MXDOTP: A RISC-V ISA Extension for Enabling Microscaling (MX) Floating-Point Dot Products

Gamze İslamoğlu*, Luca Bertaccini*, Arpan Suravi Prasad*, Francesco Conti†, Angelo Garofalo†, Luca Benini*†
*IIS, ETH Zurich, Switzerland; †DEI, University of Bologna, Italy
{gislamoglu, lbertaccini, prasadar, lbenini}@iis.ee.ethz.ch, {f.conti, angelo.garofalo}@unibo.it

Abstract*—Fast and energy-efficient low-bitwidth floating-point (FP) arithmetic is essential for Artificial Intelligence (AI) systems. Microscaling (MX) standardized formats have recently emerged as a promising alternative to baseline low-bitwidth FP formats, offering improved accuracy with a block-wise shared exponent scale combined with per-element values. However, efficiently executing the key linear algebra primitives for AI applications on MX formats requires specialized hardware support for the fundamental operators such as scaled dot product. In this work, we propose `MXDOTP`, the first RISC-V ISA extension for MX dot products, focusing on the 8-bit MXFP8 FP format. We extend the open-source Snitch RISC-V core with a dedicated MXFP8 dot product-accumulate unit, which fully consumes blocks of eight 8-bit operands packed into 64-bit inputs. To feed `MXDOTP` at full utilization with four operands per cycle, including block scales, we exploit Snitch's Stream Semantic Registers (SSRs), achieving up to 80 % utilization with minimal impact on the Snitch core's architecture and no modification to the register file. Implemented in 12 nm FinFET, a cluster with eight `MXDOTP`-extended cores reaches up to 356 GFLOPS/W when computing MXFP8 matrix multiplications at 0.8 V, 1 GHz. Compared to a software baseline, where MX dot products are computed by type casting FP8 inputs to FP32 for higher accumulation precision and applying explicit block scaling, the cluster achieves 25 speedup and 12.5 better energy efficiency at a minimal 5.1 % area increase.

I. Introduction

Compact floating-point (FP) arithmetic has gained significant attention for energy-efficient computing, particularly in machine learning workloads [1]–[4]. Deep learning models, like Large Language Models (LLMs), demand massive computational and memory resources, making both compute efficiency and memory bandwidth key concerns. In low-bitwidth formats like FP8, increasing the exponent range improves dynamic range but reduces precision, while a larger mantissa improves precision at the cost of a smaller dynamic range. Block formats mitigate this hard tradeoff with a shared exponent, preserving both range and precision.

While block floating-point (BFP) formats have been widely studied [1], Microscaling (MX) formats offer greater flexibility with a two-level exponent representation, where groups of FP elements share a wide exponent while each low-bitwidth element retains a narrower individual exponent to achieve finer-grained scaling. This reduces precision loss while retaining the advantages of low-bitwidth FP. MX formats have demonstrated high accuracy in workloads such as Convolutional Neural Networks (CNNs), LLMs, and Vision Transformers (ViTs) [2], leading to their adoption in a standard specification proposal developed by the Open Compute Project (OCP) [5].

This work is supported in part by the NeuroSoC project funded under Horizon Europe Grant Agreement n° 101070634.

Low-bitwidth formats significantly reduce memory and bandwidth needs by representing model parameters with fewer bits. However, fully realizing their efficiency gains requires native hardware support for scaled dot products, which dominate execution time in deep learning models. In large-scale data centers, reducing memory footprint and bandwidth is often prioritized, even at the cost of decompression overhead. In contrast, embedded Artificial Intelligence (AI) systems face strict power and compute constraints, where the overhead of format conversion can outweigh its benefits. This necessitates direct execution on MX formats to maximize efficiency.

NVIDIA's Blackwell Tensor Cores introduce micro-tensor scaling, incorporating community-defined MX formats [6]. Similarly, AMD's Versal AI Edge Series Gen 2 employs MX6 and MX9 formats in its AI engines [7]. In the RISC-V ecosystem, Tenstorrent's Tensix cores support BFP formats [8], but unlike MX, they rely solely on shared exponents per block, lacking the finer-grained scaling of MX (except for MXINT8). All these solutions are proprietary and tightly coupled to specialized tensor processing units. In contrast, our approach enables flexible and efficient execution of MX computations on RISC-V cores without dedicated tensor accelerators.

