2\times$ better utilization and up to $2\times$ better performance compared to the benchmark design.

Table IV shows Full-Pipe achieved clock frequencies of 540 MHz on the Virtex-7 (xc7vx485-2), and 737 MHz on the Alveo U55 (xcu55c, -2 speed grade). The data sheets for these devices list 543.77 MHz and 737 MHz, respectively as the maximum BRAM clock frequencies. Surprisingly, this is an improvement over the custom designs reported in [1], [2]. The technology node of U55 (16 nm) is comparable to that of the designs proposed in CCB (Stratix 10, 14 nm) and CoMeFa (Arria 10, 20 nm). Yet, PiCaSO runs $1.62\times$ and $1.25\times$ faster than the fastest configurations of CCB and CoMeFa, respectively. This is due to the pipelined architecture of PiCaSO, where the slowest stage is the BRAM. Thus, it can run as fast as the maximum frequency of the BRAM.

B. Reduction Network

Both PiCaSO and SPAR-2 [26] use Booth's Radix-2 algorithm for multiplication. Thus, the cycle latencies for the ADD/SUB and MULT operations in Table V are identical. SPAR-2 uses a standard NEWS network to copy operands between PEs when summing the partial products during multiply-accumulate (MAC) operations. The Accumulation row compares the number of clock cycles for SPAR-2's NEWS network and PiCaSO's reduction network. The last row in Table V shows the PiCaSO-F reduction network provides a $17\times$ improvement in accumulation latency for the test configuration reported in [25]. This improvement is due to

the careful design of the binary-hopping network discussed in Section III-D, which overlaps data transfer with computation during accumulation.

C. Scalability

A primary design goal for PiCaSO was to make it scale linearly with the BRAM capacity of any FPGA. To evaluate scalability, the largest-sized array of PIM blocks that could fit into the target devices was constructed. The results of this study are shown in Table VI.

In the Virtex-7 FPGA, the largest array of SPAR-2 [26] PIM blocks contained 24K PEs. This did not achieve the full capacity of the Slices or BRAM resources available in that device. The implementation tool failed at the placement step for larger arrays due to a high utilization (32.1%) of Unique Control Sets. Control sets are the collection of control signals for slice flip-flops. Flip-flops must belong to the same control set to be packed into the same slice. A large number of unique control sets makes it difficult to find a valid placement, even with enough available slices. In contrast, PiCaSO-F fully utilized the BRAM resources to fit 33K PEs, a 37.5% improvement over SPAR-2 in the same device. PiCaSO does not suffer from the placement issues observed in SPAR-2 due to a very low (2.1%) utilization of the unique control sets.

In the U55 FPGA, SPAR-2 almost achieved the full BRAM capacity for an array size of 63K PEs. This is due to the U55 FPGA offering significantly more slices and routing resources compared to the Virtex-7 FPGA. PiCaSO achieved 100% utilization of BRAM with $2\times$ better slice utilization over SPAR-2.

Our results showed that the scalability of the benchmark design, SPAR-2, is dependent on the Slice-to-BRAM ratio and cannot guarantee the creation of a PIM array that scales with the BRAM capacity. Conversely, our results showed PiCaSO scaling with the BRAM capacity independent of the Slice-to-BRAM ratio across multiple devices of Virtex-7 and Ultrascale+ FPGA families. Table VII lists representative devices we evaluated based on the following two criteria: BRAM capacity and LUT-to-BRAM ratio. Each device is assigned an ID as a short name to be used in this paper.

