

# Scalable Matrix Architecture (Fundamentals and Vector-Matrix Facility)

*José Moreira*  
IBM Corporation

*Abel Bernabeu*  
Esperanto

# Comments on the state-of-the-art for matrix computing

- Most high-performance matrix computations rely on the following kernel:

$$\mathbf{C}_{m \times n} = \mathbf{A}_{m \times K} \times \mathbf{B}_{K \times n}$$

$\mathbf{C}_{m \times n}$  is a register-resident matrix, with  $m \approx n$ , and  
 $\mathbf{A}_{m \times K}$  and  $\mathbf{B}_{K \times n}$  are streamed from memory, with  $m \ll K$  and  $n \ll K$

- Matrix  $\mathbf{A}_{m \times K}$  is organized as a vector of  $K$  elements, each element a column vector of  $m$  elements of  $\mathbf{A} = [\mathbf{A}^0 \quad \mathbf{A}^1 \quad \dots \quad \mathbf{A}^{K-1}]$

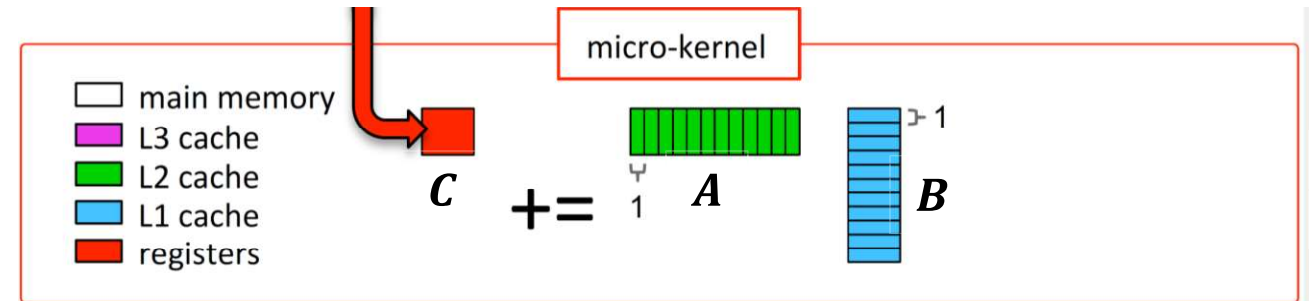
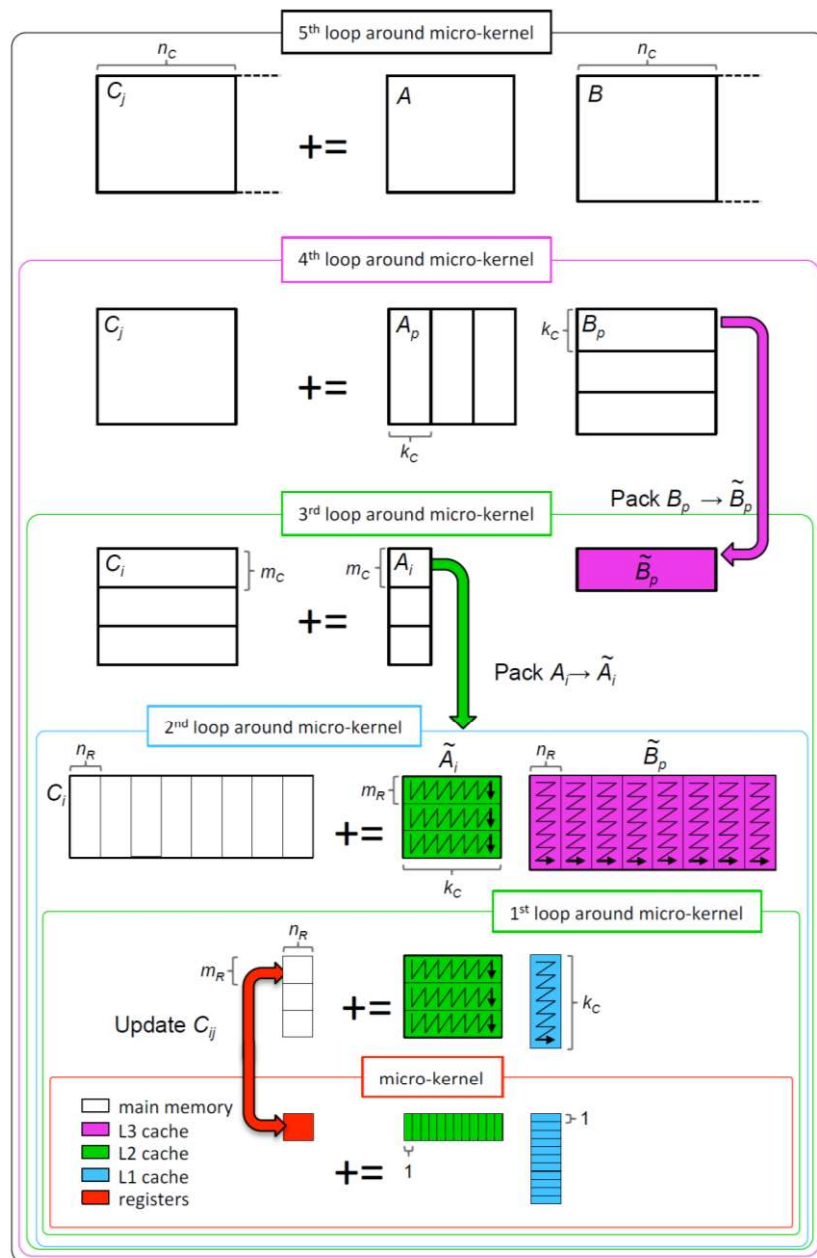
- Matrix  $\mathbf{B}_{K \times n}$  is organized as a vector of  $K$  elements, each element a row of  $n$

elements of  $\mathbf{B} = \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{K-1} \end{bmatrix}$