Existing RISC-V architectures lack native support for efficient execution of mixed-precision MX dot products due to the need for concurrent access to multiple types of data—scales, vector elements, and accumulators. Handling these dependencies in software requires frequent format conversions and explicit scaling operations, causing significant latency and energy overhead. To address these limitations, we propose `MXDOTP`, a dedicated ISA extension that natively integrates scaling within a dot product-accumulate operation. By eliminating redundant data movement and format conversions, `MXDOTP` enables high-throughput, energy-efficient execution of MX matrix multiplications. The contributions of this paper are:

- We design a dot product-accumulate unit to support MXFP8 (E5M2, E4M3) data formats with 8-bit scales and an FP32 accumulator to minimize overflows and precision losses.
- We introduce `MXDOTP`, a RISC-V ISA extension that fuses scaling and dot product into a single four-operand instruction. To sustain four operand fetches per cycle, we exploit Stream Semantic Registers (SSRs) in the Snitch core [9] to supply block scales without modifying the register file, which is typically limited to three read ports.
- We integrate `MXDOTP` into an 8-core Snitch cluster, reaching up to 80 % utilization. Implemented in 12 nm FinFET, it incurs only 11 % core-level and 5.1 % cluster-level area, and 1.9 % power overhead when idle.

- We demonstrate a 25 speedup and 12.5 improvement in energy efficiency over software-based MX matrix multiplication on the baseline cluster.

II. Background

A. Microscaling (MX) Formats

MX formats, proposed by OCP as a standard, are low-bitwidth data formats optimized for AI, co-developed by major industry players such as AMD, Arm, Intel, Meta, Microsoft, NVIDIA, and Qualcomm. These formats aim to enhance computational efficiency and memory footprint while maintaining accuracy in deep learning models. Unlike traditional per-tensor scaling or BFP, which share a single exponent across large blocks, MX offers finer-grained scaling by allowing individual elements or smaller blocks to retain independent scales. This reduces numerical errors while preserving memory efficiency. MX serves as a drop-in replacement for FP32 in inference and enables sub-8-bit training for large generative models with negligible accuracy loss [2].

An MX-compliant format consists of three components: *a scale factor*, *private elements*, and *a block size*. The specification defines four concrete formats: MXFP8, MXFP6, MXFP4, and MXINT8. Each block of k elements shares a scale factor, while elements are encoded in FP8 (E5M2/E4M3), FP6 (E3M2/E2M3), FP4 (E2M1), or INT8. The scale is encoded using an E8M0 format with a fixed block size of 32.

The MX specification defines the dot product operation as:

$$C = Dot(A, B) = X^A X^B \sum_{i=1}^{k} (P_i^A \times P_i^B) \tag{1}$$

where X^A and X^B are scale factors, and P_i^A and P_i^B are private elements of vectors A and B. The internal precision of the dot product and the order of operations are implementation-defined. The general dot product for two MX-compliant vectors is defined as:

$$C = DotGeneral(A, B) = \sum_{j=1}^{n} Dot(A_j, B_j) \tag{2}$$

where A and B consist of n MX blocks of size k, and result C *should* be an FP32 number, as recommended by the MX specification.

In this work, we focus on the MXFP8 general dot product to evaluate the efficiency of MX formats in low-bitwidth arithmetic operations. Specifically, we utilize the two MXFP8 formats, E5M2 and E4M3, to analyze their effects on power, performance, and area, as their FP8 counterparts are widely adopted in deep learning and supported by existing FPUs.

B. Snitch Cluster

The Snitch cluster is a high-performance, energy-efficient RISC-V multi-core architecture optimized for FP workloads [10]. It features eight RISC-V RV32IMAFD cores, each equipped with a 64-bit Floating Point Unit (FPU) supporting formats from FP64 to FP8. A 128 KiB shared L1 Scratchpad Memory (SPM) of 32 banks is interconnected via a single-cycle logarithmic interconnect, providing high-bandwidth, low-latency access. A dedicated DMA control core handles memory transfers between the SPM and external memory, reaching 512-bit peak bandwidth for data and instruction access through a 512-bit crossbar, while a separate 64-bit crossbar manages peripheral communication.

