Option D: streaming registers

Abel Bernabeu <abel.bernabeu@gmail.com>

October 23rd, 2023

Option D: motivation

- Outer product of vectors (as opposed to inner product of matrices)
- Only adding new registers for source operands
- Credit: symmetry with a proposal from Jose Moreira.

  Whereas Jose was adding new registers for accumulators and reusing vector registers as sources, here we are adding new registers for sources and reusing the existing vector registers as accumulators
- Big accumulators and high reuse of the existing vector registers
- Designed with **BLIS** framework in mind

Why outer product of vectors?

- IBM's Power 10 uses it as basic primitive and allows them to achieve 90% or more of peak performance (Jose Moreira can tell you)
- Exactly what is demanded by the BLIS framework
- Abel's analytical argument

Why outer product of vectors?

Loaded elements: $mk + nk = (m + n)k$

Multiply-accumulate operations:

$$mkn = (mn)k$$

Computation intensity (macs per load):

$$mn/(m + n)$$

Lemma 1: The computation intensity is optimal for m=n, irrespective of k

Proof: start with m=n and try to subtract “d” from m and add “d” to “n”, with “d<m”. That is a way like any other to make “m” and “n” non-equal without loss of generality.

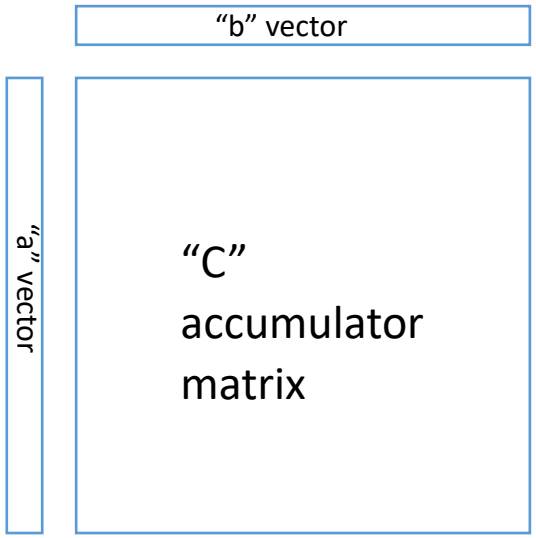
The new computation intensity gets worsened.

Lemma 2: The computation intensity improves with m and n

Proof: start with m=n=m0 and consider m=n=m0+d.

The computation intensity gets better.

Why outer product of vectors?



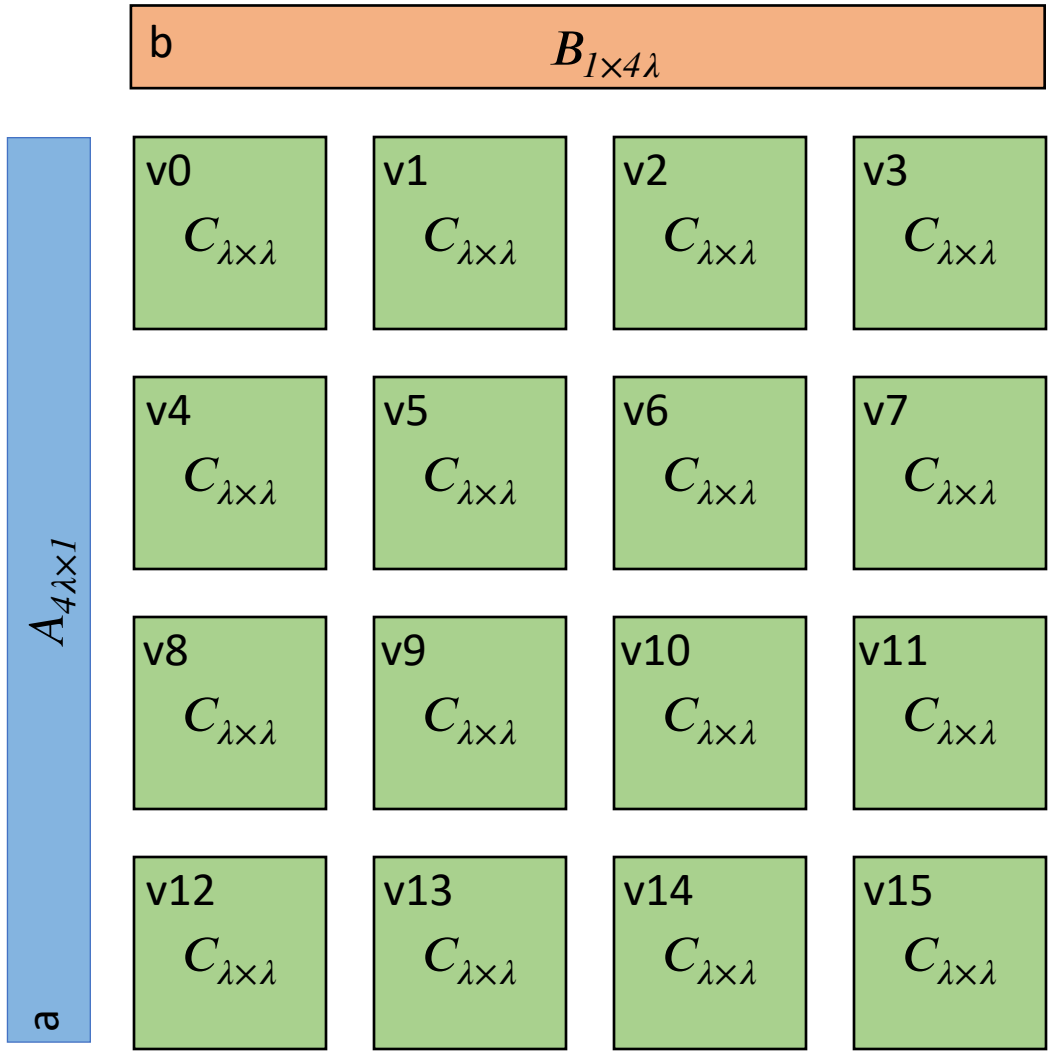
Conclusion:

k=1 is as good as any other value

m=n makes for the optimal computation intensity

The bigger the (square) accumulator the more efficient the solution

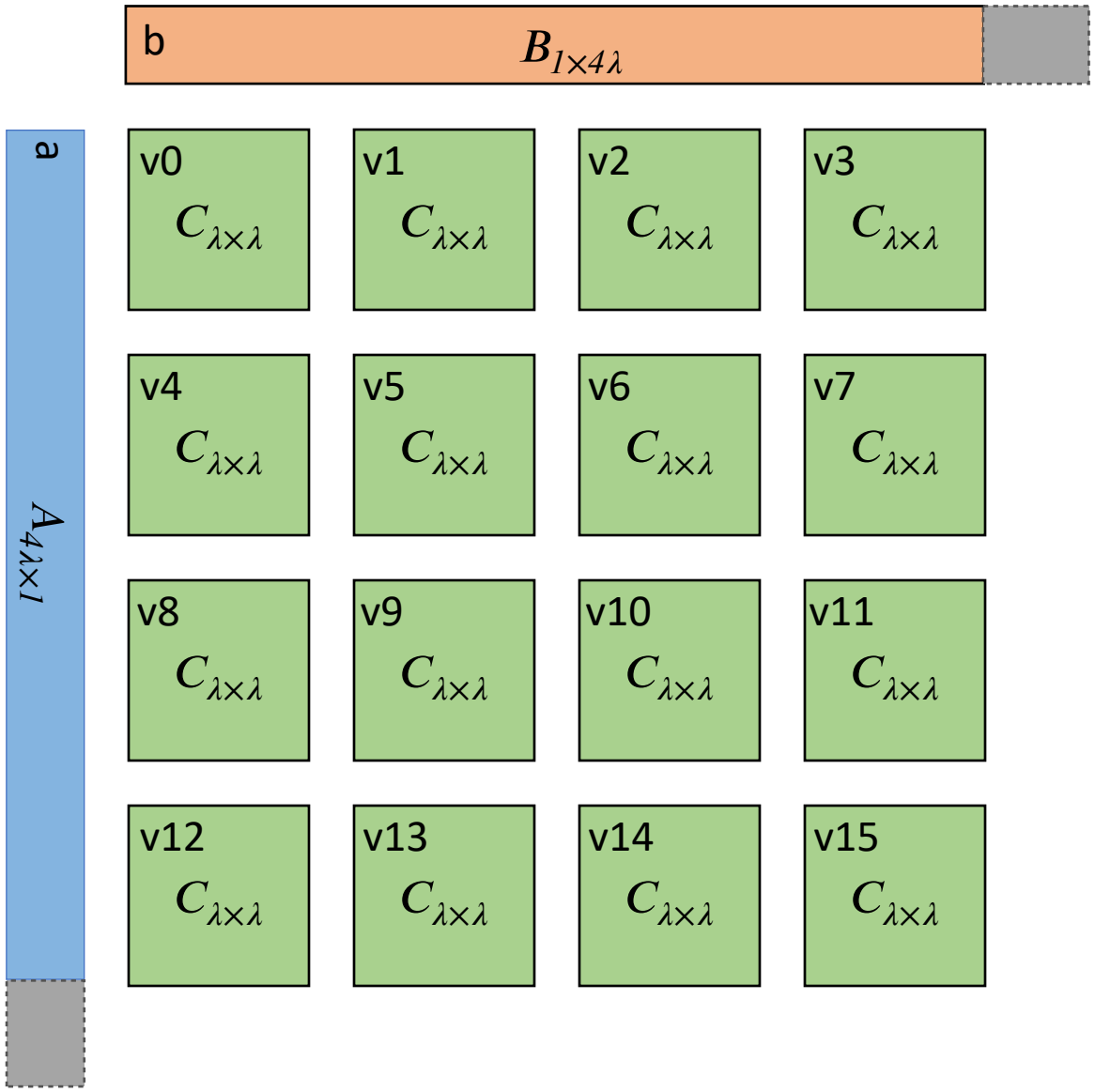
Option D: Tailoring for BLIS



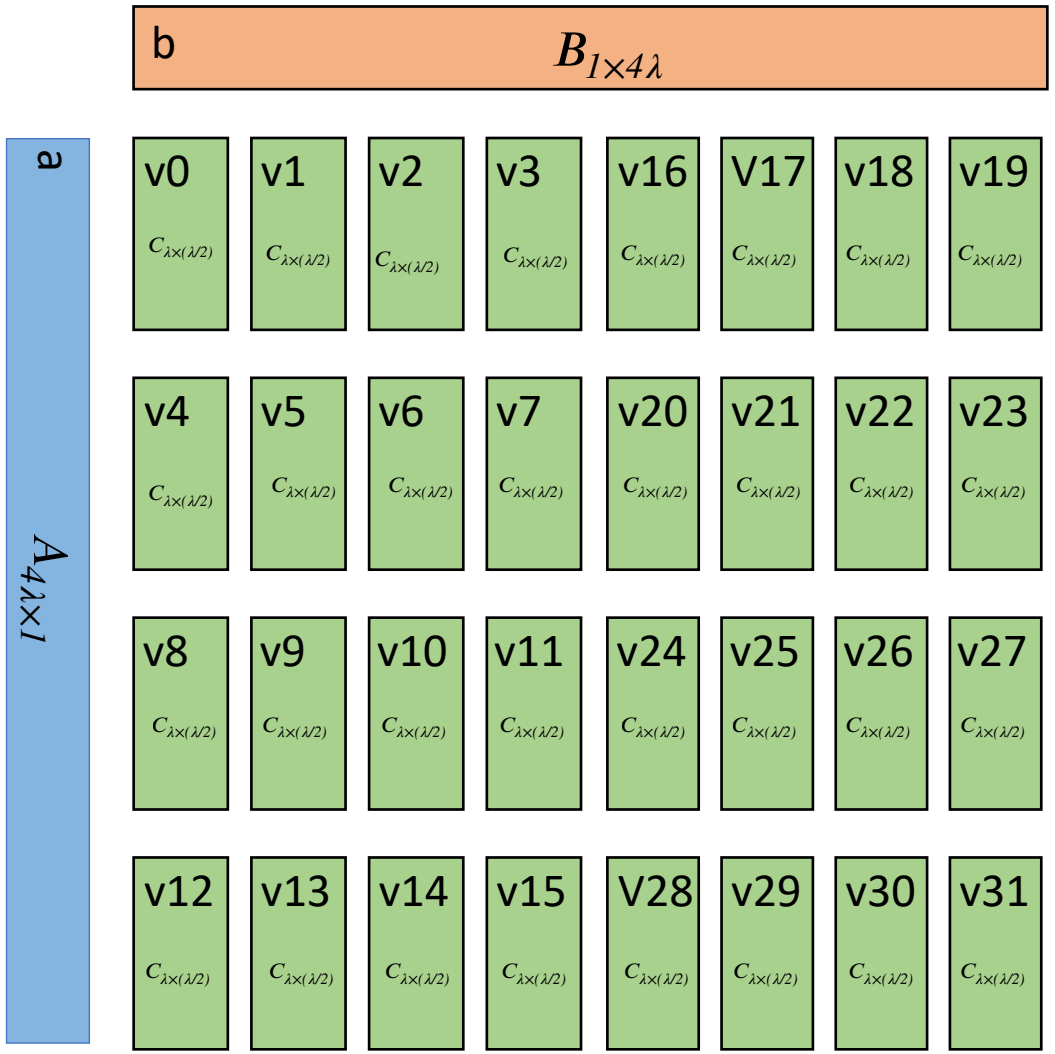
- Computational intensity $\eta = \frac{16\lambda^2}{8\lambda} = 2\lambda$ madds/load
- Instructions tailored for BLIS microkernel:
  - **load a, (rs1)** # load column on new reg, potentially caching only up to L2
  - **load b, (rs1)** # load B row on new reg, potentially caching only up to L1
  - **mac v0, a, b** # Multiply/acculate
    - destination is v0 with LMUL=16, when 32 bits
    - destination is v0 with LMUL=32, when 64 bits
    - No masking is possible (or needed)
  - **store v0, rs1(rs2)** # Store accumulator region
- Special purpose register for accumulators happening on a future revision carried on by another Task Group. Reserved bits on “mac” to be allocated by the TG.

Option D: fp32 vs. fp64 accumulators

4 lamda x 4 lamda floats, implicit LMUL=16



4 lamda x 4 lamda doubles, implicit LMUL=32



Option D: layout for A register

Key idea:

to layout so that it is possible for microarchitecture to reuse the wires from 64 bits data types for all all the other data types.

If bit from “i” from A is needed for producing tile in vector reg v[x] for 4 bits inputs,
then also needed for 8 bits inputs.
If “i” needed for 8 bits, then also needed for 16 bits.
If “i” needed for 16 bits, then also needed for 32 bits.
If “i” needed for 32 bits, then also needed for 64 bits.

Similar layout for B register.