- We will consider only the case of deterministic values of  $\mathbf{C}$ , as produced by the following algorithm:


$$\mathbf{C} \leftarrow 0; \text{for } (k = [0, K)) \mathbf{C} \leftarrow \mathbf{A}^k \times \mathbf{B}_k + \mathbf{C}$$

# Anatomy of a high-performance matrix multiply



# Performance bounds

- Let  $\Delta$  be the latency (in cycles) of an elemental multiply-add operation
- Computing each element of  $\mathbf{C}$ , as described above, requires evaluating a dependence chain of  $K$  multiply-adds, and therefore takes time  $K\Delta$

$$\mathbf{C} \leftarrow \mathbf{A}^k \times \mathbf{B}_k + \mathbf{C}$$


The diagram illustrates a feedback loop in the computation of  $\mathbf{C}$ . A blue bracket connects the  $\mathbf{C}$  on the right (the result of the previous step) to the  $\mathbf{C}$  on the left (the input to the current step). Below the bracket is the symbol  $\Delta$ , representing the latency of a single multiply-add operation.

- The computation of  $\mathbf{C}$  requires  $mnK$  multiply-adds
- The maximum computation rate that can be sustained is

$$R = \frac{mnK}{K\Delta} = mn/\Delta \quad \text{madds/cycle}$$

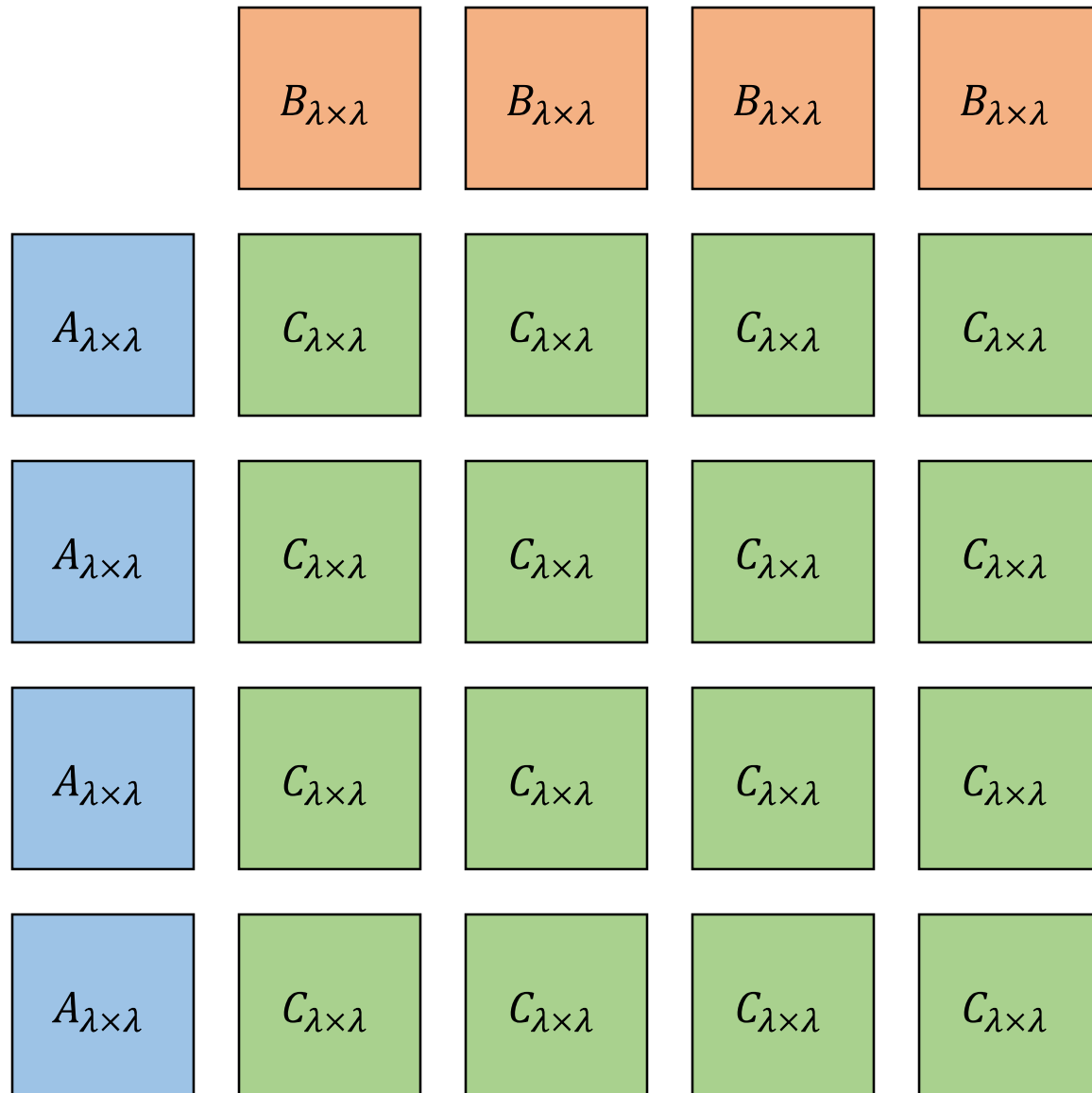
- This sets an upper bound on performance, dictated by the size of the  $\mathbf{C}$  panel that can be kept in registers (an architectural parameter) and the latency of a multiply-add (a design/technology parameter) – *Note*: integer arithmetic is more forgiving
- For modern server processor and 32-bit floating-point elements, a reasonable value of  $\Delta$  is 4 cycles – the range  $\Delta \in [2,8]$  should cover most cases

