Integrated Matrix Extension Option C (Common-Type Variant)

José E. Moreira (IBM)

February 2024

1 Introduction

This is a strawman for Option C (Multiple fixed-size matrix tiles per vector register) of the Integrated Matrix Extension (IME). As such, it is meant to guide and facilitate discussion by providing a concrete draft specification. There should be no expectation that any content in the strawman will make into the final specification, particularly in the early stages of the work. As we evolve our work in the IME Task Group, refined concepts from this strawman may be promoted to an actual draft specification.

This document focuses on the *common-type* variant of the extensions. That is, all matrix data types involved in one instruction are the same. This is distinct from a *mixed-type* variant that supports matrices of different data types in one instruction.

Notation: Matrices are represented by bold-face capital letters ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$). The element at row i, column j of matrix $\mathbf{A}$ can be denoted as either $\mathbf{A}(i,j)$ or a_{ij}. $\mathbf{A}(i,:)$ represents the i-th row of matrix $\mathbf{A}$ and $\mathbf{A}(:,j)$ its j-th column. $\mathbf{A}(i:h, j:w)$ denotes the two-dimensional section of matrix $\mathbf{A}$ consisting of the h rows beginning in row i and the w columns beginning in column j.

2 Matrix tiles

A *matrix tile* is a contiguous rectangular section of a matrix. Matrix tiles are defined by the scalar data type of their elements (same as the type of the matrix elements) and their *shape* $\lambda \times \kappa$, where λ is the number of rows of the tile and κ is the number of columns.

The shape of the tiles stored in a vector register depends on two vector configuration parameters: The *selected element width* (SEW) and the *matrix element width* (MEW). The SEW is an existing configuration parameter in the RISC-V "V" Vector Extension (RVV), but we extend it to include four additional widths: 128, 256, 512, and 1024 bits. These additional widths, together with some of the existing widths, are intended to store matrices of elements. (We do not expect those extended widths to be supported by any scalar instruction.) We introduce the new MEW parameter to represent the width in bits of an individual matrix element. The number of elements of width MEW in a matrix of width SEW is given by $\frac{\mathsf{SEW}}{\mathsf{MEW}}$.

Table 1 shows the tile shapes ($\lambda \times \kappa$) for all valid values of SEW and MEW. We must always have either $\kappa = \lambda$ or $\kappa = 2\lambda$. A vector registers of VLEN bits stores up to $\frac{\mathsf{VLEN}}{\mathsf{SEW}}$ tiles.

3 Matrix instructions

All matrix instructions in the common-type variant operate on matrix tiles of a common scalar data type. Each architected vector register stores a vector of matrix tiles, and instructions take as arguments either one, two, or three vector register identifiers. Tiles in a vector register $\mathbf{A}$ are identified as $\mathbf{A}[0], \mathbf{A}[1], \ldots, \mathbf{A}[\mathsf{vlene} - 1]$. Additional arguments to matrix instructions can include *index registers*, containing unsigned integer values, and/or vector masks.

	MEW			
SEW	8	16	32	64
32	2×2	1×2	–	–
64	2×4	2×2	1×2	–
128	4×4	2×4	2×2	1×2
256	4×8	4×4	2×4	2×2
512	8×8	4×8	4×4	2×4
1024	8×16	8×8	4×8	4×4

Table 1: Matrix tile shapes for different combinations of selected element width (SEW) and matrix element widht (MEW). For each case, we show the shape of matrix tile ($\kappa \times \lambda$).

3.1 Matrix computation instructions

Matrix computation instructions have the general form

$$\mathsf{m\{comp\}}\langle T, \mathsf{SEW}, \otimes, \oplus\rangle(\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots)$$

where m identifies this as a matrix instruction and comp identifies the specific computation, usually following the nomenclature established by BLAS. T is the matrix element data type, which specifies MEW. The MEW, together with the SEW parameter specifies the $\lambda \times \kappa$ shape of the tiles. The multiplication/addition pair $(\otimes, \oplus)$ forms a semiring on type T. (The precision of the operations may be different than the precision of the data type. For example, addition and multiplication for data type bf16 can be performed in IEEE single-precision.) $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are vector register identifiers that are used to store the tiles from matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ used in the operation, respectively. $\mathbf{A}$ and $\mathbf{B}$ are input matrices. $\mathbf{C}$ is the result matrix of the computation. A matrix instruction specifies a single group multiplier LMUL. The effective multipliers for each matrix ($\mathsf{EMUL}(\mathbf{A})$, $\mathsf{EMUL}(\mathbf{B})$, and $\mathsf{EMUL}(\mathbf{C})$) are derived from LMUL. For the rest of this document we adopt $\mathsf{LMUL} = 1$.