TABLE VII
REPRESENTATIVE OF VIRTEX-7 AND ULTRASCALE+ DEVICES

Device	Tech	BRAM#	Ratio[1]	Max PE#[2]	ID
xc7vx330tffg-2	V7	750	272	24K	V7-a
xc7vx485tffg-2	V7	1030	295	32K	V7-b
xc7v2000tfhg-2	V7	1292	946	41K	V7-c
xc7vx1140tflg-2	V7	1880	379	60K	V7-d
xcvu3p-ffvc-3	US+	720	547	23K	US-a
xcvu23p-vsva-3	US+	2112	488	67K	US-b
xcvu19p-fsvb-2	US+	2160	1892	69K	US-c
xcvu29p-figd-3	US+	2688	643	86K	US-d

[1] LUT-to-BRAM ratio
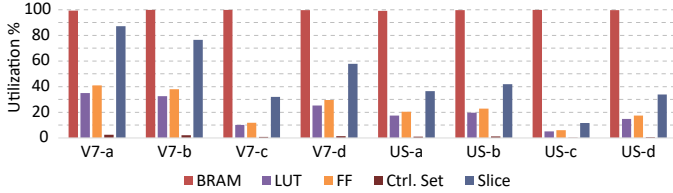[2] Maximum number of PEs if all BRAMs are utilized

Fig. 4. Scalability study on Virtex-7 and Ultrascale+ FPGA families

Fig. 4 shows that PiCaSO utilized the full BRAM capacity in all devices, and achieved the maximum number of PEs the device can fit based on BRAM density. Results showed for the smallest device (V7-a) and lowest LUT-to-BRAM ratio, the LUT and flip-flop utilization is around 40%. For one of the largest devices with a high LUT-to-BRAM ratio (US-c), these utilization numbers are negligible, around 5%. These results strongly support that PiCaSO scales linearly with the BRAM capacity of the device.

V. COMPARISON WITH CUSTOM DESIGNS

Fig. 5 shows the relative MAC latency of the custom designs w.r.t PiCaSO. The latency is computed for 16 parallel MULTs followed by the accumulation of the products. The clock speeds of custom designs are adjusted based on the performance degradations reported in [1], [2]. With the exception of CoMeFa-D at 16-bit precision, PiCaSO has the shortest latency due to faster clock speed and accumulation. CCB and CoMeFa extend the clock period to allow a complete read-modify-write per clock cycle. This allows a complete MULT to finish in half the number of cycles compared to PiCaSO and can reduce latencies at higher precisions. Still, PiCaSO runs $1.72\times - 2.56\times$ faster than CoMeFa-A, which is reported as the most practical design in [1].

Peak TeraMAC/sec throughputs on the U55 FPGA are shown in Fig. 6. CCB and CoMeFa design the BRAM IP to support one PE per bitline. With a column muxing factor of 4 [1], a Virtex 36Kb BRAM would be redesigned as a 256×144 array with 144 PEs per BRAM. The use of standard BRAM IP prevents PiCaSO (and all overlays) from making this modification. Yet PiCaSO still achieves 75% – 80% of CoMeFa-A's peak throughput, the most practical of the two CoMeFa designs. This results from PiCaSO not sacrificing the same degradation of clock speed seen in all custom designs.

The memory utilization efficiency of BRAMs is not discussed in [1], [2] but we feel is an important metric for PIM
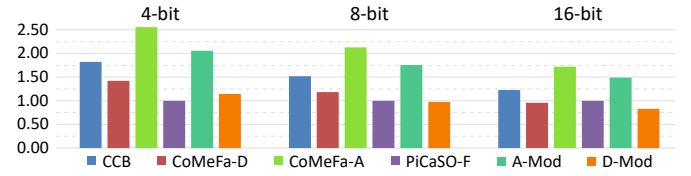

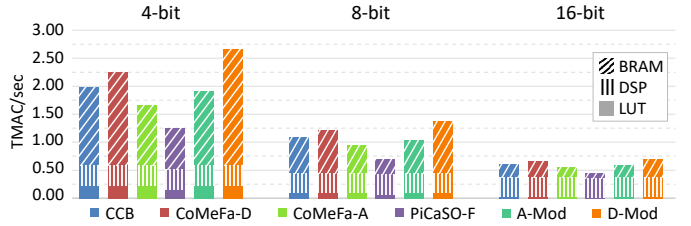Fig. 5. Relative MAC latency of custom designs w.r.t PiCaSO