## Some configurations (just to get a feel)

---

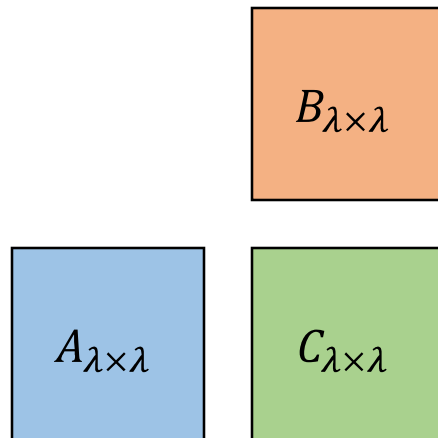
- Let the architected space consist of 32 vector registers of size  $L$  words (32-bit)
  - Then,  $mn \leq 32L$  and the maximum computation rate is  $R = 32L/4 = 8L$  madds/cycle (single-precision floating-point)
- The maximum computation rate of a vector-based matrix facility, using vector registers only, scales with  $L$
- For  $L = 4$ ,  $R = 32$  madds/cycle
- For  $L = 16$ ,  $R = 128$  madds/cycle
- *Note:* In the above, we use all architected space for  $C$ , but in a load/store architecture we need to reserve space for  $A$  and  $B$ , so cut the bound in half
  - $mn \leq 16L$  and the maximum computation rate is  $R = 16L/4 = 4L$  madds/cycle
  - For  $L = 4$ ,  $R = 16$  madds/cycle
  - For  $L = 16$ ,  $R = 64$  madds/cycle
- *Note:*  $4L$  madds/cycle is the performance equivalent of 4 vector pipes operating concurrently

## Option A: 1 Matrix per vector register

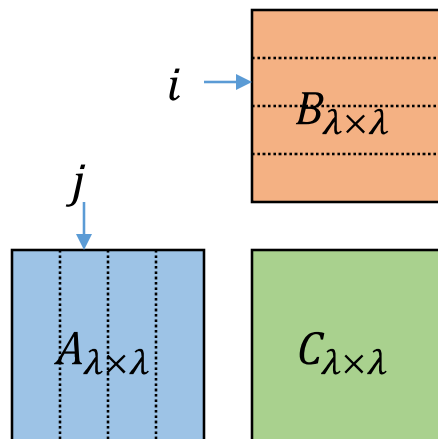


- An  $L$ -word vector register holds an  $\lambda \times \lambda$  matrix,  $\lambda = \sqrt{L}$
- 16 registers hold a  $4\lambda \times 4\lambda$  panel of  $C$
- 4 registers hold a  $4\lambda \times \lambda$  panel of  $A$
- 4 registers hold a  $\lambda \times 4\lambda$  panel of  $B$
- We compute  $C_{4\lambda \times 4\lambda} \leftarrow A_{4\lambda \times \lambda} \times B_{\lambda \times 4\lambda} + C_{4\lambda \times 4\lambda}$
- Total of  $4\lambda \times 4\lambda \times \lambda = 16\lambda^3$  multiply-adds
- Minimum time =  $\lambda\Delta$  cycles
- Maximum computation rate  $R = \frac{16\lambda^3}{\lambda^4} = 4\lambda^2 = 4L$
- This is the upper bound with 16 registers for  $C$
- Total of  $8\lambda^2$  elements loaded ( $8\lambda/\Delta$  words/cycle)
- Computational intensity  $\eta = \frac{16\lambda^3}{8\lambda^2} = 2\lambda$  madds/word
- This works for  $L = 4, 16, 64, \dots$  words
- Single- and double-precision are incompatible

# Option A: Compute instructions

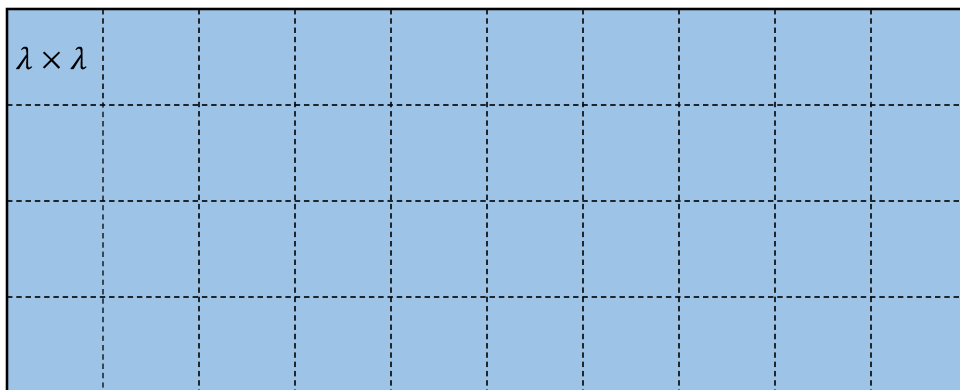
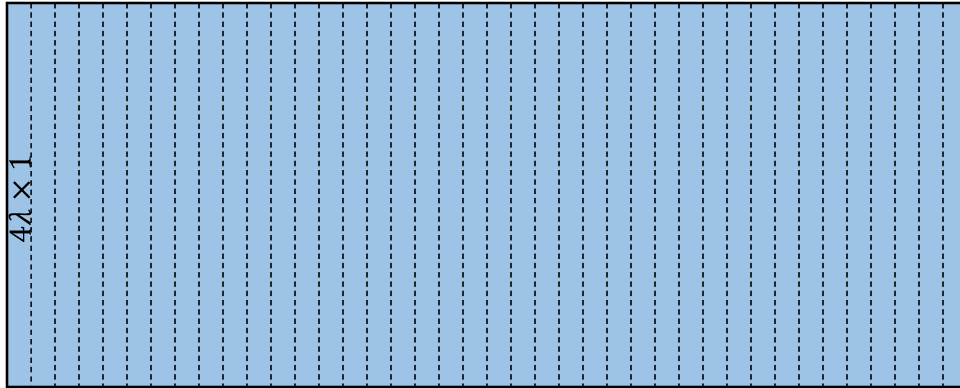


