RISC-V SUMMIT EUROPE • BOLOGNA • 8-12 JUNE 2026

All the Scaling, No New State

Dr. Philipp Tomsich (VRULL GmbH)

Dr. Erich Focht (Openchip)

with **Dr. Jose Moreira** (IBM Research) and **Bing Yu** (Andes Technology)

THE PROBLEM

Every matrix ISA forces a binary fork

Every matrix ISA outlives its first microarchitecture — the same binary must run on silicon, and at vector widths, not yet designed. Hard-coding tile shape or adding dedicated state freezes those choices at spec time and forces a fork whenever the datapath changes. The design goal instead: encode intent (streaming depth, precision, granularity), and leave hardware free to choose how to deliver it.
One binary, the whole continuum — the market selects the implementation point.

Arm SME

Dedicated state, dedicated mode

- Dedicated ZA tile register file
  New architectural state the OS must save, restore and context switch
- A separate streaming mode, with explicit entry/exit the programmer model has to manage
- Tile geometry is tied to the implementation's SVL — wider hardware needs a different binary
- Microscaling bolted-on SME2 instructions, not part of original design

Intel AMX

Geometry frozen at design time

- Fixed 16×16 tiles
  Tile shape is hard-coded into the ISA, not chosen by software
- Eight dedicated TMM tile registers, plus a TILECFG state that must be configured and restored
- Cannot exploit wider datapaths without an ISA revision: no path to scale a binary forward
- FP8 microscaling added as a later extension (AMX-FP8)

The ecosystem fragments — per-platform tuning and multi-version liibraries at every architecture boundary.

Geometry is derived, not declared

Zvvm encodes that intent directly: no new state to freeze, no shape to hard-code.
Tile state reuses the vector register file, and tile shape is derived from VLEN, SEW, and the aspect-ratio field λ — never declared.
The same binary expresses streaming depth, precision, and granularity as intent; each implementation chooses the geometry that fits its datapath.
Nothing new is added, and nothing is frozen at spec time.

No fixed state

Matrix tiles are the standard vector register file. No dedicated tile file, no separate accumulator state. The matrix extension adds capability, not architectural state.

No fixed shape

There are no tile-shape instructions and no shape constants. The programmer never declares M, N, or K: we will never outgrow a frozen into the binary at spec time.

The tile fits the datapath

The dimensions are derived with λ shaping the aspect ratio. Each implementation's geometry falls out of its own VLEN. Hardware chooses the shape that fits, not the other way around.

So scaling comes for free. Next up: The algebra that explains how.

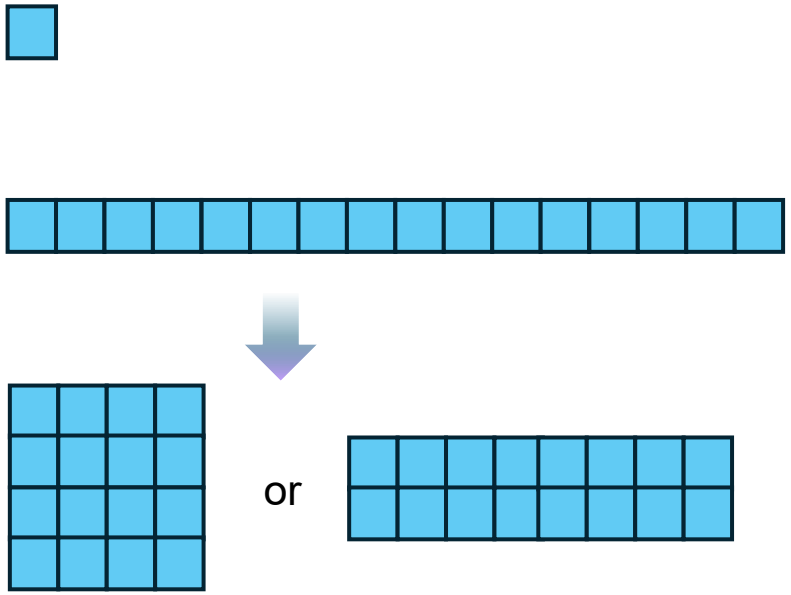
Scalars – 0-dimensional compute
(single values processed sequentially)

Vectors – 1-dimensional data parallelism
(RVV enables scalable SIMD operations)

Matrices – 2-dimensional data parallelism
(the natural abstraction for ML workloads)

Modern AI and some HPC workloads are fundamentally matrix computations.

