Considerations on Supporting Open Compute Project Microscaling Formats (MX) in RISC-V

José Moreira
*C*hair, RISC-V Vector SIG

Resources

- Open Compute Project • OCP Microscaling Formats (MX) Specification
- NVIDIA Blackwell Architecture Technical Brief
- CUDA 12.6 Update 2 Release Notes
- NVIDIA cuDNN Library
- MX Pytorch Emulation Library

Note: I added some references to NVIDIA material above, because NVIDIA's Blackwell is currently the only commercially available system (that I know of) that supports MX format natively. Nevertheless, I have found very little content in their existing documentation. Most information I have is from the OCP format specification.

Microscaling formats (MX) specification

OPEN Compute Project

OCP Microscaling Formats (MX) Specification

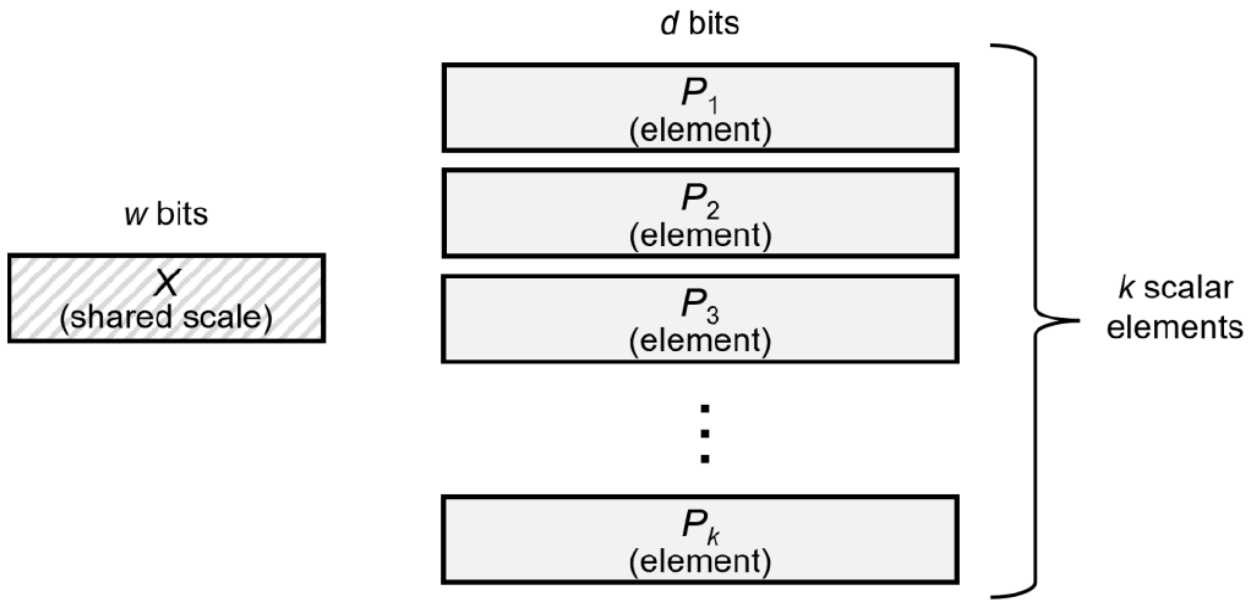
Version 1.0

Author: Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Jasmine Klar, Eric Chung, Yuan Yu, Microsoft
Author: Michael Schulte, Ralph Wittig, AMD
Author: Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Arm
Author: Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Intel
Author: Summer Deng, Maxim Naumov, Meta
Author: Paulius Micikevicius, Michael Siu, NVIDIA
Author: Colin Verrilli, Qualcomm

Microscaling

An MX-compliant format is characterized by three components:

- Scale (X) data type / encoding
- Private elements (P_i) data type / encoding
- Scaling block size (k)



All k elements (P_i) have the same data type and, therefore, the same bit-width. The scale factor X is shared across all k elements. The data types of the elements and scale are chosen independently. In this sense, MX can be seen as a mechanism to build a vector data type from scalar data types.

Concrete MX-compliant formats

A concrete MX-compliant format consists of a specific block size k and data types of X and P_i. The following concrete MX-compliant formats are part of this specification. The element and scale data types listed in this table are described in the next section.

Format Name	Element Data Type	Element Bits (d)	Scaling Block Size (k)	Scale Data Type	Scale Bits (w)
MXFP8	FP8 (E5M2)	8	32	E8M0	8
	FP8 (E4M3)				
MXFP6	FP6 (E3M2)	6	32	E8M0	8
	FP6 (E2M3)				
MXFP4	FP4 (E2M1)	4	32	E8M0	8
MXINT8	INT8	8	32	E8M0	8

Table 1. Format names and parameters of concrete MX-compliant formats.