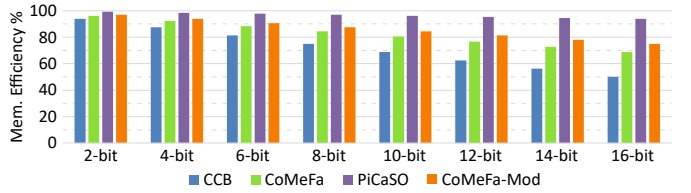Fig. 6. Peak MAC throughput of PiCaSO and custom designs on Alveo U55

Fig. 7. BRAM memory utilization efficiency on Virtex devices

architectures. Memory utilization efficiency can be defined as the fraction of BRAM memory that can be used to store model weights. Both CCB and CoMeFa follow the computation techniques used in [11] which requires scratchpad memory. For N-bit operands, CCB requires 8N reserved wordlines. CoMeFa only needs 5N wordlines using the "One Operand Outside RAM (OOOR)" technique. PiCaSO requires only 4N wordlines, as it does not require copying operands to the same bitline as in CoMeFa. In the widest mode of a Virtex 36Kb BRAM, each PE of CCB and CoMeFa would have 256 bits of storage in its register file (bitline). For PiCaSO, each register file has 1024 bits. Fig. 7 shows the memory utilization efficiency of these architectures. As observed, at higher precisions the memory efficiency drops significantly for CCB and CoMeFa. For 16-bit operands, CCB and CoMeFa have only 50% and 68.8% efficiencies, respectively, while PiCaSO has 93.8% efficiency.

Table VIII summarizes comparisons between PiCaSO and the custom designs. The custom designs significantly degrade the BRAMs maximum clock frequency, whereas PiCaSO runs at the maximum clock speed of the BRAM. However, PiCaSO has 1/4th the number of parallel MACs, as it cannot access all the bitlines. Multiplication in PiCaSO is $2\times$ slower, as it requires 2 cycles to process a single bit. However, accumulation is $2\times$ faster in PiCaSO. PiCaSO supports Booth's radix-2 multiplication. In Booth's algorithm, half of the intermediate steps are NOPs on average. Thus, PiCaSO can potentially further reduce the multiplication latency by 50% on average. CoMeFa can use Booth's algorithm only in OOOR mode and CCB does not support it at all.

As discussed earlier, the memory utilization efficiency of CCB is significantly low, PiCaSO is high, and CoMeFa lies

TABLE VIII
COMPARISON WITH CUSTOMIZED BRAM PIM ARCHITECTURES

	CCB	CoMeFa-D	CoMeFa-A	PiCaSO-F	A-Mod
Architecture	Custom	Custom	Custom	Overlay	Custom
Clock Overhead	60%	25%	150%	0%	150%
Parallel MACs	144	144	144	36	144
Mult Latency[1]	(a)	(a)	(a)	(b)	(a)
N = 8	86	86	86	144	86
Accum. Latency[2]	(c)	(c)	(c)	(d)	(e)
q = 16, N = 8	80	80	80	48	40
Support Booth's	No	Partial	Partial	Yes	Yes
Mem. Efficiency	Low	Medium	Medium	High	Medium
Complexity	High	Medium	Medium	No	Medium
Practicality	Low	Medium	High	Very High	High

[1] (a) $N^2 + 3N - 2$; (b) $2N^2 + 2N$
[2] (c) $(2N + \log_2 q)\log_2 q$; (d) $(N + 4)\log_2 q$; (e) $(N + 2)\log_2 q$

in between. CCB has the highest design complexity mainly due to its need for a modified voltage supply. CoMeFa has medium complexity since it requires modifications to the SAs, additional flip-flops, and SA cycling. Being an overlay, PiCaSO does not have such design complexities. As reported in [1], the practicality of CCB is low, CoMeFa-D is medium, and CoMeFa-A is high. In that reference, the practicality of PiCaSO is very high. It offers 80% of CoMeFa-A's peak throughput with 2.56× shorter latency, 25% better memory efficiency, can be implemented using off-the-shelf FPGAs, and is tested on real devices, while CCB and CoMeFa numbers are mainly based on simulations.

