A Customizable Domain-Specific Memory-Centric FPGA Overlay for Machine Learning Applications

Atiyehsadat Panahi*, Suhail Balsalama*, Ange-Thierry Ishimwe*, Joel Mandebi Mbongue$, David Andrews*
Department of Computer Science and Computer Engineering
**University of Arkansas,* $*University of Florida*
Fayetteville, Arkansas
{apanahi, sebasala, ai006}@uark.edu, jmandebimbongue@ufl.edu, dandrews@uark.edu

***Abstract*—This paper presents an overview and performance analysis of a software-programmable domain-customizable System-on-Chip (SoC) overlay for low-latency inferencing of variable and low-precision Machine Learning (ML) networks targeting Internet-of-Things (IoT) edge devices. The SoC includes a 2-D processor array that can be customized at design time for FPGA logic families. The overlay resolves historic issues of poor designer productivity associated with traditional Field Programmable Gate Array (FPGA) design flows without the performance losses normally incurred by overlays. A standard Instruction Set Architecture (ISA) allows different ML networks to be quickly compiled and run on the overlay without the need to resynthesize. Performance results are presented that show the overlay achieves $1.3\times$-$8.0\times$ speedup over custom designs while still allowing rapid changes to ML algorithms on the FPGA through standard compilation.**

***Index Terms*—FPGA, overlay, processor array, machine learning, SIMD, bit-serial, fixed-point, MLP, CNN, LSTM, GRU**

I. Introduction

The explosive growth of the Internet-of-Things (IoT) is changing how we must store and analyze data [1]. Processing is moving out to where the data is produced, domain-specific hardware accelerators are becoming ubiquitous infrastructure, and latency is replacing throughput as the driving system performance requirement. The machine learning algorithms that are migrating out to the IoT edge are driving architects to increase the capacity of on-chip fast memory to store temporary results and reduce the memory transfer overhead. Computational latency, as well as energy concerns, are being addressed through 2-D Single-Instruction-Multiple-Data (SIMD) and systolic arrays of low-precision fixed-point Arithmetic Logic Unis (ALUs) [2].

Field Programmable Gate Arrays (FPGAs) are in a unique position to facilitate the transfer of Machine Learning (ML) algorithms out into IoT edge devices. The malleability of the compute fabric allows the creation of architectures to match the diverse spectrum of lower precision ML network topologies needed by the applications running at the IoT edge. The maturation of High-Level Synthesis (HLS) tools and increased numbers of diffused DSPs within the FPGAs have opened new opportunities for non-hardware experts to quickly translate algorithms into an FPGA-based hardware accelerator.

Despite these advantages, the historical barrier of poor programmer productivity must be eliminated if FPGAs are to gain universal acceptance within the broad programmer community and serve as transparent IoT edge infrastructure with other programmable components. From a programmer productivity perspective, the debate between using HLS languages or traditional Hardware Description Languages (HDLs) is moot. Either front-end language requires programmers to understand low-level hardware design, use hardware-centric CAD tools, and for even minor design changes, suffer through the time-consuming synthesis, place, and route. FPGAs will achieve full acceptance with other programmable IoT edge components by programmers when they are abstracted under familiar software development tools that remove the time-consuming step of hardware synthesis.

This paper presents a new overlay architecture that brings software levels of programmer productivity, code portability, and reuse to the design of any neural network configuration within the FPGA. The overlay contains a new Processor-In-Memory (PIM) SIMD processor array architecture designed to reduce communication latency and increase computational concurrency for the low-precision fixed-point arithmetic used in IoT edge devices. The contributions of this paper are:

- A System-on-Chip (SoC) design that includes a fully programmable memory-centric overlay and Instruction Set Architecture (ISA). The ISA allows the overlay to be programmed to support reduced precision, low-latency inferencing of any ML network.
- A new "elastic" memory-centric compute tile that can be composed by scripts to form any size 2-D compute array within the overlay for different FPGA devices. The memory-centric architecture reduces both computation and communication latencies with the 2-D array of mixed variable and low-precision multiply-accumulate units.
- Performance analysis of standard MLP, CNN, LSTM, and GRU benchmarks implemented on Xilinx Virtex-7 and Ultrascale FPGAs. Run time results show the overlay achieves competitive and even lower inference latencies compared to custom FPGA-based accelerators.