- Vectors already express 1-D parallelism
- Next step: lift vector computation to 2-D matrix operations
- Same state, different meaning



Beyond Vectors

Scalars – 0-dimensional compute
(single values processed sequentially)

Vectors – 1-dimensional data parallelism
(RVV enables scalable SIMD operations)

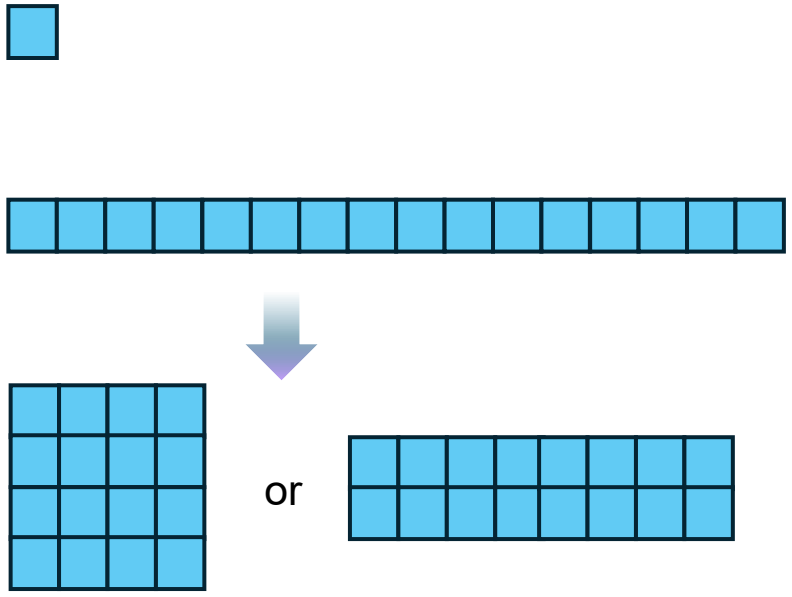
Matrices – 2-dimensional data parallelism
(the natural abstraction for ML workloads)

Modern AI and some HPC workloads are fundamentally matrix computations.

Vectors already express 1-D parallelism

Next step: lift vector computation to 2-D matrix operations

Same state, different meaning



Matrix Multiplication

$A \cdot B^T + C \rightarrow C$

A, B VRs are matrix tiles of dimension ***σ x λ***

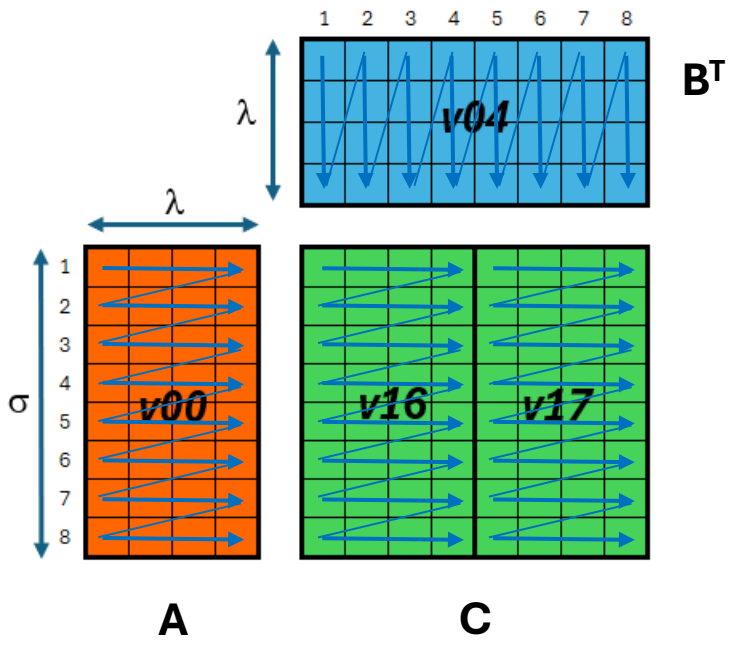
- **λ** must be one of 1, 2, 4, 8, 16, 32, 64
  - choice of implementor
  - is the **only geometry parameter**
- Any VLEN is supported (up to 64kbits)
- Elements of width SEW in a vector register
  $VL_{MAX} = VLEN/SEW$

Output C matrix tile:

- One or multiple VRs
- $MUL_C = VL_{MAX} / \lambda^2$
- At VL_{MAX} C is square! ***σxσ***
  - Maximizes compute intensity