The cluster has two key ISA extensions. The `FREP` (Floating-point Repetition) extension [10] executes a predefined sequence of FP instructions, removing the overhead of branching in tight arithmetic loops. The `SSR` (Stream Semantic Register) extension [9] eliminates explicit load/store instructions by enabling hardware-managed memory streams. Instead of issuing individual memory instructions, SSRs autonomously fetch and store data with configurable affine access patterns, reducing address calculation overhead and instruction count. Each core has three SSRs, programmable with a base address, stride, and loop bounds across up to four dimensions. After configuration, the SSR logic manages memory transfers directly into the core registers, while the core focuses on computation. This improves utilization and overall efficiency.

III. Architecture

A. MXDOTP Datapath and Integration into FPU

To efficiently implement the general dot product operation for the MXFP8 format, we adopt the principle of early accumulation introduced in [3]. A simplified block diagram of the `MXDOTP` datapath is shown in Figure 1a. To accommodate both FP8 formats, i.e. E5M2 and E4M3, we utilize an FP9 (E5M3) intermediate format. Instead of converting the sum of products to FP32 and adding it to the FP32 accumulator with a dedicated FP32 adder, early accumulation directly adds the shifted accumulator to the sum of products in fixed-point before the final conversion to FP32. The sum of multiplied elements and the accumulator is represented using a 95-bit fixed-point format with an anchor at 34, ensuring it can accommodate the full range of the sum of eight products along with the shifted accumulator, including sign and rounding bits. Since the MX specification leaves internal precision as implementation-defined, we conservatively select the minimum bitwidth required to guarantee an exact result. We implement roundTiesToEven (RNE), the most commonly used rounding mode in AI workloads, ensuring compliance with the specification, which leaves other modes optional.

The `MXDOTP` datapath is integrated into the FPU of the Snitch cluster as an additional operation group[1]. Since the FPU features a 64-bit data interface to support double-precision FP, the `MXDOTP` datapath efficiently computes the dot product between vectors of eight elements with a throughput of one result per cycle, while its latency depends on the parametric number of pipeline stages.

B. Instruction and Integration into Snitch

We integrate the FPU with `MXDOTP` into the Snitch core and extend the ISA with the `MXDOTP` instruction. The instruction operates on five inputs and produces an accumulated output, as detailed in Table I. Unlike a traditional three-operand Fused Multiply-Add (FMA) instruction, `MXDOTP` requires block scales as an additional input. However, the Snitch core and its FPU are designed to handle up to three input operands, and the floating-point register file (FP RF) has three read ports and one write port, compliant with the RISC-V ISA. While adding another read port is a possible solution, it would

[1]Available at: https://github.com/pulp-platform/cvfpu/tree/feature/mxdotp

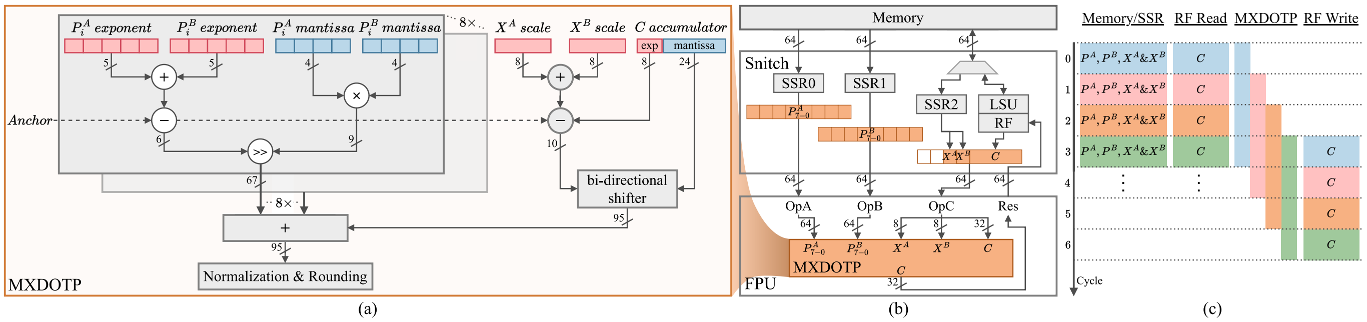


Fig. 1: (a) Simplified `MXDOTP` datapath (sign bits omitted). (b) Integration of `MXDOTP` into the Snitch core. (c) Execution of `MXDOTP` over multiple cycles with three pipeline stages. SSRs enable efficient operand streaming, ensuring full utilization without stalls.