- rank- $\lambda$  update (matrix multiply)
  - $C_{\lambda \times \lambda} \leftarrow \pm A_{\lambda \times \lambda} \times B_{\lambda \times \lambda} \pm C_{\lambda \times \lambda}$
  - Computations:  $\lambda^3$  madds/instruction
  - Latency:  $\lambda\Delta$
  - Must dispatch/issue 4 computational instructions/operations every  $\lambda$  cycles to achieve maximum computation rate ( $4L$  madds/cycle)



- rank-1 update (outer product)
  - $C_{\lambda \times \lambda} \leftarrow \pm A^j \times B_i \pm C_{\lambda \times \lambda}$  (usually  $i = j$ )
  - Computations:  $\lambda^2$  madds/instruction
  - Latency:  $\Delta$
  - Must dispatch/issue 4 computational instructions/operations every cycle to achieve maximum computation rate ( $4L$  madds/cycle)
  - rank- $\lambda$  update can be cracked into  $\lambda$  rank-1 updates

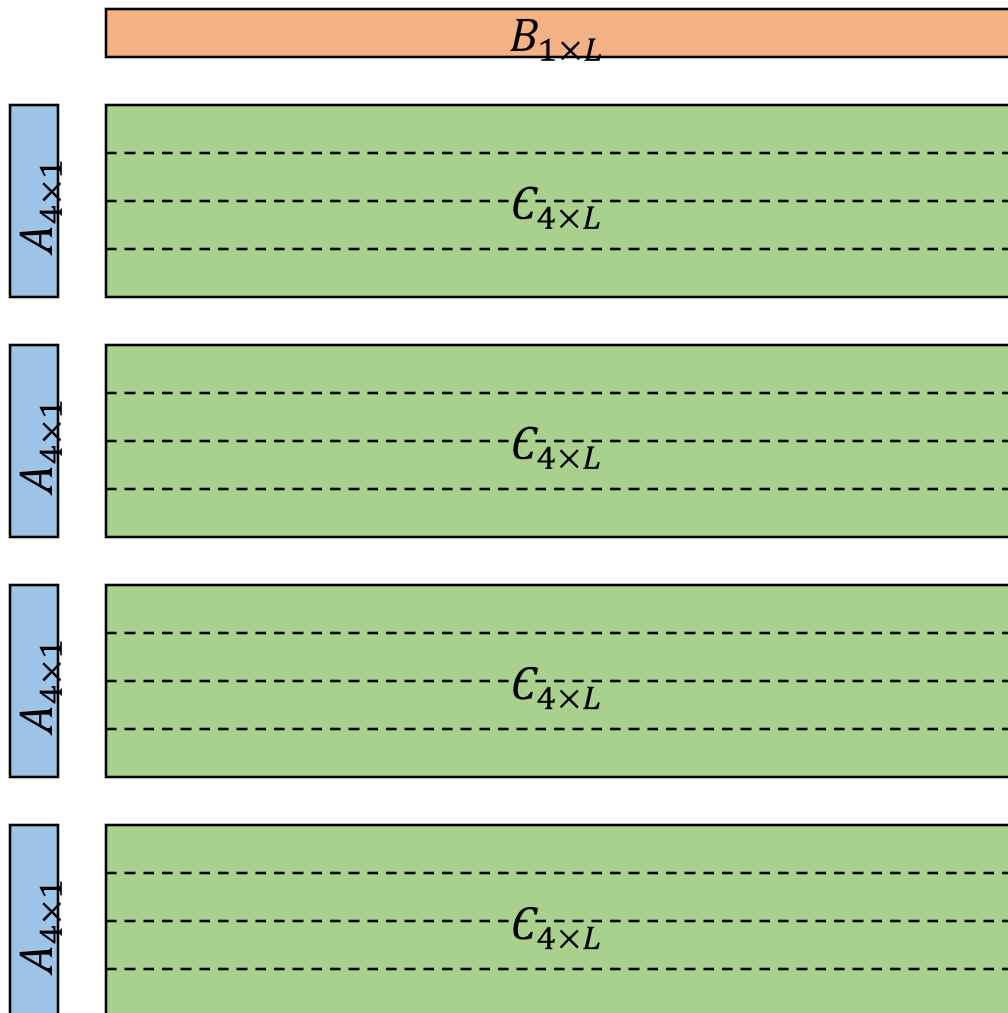
# Option A: Software impact



- Before the compute kernel is executed, the input matrices are *packed* to optimize streaming performance
- $A$  panels are typically formatted as column-major matrices of shape  $4\lambda \times K$
- $B$  panels are typically formatted as row-major matrices of shape  $K \times 4\lambda$
- For Option A to work, both  $A$  and  $B$  must be packed into  $\lambda \times \lambda$  blocks
- Both the compute and packing kernels must be modified to support matrix operations
- $C$  panel must also be reformatted for load/store

