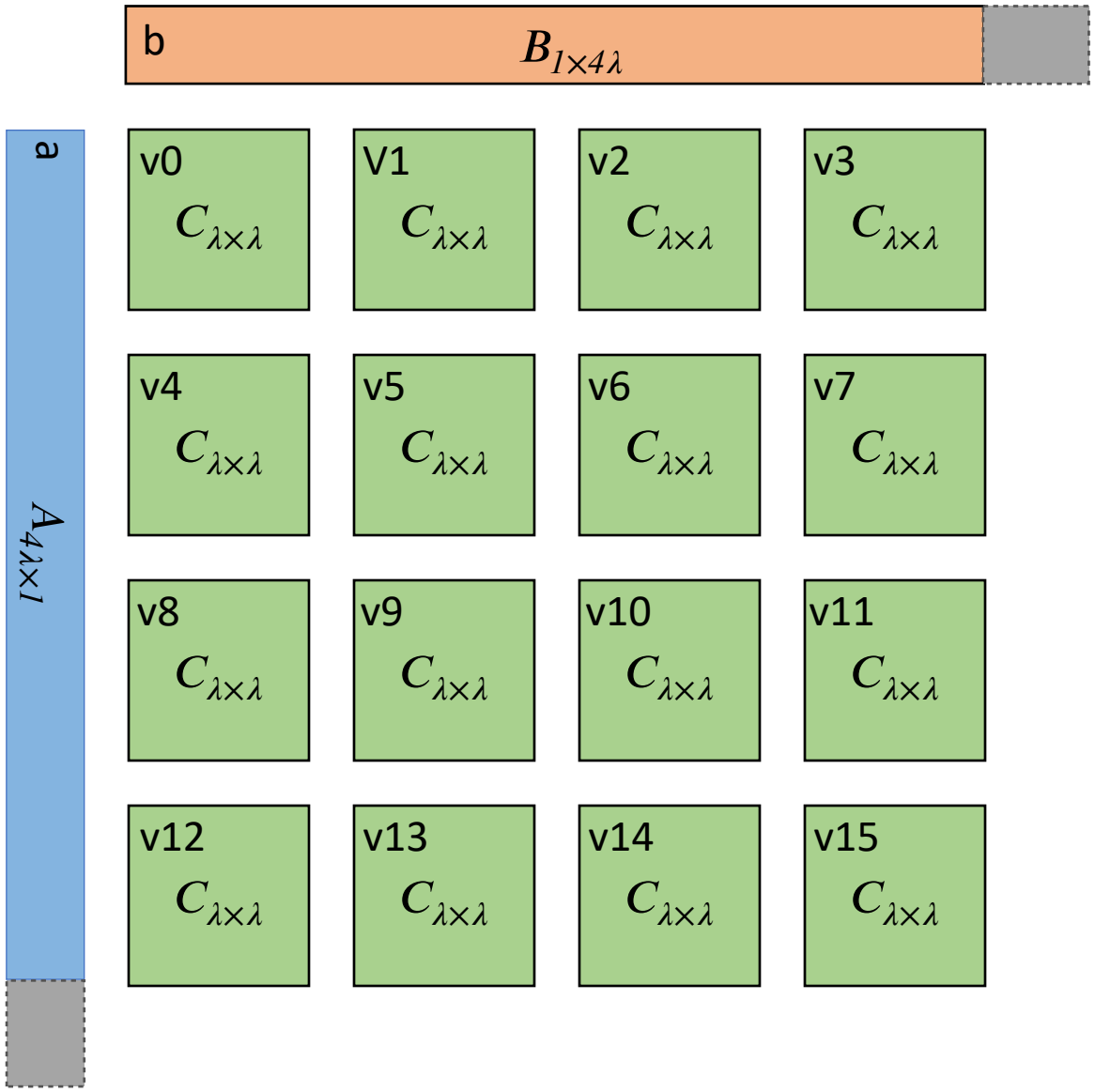
Option D: accumulator tiles in vector registers