TABLE I: Operands of the `MXDOTP` instruction.

Symbol	Operand	Role	Data Type
P^A / P^B	Elements of A / B	Input	8 × FP8
X^A / X^B	Scale factor of A / B	Input	INT8
C	Accumulator	Input/Output	FP32

TABLE II: Encoding format of the `MXDOTP` instruction.

Bits	31-27	26-25	24-20	19-15	14-12	11-7	6-0
Field	`rs3`	`sl`	`rs2`	`rs1`		`rd`	`opcode`
`MXDOTP`	$X^A\&X^B$	*Select*	P^B	P^A		C	1110111

introduce additional area overhead (around 12 % to the FP RF). Instead, we leverage the SSR extension of Snitch to stream scales without increasing area while also eliminating explicit load instructions for scales.

Figure 1b and c illustrate how operands are provided to the FPU and the `MXDOTP` datapath. To efficiently manage operand fetching while adhering to register file constraints, we utilize three SSRs. Two SSRs stream the A and B vectors directly into the FPU via its two 64-bit input interfaces, and the third SSR streams both scale factors for A and B. Since the scales require an additional operand beyond the standard three-operand FPU design, we merge the block scales with the FP32 accumulator and transfer them via the third 64-bit input interface of the FPU. The final FP32 accumulated result is then written back to the FP RF. The only hardware modification required in the Snitch core for `MXDOTP` is the merging of scales with the accumulator.

A detailed representation of the encoding is shown in Table II for the instruction `mxdotp rd, rs1, rs2, rs3, sl`. Since the register file only has three ports, this instruction requires at least one operand to be sourced via an SSR. Bits 26-25 select the appropriate scales from the register `rs3`, as each 64-bit register stores four sets of scales. To distinguish between E5M2 and E4M3 FP8 formats, we introduce a dedicated Control Status Register (CSR), which allows configuring the format prior to computation.

IV. Results

A. Implementation Setup and Area Analysis

We synthesize, place, and route the `MXDOTP`-extended Snitch cluster in GlobalFoundries' 12 nm FinFET technology with a target frequency of 0.95 GHz under worst-case conditions (SS/0.72 V/125 °C) using Synopsys Fusion Compiler. To sustain frequency, the `MXDOTP` unit is implemented with three levels of pipeline registers. The cluster achieved 1.09 GHz under typical conditions (TT/0.8 V/25 °C) without introducing any new critical paths. We use Synopsys' PrimeTime to estimate power consumption under typical conditions, with switching activities extracted from a post-layout gate-level simulation at 1 GHz. We average power consumption over ten samples extracted from DeiT-Tiny [11], quantized to MXFP8 using Microsoft's MX PyTorch Emulation Library[2].

The total area of the cluster with `MXDOTP`-extended cores is 4.89 MGE, representing a minimal 5.1 % increase over the baseline Snitch cluster. The Snitch core complex consists of the core itself, its private instruction cache, SSRs, and the FP subsystem, which includes the FPU, FREP logic, and FP RF. The area breakdown of these components is shown in Figure 3. `MXDOTP` accounts for 17 % of the FPU area and contributes 9.5 % to the core complex. Moreover, it adds only 1.9 % power overhead to the cluster when idle.

B. Software Benchmark Setup

To evaluate performance and energy efficiency, we compare three matrix multiplication (MM) kernels: (1) *FP32*, a baseline kernel using standard 32-bit FP multiply-accumulate (MAC) operations, (2) *FP8-to-FP32*, a software-based MX implementation that converts FP8 inputs to FP32, performs MAC operations, and applies block scale multiplications post-accumulation, and (3) *MXFP8*, leveraging the proposed `MXDOTP` ISA extension to directly compute MX dot products in hardware, the block size remains configurable in software.