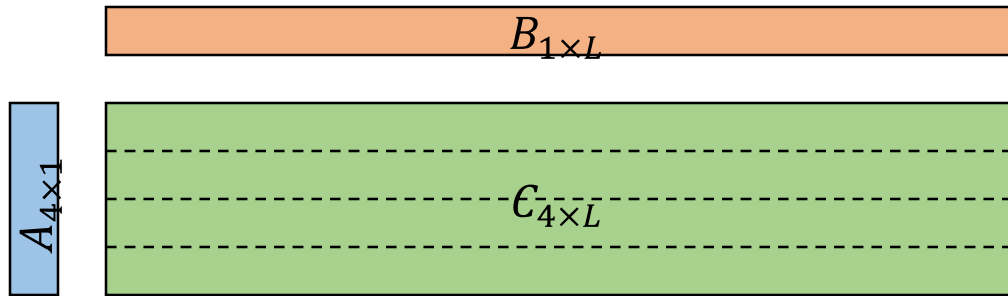


## Option B: 1 Matrix in 4 vector registers



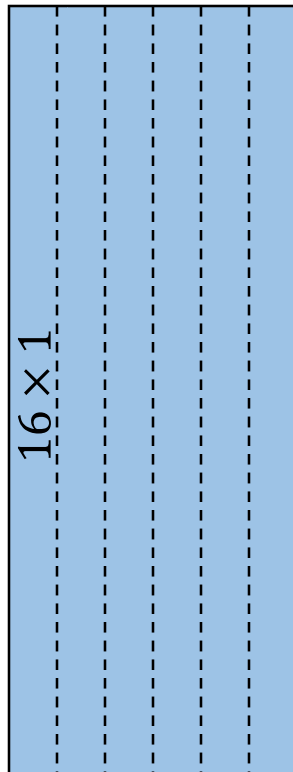
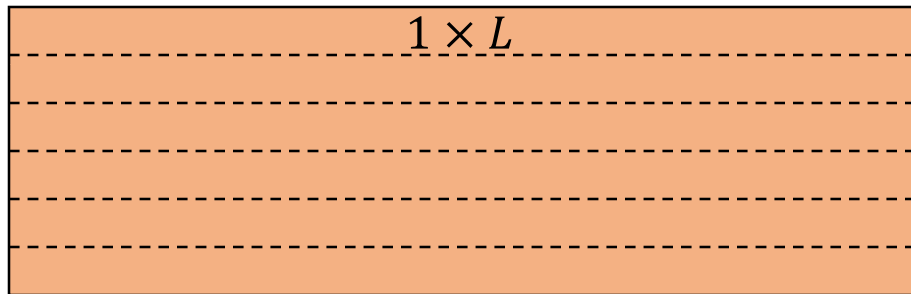
- An  $L$ -word vector register holds a row of  $C$
- 4 registers hold a  $4 \times L$  panel of  $C$
- 16 registers hold a  $16 \times L$  panel of  $C$
- An  $L$ -word vector register holds a row of  $B$
- “Some” combination of registers holds a column of  $A$
- We compute  $C_{16 \times L} \leftarrow A_{16 \times 1} \times B_{1 \times L} + C_{16 \times L}$
- Total of  $16L$  multiply-adds
- Minimum time =  $\Delta$  cycles
- Maximum computation rate  $R = \frac{16L}{\Delta} = 4L$  madds/cycle
- This is the upper bound with 16 registers for  $C$
- Total of  $L + 16$  words loaded ( $\frac{L+16}{\Delta}$  words/cycle)
- $\eta = \frac{16L}{L+16} = \left[ \frac{16}{5}, 16 \right)$  madds/word
- This works  $\forall L \geq 4$  elements
- Single- and double-precision are compatible  $\forall L \geq 8$

## Option B: Compute instructions



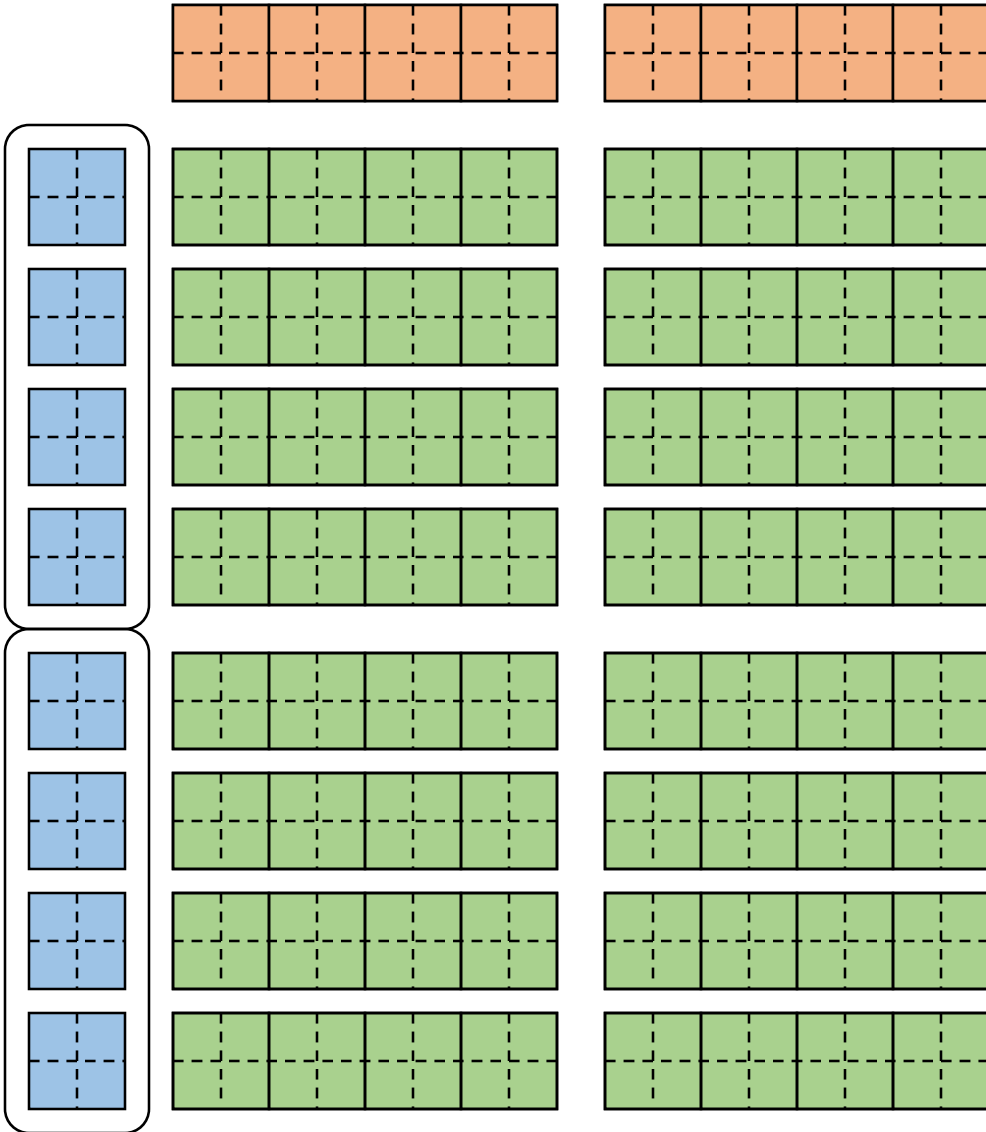
- rank-1 update (outer product)
  - $C_{4 \times L} \leftarrow \pm A_{4 \times 1} \times B_{1 \times L} \pm C_{4 \times L}$
  - Computations:  $4L$  madds/instruction
  - Latency:  $\Delta$
  - Must dispatch/issue 1 computational instruction/operation every cycle to achieve maximum computation rate ( $4L$  madds/cycle)

