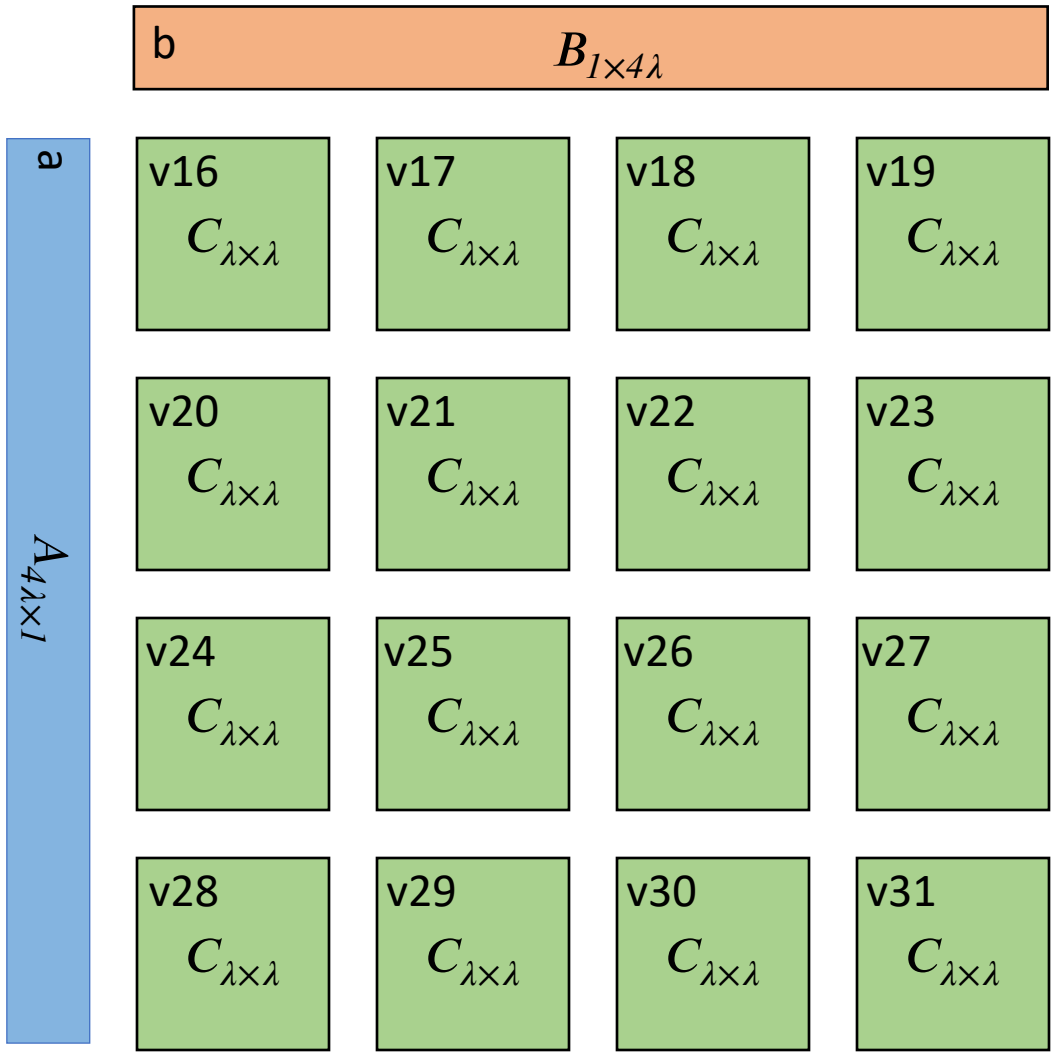
Option D: accumulator tiles in vector registers