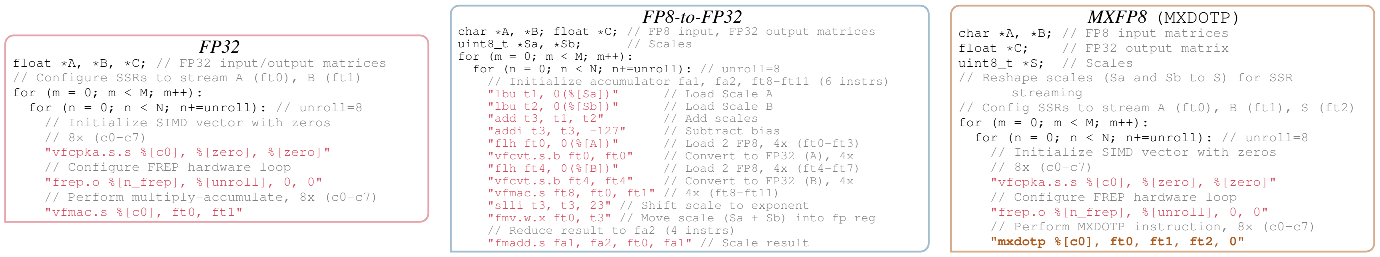
As shown in Figure 2, the *FP32* kernel operates on FP32 inputs and accumulators using a 2-way SIMD MAC, while the *FP8-to-FP32* kernel introduces significant overhead due to FP8 to FP32 format conversions and explicit scale operations. In contrast, the *MXFP8* kernel eliminates these overheads by executing the entire scaled dot product in a single instruction, efficiently integrating the scaling within the hardware pipeline.

C. Performance and Energy Efficiency

As shown in Figure 4, `MXDOTP` achieves a speedup of 3.1 to 3.4 over *FP32* and 20.9 to 25.0 over the *FP8-to-FP32* software baseline; it also improves energy efficiency by 3.0 to 3.2 and 10.4 to 12.5 , respectively. The *MXFP8* kernel achieves up to 102 GFLOPS and 356 GFLOPS/W, reaching 79.7 % of the ideal throughput, factoring in SSR and FREP configuration and loop overheads, accumulator initializations, and stores for final results.

While MX reduces memory footprint even without a dedicated dot product unit, our results show that the full performance and energy benefits of MX cannot be realized without hardware support for the general dot product operator. Compared to *FP32*, `MXDOTP` achieves higher throughput by leveraging lower bitwidth, allowing more data to be fetched and processed per cycle. Although *FP8-to-FP32* operates

[2]https://github.com/microsoft/microxcaling



FP32

```
float *A, *B, *C; // FP32 input/output matrices
// Configure SSRs to stream A (ft0), B (ft1)
for (m = 0; m < M; m++):
  for (n = 0; n < N; n+=unroll): // unroll=8
    // Initialize SIMD vector with zeros
    // 8x (c0-c7)
    "vfcpka.s.s %[c0], %[zero], %[zero]"
    // Configure FREP hardware loop
    "frep.o %[n_frep], %[unroll], 0, 0"
    // Perform multiply-accumulate, 8x (c0-c7)
    "vfmac.s %[c0], ft0, ft1"
```

FP8-to-FP32

```
char *A, *B; float *C; // FP8 input, FP32 output matrices
uint8_t *Sa, *Sb;      // Scales
for (m = 0; m < M; m++):
  for (n = 0; n < N; n+=unroll): // unroll=8
    // Initialize accumulator fa1, fa2, ft8-ft11 (6 instrs)
    "lbu t1, 0(%[Sa])"       // Load Scale A
    "lbu t2, 0(%[Sb])"       // Load Scale B
    "add t3, t1, t2"         // Add scales
    "addi t3, t3, -127"      // Subtract bias
    "flh ft0, 0(%[A])"       // Load 2 FP8, 4x (ft0-ft3)
    "vfcvt.s.b ft0, ft0"     // Convert to FP32 (A), 4x
    "flh ft4, 0(%[B])"       // Load 2 FP8, 4x (ft4-ft7)
    "vfcvt.s.b ft4, ft4"     // Convert to FP32 (B), 4x
    "vfmac.s ft8, ft0, ft1" // 4x (ft8-ft11)
    "slli t3, t3, 23" // Shift scale to exponent
    "fmv.w.x ft0, t3" // Move scale (Sa + Sb) into fp reg
    // Reduce result to fa2 (4 instrs)
    "fmadd.s fa1, fa2, ft0, fa1" // Scale result
```