3.1.1 mgemm – matrix multiplication

Syntax: $\mathsf{mgemm}\langle T, \mathsf{SEW}, \otimes, \oplus\rangle(\mathbf{A}, \mathbf{B}, \mathbf{C})$

Arguments: $\mathbf{A}$ is a vector register ($\mathsf{EMUL}(\mathbf{A}) = 1$) with vlene matrix tiles of shape $\lambda \times \kappa$. $\mathbf{B}$ is a pair of vector registers ($\mathsf{EMUL}(\mathbf{B}) = 2$) with $2 \times$ vlene matrix tiles of shape $\lambda \times \kappa$. If $\kappa = \lambda$, only the first vlene tiles of $\mathbf{B}$ are used by the instruction. $\mathbf{C}$ is a vector register ($\mathsf{EMUL}(\mathbf{C}) = 1$) with vlene matrix tiles of shape $\lambda \times \kappa$.

Semantics: If $\kappa = \lambda$, this instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \mathbf{A}[i] \overset{\oplus}{\otimes} \mathbf{B}[i]^{\mathsf{T}}, \forall i \in [0, \mathsf{vlene})$. If $\kappa = 2\lambda$, this instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \left[\mathbf{A}[i] \overset{\oplus}{\otimes} \mathbf{B}[2i]^{\mathsf{T}}, \mathbf{A}[i] \overset{\oplus}{\otimes} \mathbf{B}[2i+1]^{\mathsf{T}}\right], \forall i \in [0, \mathsf{vlene})$. (Since $\mathbf{A}[i] \overset{\oplus}{\otimes} \mathbf{B}[j]^{\mathsf{T}}$ has shape $\lambda \times \lambda$, we can pack two of those results in a tile of shape $\lambda \times \kappa$.)

Example: Let $\mathsf{VLEN} = 256$, $\mathsf{SEW} = 128$, and $\mathsf{MEW} = 64$. In this case, vector registers $\mathbf{A}$ and $\mathbf{C}$ contain two 1×2 tiles of 64-bit elements each. The vector register pair $\mathbf{B}$ contains four 1×2 tiles of 64-bit elements. The computation performed is illustrated as follows:

$$\begin{array}{r|c}
 & \mathbf{B}[0] : [\ b_{00} \quad b_{10}\ ] \quad \mathbf{B}[1] : [\ b_{01} \quad b_{11}\ ] \quad \mathbf{B}[2] : [\ b_{02} \quad b_{12}\ ] \quad \mathbf{B}[3] : [\ b_{03} \quad b_{13}\ ] \\
\hline
\mathbf{A}[0] : [\ a_{00} \quad a_{01}\ ] & \mathbf{C}[0] : [\ c_{00} + a_{00}b_{00} + a_{01}b_{10} \quad c_{01} + a_{00}b_{01} + a_{01}b_{11}\ ] \\
\mathbf{A}[1] : [\ a_{10} \quad a_{11}\ ] & \mathbf{C}[1] : [\ c_{12} + a_{10}b_{02} + a_{11}b_{12} \quad c_{13} + a_{10}b_{03} + a_{11}b_{13}\ ]
\end{array}$$

3.1.2 mgemm0 – matrix multiplication with $\mathbf{A}[0]$

Syntax: $\mathsf{mgemm}\langle T, \mathsf{SEW}, \otimes, \oplus\rangle(\mathbf{A}, \mathbf{B}, \mathbf{C})$

Arguments: $\mathbf{A}$ is a vector register ($\mathsf{EMUL}(\mathbf{A}) = 1$) with vlene matrix tiles of shape $\lambda \times \kappa$. $\mathbf{B}$ is a pair of vector registers ($\mathsf{EMUL}(\mathbf{B}) = 2$) with $2 \times$ vlene matrix tiles of shape $\lambda \times \kappa$. If $\kappa = \lambda$, only the first vlene tiles of $\mathbf{B}$ are used by the instruction. $\mathbf{C}$ is a vector register ($\mathsf{EMUL}(\mathbf{C}) = 1$) with vlene matrix tiles of shape $\lambda \times \kappa$.

Semantics: If $\kappa = \lambda$, this instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \mathbf{A}[0] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[i]^{\mathsf{T}}, \forall i \in [0, \mathsf{vlene})$. If $\kappa = 2\lambda$, this instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \left[\mathbf{A}[0] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[2i]^{\mathsf{T}}, \mathbf{A}[0] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[2i+1]^{\mathsf{T}}\right], \forall i \in [0, \mathsf{vlene})$. (Since $\mathbf{A}[0] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[j]^{\mathsf{T}}$ has shape $\lambda \times \lambda$, we can pack two of those results in a tile of shape $\lambda \times \kappa$.)

