2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID) | 978-1-6654-8505-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/VLSID2022.2022.00058

A Soft RISC-V Vector Processor for Edge-AI

V. Naveen Chander, Kuruvilla Varghese, Senior Member, IEEE Indian Institute of Science, Bengaluru{naveenv,kuru}@iisc.ac.in

***Abstract*—Edge computing is the key to unlocking the power of deep neural networks on edge devices. However, deploying power-hungry deep neural network inference on resource-constrained and power-limited devices poses serious challenges in delivering real-time performance. With the advent of RISC-V Vector extension, there has been a renewed interest in vector processors to exploit data-parallel workloads. General purpose processors featuring vector coprocessors are riddled with complex control mechanisms such as instruction schedulers, operand queues, and scoreboards which have largely inhibited their presence in the realm of low-power microcontrollers. This work features a systolic array based vector unit that is closely integrated into the pipeline of a 32-bit, in-order, single-issue RISC-V scalar core that runs at 50 MHz. The robustness of neural networks coupled with the flexibility offered by the RISC-V Vector extension instruction set is used to significantly reduce several architectural complexities of the vector unit. The vector processor is implemented on Xilinx Virtex 7 (XC7VX485T) FPGA. Benchmarking the RISC-V Vector processor shows a speedup of up to 40.7x over the scalar RISC-V core on image recognition tasks at the cost of 1.2x power consumption and 1.8x hardware resources. The soft core vector processor also compares well with similar processors that use data-level parallelism.**

I. Introduction

Moore's law may no longer govern the growth curve of today's computers, but it seems to apply well on the number of new articles published on arXiv.org per year, in the field of machine learning (ML) [5]. ML's growth is largely fueled by the emergence of mobile and IoT devices as avenues for deploying deep neural networks (DNN), a key ML technique. Also, the growing volume of data generated by these edge terminals is gradually making it difficult for large-scale cloud data-centers to deliver the required performance due to rapidly increasing computational workloads. Edge computing has emerged as an attractive alternative as it reduces costs, latency, dependence on network availability and security concerns [14].

A. Understanding the complexity of DNNs

DNNs are computational and memory intensive. A six- layer convolutional neural network (CNN) shown in fig. 1 to classify handwritten digits from the MNIST [12] dataset was found to take about 94 ms to classify one image when run on a 32-bit RISC-V microcontroller developed by [1]. Real-time inference systems typically tolerate latencies up to 7 ms [13], which severely inhibits usage of microcontrollers for such applications. To understand the nature of computational workloads involved in DNN inference, program analysis of some commonly used inference topologies was carried out.

ML inference largely consists of data-parallel workloads [5], [6]. Computational bottlenecks in ML inference were examined by profiling common neural network topologies such

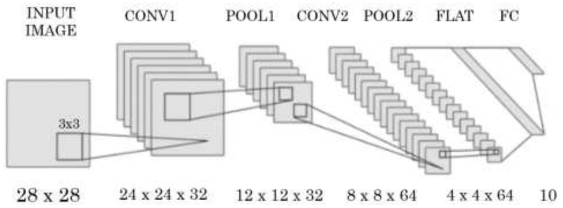

Fig. 1: Convolutional Neural Network used for profiling

TABLE I: Distribution of Major Operations on CNN and RNN

CNN				RNN	
CONV2D	ReLU	Maxpool	VMATMUL	VMATMUL	Activations
98.78%	0.26%	0.21%	0.13%	98.08%	1.43%

as multi-layer perceptron (MLP), recurrent neural network (RNN) and CNN on *Intel Xeon E5-1607-v3* CPU. Following are the observations when their C-codes were profiled on Intel Xeon CPU -

1) Most neural networks consist of linear operations such as matrix-vector multiplication, convolution, vector scaling and vector exponent computations. Percentage distribution of the major operations is shown in table I.
2) CPU is busy with matrix/vector operations for > 99% of the total execution time for all these topologies.
3) For the six layer CNN shown in fig. 1 with 5x5 kernels, the CPU spends about 99% of the total execution time in performing 2D convolutions. About 98% of the total execution time for an RNN goes into matrix vector multiplications.
4) Even though many neural network topologies feature complex, non-linear computations such as *sigmoid* and *tanh*, they occupy a negligible fraction of the total execution time. For example, profiling of an RNN shows that exponent computations constitute only 1.43% of the total execution time. Going by Amdahl's law, any investment on hardware accelerators for exponential computations is bound to provide only limited pay-off.