MXFP8 (MXDOTP)

```
char *A, *B; // FP8 input matrices
float *C;    // FP32 output matrix
uint8_t *S;  // Scales
// Reshape scales (Sa and Sb to S) for SSR
     streaming
// Config SSRs to stream A (ft0), B (ft1), S (ft2)
for (m = 0; m < M; m++):
  for (n = 0; n < N; n+=unroll): // unroll=8
    // Initialize SIMD vector with zeros
    // 8x (c0-c7)
    "vfcpka.s.s %[c0], %[zero], %[zero]"
    // Configure FREP hardware loop
    "frep.o %[n_frep], %[unroll], 0, 0"
    // Perform MXDOTP instruction, 8x (c0-c7)
    "mxdotp %[c0], ft0, ft1, ft2, 0"
```

Fig. 2: Comparison of *FP32*, *FP8-to-FP32*, and *MXFP8* MM kernels. M and N represent the output matrix's row and column dimensions, respectively.

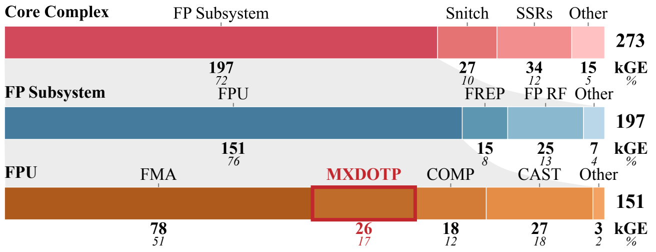


Fig. 3: Area breakdown of the `MXDOTP`-extended Snitch core.

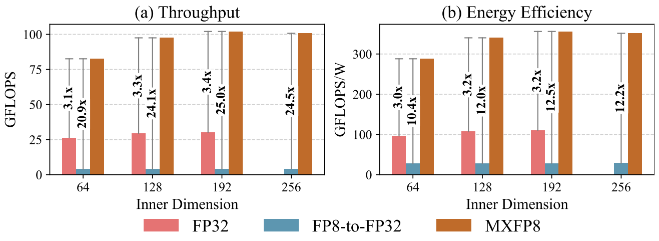


Fig. 4: (a) Throughput and (b) Energy Efficiency of *FP32*, *FP8-to-FP32*, and *MXFP8* MM kernels for varying inner dimensions, with rows and columns fixed at 64. *FP32* does not fit into L1 with inner dimension of 256.

at the same low bitwidth, `MXDOTP` improves efficiency by fusing scaling and format conversions into a single operation. Moreover, frequent FP8 to FP32 format conversions and scale operations in *FP8-to-FP32* introduce significant computational overhead, making it less energy-efficient than even the *FP32* baseline. These findings highlight that dedicated MX dot product hardware is essential to fully unlock the performance and energy efficiency benefits of MX formats in AI workloads. Implemented as an ISA extension, we provide full software flexibility, enabling seamless mixed-precision execution.

D. Comparison to State-of-the-Art

We compare our work against prior FP8 dot product units and compute clusters in Table III. We first analyze standalone dot product units, corresponding to the first four rows of the table. ExSdotp [4] performs FP8 dot products with FP16 accumulation, while Desrentes et al. [12] support exact accumulation and FP32 conversion. However, both lack scaling support, whereas Lutz et al. [3] introduce a single 7-bit scale. Our design advances beyond these approaches by incorporating two independent 8-bit scales, fully enabling MX general dot product. With an energy efficiency of 2 TFLOPS/W, it outperforms ExSdotp. The higher efficiency of [12] is primarily due to three factors: (1) our design includes additional scaling operations, which are not counted as OPs, (2) [12] reports synthesis estimates at different frequencies, while we provide post-layout area and power results, and (3) we report only E5M2 results for [12], as their design does not support a combined unit for E5M2 and E4M3, unlike `MXDOTP`.

