Understanding RISC-V® Extensions for AI

Krste Asanovic
Chief Architect, RISC-V International

AI Forum
RISC-V Summit
Shanghai, China
July 2025

AI Computing

AI is pervasive, both as:

- Horizontal technology
  - Used everywhere on every device
  - Single device might run dozens of different models
- Vertical application
  - Dedicated racks of machines for training models and serving inferences

- Common attributes:
  - Large amounts of numeric computation on small specialized datatypes
  - Large memory footprint to hold model parameters
  - Large memory bandwidth to fetch parameters and communicate between layers

RISC-V and AI

- RISC-V supports a general computing model allowing a balance of **scalar**, **vector**, and **matrix** capabilities
  - Same cores can run non-AI portion of application as well as AI portion, simplifying software, lowering latency, and increasing utilization
  - Can scale to arbitrarily high performance with scale-up+scale-out multicores
  - New models can run reasonably well on RISC-V hardware designed for earlier models
  - Different balance of scalar/vector/matrix possible with different microarchitectures in current or future generations of RISC-V, without changing programming model

RISC-V Vector Extension (RVV)

- Vectors needed to support matrix acceleration, as many critical steps in AI algorithms cannot be performed as matrix multiplies
- Vector performance limits AI throughput, especially if matmul accelerated

- RVV designed to support many datatypes and mixed-precision operations efficiently through LMUL
  - E.g., widening multiply-add, narrowing converts
- RVV designed to scale to large vector lengths
  - Already, implementations up to 16Kb per vector register
- RVV LMUL provides way to gang vector registers
  - Provide larger operands to a vector instruction
- RVV designed as compiler target
  - Now supported in backends of major compiler frameworks

New RVV Datatypes for AI

- Original RVV 1.0 supports INT{8,16,32,64} and IEEE FP{16,32,64}
- Ratified Zvfbfmin supports vector BF16 conversions
- Ratified Zvbfwma adds vector widening BF16 muladd into FP32

- Proposal (Zvfbfa) for further support for BF16 vector arithmetic
- Proposal (Zvfofp8min) for vector OFP8 support through conversions
- Current discussion around use of new altfmt bit in vtype CSR to encode new datatypes in clean way without expanding instruction length or taking too many opcodes

- Expect considerable activity around new vector datatypes for AI, including discussions on MXFP8/6/4
  - We will need to consider expanding instruction space and/or vtype bits to encode

RISC-V Matrix Extensions

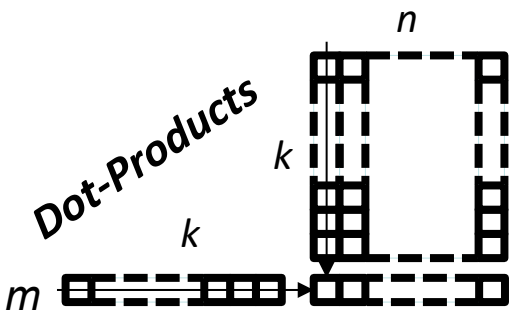
- Matrix multiply (matmul) instructions can provide large speedups on key compute in portions of AI applications
- Across universe of RISC-V implementations and application domains, different matmul microarchitectures make sense, and may result in different ISA choices (or no matmul at all)
- Only a relatively small number of matrix multiply routines exist, so software less affected by ISA choices

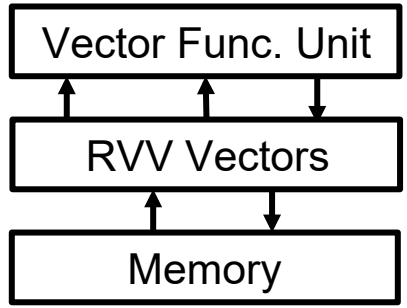
RISC-V Matrix Extensions, 1+3 Approaches