II. Processor Array Architecture

Fig. 1(a) shows the block diagram of our SoC overlay architecture with an expanded view of the processor array. The processor array executes as a decoupled accelerator sequenced by a standard processor within a larger SoC overlay. Sections

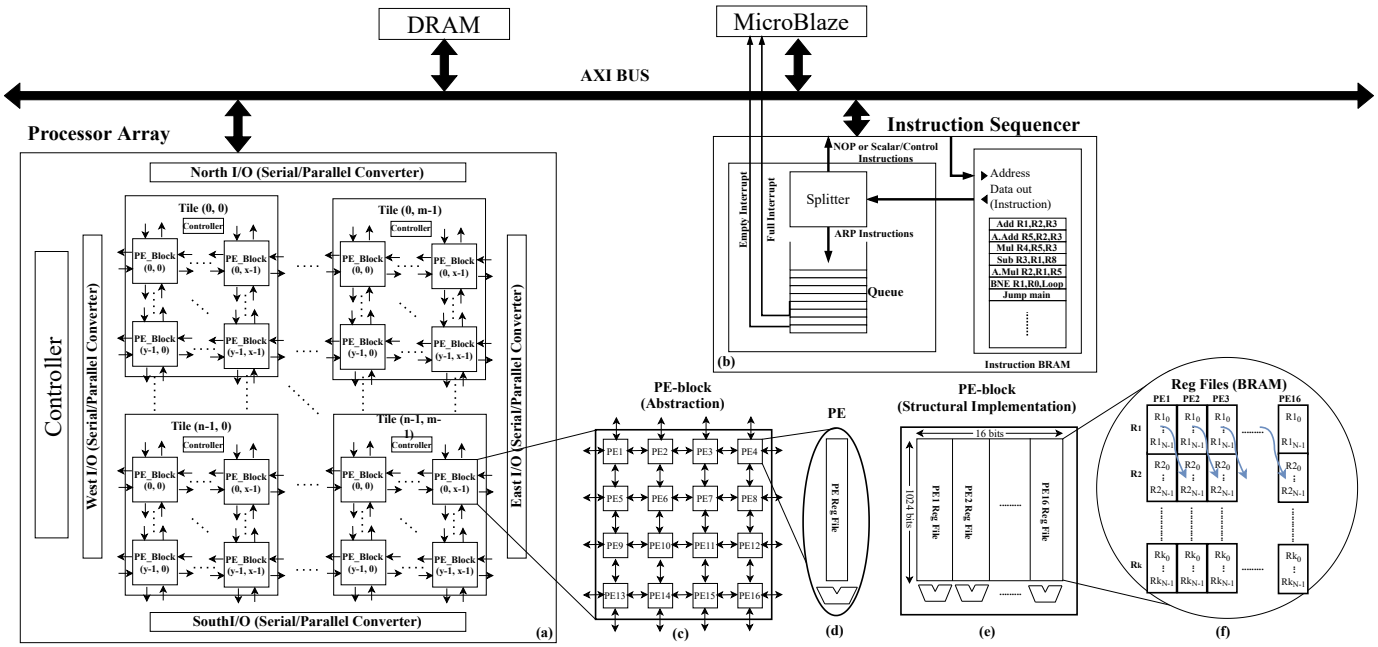

Fig. 1. PIM processor array.

of the application program to be executed on the processor array are written, compiled, and linked as normal functions and subroutines within the main program running on the standard processor. The processor array uses the instruction format of the controlling processor (MicroBlaze) and contains a decoder that issues Micro-Operations (uops) that sequence the Processing Elements (PEs) in the array. This allows programs to be written and compiled for the processor array using a standard compiler (such as for the MicroBlaze).

The overlay includes an $Instruction\ Sequencer$ that snoops instructions from the address range of the $Instruction\ BRAM$ (Fig. 1(b)). The sequencer places instructions for the processor array into a queue and returns a NOP to the MicroBlaze. The queue enables the Microblaze to prefetch sequences of instructions for the processor array. This decoupling allows the MicroBlaze and the processor array to asynchronously and concurrently continue execution. When needed, the MicroBlaze and the processor array synchronize through an interrupt-driven protocol.