VL = 32
λ = 4
σ = 8

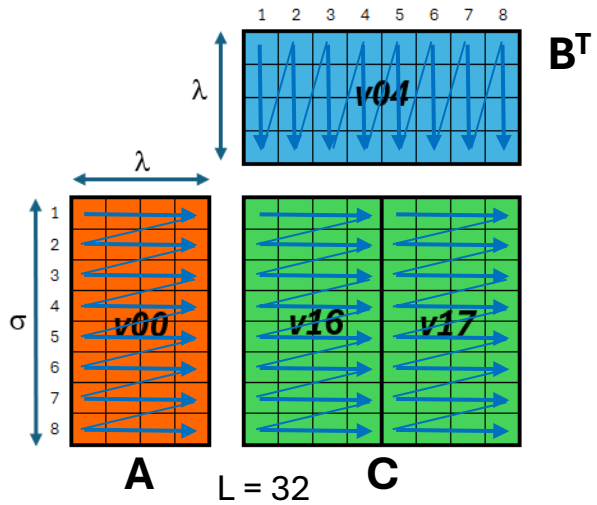
Example



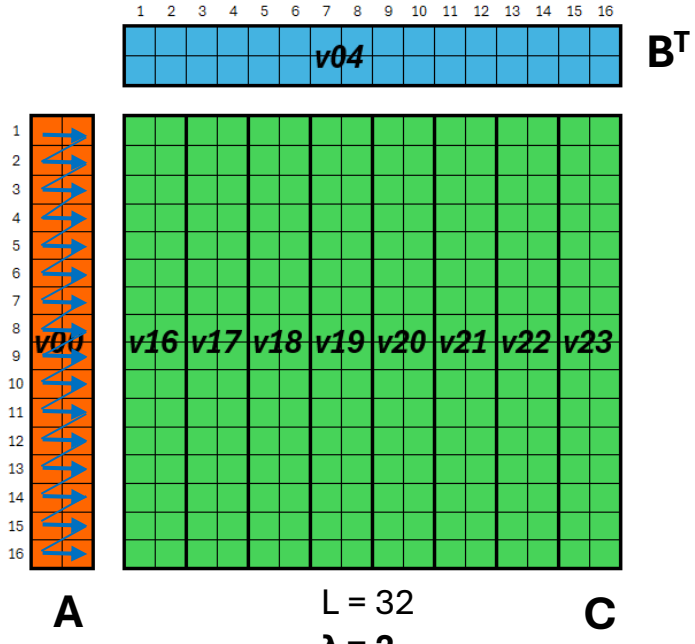
Multiply: ***vmmacc.vv vd,vs1,vs2***
Tile Load: ***vmtl vd,(base),ld***
Transposed Load: ***vmttl vd,(base),ld***

Compute Intensity

(C remains in place)
2*32 = 64 A, B el. loaded
8*8*4 = 256 MAC ops
➔ 4.0 MACs per load



(C remains in place)
2*32 = 64 A, B el. loaded
16*16*2 = 512 MAC ops
➔ 8.0 MACs per load

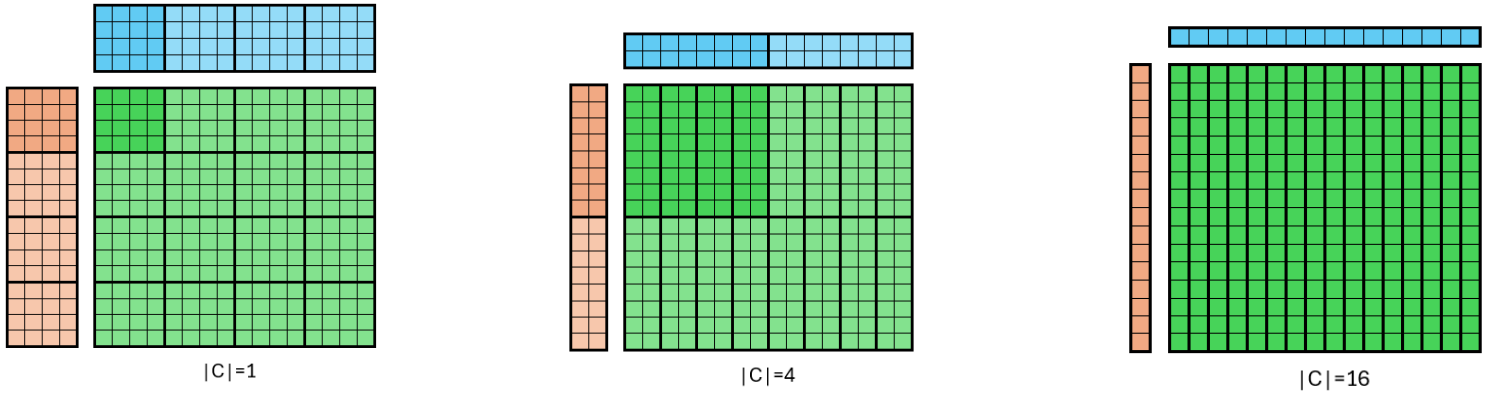


Implementation may alias C to VRs or use a hidden accumulator: the ISA is agnostic.