Basic operation: Dot-product of two MX vectors

- Let $A = \left\{X^{(A)}, \left[P_i^{(A)}\right]_{i=1}^{k}\right\}$, and $B = \left\{X^{(B)}, \left[P_i^{(B)}\right]_{i=1}^{k}\right\}$
- Compute $C = \mathrm{Dot}(A, B) = X^{(A)}X^{(B)}\sum_{i=1}^{k}\left(P_i^{(A)} \times P_i^{(B)}\right)$
- If we define
  - $P^{(A)} = \left[P_i^{(A)}\right]_{i=1}^{k}$
  - $P^{(B)} = \left[P_i^{(B)}\right]_{i=1}^{k}$
  - $\left\langle P^{(A)}, P^{(B)}\right\rangle = \sum_{i=1}^{k}\left(P_i^{(A)} \times P_i^{(B)}\right)$
- Then $\mathrm{Dot}(A, B) = X^{(A)}X^{(B)}\left\langle P^{(A)}, P^{(B)}\right\rangle$
- Possible approach:
  - The P blocks are the “elements” stored in vector registers
  - $\left\langle P^{(A)}, P^{(B)}\right\rangle$ is the basic arithmetic operation

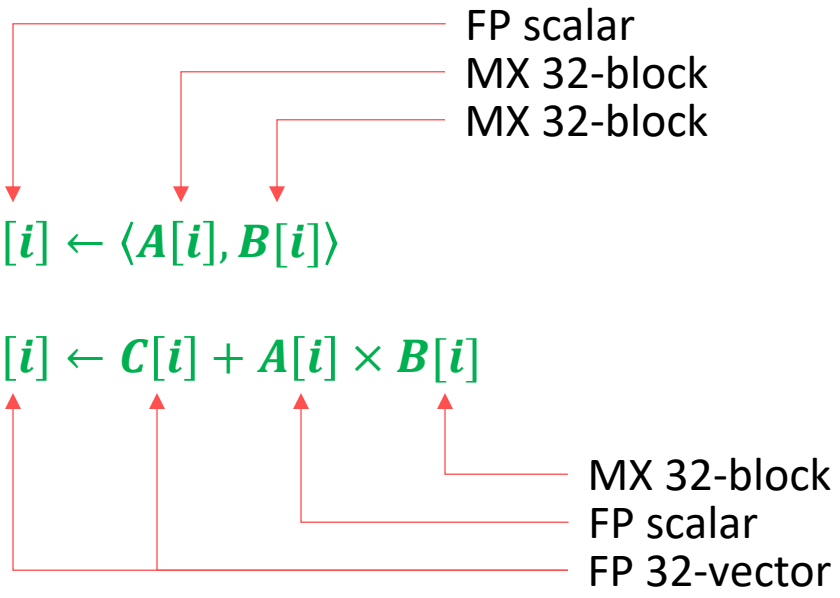
Concrete MX formats ($k = 32$)

Scalar element width for different MX formats

Format(P)	SEW(P)
FP8	256
FP6	192 (→ 256 ?)
FP4	128
INT8	256

Minimum data type for exact result (4-8× narrowing)

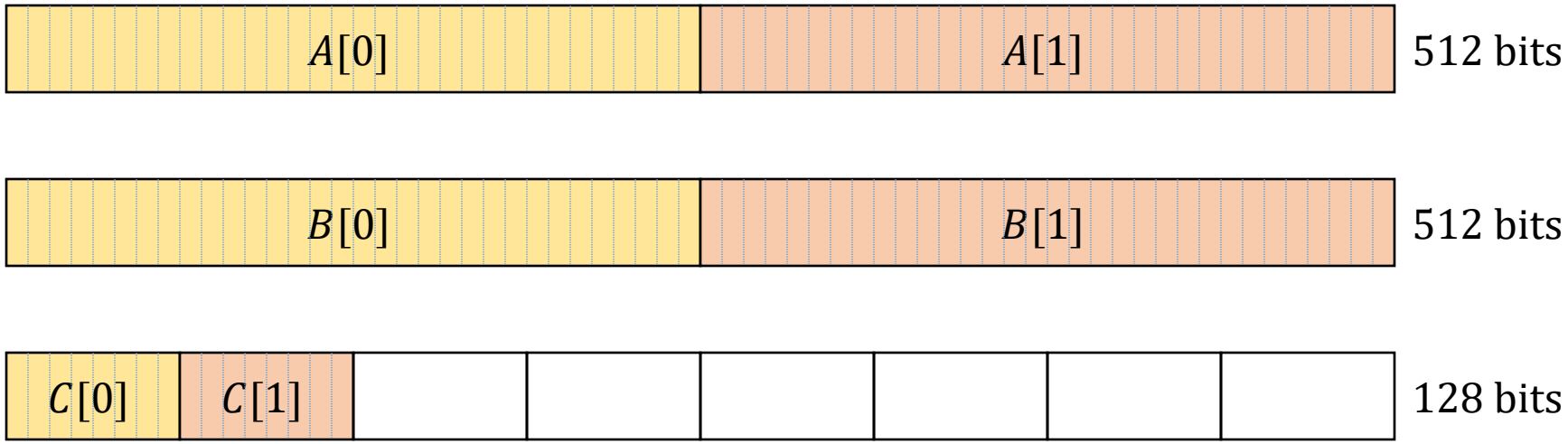
$\langle P^{(A)}, P^{(B)} \rangle$	FP8	FP6	FP6	INT8
FP8	FP64	FP64	FP64	FP64
FP6	FP64	FP32	FP32	FP32
FP4	FP64	FP32	FP32	FP32
INT8	FP64	FP32	FP32	FP32