Abel Bernabeu <abel.bernabeu@esperantotech.com>

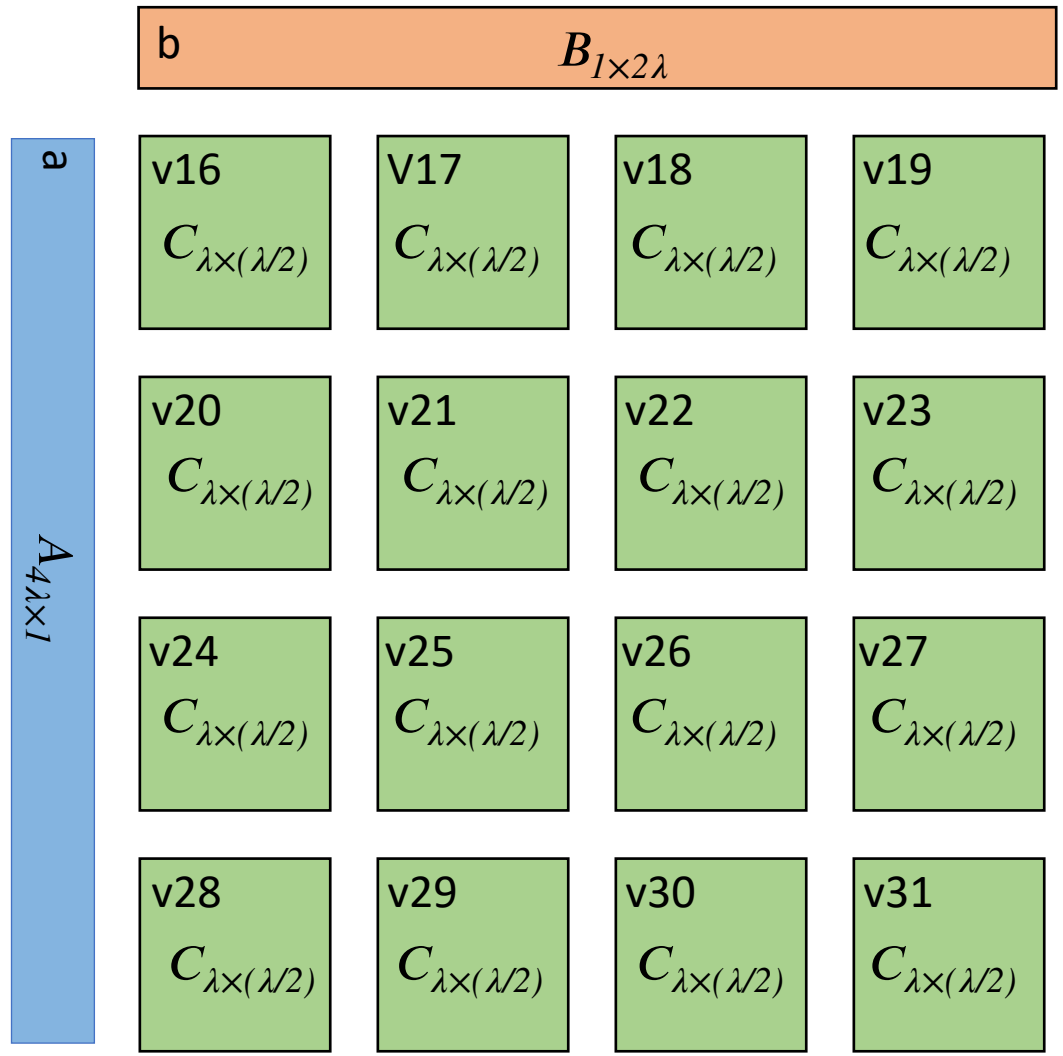
July 15th, 2024

Accumulator tiling: fp32 vs. fp64

4 lamda x 4 lamda floats



4 lamda x 2 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
- Reuse of v0..v15 as source vectors for outer product. - Reuse of regs v16..v31 as accumulator tiles Footprint scales with vscale >= mnscale	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:

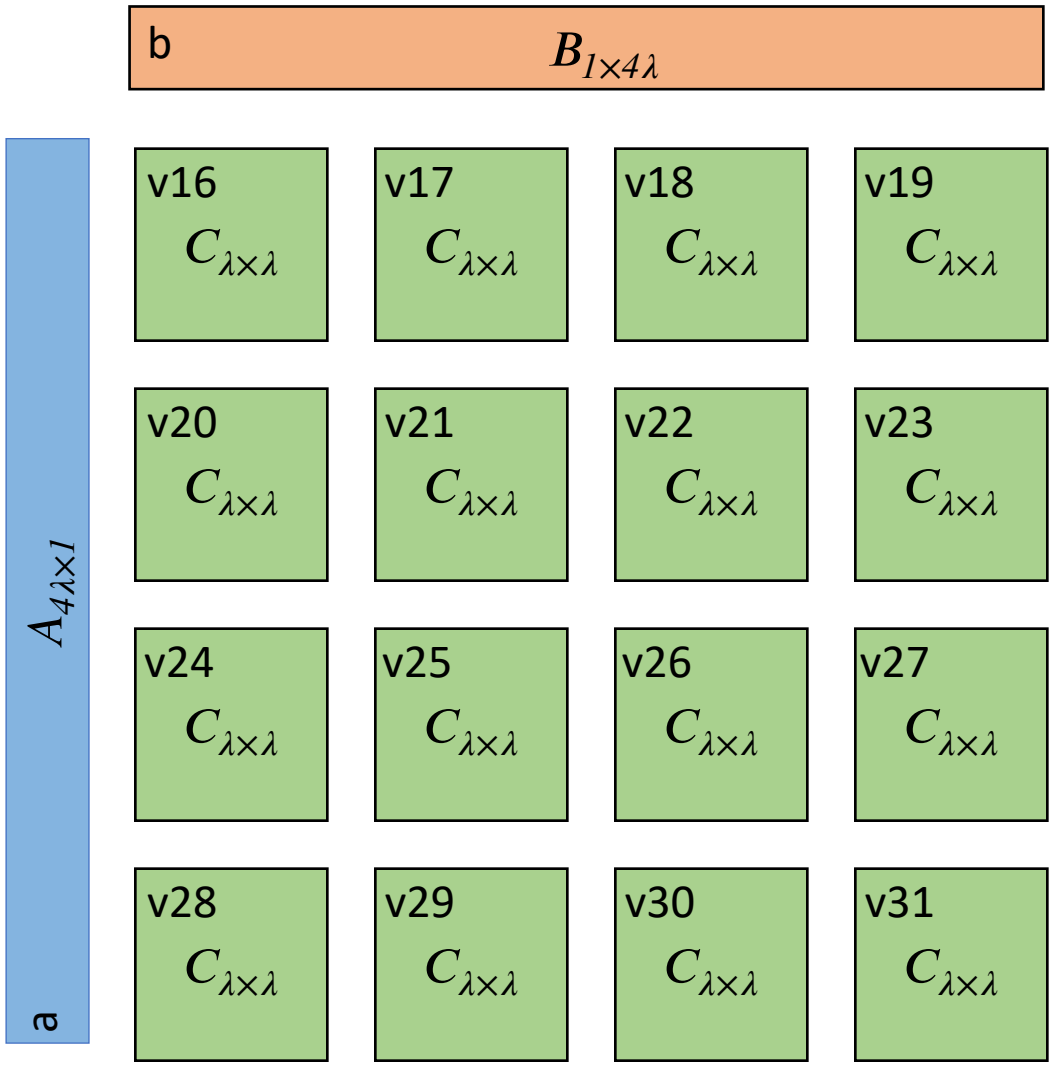
(4 lambda x 4 lambda) accumulator
split in 4x4 tiles
Lambda x lambda elements each tile

For 32 bits:

(4 lambda x 2 lambda) accumulator
split in 4x4 tiles
Lambda x (lambda/2) elements each tile

Lambda	Matrix tile height	Matrix tile width for 32 bits	Matrix tile width for 64 bits	Matrix dims for 32 bits	Matrix dims for 64 bits	Min VLEN
2	2	2	1	8x8	8x2	Zvl64b
4	4	4	2	16x16	16x8	Zvl128b
6	6	6	3	24x24	24x12	Zvl192b
8	8	8	4	32x32	32x16	Zvl256b