Compute Intensity vs. Multiply Instructions

With 16 VRs for C

When VLMAX is even power of 2



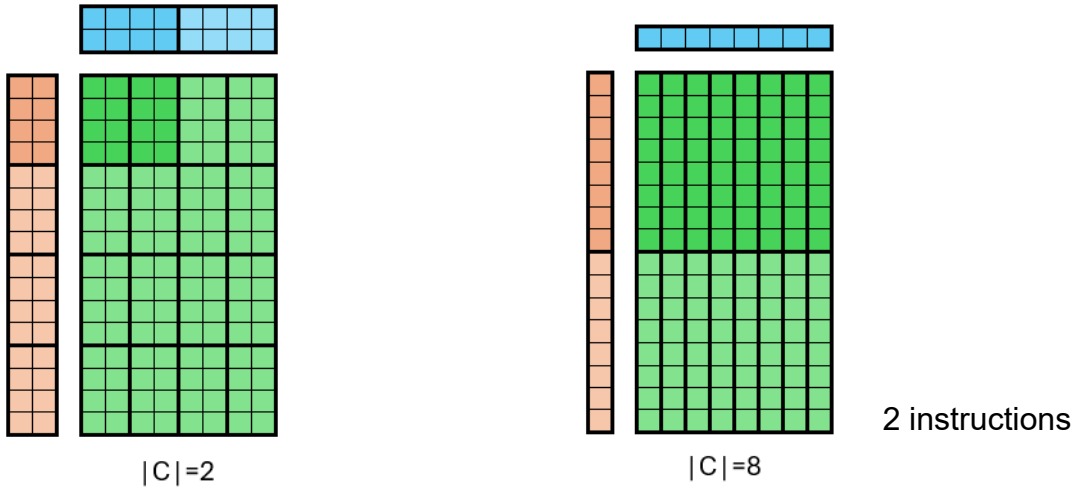
16 instructions

4 instructions

1 instruction

When VLMAX is odd power of 2

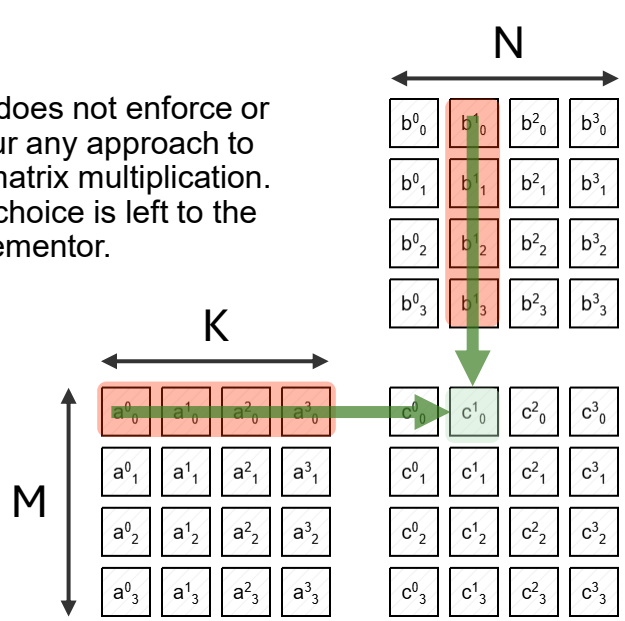
8 instructions



2 instructions

Inner versus Outer Product

IME does not enforce or favour any approach to the matrix multiplication. The choice is left to the implementor.