A. Memory-Centric Configurable Processor Array

Fig. 1(a) shows how the processor array is configured as a 2-D SIMD array of $m \times n$ compute tiles. Each tile includes a local controller. Fig 1(b) shows how each tile is configured as an $x \times y$ array of PE-blocks. The $m \times n$ dimensions of compute tiles and $x \times y$ dimensions of PE-blocks are parameters in a build script. Designers can set these parameters to configure and optimize the array for different use scenarios as well as logic family-specific resource configurations and capacities.

The hierarchical organization of tiles, PE-blocks, and controllers has been defined to localize signal fanouts. This eases routing congestion and reduces clock and signal delays for arrays that approach high per chip resource utilizations. The tiles and PE-blocks are automatically created using the build script within a default North-East-West-South ($NEWS$) interconnect network. This default network can be customized or replaced by different domain-specific interconnects. In section III-D, we show how communications latency affects end-to-end inference latency for certain classes of neural networks and can be reduced by replacing the ($NEWS$) network with a binary reduction tree interconnect network.

B. PIM PE-block Architecture

Fig. 2(a) shows a typical configuration of BRAM-DSP blocks configured to maximize throughput of streaming data. Fig. 2(b) contrasts this configuration with the proposed PIM architecture configured to reduce latency by forming groups of ALUs with concurrent access to a block of common shared local storage. Fig. 3 shows a $18kb$ BRAM configured as 16×1024 bits distributed memory. In our PIM architecture, the width of a row sets the number of ALUs that can be connected in parallel to one BRAM. Parallelism can be increased by widening the row width of the BRAM. Designers can trade-off ALU concurrency versus precision and local storage capacity for different logic families.

Regardless of ALU precision or logic family, the processor array's decoder is made aware of the data-widths specified in the application code and issues the necessary uops to sequence the ALUs accordingly, transparently to the programmer. This decouples algorithm optimization from hardware optimization and supports the objective of bringing software levels of productivity to FPGA hardware design. Conversely, changing the data-widths in the high-level application, particularly when

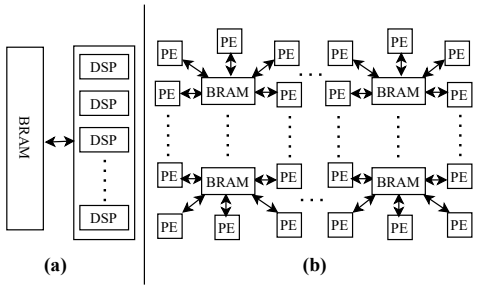

Fig. 2. (a) DSP-based method vs. (b) Proposed memory-centric method.

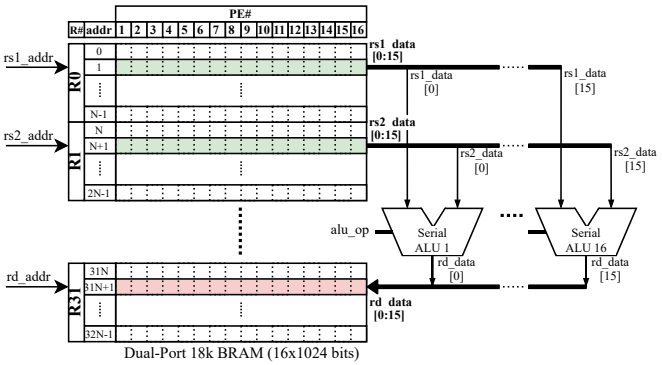

Fig. 3. Bit-serial ALUs.

dealing with low-precision widths, in traditional custom accelerator flows can require a resynthesis of the complete design.

1) Local Storage: Each ALU is provided with 1024 bits of local storage. Dynamic typing based on the operand width is used to set the iteration count, bit-width, and the number of storage registers in the BRAM for a particular ALU bit-width. This allows programmers to compile domain optimizations that effectively scale resources to arithmetic precision.

2) Data Movement: Fig. 1(c) show the PEs configured within a $NEWS$ interconnect network. The sharing of a memory block by multiple PEs eliminates the need to embed an additional physical interconnect network. Inter-PE communications are realized through rotating data (logical shifts) in the BRAM (Fig. 1(f)). Moving data by logical shifts in the BRAM eliminates the additional data paths and buffers required to implement a separate interconnect network.