Abel Bernabeu <abel.bernabeu@esperantotech.com>

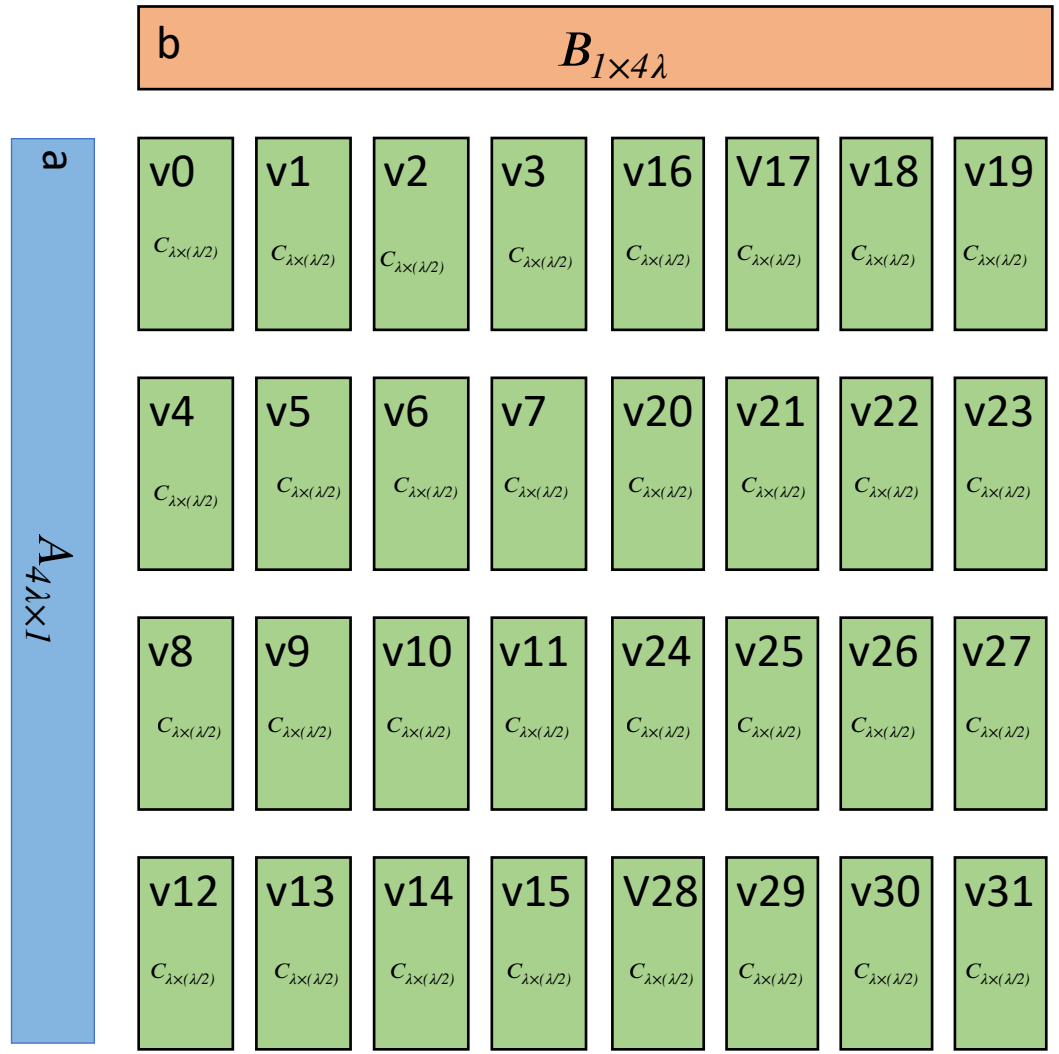
July 1st, 2024

Accumulator tiling: fp32 vs. fp64

4 lamda x 4 lamda floats



4 lamda x 4 lamda doubles



Scalable size concept

Scalable size = minimum value X scale

With scales being:

- one-scale: non-scalable amount
- vscale: a vector length scale (VLENB CSR / 8)
- mscale: a matrix height (MATCOLB CSR?)
- nscale: a matrix width (MATROWB CSR?)
- mnscale: a matrix height times matrix width (MATB CSR?)

Conceptual fitting in the ecosystem

Registers	Extension
- Vectors v0..v31: footprint scales with vscale, holding regular vectors	Zv
- New vector **reg a**. Footprint scales with **mscale**. Source for outer vector product - New vector **reg b**. Footprint scales with **nscale**. Source for outer vector product - Reuse of **regs v0..v31**: footprint scales with **vscale>=mnscale**. Tiles of accumulator, destination for outer product of vectors	IME TG Option D
- Additional mn-scaled matrix registers for supporting: * Inner product of matrices * Convolution	AME TG

Minimum VLEN for different configurations

A vector register fits an accumulator tile:

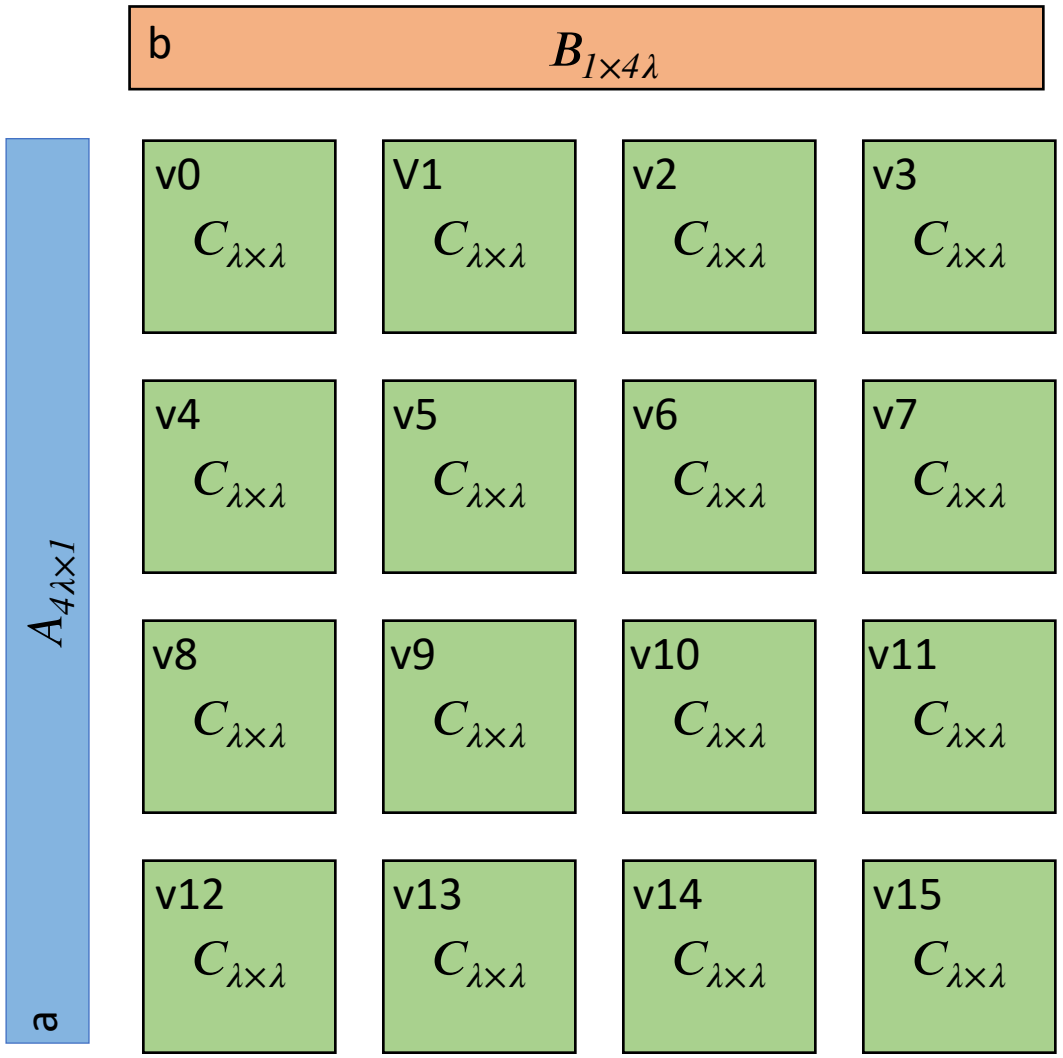
vl >= matrix_tile_height * matrix_tile_width

For 32 bits, the accumulator is split in 4x4 tiles, each tile fitting in a vector registers

For 64 bits, the accumulator is split in 8x4 tiles, each tile fitting in a vector registers

Lambda	Matrix tile height	Matrix tile width for 32 bits	Matrix tile width for 64 bits	Matrix dims for 32 bits and 64 bits	Min VLEN
2	2	2	1	8x8	Zvl64b
4	4	4	2	16x16	Zvl128b
6	6	6	3	24x24	Zvl192b
8	8	8	4	32x32	Zvl256b