While these experiments reinforce the notion of neural network workloads being largely data-parallel, they also clearly indicate that devising architectures to speedup data parallel operations alone is sufficient to accelerate ML inference.

B. Vector Processors

Vector processors handle data level parallelism by executing several instructions across multiple lanes. While a vector instruction can operate on only one element at a time, it can execute n instructions simultaneously, where n is the number of lanes. Vector processors feature a vector-length register to set the length of the vector at run time.

VIPERS [16] demonstrated the use of soft core vector processors as a data-parallel accelerator. VESPA [15] is another general purpose soft core vector processor that is built on a vector extension of MIPS ISA. The authors customized the vector processor for domain-specific architectures to show a significant reduction in the hardware resources with negligible impact on performance. VEGAS [4] soft core vector processor demonstrates how vector architectures can be targeted for embedded systems. Instead of caches, it features a software-controlled scratchpad memory to store vectors. VEGAS demonstrates improvements between 10x and 208x over Intel's 32-bit NIOS-II embedded microprocessor.

ARA [2] is a 64-bit vector unit that supports the RISC- V Vector extension (RVV) and is interfaced to a 64-bit application class processor that runs at 1GHz. The authors bring out computational bottlenecks related to matrix/vector kernels on vector processors.

Work in [11] proposes a low-power vector processor which features a tightly-coupled vector unit that is directly interfaced to the pipeline of a scalar RISC-V processor. It features a minimal subset of instructions from the RVV ISA and demonstrates a speedup of up to 5.8x over a scalar RISC-V core at 50 MHz on Xilinx Spartan FPGA. While the speedup is not appreciably high, the vector unit alone adds about 160% more hardware to the scalar RISC-V core.

Klessydra-T [3] is a multi-threaded soft core vector coprocessor targeting edge computing applications. With three hardware threads and a vector coprocessor employing custom vector instructions, Klessydra-T demonstrates a speedup of about 36x over the packed SIMD based RI5Cy Core [9] on matrix multiplication kernels. Heavy usage of custom vector instructions restricts the portability of vector applications across vector processors.

Several studies related to neural network optimization show that neural networks are robust to accommodate loss in precision and variation in size. Popular techniques such as neural pruning and quantization have demonstrated significant reduction in the size of neural networks without noticeable loss in accuracy. [8], [10]. The vector unit designed in this paper uses the robustness of neural network topologies as a means to simplify various architectural complexities which are normally found in most vector processors. This work presents two key contributions:

- A lightweight, portable vector microarchitecture targeting ML inference which is integrated to a scalar RISC-V pipeline to obtain a speedup of up to 40x with a cost of 25% more power and about 85% more resources the scalar RISC-V core
- A custom benchmark consisting of commonly used kernels in ML inference which can be used to evaluate architectures targeted at edge-AI.

II. Microarchitecture

The vector unit is designed as a *systolic array* of eight Processing Elements (PEs) spread across eight lanes. These PEs execute in a tightly coupled and periodic manner sharing a common pool of resources such as vector register file and scratchpad memory. The vector unit is tightly coupled with the scalar RISC-V pipeline [1] to obtain the vector processor.

A. Instruction Set

The vector unit implements about 50 instructions from the RISC-V Vector Extension [7] comprising of vector-integer arithmetic, logical and vector memory operations. Since neural networks are robust to loss in precision, vector floating point support is not provided. The vector unit handles only 32-bit integers and does not support mixed-width operations. In the context of ML inference, softmax is often implemented as a function that finds the arg-max of a vector. Hence, four custom instructions for finding arg-max/min of a vector have been added to the instruction set.