III. Experimental Results

A. Experimental Setup

We create the SoC system shown in Fig. 1 with $10k$ PEs to serve as our base experimental hardware platform using Vivado 2018.3 tools. The same code was run on the two FPGAs (Virtex-7 and Virtex Ultra), informally validating the overlay's ability to support code portability and reuse. Along with abstraction, code portability, and reuse are the cornerstones necessary to achieve software levels of productivity. Design goals for IoT edge devices are to reduce latencies and not maximize peak throughput. The implemented benchmarks were written in C, compiled, and linked with the overlay ISA.

B. Compilation, Code Portability, and Reuse

As shown in Table I, qualitatively, we were able to compile and run all network types on our single processor array. This validates that rapid changes to the network structure can be realized through compilation without requiring CAD tools or synthesis. Importantly, this also opens the use of these devices to programmers with no hardware design expertise. The same application code was then run on both FPGA devices, demonstrating code portability and reuse. Finally, the latencies and speedups show that enabling programmer productivity levels, code portability, and reuse does not necessarily need to come at the cost of reduced performance.

C. Performance Analysis

Column three in Table I shows the speedups achieved over reported single network custom accelerators and an overlay design. Column eight shows that the arithmetic operations in the reported designs were instantiated in DSPs. Our processor array implemented bit-serial circuits in the gates and used no DSPs. Regardless of how arithmetic circuits are implemented, latency degrades when the circuit is forced to idle, waiting for operands. Incorporating BRAMs in the system's linear global address space outside of user logic IP limits achievable bandwidth that then limits the number of DSPs that can operate concurrently. This results in the serialization of operations that could have been computed concurrently. The structural definition of our PIM architecture matched the memory bandwidth to the $10k$ bit-serial PEs. Table I shows the processor-in-memory architecture with bit-serial PEs delivers competitive performance with custom designs. The precision of the operations can be lowered to match the needs at the IoT edge in software by changing the data types and recompiling. Programmers can change the design without having to resynthesize, and the inference latency will scale down as the precision is lowered.

D. Domain Customization

Ultimately, the computation/communication ratio determines the end-to-end inference latency seen by a user. Table II breaks down the percentage of cycles spent in computations (Array Active Cycles) and communications (Internal Data Movement and Weight Stall Cycles) within our overlay. Consistent with results reported in [3], the MLP/LSTM/GRU benchmarks are communication-bound. Without any loss to the generality, a system designer targeting such communication-bound networks may want to perform some additional domain customizations to further reduce inference latency for their applications. Amdahl's law would point in the direction of the communications subsystem. Table III shows how these data movement cycles can be reduced by augmenting the $NEWS$ network with a binary tree reduction/interconnect network along with the additional resources for a Virtex Ultra. These customizations can be encapsulated in software macros as part of a domain-specific library available to programmers. The CNN network is not included in Table III as the Internal Data Movement is a small portion of its total execution cycles.