3.1.3 mgemmx – matrix multiplication with $\mathbf{A}[x]$

Syntax: $\mathsf{mgemm}\langle T, \mathsf{SEW}, \otimes, \oplus\rangle(\mathbf{A}, \mathbf{B}, \mathbf{C}, x)$

Arguments: $\mathbf{A}$ is a vector register ($\mathsf{EMUL}(\mathbf{A}) = 1$) with vlene matrix tiles of shape $\lambda \times \kappa$. $\mathbf{B}$ is a pair of vector registers ($\mathsf{EMUL}(\mathbf{B}) = 2$) with $2 \times$ vlene matrix tiles of shape $\lambda \times \kappa$. If $\kappa = \lambda$, only the first vlene tiles of $\mathbf{B}$ are used by the instruction. $\mathbf{C}$ is a vector register ($\mathsf{EMUL}(\mathbf{C}) = 1$) with vlene matrix tiles of shape $\lambda \times \kappa$. The index register x identifies the matrix tile $\mathbf{A}[x], x \in [0, \mathsf{vlene})$.

Semantics: If $\kappa = \lambda$, this instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \mathbf{A}[x] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[i]^{\mathsf{T}}, \forall i \in [0, \mathsf{vlene})$. If $\kappa = 2\lambda$, this instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \left[\mathbf{A}[x] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[2i]^{\mathsf{T}}, \mathbf{A}[x] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[2i+1]^{\mathsf{T}}\right], \forall i \in [0, \mathsf{vlene})$. (Since $\mathbf{A}[x] \overset{\oplus}{\underset{\otimes}{}} \mathbf{B}[j]^{\mathsf{T}}$ has shape $\lambda \times \lambda$, we can pack two of those results in a tile of shape $\lambda \times \kappa$.)

4 Case study: dgemm

The BLAS dgemm routine computes $\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$, where $\mathbf{C}$ is a $M \times N$ matrix, $\mathbf{A}$ is a $M \times K$ matrix, $\mathbf{B}$ is a $K \times N$ matrix, α and β are scalars. All data types are double-precision floating-point numbers. (We are ignoring the matrix transpose variants.)

Pseudo code for a vlen agnostic implementation of dgemm is shown in Figure 1. It consists of a double-nested parallel loop across the rows and columns of result matrix $\mathbf{C}$, Each iteration of the loop nest computes an $m \times n$ panel of $\mathbf{C}$ in the *micro-kernel* μdgemm, with $m = 4\lambda$ and $n = 4\kappa\mathsf{vlene}$. (For simplicity, we let M and N be integer multiples of m and n, respectively, so there are no loop remainders.)

Pseudo code for the micro-kernel in shown in Figure 2. It corresponds to the mapping of matrices to vector registers shown in Figure 3. The code, including the corresponding binary code, produces the correct result independent of the values of vlen and SEW, as long as they are both valid. We call this code vlen agnostic.

The micro-kernel works as follows. We first zero the contents of the $m \times n$ panel of $\mathbf{C}$ in registers v16–v31, using standard RISC-V vector instructions. We then iterate over the inner dimension. The outer loop proceeds in chunks of κvlene, while the inner loop proceeds in chunks of κ. At each iteration of the outer (k) loop, we load 4vlene tiles of matrix $\mathbf{A}$ in registers v04–v07. This can be accomplished either with existing RISC-V indexed vector loads or with a new two-dimensional strided vector load. At each iteration of the inner (x) loop, we process one tile of $\mathbf{A}$ from each of those registers. At the beginning of each x iteration we load a $\kappa \times 4\kappa$vlene panel of $\mathbf{B}$, corresponding to either 4vlene (if $\kappa = \lambda$) or 8vlene (if $\kappa = 2\lambda$) tiles of shape $\lambda \times \kappa$. These can also be viewed as 4vlene tiles of shape $\kappa \times \kappa$. We then multiply each of the 4 tiles of shape $\lambda \times \kappa$ in position x of registers v04–v07 by the 4vlene tiles of shape $\kappa \times \kappa$ in registers v08–v15. This updates the 16vlene tiles of $\mathbf{C}$ in registers v16–v31. At the end of the loop we will have computed $\mathbf{AB}$. The scaling by scalar α can be accomplished with existing RISC-V vector instructions, as can the addition of

```
procedure dgemm($M, N, K, \alpha, \mathbf{A}, \mathbf{B}, \beta, \mathbf{C}$)
    $m \leftarrow 4\lambda$
    $n \leftarrow 4\kappa$
    foreach($I \leftarrow 0; I < M; I \leftarrow I + m$)
        foreach($J \leftarrow 0; J < N; J \leftarrow J + n$)
            $\mu$dgemm($K, \alpha, \mathbf{A}(I : m. :), \mathbf{B}(:, J : n), \beta, \mathbf{C}(I : m, J : n)$)
        end
    end
end
```

Figure 1: The (M, N, K) BLAS routine dgemm can be implemented as a two-dimensional parallel loop, each iteration computing a $m \times n$ panel of the result matrix $\mathbf{C}$. The computation of each panel is performed by a micro-kernel that takes as input a block of m rows of $\mathbf{A}$ and a block of n columns of $\mathbf{B}$.