At the cluster level (last two rows), our implementation achieves competitive throughput and efficiency compared to MiniFloat-NN [4], which integrates ExSdotp as an ISA extension. However, MiniFloat-NN does not include a dedicated mechanism for handling block scaling, requiring an additional software stage. In contrast, our approach natively integrates block scaling within the dot product operation while achieving the same frequency-normalized throughput as MiniFloat-NN.

TABLE III: Comparison of FP8 dot product units (first four rows) and compute clusters (last two rows).

Design	**Tech.** nm	**Voltage** V	**Freq.** GHz	**Area** mm^2	**Scale Support**	**Accum. Format**	**Throughput**[*] GFLOPS	**Efficiency**[*] GFLOPS/W
ExSdotp[†] [4]	12	0.8	1.26	5.13×10^{-3}	✗	FP16	20.2	1631
Desrentes *et al.*[‡] [12]	16	—	1.25	9.81×10^{-3}	✗	FP32	80.0	11 300
Lutz *et al.* [3]	5	—	3.6	6.74×10^{-4}	✓ (1 × 7b)	FP32	28.8	—
This work	12	0.8	1.09	3.15×10^{-3}	✓ (2 × 8b)	FP32	17.4	2035
MiniFloat-NN [4]	12	0.8	1.26	0.52	✗	FP16	128	575
This work	12	0.8	1.00	0.59	✓ (2 × 8b)	FP32	102	356

[*] 1 FLOP = 1 FP multiplication or 1 FP addition.
[†] Reported for FP8 with FP16 accumulation, though the unit also supports FP16 with FP32 accumulation.
[‡] We report the combined dot product (synthesized at 250 MHz assuming it can be pipelined with four stages) and FP32 compression (synthesized at 1.25 GHz) units for E5M2 format.

V. Conclusion

We presented `MXDOTP`, a RISC-V ISA extension enabling efficient execution of Microscaling (MX) dot products. By integrating a dedicated dot product-accumulate unit into the Snitch core and leveraging SSRs to handle scaling factors, we enable hardware acceleration of MX-based matrix multiplications. Implemented in a 12 nm FinFET technology, our design achieves 102 GFLOPS and 356 GFLOPS/W, 25 speedup and 12.5 improvement in energy efficiency over a software-based MX matrix multiplication kernel. These results highlight the necessity of dedicated MX ISA and micro-architectural support for aggressively block-quantized AI workloads.

References

[1] B. D. Rouhani *et al.*, "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point," in *NIPS*, 2020.

[2] B. D. Rouhani *et al.*, "Microscaling Data Formats for Deep Learning," 2023. [Online]. Available: http://arxiv.org/abs/2310.10537

[3] D. R. Lutz *et al.*, "Fused FP8 4-Way Dot Product With Scaling and FP32 Accumulation," in *ARITH*, 2024.

[4] L. Bertaccini *et al.*, "MiniFloats on RISC-V Cores: ISA Extensions With Mixed-Precision Short Dot Products," *IEEE Trans. on Emerging Topics in Computing*, vol. 12, no. 4, pp. 1040–1055, 2024.

[5] B. D. Rouhani *et al.*, "OCP Microscaling Formats (MX) Specification," 2023. [Online]. Available: https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf

[6] "NVIDIA Tensor Cores: Versatility for HPC & AI." [Online]. Available: https://www.nvidia.com/en-us/data-center/tensor-cores/

[7] "AMD Versal™ AI Edge Series Gen 2 Product Brief." [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/products/adaptive-socs-and-fpgas/versal/versal-ai-edge-gen2-product-brief.pdf

[8] "Data Formats and Math Fidelity — TT Buda documentation." [Online]. Available: https://docs.tenstorrent.com/pybuda/latest/dataformats.html

[9] F. Schuiki *et al.*, "Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 212–227, 2021.

[10] F. Zaruba *et al.*, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," *IEEE Trans. Comput.*, vol. 70, no. 11, pp. 1845–1860, 2021.

[11] H. Touvron *et al.*, "Training data-efficient image transformers & distillation through attention," in *ICML*, 2021.

[12] O. Desrentes *et al.*, "Exact Dot Product Accumulate Operators for 8-bit Floating-Point Deep Learning," in *DSD*, 2023.