Element in existing RVV vector register

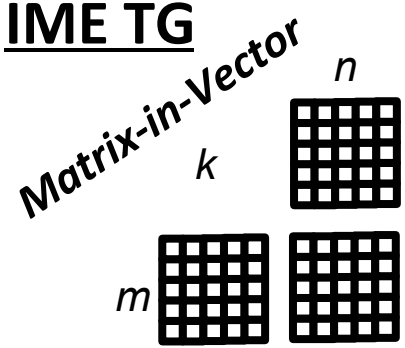
Element in added matrix registers

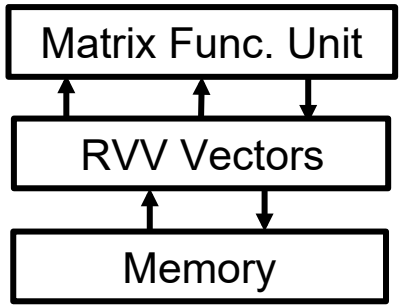
Zvbdot Fast-Track



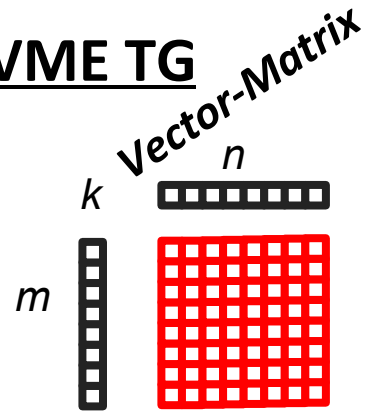


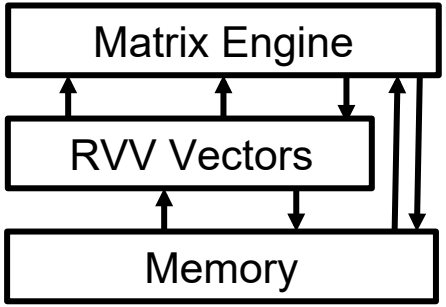
IME TG



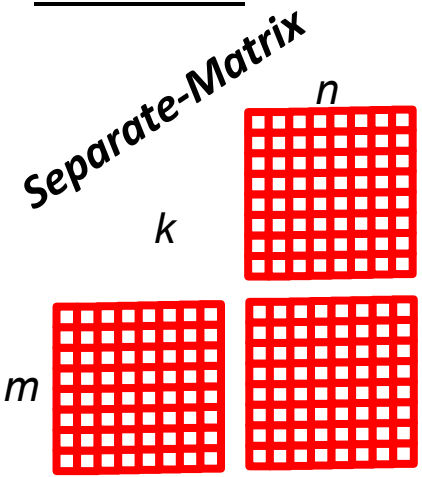


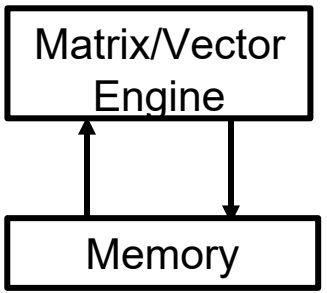
VME TG



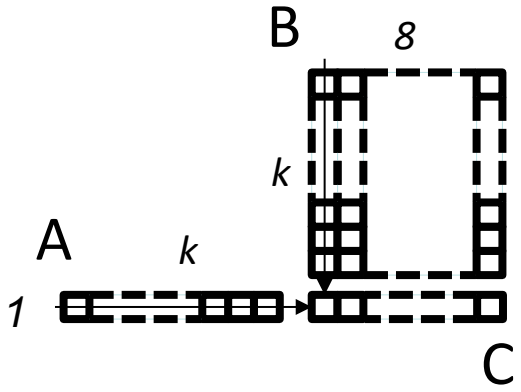


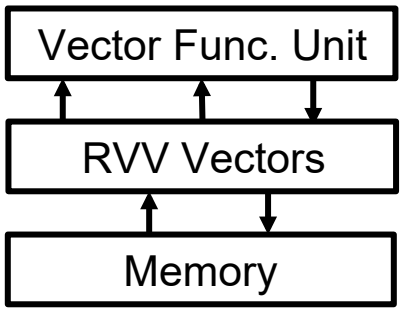
AME TG





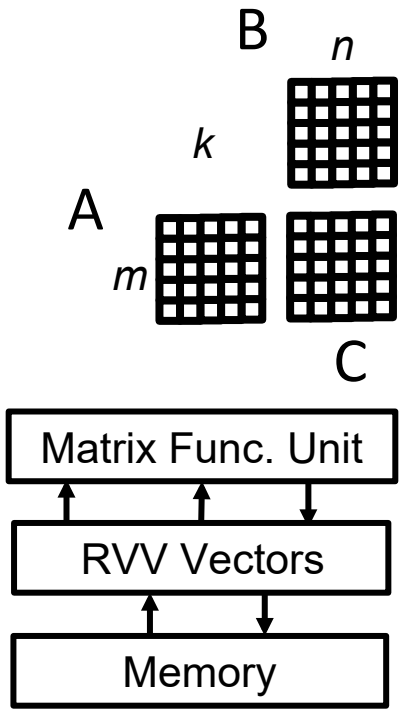
Vector Batch Dot-Products
(Zvbdot Fast-Track Extension)