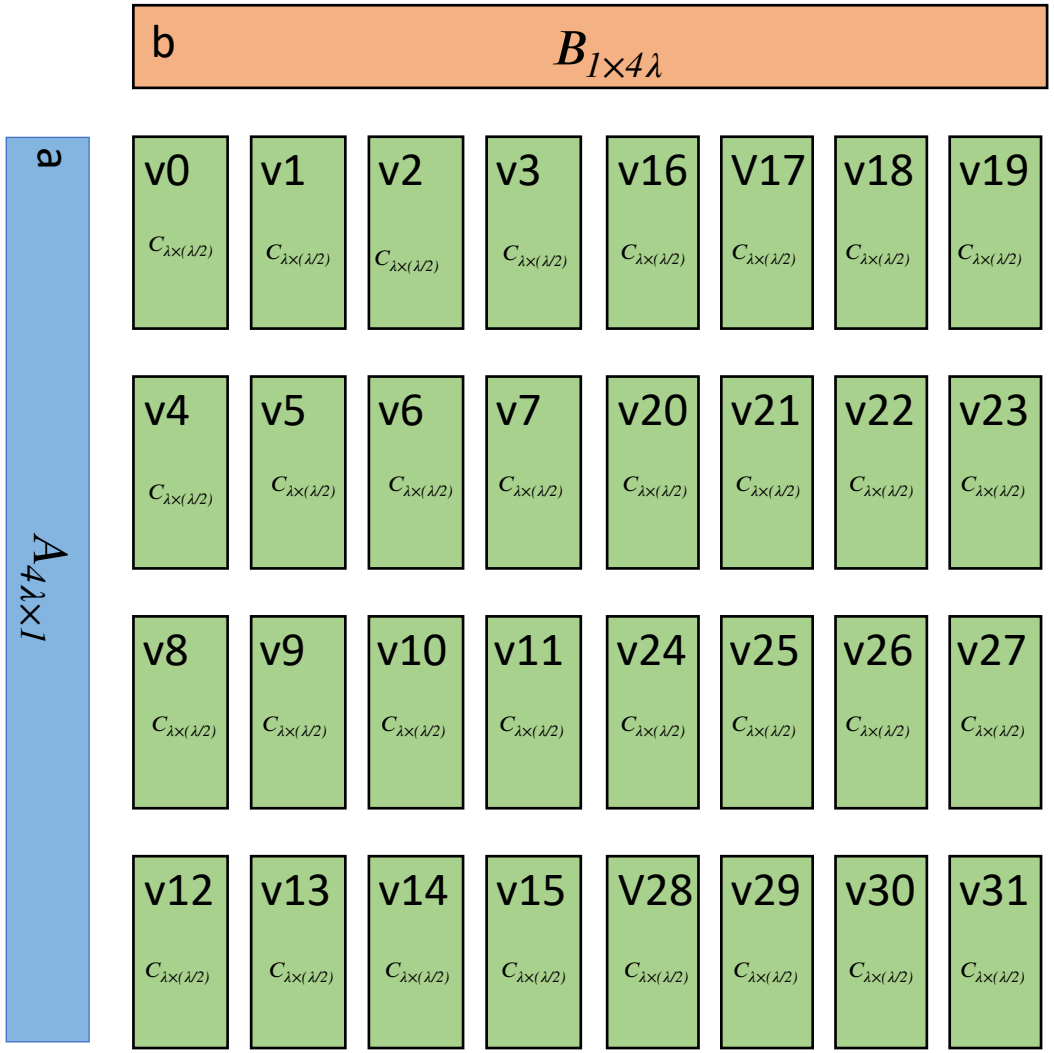
Option D: Computation intensity for 32 bits



- Computational intensity (madds/loads):

$$\eta = \frac{16\lambda^2}{8\lambda} = 2\lambda \text{ madds/load}$$

Option D: Computation intensity for 64 bits



- Computational intensity (madds/loads):

$$\eta = \frac{16\lambda^2}{8\lambda} = 2\lambda \text{ madds/load}$$

- The same as for 32 bits!

Instructions (straw man)

- Instructions based on outer product of vectors:
  - **load{32|64} a, rs1, rs2** # load column on new reg, potentially caching only up to L2 with Zihintntl
    rs1 is X coordinate as uint
    rs2 is Y coordinate as uint
  - **load{32|64} b, rs1, rs2** # load B row on new reg, potentially caching only up to L1 with Zihintntl
    rs1 is X coordinate as uint
    rs2 is Y coordinate as uint
  - **mac{32|64}** # Multiply/accelerate, suitable for a sequencer expanding microinstructions
    EMUL=16 for mac32, EMUL=32 for mac64
    sources are a and b
    destinations are v0, v1, v2, … , v15, when 32 bits
    destinations are v0, v1, v2, … , v31, when 64 bits
    no masking is possible (or needed)
    vtype and vl CSRs ignored
  - **store{32|64} v0, rs1, rs2** # Store accumulator region
    rs1 is X coordinate as uint
    rs2 is Y coordinate as uint
- For loads and stores:

  base address is taken from **MATBASE** CSR
  stride is taken from bits **62..0 of MATSTRIDE CSR**
  whether loads/stores do transpose is encoded in bit **63 of MATSTRIDE CSR**
  width for clipping is taken from **MATWIDTH** CSR
  height for clipping is taken from **MATHEIGHT** CSR