A. Fusing PiCaSO Optimizations into Custom Designs

Fig. 8 shows how modifications highlighted in red can accelerate CoMeFa-A [1]. We refer to this implementation as *A-Mod*. PiCaSO's OpMux module per bitline consists of a 2-to-1 mux and a 4-to-1 mux. This can be implemented using a few CMOS pass transistors. OpMux then saves both the cycles and memory needed to copy operands during accumulation [2], [11]. PiCaSO's network module can overlap data movement with computation between different PIM blocks. The network module can be embedded within the PIM block or can be implemented using logic slices from the FPGA. A single-bit port connection to the network module is enough to support row-wise accumulation.

Although [1] mentions that the PE does not add additional delay to the extended clock, in a practical circuit, there will always be an additional delay. This delay can be hidden using one of the pipelining schemes of PiCaSO. A single stage of registers could be enough to hide the PE delay. As BRAM blocks already contain output registers, this should not add any additional area overhead on top of what is reported in [1]. The PE circuit can be placed between two stages of registers if the delay is too long. This is illustrated using the dashed flip-flops in Fig. 8. Similar modifications can be performed on CoMeFa-D referred to as implementation *D-Mod.*

These modifications can significantly improve the performance of the custom designs. The extrapolated performance numbers for A-Mod and D-Mod are presented in Fig. 5 and Fig. 6. As observed in Fig. 5, the adoption of PiCaSO's
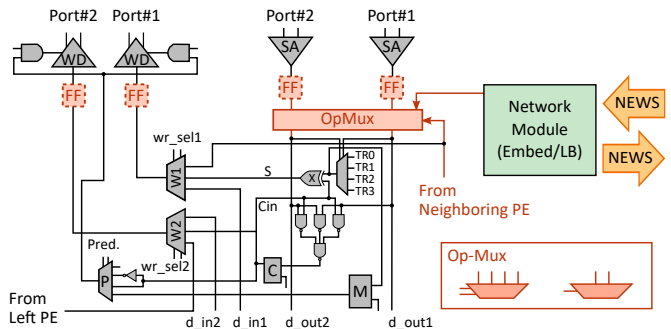

Fig. 8. Modified CoMeFa-A [1] with PiCaSO adoption (A-Mod)

OpMux and network modules can improve their MAC latency by 13.4% – 19.5% due to faster accumulation. This consequently improves their throughput by 5% - 18% over different precisions. In Fig. 7, CoMeFa-Mod represents both A-Mod and D-Mod implementations. Due to OpMux, A-Mod and D-Mod no longer requires scratchpad storage to copy operands for accumulation. This improves their memory utilization efficiency by 6.2%. This means at 4-bit precision, 1.6 million more weights can be stored in a device with 100 Mb of BRAM. This would significantly reduce weight stall cycles [3] and allow bigger models to be stored on chip. In Table VIII, the A-Mod column shows the architectural enhancements due to these modifications. A-Mod retains the high parallelism and fast Mult latency of the original CoMeFa design and offers 2× faster accumulation and full support for Booth's algorithm.

VI. CONCLUSIONS

This paper presented PiCaSO, an open-source scalable and portable Processor in Memory (PIM) overlay architecture. As an overlay, PiCaSO brings software levels of productivity to the design of FPGA machine-learning accelerators across AMD devices. The PIM architecture addresses the needs of machine learning and big data analytic applications that are memory intensive.

A scalability study was presented that established PiCaSO scaled linearly with the BRAM capacity across a range of devices with varying LUT-to-BRAM ratios. Analysis on Virtex-7 and Ultrascale+ devices showed PiCaSO runs as fast as the BRAM maximum frequency. Comparisons against SPAR-2, a state-of-the-art SIMD array processor overlay, showed improvements in slice utilization and achievable clock frequency by 2× and accumulation latency reduction by 17×.

Comparative analysis against custom designs showed PiCaSO achieves up to 80% of the peak throughput and up to 2.56× shorter latency and 25% – 43% better memory utilization.

We showed that the proposed architecture can be adopted into custom PIM designs, and can improve the throughput by 18%, latency by 19.5%, and memory utilization by 6.2%.

Our future efforts are focused on automating and applying application-specific and logic-family customizations to the generation of both PiCaSO-based accelerator and compiler-generated executables.