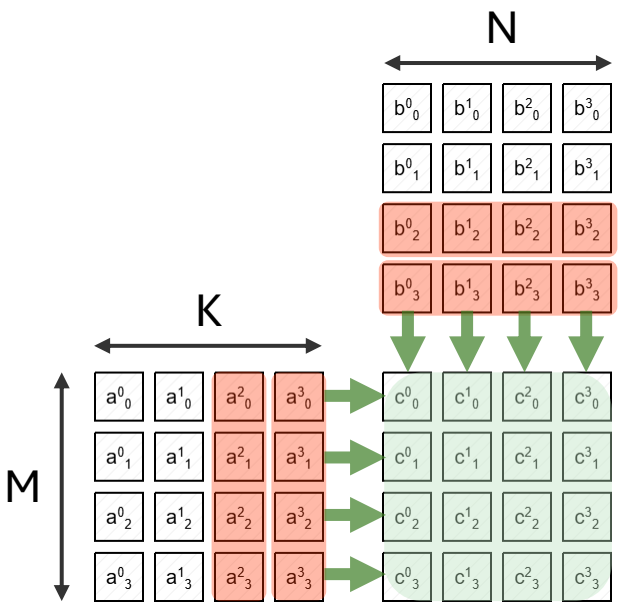


Inner Product

Core operation: **Dot Product**

Examples: systolic array;
one or several „fat“ dot products

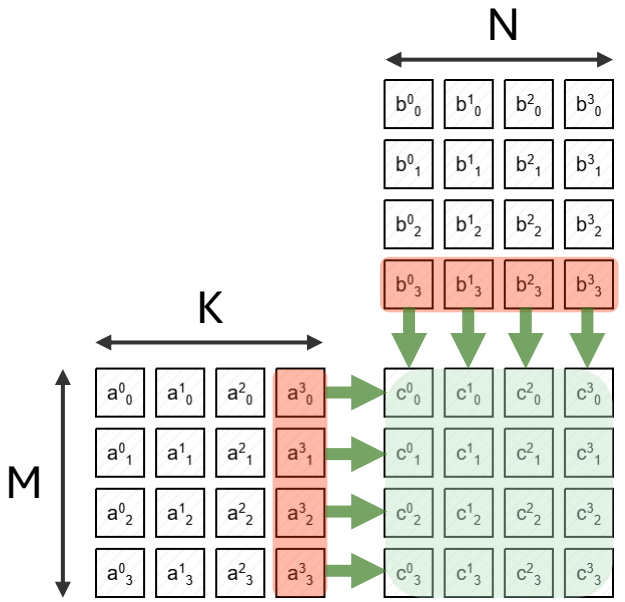
```
for (i=0;i<N;i++)
  for (j=0;j<M;j++)
    sum = 0
    for (k=0;k<K;k++)
      sum += a^k_j*b^i_k
    c^i_j = sum
```



Outer Product

Core operation: **Rank-2 Update**

```
for (k=0;k<K;k+=2)
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      c^i_j += a^k_j*b^i_k+ a^k+1_j*b^i_k+1
```



Outer Product

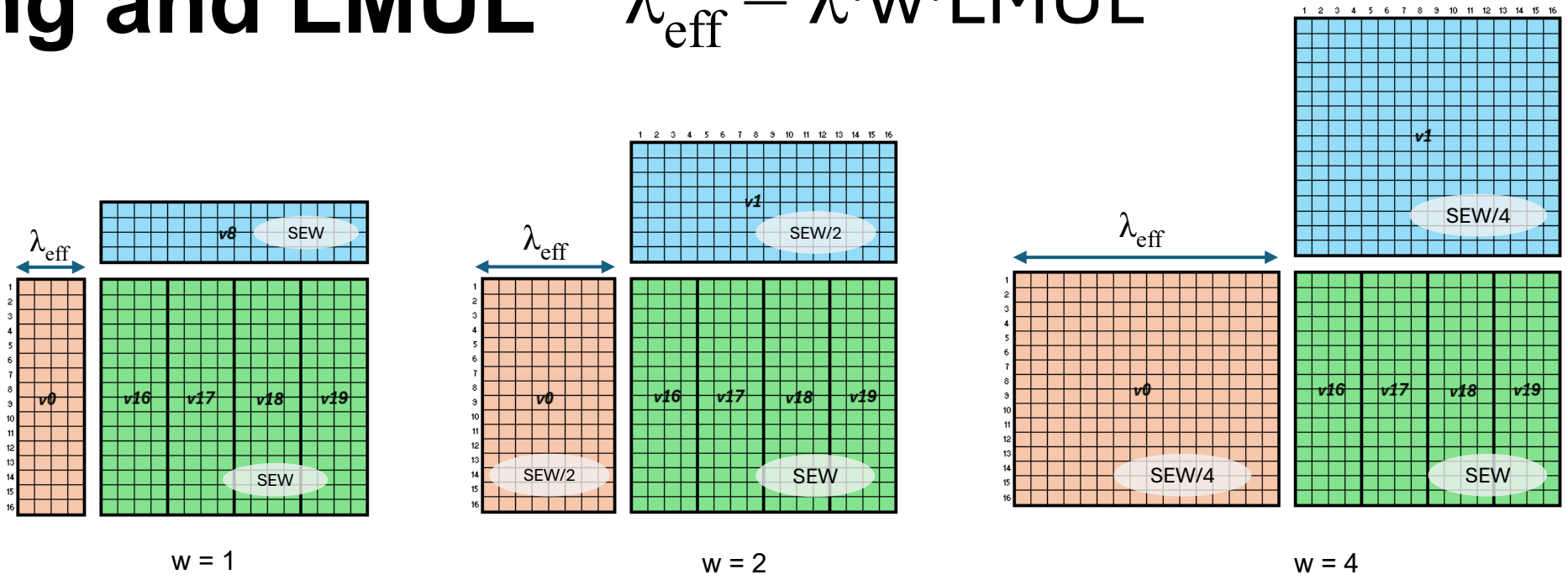
Core operation: **Rank-1 Update**

```
for (k=0;k<K;k++)
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      c^i_j += a^k_j*b^i_k
```