TABLE I
IMPLEMENTATION RESULTS

Name	Latency	Speedup	Data [a] Format	LUTs	FFs	BRAMs	DSPs	Freq. (MHz)	FPGA	Method
LSTM(1) (61, 250, 250, 250, 39) on TIMIT dataset										
Guan [4]	390 ms		FLP 32	198280	182646	1072	1176	150	Virtex-7	HLS
This	17.5 ms	22.2	FxP 32	138380	67801	313	**0**	130	Virtex-7	
Work	11.4 ms	34.2	FxP 32	133890	56207	313	**0**	200	Virtex Ultra	Overlay
LSTM(2) (64, 128, 128, 64) on CharRec dataset										
Chang [5]	900 us		FxP 16	7201	12960	16	50	142	Zynq	HDL
This	395.9 us	2.3	FxP 16	138380	67801	313	**0**	130	Virtex-7	
Work	257.1 us	3.5	FxP 16	133890	56207	313	**0**	200	Virtex Ultra	Overlay
MLP (874, 100, 100) on MNIST dataset										
SNN [6]	1.3 ms		FxP 25	139562	175604	50	400	100	Virtex-7	HDL
This	0.5 ms	2.6	FxP 32	138380	67801	313	**0**	130	Virtex-7	
Work	0.3 ms	4.3	FxP 32	133890	56207	313	**0**	200	Virtex Ultra	Overlay
CNN SqueezeNet v1.1 on ImageNet dataset										
CNN-Grinder [7]	70.5 ms		FxP 8	34489	25036	97.5	172	100	Zynq	HLS
Light-OPU [8]			FxP 8	173522	241175	193.5	704	200	Kintex-7	Overlay
This	51.0 ms	1.3	FxP 8	138380	67801	313	**0**	130	Virtex-7	
Work	33.1 ms	2.1	FxP 8	133890	56207	313	**0**	200	Virtex Ultra	Overlay
GRU (39, 256, 200, 10) on DeepSpeech dataset										
DeltaRNN [9]	26.4[b] ms		FxP 16	261357	119260	768	457.5	125	Zynq-7000	HDL
This	3.3 ms	8.0	FxP 16	138380	67801	313	**0**	130	Virtex-7	
Work	2.1 ms	12.5	FxP 16	133890	56207	313	**0**	200	Virtex Ultra	Overlay

[a] FxP := fixed-point and FLP := floating-point.
[b] The reported latency is for a delta threshold of $0x00$ [9], which is equivalent to what is implemented in the proposed design.

TABLE II
BREAKDOWN OF EXECUTION CYCLES

Operation	LSTM(1)	LSTM(2)	MLP	CNN	GRU
Array Active Cycles[a]	33.6%	23.9%	54.2%	84.1%	10.8%
Internal Data Movement[b]	37.8%	76.1%	45.8%	0.1%	27.6%
Weight Stall Cycles	28.6%	0%[c]	0%[c]	15.8%	61.6%

[a] Multiply-Accumulate (MAC) operations.
[b] $NEWS$ operations.
[c] No data movement from DRAM to BRAM.

TABLE III
EFFECTS OF BINARY TREE INTERCONNECT

Benchmark	Binary Tree add Linear shift	Binary Tree add Binary Tree shift
Execution Time		
LSTM(1) (ms)	11.4	8.2
LSTM(2) (us)	257.1	123.8
MLP (ms)	0.3	0.2
GRU (ms)	2.1	1.2
Resource Utilization (10k PEs)		
LUTs	133890	492937
FFs	56207	76501
BRAMs	313	313
DSPs	0	0

CONCLUSION

This paper presented a domain customizable processor-in-memory overlay to enable programmers with no hardware design expertise to program any type of machine learning networks into FPGAs. The proposed overlay can be configured through software to operate on different data-widths as well as network configurations. The presented processor-in-memory architecture matches memory bandwidth with large numbers of ALUs. This allows inference latency to be reduced through concurrency and reduced precision operations, which is essential in user-facing and real-time IoT applications. Experimental results showed that the proposed single processor array achieved speedups over a range of custom network designs.

REFERENCES

[1] A. Ishfaq. "Discover Internet of Things editorial," *Discover Internet of Things*, 10.1007/s43926-021-00007-6, 2021.

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

[3] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, et al., "In-datacenter performance analysis of a tensor processing unit," *In Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017.

[4] Y. Guan, Z. Yuan, G. Sun, and Cong, J., "FPGA-based accelerator for long short-term memory recurrent neural networks," *In 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 629–634), 2017.

[5] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on FPGA," *arXiv preprint arXiv:1511.05552*, 2015.

[6] W. uo, H. E. Yantir, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Toward the Optimal Design and FPGA Implementation of Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[7] P. G. Mousouliotis, and L. P. Petrou, "CNN-Grinder: From Algorithmic to High-Level Synthesis Descriptions of CNNs for Low-end-low-cost FPGA SoCs," *Microprocessors and Microsystems*, vol. 102990, 2020.

[8] Y. Yu, T. Zhao, K. Wang, and L. He, "Light-OPU: An FPGA-based Overlay Processor for Lightweight Convolutional Neural Networks," *In The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 122–132, 2020.

[9] C. Gao, D. Neil, E. Ceolini, S. C. Liu, and T. Delbruck, "DeltaRNN: A power-efficient recurrent neural network accelerator," *In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 21–30, 2018.