B. Scalar RISC-V Processor

The scalar RISC-V processor is an in-order, single-issue, 32-bit microcontroller class RV32G core with 5-stage pipeline [1] that runs at 50 MHz. It features a 1MB block RAM with SECDED ECC as main memory and a uART peripheral which are interfaced to the host CPu pipeline using as Wishbone- B3 bus. From fig. 2, the Instruction Fetch (IF) includes an 8kB I-cache, TLB and a two-bit G-share branch predictor that drives the program counter. The Instruction Decoder (ID) decodes instructions to derive control signals. The Vector Interface unit (VIu) establishes a handshake with the Vector unit and supplies all decoded control signals and scalar operands to the vector unit. Execute (EXE) stage performs scalar ALu operations on integers and floating point numbers and computes branch targets. The Memory (M) stage contains an 8kB D-cache and control and status registers. The results of a scalar instruction are written into a scalar register file containing registers *r0-r31* in the Write-back (WB) stage.

C. Vector Interface Unit (VIU)

The Vector unit is interfaced to ID stage of the host processor's pipeline by means of a VIU as shown in fig. 2. The VIu features a vector instruction decoder that checks for vector instructions and fully decodes them only if the master decoder decodes an incoming instruction as a vector instruction. As the vector unit is closely coupled to the host processor, the instruction execution between the scalar core and the vector units is mutually exclusive, i.e., when scalar core executes instructions, the vector unit is stalled and vice-versa. This arbitration is done by the VIU. The VIU ensures that order of program execution is preserved. The VIU provides a native bus interface between the vector unit and the scalar processor pipeline. The VIU bundles up to eight vector instructions before they are dispatched to the vector unit. Decoded instructions are stored in a local buffer until they are bundled. If the incoming stream of instructions has $n > 8$ consecutive vector instructions, vector instructions are dispatched to the vector unit in batches of eight. If the incoming instruction stream has $n < 8$ consecutive vector instructions, then the VIU bundles only n instructions and dispatches them to the vector unit. This process incurs a delay of 26 clock cycles before the vector convoy can commence execution. The effect of this overhead is examined in the results section.

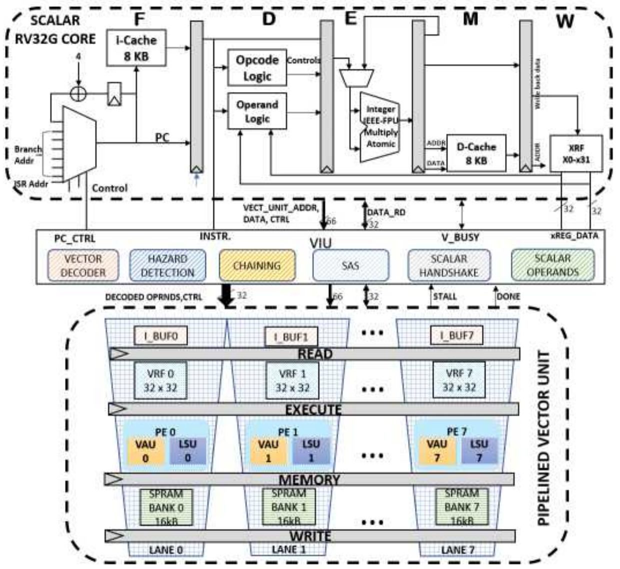

Fig. 2: Microarchitecture diagram of RISC-V Vector Processor. RV32G designed in [1] is interfaced with the 4-stage pipelined vector unit using Vector Interface Unit (VIU). Each Lane of the vector unit consists of (|)th Vector Register File, Vector Arithmetic Unit (VAU), Load Store Unit (LSU) and Scratchpad Memory (SPRAM)

D. *Vector Register File (VRF)*

Since RISC-V is a load-store ISA, all operations are carried out on vectors residing in vector registers [7]. The VRF consists of 32 vector registers *v0-v31.* Each vector register is 256-bits wide and can accommodate eight 32-bit integers. Also, up to eight consecutive vector registers can be grouped together to store a single vector. Thus, a vector instruction can operate on a maximum of 64 elements, which is referred to as the maximum vector length (MVL).