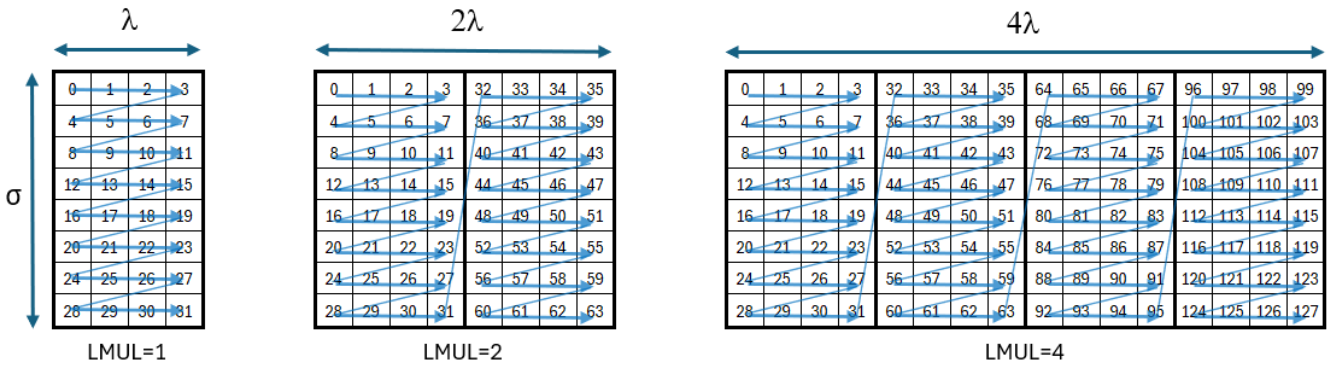
Widening and LMUL

$$\lambda_{eff} = \lambda \cdot w \cdot LMUL$$

Widening multiply & accumulate up to 8x



LMUL only applies to A and B tiles!



Performance Scales

$$CI = \frac{M \cdot M \cdot VL/VLMAX \cdot \mathrm{K_{eff}}}{M \cdot K_{eff}(1 + VL/VLMAX)} = \frac{VLMAX \cdot VL}{\lambda\,(VLMAX + VL)} \rightarrow \frac{VLMAX}{2\lambda}$$

Compute intensity
MACs/load

HW	VLEN	λ	BW
SW	VL	w	LMUL

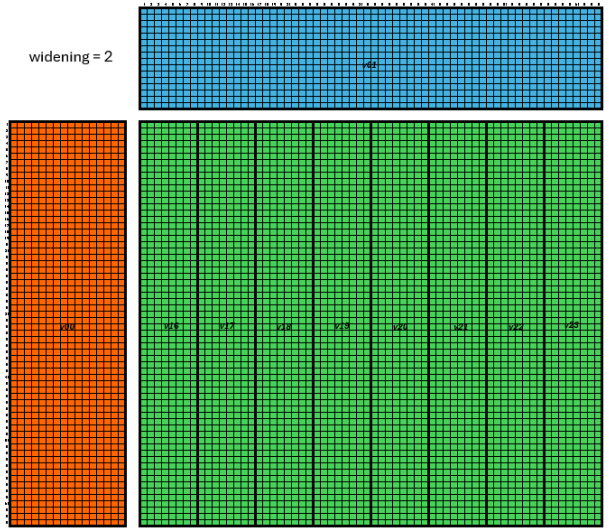
$$Performance = CI \cdot BW = \frac{VLMAX \cdot BW}{2\lambda}$$

With the same Cache-to-VR bandwidth: a wider implementation sustains full utilization with no recompilation. Same binary, VLEN=256 → 65536, 16× throughput.