## Option B: Software impact



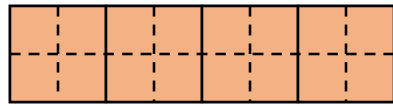
- Conventional BLAS formatting
  - $A$  panel formatted as column-major  $16 \times K$  matrix
  - $B$  panel formatted as row-major  $K \times L$  matrix
  - $C$  panel does not have to be reformatted
- Easier pre/post-processing of rows/columns of matrices
  - Does not require reformatting data from/to vector format to/from matrix format
  - Although not that critical for matrix multiplication, insert/extract of rows from matrix registers from/to vector registers is useful in other algorithms (e.g., DFT)

## Option C: Matrix as an element type

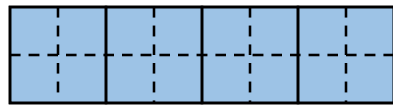


- We fix a  $\lambda$  and define a new “vector element type” – a  $\lambda \times \lambda$  matrix of words (e.g.,  $\lambda = 2$ )
- A vector register of length  $L$  holds  $L/\lambda^2$  of these matrices (e.g.,  $L = 16, L/\lambda^2 = 4$ )
- “Some” number of registers hold an  $8\lambda \times \lambda$  panel of **A** ( $8 \lambda \times \lambda$  matrices)
- 2 registers hold a  $\lambda \times 2L/\lambda$  panel of **B** ( $2L/\lambda^2 \lambda \times \lambda$  matrices)
- 16 registers hold an  $8\lambda \times 2L/\lambda$  panel of **C** ( $16L$  words)
- We compute  $\mathbf{C}_{8\lambda \times 2L/\lambda} \leftarrow \mathbf{A}_{8\lambda \times \lambda} \times \mathbf{B}_{\lambda \times 2L/\lambda} + \mathbf{C}_{8\lambda \times 2L/\lambda}$
- Total of  $16L\lambda$  multiply-adds
- Minimum time =  $\lambda\Delta$  cycles
- Maximum computation rate  $R = \frac{16L}{\Delta} = 4L$  madds/cycle
- This is the upper bound with 16 registers for **C**
- Total of  $2L + 8\lambda^2$  words loaded ( $(2L+8\lambda^2)/\lambda\Delta$  words/cycle)
- $\eta = \frac{16L\lambda}{2L+8\lambda^2} = \lfloor \frac{8\lambda}{5}, 8\lambda \rfloor$  madds/word
- This works  $\forall L \geq \lambda^2$  words
- Single- and double-precision are compatible  $\forall L \geq 2\lambda^2$  words