The 256-bit wide VRF is organized into eight register banks such that each vector lane contains a 32-bit wide register file. Bank-conflicts can occur when multiple lanes attempt to access the same VRF bank simultaneously. The vector control unit, termed as *Systolic Array Sequencer (SAS),* schedules the instructions across vector lanes in such a way that no two lanes contend for the same register bank at a given time. This has been made possible by providing each VRF with five read- ports and one write-port as shown in fig. 3. Details on SAS are discussed in subsequent sections.

VRFs are realized as distributed memory blocks in the FPGA which can provide data with *zero-cycle* latency.

E. *Vector Arithmetic Unit*

Vector arithmetic unit (VAU) is spread across eight lanes. VAU in each of these lanes performs the two types of operations on vectors-

1) Multi-operand operations which operate on two or three input vectors to produce an output vector. Eg. *vadd, vmacc* etc.,
2) Uni-operand operations which operate on a single vector operand to produce a scalar or a vector output. Eg. *vmin*, *vmax*, *vargmax*, *vslide* etc.,

VAU supports fractional numbers in fixed point format. Allowed representations are Q32.0, Q24.8, Q16.16 and Q8.24.

F. *Vector Memory System*

Since vector registers offer only a limited capacity to hold vectors, a vector memory is used to hold longer vectors and matrices such as the weights of a neural network.

Each lane in the vector unit has a vector load-store unit (VLSU) and one bank of a 16 KB software-controlled scratchpad memory (SPRAM). The VLSU generates addresses to support three addressing modes- unit-stride, strided and vector- indexed. Up to 64 elements can be loaded/stored with a single vector instruction. Vector load stores are masked i.e., vector elements can be selectively loaded/stored into/from vector registers.

FPGA Block RAMs are used to provide a total capacity of 128 KB for the vector memory with a read latency of one clock cycle.

G. *Systolic Array Vector Execution Unit*

A Systolic Array is a network of identical PEs that can process data systematically by sharing a pool of resources among themselves in a coordinated manner. Systolic arrays are extremely efficient when it comes to carrying out computations of simple arithmetic operations on large data such as matrix multiplication, convolution etc., and has been used in execution units of high-throughput machine learning processors such as Google TPU [6].

Fig. 3 shows a skeletal representation of the eight-lane vector unit with the VRF and SPRAM data interleaved across eight banks. Each lane can access data from any of the eight memory banks through crossbar switches shown. Each PE block consists of a VAU and a VLSU. The SAS controls crossbar switches *x1, x2, x3* and *x4* to transfer data across vector lanes as shown in fig. 3.

1) Systolic Array Sequencer: Once the VIU provides decoded vector instructions and other required operands to the vector unit, the SAS distributes instructions across vector lanes, schedules their execution and facilitates usage of shared resources. The design of this sequencer is critical as it determines the throughput of vector unit and prevents bank-conflicts in VRF.