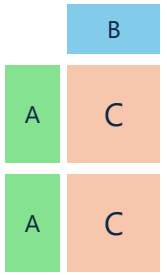
HPC example:

- VLEN=16384
- FP32 += FP16*FP16 (w=2)
- VL = 1024 FP16 elements → **VRs fully utilized**
- Compute intensity: 64 ops/load

Performance is limited by the 32 architectural VRs, more would enable larger or more C panels.

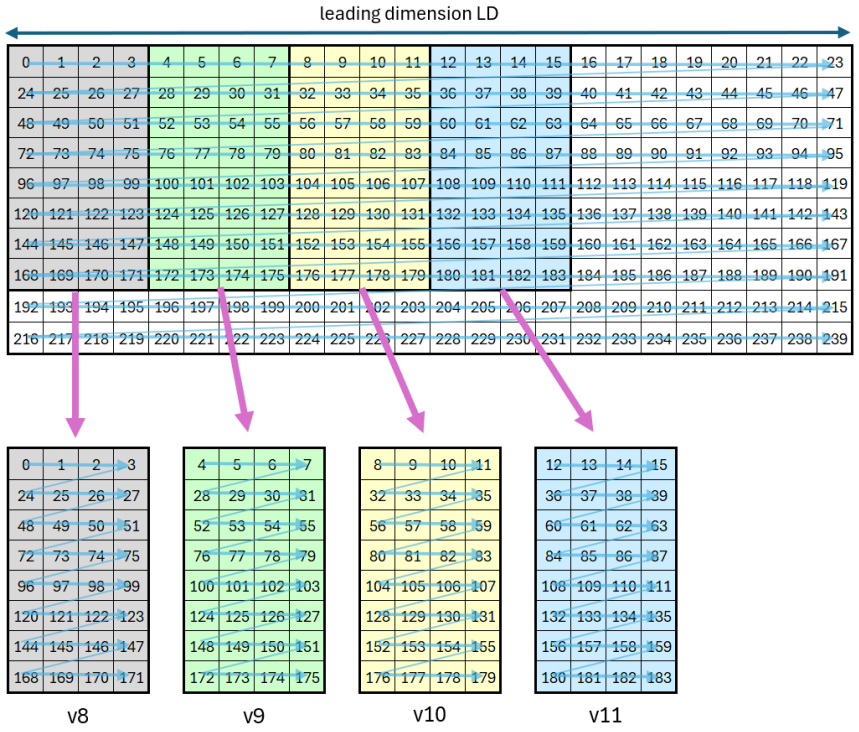


E.g. FP32 += FP16*FP16



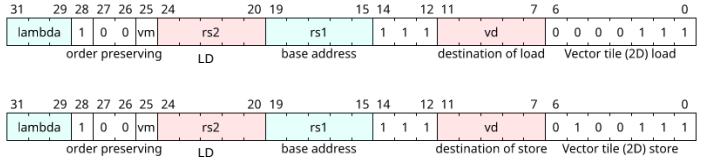
Multiple panels
compute intensity:
85.3 ops/load