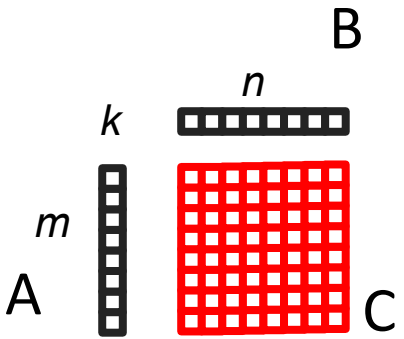
- No new state, uses existing vector registers
- Computes 8 dot-products, each of VLEN bits of input.
- One input from vector register A
- Eight inputs from vector register group B (each column of B in a single vector register)
- Eight outputs accumulated in vector register group C
  - A 3b immediate offset allows more of C to be addressed to hold accumulators (up to 64 results in C)
- E.g., VLEN=1024K, FP8 inputs, FP32 outputs
  - 1024/8 => k=128 elements per dot product
  - 8 dots*128 elements*2 ops/MACC = 2K FLOPS per instruction
  - Writes back 8*32b = 256b per instruction to C
- Good match for vector unit – more reads than writes
- Accelerates GEMV as well as GEMM

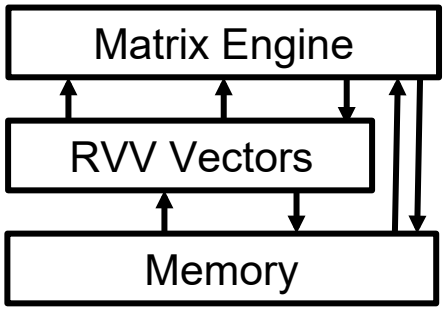
IME TG: Matrix-in-Vector Approach



- No new state, uses existing vector registers
- Treats vector registers as holding one or more matrix tiles
  - Different sub-proposals handle tiles differently
- Outer-product style matrix multiply
  - some treat A and B as vectors not tiles
- Greater arithmetic intensity with longer vector registers
- Write bandwidth requirements for result C matrix might push implementers to rename vector registers into matrix multiply unit, transparent to software
- Most sub-proposals assume new matrix tile load/store instructions

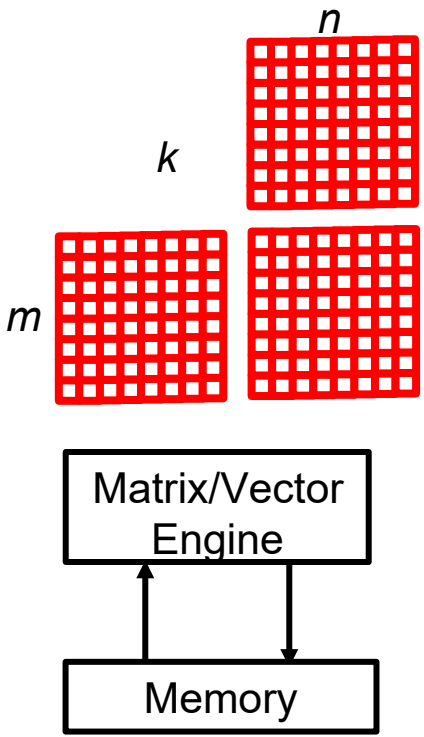
VME TG: Vector-Matrix Approach





- Adds new large matrix accumulator state to hold C
- C state might be divided into one or more tiles
- Uses RVV vectors to provide source operands A and B for matmul
- Outer-product style matrix multiply into C acc
  - Might have ”fat” k>1 support for narrower input types
- Support for C row/column move to vector registers and/or load/store to memory
- Matrix engine can provide very high throughput with C accumulator state located near arithmetic units

AME TG: Separate Matrix Engine



- Adds new large matrix state to hold A, B, C
- Does not require RVV vectors
  - If no RVV, have to define new vector operations operating on matrix state, large software effort
  - If RVV present, would like to use
- Requires separate matrix load/store path to memory

- Large design space, fewest constraints, theoretically allows the greatest performance
- No clear consensus on proposal yet

Alternative Matrix Instruction Extensions

- **Batch Dot-Products:** No additional matrix state. Small and effective extension at smaller vector lengths, also meets some DSP needs, but doesn't scale performance with larger vector lengths.
  - ***Fast-track, simple vector extension– likely to ratify first***
- **Matrix-in-Vector:** No additional matrix state. Allows larger throughputs than dot-products for longer vector lengths.
  - ***Large design space, will likely take some time to converge***
- **Vector-Matrix:** Add matrix state to processors that already have RVV. Maintains vector ISA and memory model.
  - ***New TG, but smaller design space than other matrix extensions, likely to converge soon***
- **Separate-Matrix:** Unconstrained matrix and vector design.
  - ***Largest design space and software effort, will take most time to converge.***