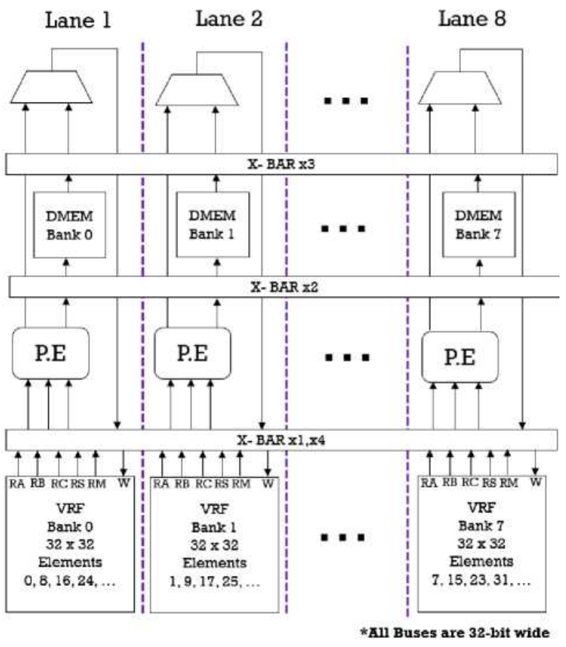
Lane 1
Lane 2
Lane 8
X- BAR x3
DMEM Bank 0
DMEM Bank 1
DMEM Bank 7
X- BAR x2
P.E
P.E
P.E
X- BAR x1,x4
RA RB RC RS RM W
VRF Bank 0 32 x 32 Elements 0, 8, 16, 24, ...
VRF Bank 1 32 x 32 Elements 1, 9, 17, 25, ...
VRF Bank 7 32 x 32 Elements 7, 15, 23, 31, ...
*All Buses are 32-bit wide

Fig. 3: Block diagram showing a skeleton of the vector execution unit organized into eight lanes. Each VRF bank consists of five read ports and one write-port. Ports *RA, RB* and *RC* provide operands for VAU, *VS* provides vector operand for vector slide and the vector mask is issued through *VM*

TABLE II: Variation of VRF bank access pattern by lanes as scheduled by the SAS. This pattern of access is followed in all four stages of the vector pipeline to eliminate VRF bank conflicts

Cycle / Lane	1	2	3	4	5	6	7	8	9	10
1	0	1	2	3	4	5	6	7	0	1
2	-	0	1	2	3	4	5	6	7	0
3	-	-	0	1	2	3	4	5	6	7
8								0	1	2

2) Lane-sequencers: The SAS is designed as hierarchical set of sequencers with a master sequencer controlling eight lane-sequencers. Master sequencer starts when it receives a *start* signal from VIU. Each lane has its own microsequencer which is controlled by the master sequencer. The micro-sequencer determines index of the vector element to be processed by a lane in a given pipeline stage. The master sequencer initiates lane sequencers in successive clock cycles as shown in table II. Since vectors in the VRF are aligned with bank zero as shown in fig. 3, the resulting VRF bank access pattern by all lanes works out as shown in table II. The SAS control signals are bubbled through the pipeline to ensure that bank conflicts do not occur in any stage of the vector pipeline.

When all the eight micro-sequencers run to completion, the master sequencer issues a *done* signal back to VIU to indicate completion.

H. Implementation

The designed vector processor is implemented on Xilinx Virtex 7 (XC7VX485T) FPGA with a target frequency of 50 MHz, which is the frequency of the scalar processor. Table III shows the hardware utilization in terms of FPGA primitives and the percentage of total CPU's resources used for major modules of RISC-V Vector processor. The vector unit is the largest module and uses about 46% of the total CPU's resources. In other words, addition of the vector unit, increases the CPU's resources by about 85%.

TABLE III: Hardware resource utilization of major constituents of

Module	Slice LUTs	Slice Registers	BRAMs	DSP Tiles
Vector Pipeline	35737 (46%)	19896 (47%)	32 (10 %)	43 (64%)
Scalar Pipeline	41996 (53%)	22032 (52%)	27 (9%)	24 (36%)
Main Memory, Peripherals, Bus Interconnect and I/O	1201 (1.5%)	470 (1.1%)	256 (82%)	0

RISC-V vector Processor

Power analysis was done for vector and scalar processors by the method of vector-based estimation using Xilinx Power Analyzer. The 6-layer CNN in fig. 1 is run on both scalar and vector cores in post-implementation functional simulation to generate activity vectors for all internal signals and then fed to the Xilinx Power Analyzer to estimate the power consumption. Table IV compares the power components of scalar and vector RISC-V processors. Since, device static power is due to the intrinsic leakage of various circuits inside FPGA, it remains the same irrespective of the design. From table V, it can be seen that vector unit consumes about one-fifth of total processor power. In other words, the addition of vector unit increases the power consumption by 21%.