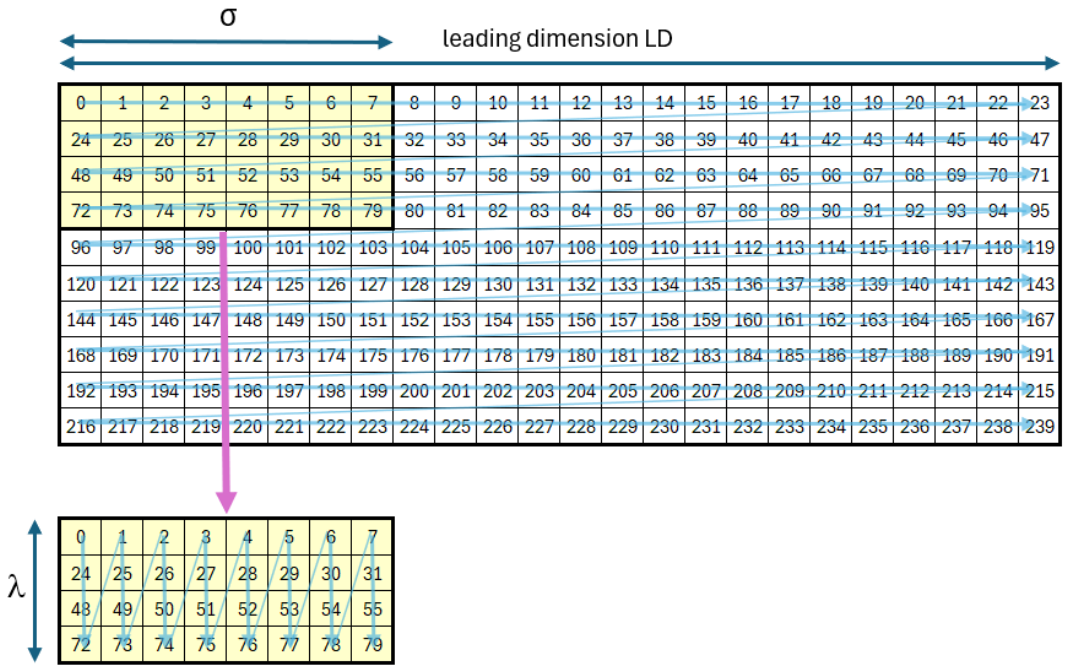
Matrix Tile 2D Load / Store



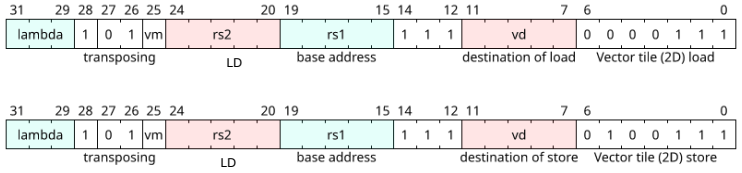
vmtl vd,(base),ld
Order-Preserving Tile Load

- Example: LMUL = 4
- Opportunity for optimization
- Load full cache lines
  example:
  cache line size: 64 byte
  SEW=32





vmttl vd,(base),ld
Transposing Tile Load



A SCALABLE SOLUTION

Microcontroller to **supercomputer**

The same opcode, the same binary — from one column of C on a two-register core to a full HPC tile on a wide-VLEN machine. Partial VL makes embedded streaming and bulk throughput two ends of one continuum, not two ISA features.

Embedded

Complete GEMM on a core with just 64bit-wide vector registers. No spill. AI inference without a dedicated NPU.

- AI INFERENCE, NO DEDICATED NPU
- SCALES WITH VLEN (FROM AS LITTLE AS 64BIT-WIDE VECTOR REGISTERS)
- VL=K_EFFL ONE COLUMN AT A TIME

Mid-range

Raise LMUL for K_eff depth. Ppartial VL balances register pressure against compute intensity.

- MODULATE LMUL FOR DEEPER K
- PARTIAL VL BALANCES MEMORY BANDWIDTH AGAINST COMPUTE
- SAME KERNEL, MORE THROUGHPUT

Wide-VLEN HPC

LMUL = 8, full VL — bulk tiling for servers and supercomputers. Erich's end of the continuum.

- BULK UPDATE
- SERVERS TO SUPERCOMPUTERS WITH THE SAME BINARY
- LMUL=8, FULL VL: MAXIMAL TILE SIZE

Embedded streaming and HPC bulk processing are two ends of one continuum, not two ISA features.

DEGREES OF FREEDOM

4 degrees of freedom + zero-cost MX

Four orthogonal knobs let software and hardware each express intent — VLEN, λ, LMUL, and VL shape every point from embedded streaming to HPC bulk. Where SME and AMX bolted microscaling on later, Zvvm gets it from one already-present bit: no new opcodes, no new registers, no new modes.

Knob	Range	Purpose
VLEN	64 → 64K	Binary-compatible throughput scaling
λ	1 → 64	Tile geometry; dot-product at λ = VL
LMUL	1 → 8	Trade register groups for K_eff depth
VL	K_eff → max	Embedded streaming → full HPC tile
W	1, 2, 4	Precision ladder: INT8→INT32, FP16→FP32

Microscaling, free

`vm = 0` aliases the FP MAC opcode onto the MX decode path.

- No new opcodes
- No new registers
- No new modes
- MXFP8 / MXFP4 / MXINT8 + mixed

TAKEAWAYS

All the scaling. No new state.

One GEMM binary runs from a microcontroller with VLEN=64, on a server with VLEN=1024, and on a supercomputer with VLEN=64k. This is possible only, because geometry is algebraic and all state reuses the V register file.

The Zvmm specification is **currently undergoing Internal Review**. We expect ratification later this year.

SOFTWARE ENABLEMENT DELIVERED AS A STANDARDIZATION ARTIFACT

QEMU model

GCC intrinsics

BLAS kernels

Test suite

Co-developed with the specification.
Lowering the barrier for implementers, accelerating the upstream adoption in open-source, and making the specification testable from the start.