$\texttt{vmxdot}\langle \mathbf{fp}(C), \mathbf{mx}(A), \mathbf{mx}(B) \rangle\texttt{.vv}\ C, A, B$ # $C[i] \leftarrow \langle A[i], B[i] \rangle$

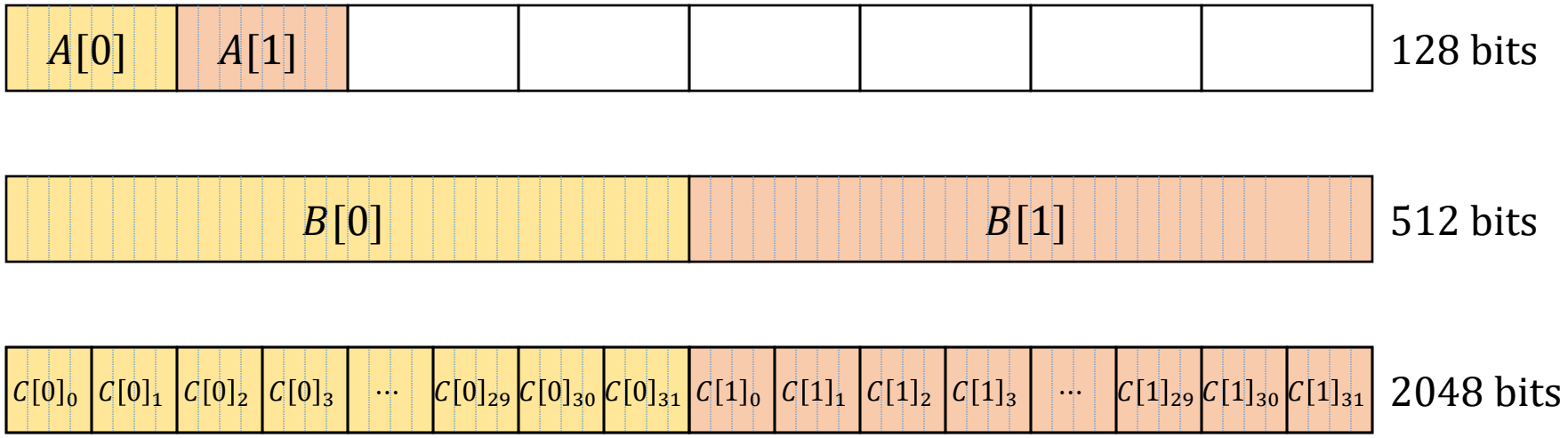
$\texttt{vmxaxpy}\langle \mathbf{fp}(C), \mathbf{fp}(A), \mathbf{mx}(B) \rangle\texttt{.vv}\ C, A, B$ # $C[i] \leftarrow C[i] + A[i] \times B[i]$

vmxdot⟨FP64, FP8, FP8⟩.vv C, A, B



- *Example*: VLEN = 512
- $C[0] \leftarrow \langle A[0], B[0] \rangle$
- $C[1] \leftarrow \langle A[1], B[1] \rangle$
- EMUL(A)=1, EMUL(B)=1, EMUL(C)=¼ (the exact values depend on VLEN, the ratios do not)
- The **vmxdot** is the building block for pure MX linear algebra libraries
- The $X^{(A)}X^{(B)}$ scale has to be applied separately, with existing vector instructions
- In general, accumulation of C can only be performed after scaling

vmxaxpy⟨FP32, FP64, FP8⟩.vv C, A, B



- *Example*: VLEN = 512
- $C[0] \leftarrow C[0] + A[0] \times B[0]$
- $C[1] \leftarrow C[1] + A[1] \times B[1]$
- EMUL(A)=¼, EMUL(B)=1, EMUL(C)=4 (the exact values depend on VLEN, the ratios do not)
- The **vmxaxpy** is the building block for mixed FP-and-MX linear algebra libraries
- The A elements include both the $X^{(B)}$ scale and the multiplier from the FP matrix
- Accumulation of C can be generally performed