TABLE IV: Power consumption by RISC-V CPU

	Total On-chip Power	Dynamic Power	Device Static Power
RISC-V Vector	597 mW	333 mW	264 mW
RISC-V Scalar	496 mW	235 mW	261 mW

TABLE V: Power consumption by constituents of RISC-V Vector CPU

Module	Hierarchical Power	Total Power
Vector Pipeline	120 mW	20%
Scalar Pipeline	95 mW	16%
Memory and Peripherals	115 mW	20%
FPGA Device Static	264 mW	44.3%

Static timing analysis shows that although the vector unit could run at 100 MHz frequency as a stand-alone unit, the bus interface logic for the host CPU to directly access the VRF and SPRAM forms the critical path in the entire design, with a positive slack of about 0.4 ns at 50 MHz.

III. Experimental Results

Edge AI requires performance with minimal hardware and power costs. Thus, the vector processor is evaluated on both speed and energy fronts by carrying out post-implementation simulations on the Xilinx Vivado simulator for various kernels to measure the execution time and energy per operation.

Commonly used mathematical operations in neural networks, which were identified during workload analysis of neural networks have been aggregated into a custom benchmark. The benchmark developed in this work consists of 30 programs that fall under four kernels as shown in table VI. It also includes two neural networks-three-layer perceptron and a six-layer CNN to classify handwritten digits based on the MNIST dataset [12]. Among the four operations, AXPY has the lowest computational intensity and the least memory footprint for a given vector size, MATMUL occupies the highest memory for a given matrix size and CONV offers the highest computational intensity. The benchmark also includes an MLP and a CNN to classify images from MNIsT handwritten digits dataset [12], which are discussed later. The results of kernels

Benchmark Class	**Description**	**Use in ML**
AXPY	$Y = aX + Y,\ a \in \mathrm{I}$	Input Scaling, Batch-normalization
Perceptron	$\mathrm{Y} = ReLU\ (WX + \mathrm{b}),\ W \in \mathrm{I}^{m\mathrm{x}n}$	MLP, RNNs, FC layers, RBMs, Autoencoders etc.,
MATMUL	Square Matrix Multiplication	Regression, SVM
CONV	2D convolution	CNN, Capsule Networks

TABLE VI: Programs in the benchmark and their uses in ML inference

run on the RISC-V scalar and vector cores are compared. The results reported by other data parallel processors such as Klessydra-T [3] and RI5Cy [9] for the benchmarking programs have been used to evaluate the processor under test.

A. Benchmarking Results

Benchmark programs are written in C language for RISC-V scalar target and compiled using *riscv-gnu-toolchain* with *-O3* optimization. Vector programs are written in RISC-V Vector assembly.

1) Performance: Speedup of the vector core over the scalar core is plotted for various values of vector lengths in fig. 4. The speedup ranges between 2.5x and 68x. We see a general trend of increasing speedup with an increase in vector length. This is attributed to startup overheads due to SAS and handshaking with VIU, which is fixed at 26 cycles. For small vector lengths such as vl = 8, the percentage overhead is as high ar 83%.