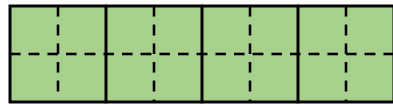
# Option C: Compute instructions



×

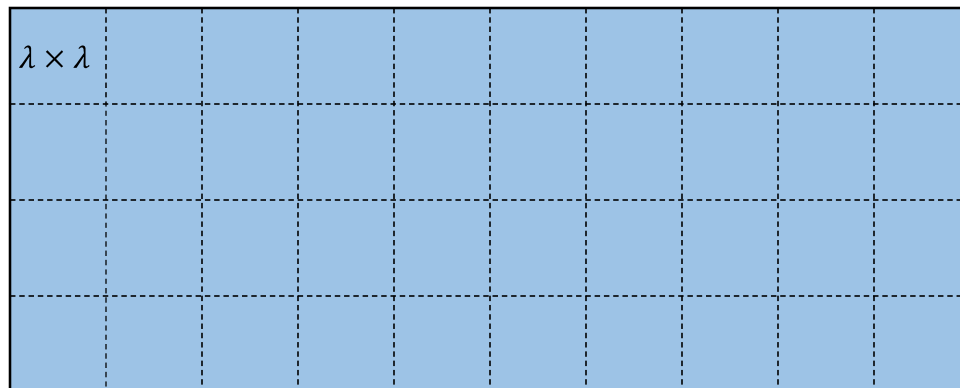
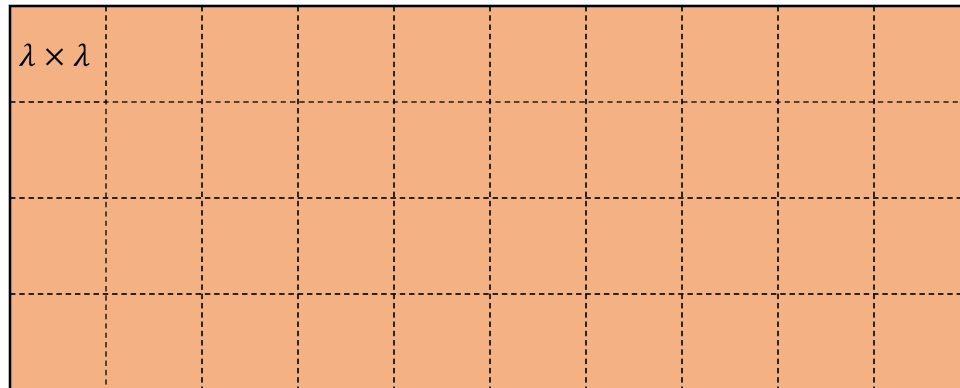


+



- A vector of matrix multiplies
  - $\mathbf{C}_{\lambda \times \lambda} \leftarrow \pm \mathbf{A}_{\lambda \times \lambda} \times \mathbf{B}_{\lambda \times \lambda} \pm \mathbf{C}_{\lambda \times \lambda}$  ( $L/\lambda^2$  times)
  - Computations:  $\lambda L$  madds/instruction
  - Latency:  $\lambda \Delta$
  - Must dispatch/issue  $4/\lambda$  computational instructions/operations every cycle to achieve maximum computation rate ( $4L$  madds/cycle)

## Option C: Software impact



- For Option C to work, both **A** and **B** must be packed into  $\lambda \times \lambda$  blocks – **A** in column-major, **B** in row-major
- Both the compute and packing kernels must be modified to support matrix operations
- **C** panel must also be reformatted for load/store

# Conclusions

---

- The size of the architected register space imposes an upper bound on the performance of deterministic matrix multiply kernels
- If we use the architected scalable vector register space ( $32L$  words) to hold the panel of  $C$  our performance upper bound, for a reasonable choice of latency parameters, is  $8L$  madds/cycle – a more realistic value is  $4L$  madds/cycle (single-precision floating-point)
- We advocate to pursue the vector register-based matrix extensions first, starting with three possible options:
  - Option A : 1 matrix/1 vector register  $(R = 4L \text{ madds/cycle with } 4 \text{ matrix pipes})$
  - Option B : 1 matrix/ $n$  vector registers  $(R = 4L \text{ madds/cycle with } 1 \text{ matrix pipe})$
  - Option C :  $n$  matrices/1 vector register  $(R = 4L \text{ madds/cycle with } 4/\lambda \text{ matrix pipes})$

# Sub-word data types on the inputs

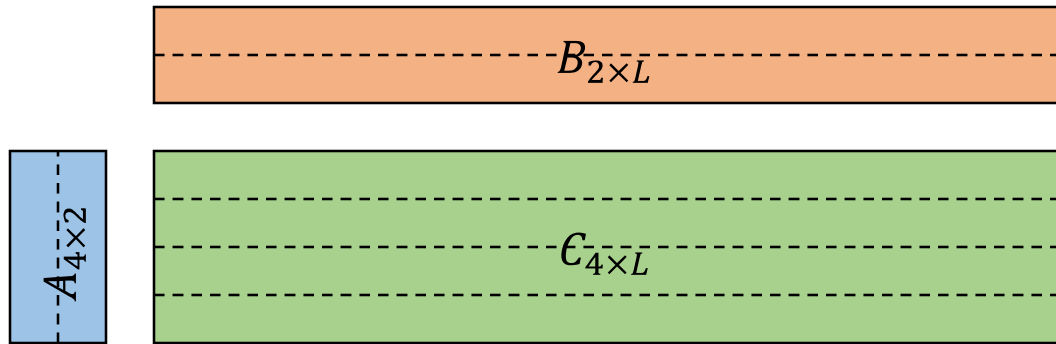
- Usually, the lower precision data types are only used on the inputs
- Result is kept in higher precision (e.g., 32 bits), requiring a *widening*
- 16-bit data types can be packed 2 per word (e.g., Option B with  $L = 4$  words/128-bit vectors)

$$\underbrace{\begin{bmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{bmatrix}}_{\substack{\text{fp32} \\ (64 \text{ bytes})}} = \underbrace{\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \\ A_{20} & A_{21} \\ A_{30} & A_{31} \end{bmatrix}}_{\substack{\text{bfloat16} \\ (16 \text{ bytes})}} \times \underbrace{\begin{bmatrix} B_{00} & B_{10} & B_{20} & B_{30} \\ B_{01} & B_{11} & B_{21} & B_{31} \end{bmatrix}}_{\substack{\text{bfloat16} \\ (16 \text{ bytes})}} + \underbrace{\begin{bmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{bmatrix}}_{\substack{\text{fp32} \\ (64 \text{ bytes})}}$$