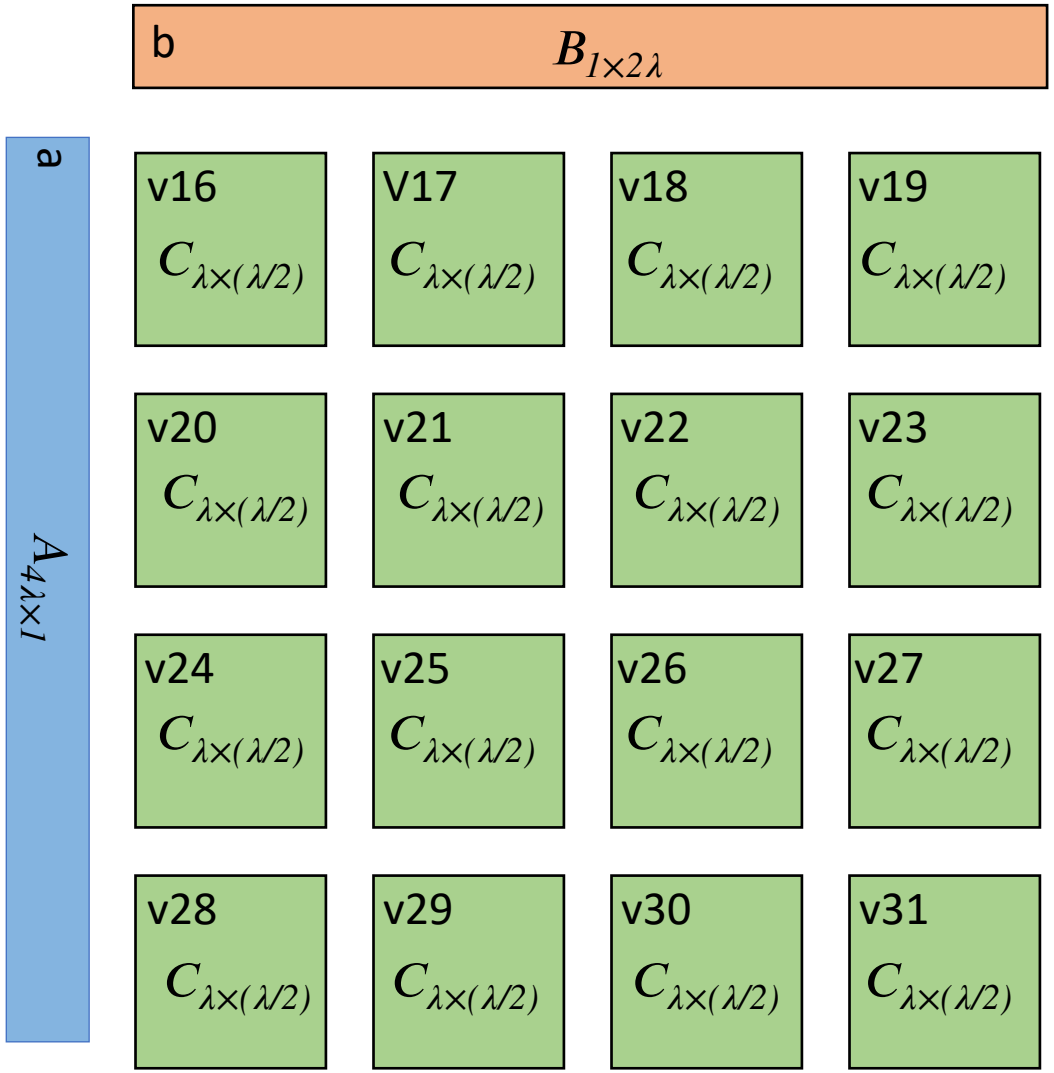
Option D: Computation intensity for 32 bits



- 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):

$$\eta = \frac{8\lambda^2}{6\lambda} = 1.333\ \lambda \text{ madds/load}$$

- Lower than for 32 bits.

Instructions (straw man)

- Instructions based on outer product of vectors:
  - **ldcol{32|64} vd, rs1, rs2** # load column on vd, potentially caching only up to L2 with Zihintntl
    rs1 is X coordinate as uint
    rs2 is Y coordinate as uint
  - **ldrow{32|64} vd, rs1, rs2** # load B row on vd, potentially caching only up to L1 with Zihintntl
    rs1 is X coordinate as uint
    rs2 is Y coordinate as uint
  - **mac{32|64} vs1, vs2** # Multiply/accelerate, suitable for a sequencer expanding microinstructions
    EMUL=16
    sources are vs1 and vs1
    destinations are v16, v1, v2, … , v31
    no masking is possible (or needed)
    vtype and vl CSRs ignored
  - **store{32|64} rs1, rs2** # Store accumulator region
    implicit sources are v16, v1, v2, … , v31
    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