The steady-state speedup of about 10x in the speedup curve for AXPY in fig. 4 does not continue indefinitely. It is limited by the capacity of vector scratchpad memory. This can be seen in the speedup plot for Perceptron in fig. 4, where the speedup increases till 67x for vl = 64 before it drastically drops to about 5x for vl = 128. Due to scratchpad memory

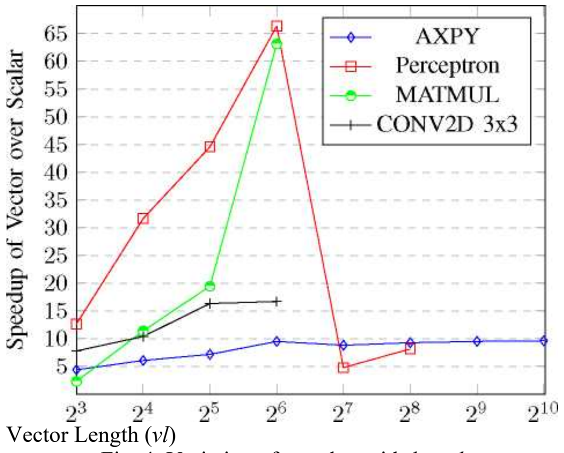

Fig. 4: Variation of speedup with $log_2 vl$

overflow, the perceptron weight matrix had to be fetched from main memory which takes about 3.4 ms, while the vector operations to compute the perceptron output take only about 268^s. Hence, the sudden drop.

B. Energy per Algorithmic Operation

Energy per algorithmic operation is calculated as the product of on-chip power consumption and execution time averaged over n algorithmic operations [9], [3]. The on-chip power consumption for algorithms was estimated by carrying out vectored power estimation on the *Xilinx Power Analyzer* tool by running post-implementation functional simulations for the algorithm on scalar and vector RISC-V cores. The last row in table VII provides the energy per operation for RISC-V vector and scalar cores and the ratio of their energies. Due to high speedup, *MATMUL* and *Perceptron* show a higher energy efficiency compared to *CONV* and *AXPY*.

C. Comparison with other Data Parallel Processors

Table VIII compares the execution time values for RISC- V vector processor with two other data parallel processors available in [3]. For convolution workloads, the RISC-V vector processor has the lowest execution time for 8x8 and 16x16 images. Klessydra-T is fastest on 32x32 images for 3x3 and 5x5 kernels. RI5Cy is fastest on MATMUL. On the whole, it can seen that the RISC-V vector processor has a comparable performance with other similar processors that use data level parallelism.

D. Image Classification

ML inference was run on RISC-V vector and scalar processors to perform image recognition on MNIST dataset using (a) 3-layer perceptron consisting of 32, 16 and 10 neurons with ReLU activations in the intermediate layers and (b) 6-layer CNN as shown in fig. 1. All weights and activations were quantized to Q8.24 format post training. The images to be

Vector Length	AXPY			Perceptron			MATMUL			CONV2D-3x3		
	#cycles		Speedup	#cycles		Speedup	#cycles		Speedup	#cycles		Speedup
	Vector RISC-V	Scalar RISC-V		Vector RISC-V	Scalar RISC-V		Vector RISC-V	Scalar RISC-V		Vector RISC-V	Scalar RISC-V	
8	48	210	4.38	3142	39772	12.66	1755	4040	2.30	250	1946	7.78
16	56	340	6.07	3662	115915	31.65	4610	52565	11.40	991	10314	10.41
32	72	516	7.17	5087	226760	44.58	20858	405855	19.46	4238	69214	16.33
64	104	988	9.50	6782	449770	66.32	113786	7181791	63.12	17750	296398	16.70
Energy per algorithmic Operation	18.38 nJ/op	146.81 nJ/op	7.98	1.26 uJ/neuron	69.71 uJ/neuron	55.10	71.24 uJ/op	1.35 mJ/op	52.43	55.13 nJ/pixel	764.9 nJ/pixel	13.87

TABLE VII: Benchmarking Results

TABLE VIII: Comparison of Kernel performance with other similar processors having Data level parallelism

CPU Name	Freq. (MHz)	Execution Time (MS)				
		CONV2D 3x3			CONV2D 5x5	MATMUL
		8x8	16x16	32x32	32x32	64x64
Klessydra-T [3]	120	9.91	21.18	59.54	112.38	2741.35
RI5Cy [3]	91.4	46.46	165.08	623.85	1969.37	1971
Vector RISC-V	**50**	**4.99**	**19.81**	**84.75**	**185.8**	**2275.71**

classified were fed to Xilinx FPGA housing the vector processor through UART. The vector processor runs MLP/CNN to classify the images and send the classified digit back to the host computer through UART.