- Different ways to compute the result, affecting floating-point and saturating integer arithmetic
  - $C_{ij} = \text{round}(A_{i0}B_{j0} + A_{i1}B_{j1} + C_{ij})$  : only round at the end (true rank-2 update operation)
  - $C_{ij} = \text{round}(\text{round}(A_{i0}B_{j0} + A_{i1}B_{j1}) + C_{ij})$  : round intermediary result
  - $C_{ij} = \text{round}(\text{round}(\text{round}(A_{i0}B_{j0}) + A_{i1}B_{j1}) + C_{ij})$  : round at every step
  - $C_{ij} = \text{round}(\text{round}(C_{ij} + A_{i0}B_{j0}) + A_{i1}B_{j1})$  : scalar algorithm ( $2 \times$  rank-1 update operations)

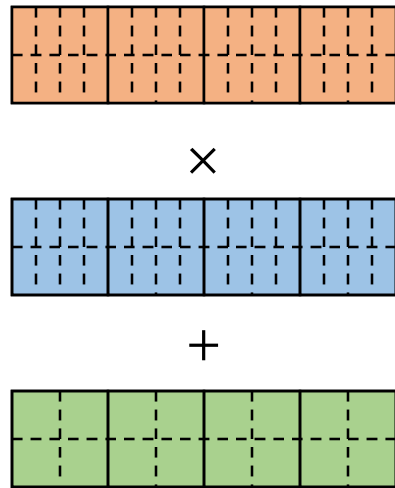


## Option B: Compute instructions (for bfloat16 inputs)



- rank-2 update (two outer products)
  - $C_{4 \times L} \leftarrow \pm A_{4 \times 2} \times B_{2 \times L} \pm C_{4 \times L}$
  - Computations:  $8L$  madds/instruction
  - Latency:  $\Delta$
  - Must dispatch/issue 1 computational instruction/operation every cycle to achieve maximum computation rate ( $8L$  madds/cycle)
  - $A$  : 4 words of a vector register
  - $B$  : 1 vector register of  $L$  words
  - $C$  : 4 vector registers of  $L$  words each

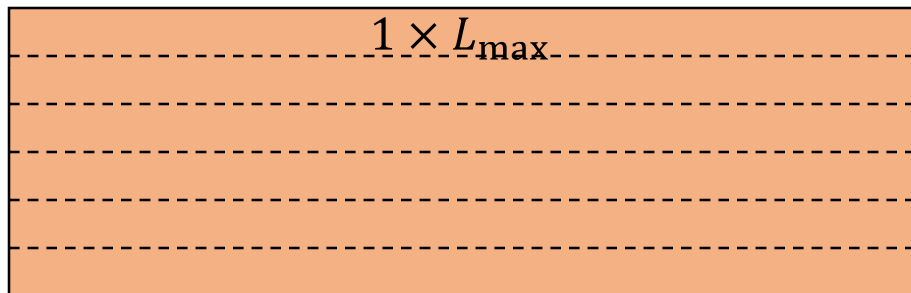
# Option C: Compute instructions (for bfloat16 inputs)



- A vector of matrix multiplies
  - $\mathbf{C}_{\lambda \times \lambda} \leftarrow \pm \mathbf{A}_{\lambda \times 2\lambda} \times \mathbf{B}_{\lambda \times 2\lambda}^T \pm \mathbf{C}_{\lambda \times \lambda}$  ( $L/\lambda^2$  times)
  - Computations:  $2\lambda L$  madds/instruction
  - Latency:  $\lambda\Delta$  (choice of rounding may impact)
  - Must dispatch/issue  $4/\lambda$  computational instructions/operations every cycle to achieve maximum computation rate ( $8L$  madds/cycle)
  - $\mathbf{A}$  : 1 vector register of  $2L$  halfwords
  - $\mathbf{B}$  : 1 vector register of  $2L$  halfwords
  - $\mathbf{C}$  : 1 vector registers of  $L$  words
  - *Note*: Typically, the same matrix in  $\mathbf{A}$  is used for all matrix multiplies, requiring either an implicit or explicit splat

# Upmigration of threads

- Upmigration means moving a running thread from a processor with vector registers of length  $L$  to another processor with vector registers of length  $\bar{L} > L$
- If  $\bar{L} < L$ , there would be loss of architected state
- After upmigration, code for Options B and C (for a fixed  $\lambda$ ) can continue to run with currently computed masks – at next iteration, masks are recomputed and code can use full  $\bar{L}$
- More difficult for Option A, as changing  $L$  changes  $\lambda = \sqrt{L}$
- There is an upper limit on maximum useful  $\bar{L}$ , as dictated by the packing routines



## Some configurations with new architected state (just to get a feel)

---

- Let the architected space for matrices consist of 32 vector registers of size  $L$  elements
  - Then,  $mn \leq 32L$  and the maximum computation rate is  $R = 32L/4 = 8L$  madds/cycle
- Let the architected space consist of 8 accumulators of size  $L \times L$  elements
  - Then,  $mn \leq 8L^2$  and the maximum computation rate is  $R = 8L^2/4 = 2L^2$  madds/cycle
- It is in this sense that the performance of an attached matrix facility, with its own set of accumulators, scales with  $L^2$ , whereas the performance of a vector-based matrix facility, using vector registers only, scales with  $L$
- For  $L = 4$ , both variants have the same upper bound on performance,  
 $R = 32$  madds/cycle
- For  $L = 16$ , the difference is considerable:  
 $R_{\text{VMF}} = 128$  madds/cycle,  $R_{\text{AMF}} = 512$  madds/cycle
- It is true that both 128 and 512 are big performance numbers, but this explains why we need to consider both variants for matrix extensions to RISC-V
- Note: In the above, we use all architected space for **C**, but in a load/store architecture we need to reserve space for **A** and **B**, so cut the bound in half