REFERENCES

[1] A. Arora, T. Anand, A. Borda, R. Sehgal, B. Hanindhito, J. Kulkarni, and L. K. John, "CoMeFa: Compute-in-Memory Blocks for FPGAs," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2022, pp. 1–9.

[2] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. Iyer, and R. Das, "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2021, pp. 88–96.

[3] 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*, 2017, pp. 1–12.

[4] P. M. Kogge, J. B. Brockman, T. L. Sterling, and G. R. Gao, "Processing In Memory: Chips to Petaflops," in *International Symposium on Computer Architecture*, vol. 97. Citeseer, 1997.

[5] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the Terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[6] H. S. Stone, "A Logic-in-Memory Computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, 1970.

[7] D. G. Elliott, W. M. Snelgrove, and M. Stumm, "Computational RAM: A memory-SIMD hybrid and its application to DSP," in *1992 Proceedings of the IEEE Custom Integrated Circuits Conference*. IEEE, 1992, pp. 30–6.

[8] T. Finkbeiner, G. Hush, T. Larsen, P. Lea, J. Leidel, and T. Manning, "In-Memory Intelligence," *IEEE Micro*, vol. 37, no. 4, pp. 30–38, 2017.

[9] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43–56.

[10] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H.-S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E.-B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 350–352.

[11] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural Cache: Bit-Serial in-Cache Acceleration of Deep Neural Networks," in *2018 ACM/IEEE 45Th annual international symposium on computer architecture (ISCA)*, 2018, pp. 383–396.

[12] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3S, pp. 336–348, 2015.

[13] N. S. Kim, D. Chen, J. Xiong, and W.-m. W. Hwu, "Heterogeneous computing meets near-memory acceleration and high-level synthesis in the post-moore era," *IEEE Micro*, vol. 37, no. 4, pp. 10–18, 2017.

[14] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 802–815.

[15] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs ," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 100–113.

[16] M. A. Kabir, E. Kabir, J. Hollis, E. Levy-Mackay, A. Panahi, J. Bakos, M. Huang, and D. Andrews, "PiCaSO: A Scalable and Fast PIM Overlay." [Online]. Available: https://github.com/Arafat-Kabir/PiCaSO

[17] A. Landy and G. Stitt, "Serial Arithmetic Strategies for Improving FPGA Throughput," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, pp. 1–25, jul 2017.

[18] D. J. M. Moss, D. Boland, and P. H. W. Leong, "A Two-Speed, Radix-4, Serial-Parallel Multiplier," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 27, no. 4, pp. 769–777, 2019.

[19] G. Csordás, B. Fehér, and T. Kovácsházy, "Application of bit-serial arithmetic units for FPGA implementation of convolutional neural networks," in *2018 19th International Carpathian Control Conference (ICCC)*, 2018, pp. 322–327.

[20] A. Landy and G. Stitt, "Revisiting Serial Arithmetic: A Performance and Tradeoff Analysis for Parallel Applications on Modern FPGAs," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 9–16.

[21] D. Walsh and P. Dudek, "A compact FPGA implementation of a bit-serial SIMD cellular processor array," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012, pp. 1–6.

[22] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, p. 65–74.

[23] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MI-CRO)*, 2016, pp. 1–12.

[24] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of low numeric precision deep learning inference using intel® fpgas," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 73–80.

[25] S. Basalama, A. Panahi, A.-T. Ishimwe, and D. Andrews, "SPAR-2: A SIMD Processor Array for Machine Learning in IoT Devices," in *2020 3rd International Conference on Data Intelligence and Security (ICDIS)*. IEEE, 2020, pp. 141–147.

[26] A. Panahi, S. Balsalama, A.-T. Ishimwe, J. M. Mbongue, and D. Andrews, "A Customizable Domain-Specific Memory-Centric FPGA Overlay for Machine Learning Applications," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, Aug. 2021, pp. 24–27.

[27] A. Panahi, E. Kabir, A. Downey, D. Andrews, M. Huang, and J. D. Bakos, "High-rate machine learning for forecasting time-series signals," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–9.