The performance results are shown in table IX. It can be seen that the vector processor gives a speedup of about 40x on MLP and 20x on CNN with 33x and 16x savings in energy as shown in table IX.

TABLE IX: Performance and Energy Results for Neural Networks

	T_{exe} (Ms)		speedup	Energy Improvement
	Vector RISC-V	Scalar RISC-V		
MLP	345	14045	40.71x	33x
CNN	4813	93999.557	19.53x	16x

IV. Conclusion

A domain-specific, microcontroller-class vector processor for accelerating ML inference at the edge has been realized by augmenting a vector unit to an existing RISC-V core. While this opens up avenues for implementing a wide class of inference algorithms on the edge, it also provides a framework for making domain-specific hardware simplifications that lower the barriers to realize vector processors. The custom benchmark suite used in this work covers a wide range of kernels that are commonly used in ML inference and can be used to evaluate the performance of processors designed for Edge-AI applications.

References

[1] S. Budi, P. Gupta, K. Varghese, and A. Bharadwaj, “A risc-v isa compatible processor ip for soc,” in *2018 International Symposium on Devices, Circuits and Systems (ISDCS),* 2018, pp. 1-5.

[2] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, “Ara: A 1-ghz+ scalable and energy-efficient risc-v vector processor with multiprecision floating-point support in 22-nm fd-soi,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 28, no. 02, pp. 530-543, feb 2020.

[3] A. Cheikh, S. Sordillo, A. Mastrandrea, F. Menichelli, G. Scotti, and M. Olivieri, “Klessydra-t: Designing vector coprocessors for multithreaded edge-computing cores,” *IEEE Micro,* vol. 41, no. 2, pp. 64-71, 2021.

[4] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, “Vegas: Soft vector processor with scratchpad memory,” *Association for Computing Machinery,* 2011. [Online]. Available: https://doi.org/10.1145/1950413.1950420

[5] J. Dean, D. Patterson, and C. Young, “A new golden age in computer ar-chitecture: Empowering the machine-learning revolution,” *IEEE Micro,* vol. 38, no. 2, pp. 21-29, 2018.

[6] J. et.al, “In-datacenter performance analysis of a tensor processing unit,” *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA),* pp. 1-12, 2017.

[7] K. A. et.al, “Risc-v ”v” extension spec v1.0,” 2021. [Online]. Available: https://github.com/riscv/riscv- v- spec/blob/master/v- spec.adoc

[8] S. H. et.al, “Eie: Efficient inference engine on compressed deep neural network,” *International Symposium on Computer Architecture (ISCA)*, 2016.

[9] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gurkaynak, and L. Benini, “Near-threshold risc- v core with dsp extensions for scalable iot endpoint devices,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 25, no. 10, pp. 2700-2713, 2017.

[10] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” 2016.

[11] M. Johns and T. J. Kazmierski, “A minimal risc-v vector processor for embedded systems,” in *2020 Forum for Specification and Design Languages (FDL),* 2020, pp. 1-4.

[12] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[13] D. Patterson, “David patterson - domain-specific architectures for deep neural networks,” https://www.youtube.com/watch?v=FSwKCL8A9JQ& t=1647s, 2019.

[14] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” *IEEE Communications Surveys Tutorials*, vol. 22, no. 2, pp. 869-904, 2020.

[15] P. Yiannacouras, J. G. Steffan, and J. Rose, “Vespa: Portable, scalable, and flexible fpga-based vector processors,” in *In CASES'08: International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2008.

[16] J. Yu, C. Eagleston, C. H.-Y. Chou, M. Perreault, and G. Lemieux, “Vector processing as a soft processor accelerator,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 2, Jun. 2009. [Online]. Available: https://doi.org/10.1145/1534916.1534922