$\beta\mathbf{C}$. The loading and storing of the panel of $\mathbf{C}$ can again use either indexed vector load instructions or a new two-dimensional strided vector load.

The binary code in Figure 2 is agnostic to the value of vlen, producing the correct result in every case. The computational intensity (η), defined as the ratio of floating-point operations by the number of elements transferred, can be computed as follows. Each iteration of the outer loop of Figure 2 loads $4\lambda\kappa\mathsf{vlene}$ elements of matrix $\mathbf{A}$. Each iteration of the inner loop of Figure 2 loads $4\kappa^2\mathsf{vlene}$ elements of matrix $\mathbf{B}$. Therefore, the total number of elements loaded in each iteration of the outer loop is $4\kappa\mathsf{vlene}(\lambda + \kappa\mathsf{vlene})$. Each mgemmx instruction in the inner loop performs $\lambda\kappa^2$ multiply-adds for each tile of matrix $\mathbf{C}$ produced, or $\lambda\kappa^2\mathsf{vlene}$ multiply-adds per instruction. Since there are 16 instructions in the inner loop body and that body is executed vlene time per iteration of the outer loop, we perform $32\lambda\kappa^2\mathsf{velene}^2$ multiply-adds per iteration of the outer loop. Therefore, the computational intensity is

$$\eta = \frac{32\lambda\kappa^2\mathsf{vlene}^2}{4\kappa\mathsf{vlene}(\lambda + \kappa\mathsf{vlene})} = 8\frac{\lambda\kappa\mathsf{vlene}}{\lambda + \kappa\mathsf{vlene}} = \left\{ \begin{array}{ll} 8\frac{\lambda\mathsf{vlene}}{1+\mathsf{vlene}} & (\kappa = \lambda) \\ 16\frac{\lambda\mathsf{vlene}}{1+2\mathsf{vlene}} & (\kappa = 2\lambda) \end{array} \right. .$$

The computational intensity scales with λ, which is proportional to the square root of SEW. Furthermore, the computational intensity is highest when the tiles are square. At the limit $\mathsf{vlene} \to \infty$, both computational intensities converge to 8λ. At the other extreme, when $\mathsf{vlene} = 1$, the computational intensities are 4λ $(\kappa = \lambda)$ and $\frac{16}{3}\lambda$ $(\kappa = 2\lambda)$.

```
procedure $\mu$dgemm$(K, \alpha, \mathbf{A}, \mathbf{B}, \beta, \mathbf{C})$
    $\begin{bmatrix} \mathsf{v16} & \mathsf{v17} & \mathsf{v18} & \mathsf{v19} \\ \mathsf{v20} & \mathsf{v21} & \mathsf{v22} & \mathsf{v23} \\ \mathsf{v24} & \mathsf{v25} & \mathsf{v26} & \mathsf{v27} \\ \mathsf{v28} & \mathsf{v29} & \mathsf{v30} & \mathsf{v31} \end{bmatrix} \leftarrow 0$
    for$(k \leftarrow 0; k < K; k \leftarrow k + \kappa \times \mathsf{vlene})$
        $\mathsf{v04} \leftarrow \mathbf{A}(0 \times \lambda : \lambda, k : \kappa \times \mathsf{vlene})$
        $\mathsf{v05} \leftarrow \mathbf{A}(1 \times \lambda : \lambda, k : \kappa \times \mathsf{vlene})$
        $\mathsf{v06} \leftarrow \mathbf{A}(2 \times \lambda : \lambda, k : \kappa \times \mathsf{vlene})$
        $\mathsf{v07} \leftarrow \mathbf{A}(3 \times \lambda : \lambda, k : \kappa \times \mathsf{vlene})$
        for$(x \leftarrow 0; x < \mathsf{vlene}; x \leftarrow x + 1)$
            $(\mathsf{v08}, \mathsf{v09}) \leftarrow \mathbf{B}(k : \kappa, 0 \times \kappa \times \mathsf{vlene} : \kappa \times \mathsf{vlene})$
            $(\mathsf{v10}, \mathsf{v11}) \leftarrow \mathbf{B}(k : \kappa, 1 \times \kappa \times \mathsf{vlene} : \kappa \times \mathsf{vlene})$
            $(\mathsf{v12}, \mathsf{v13}) \leftarrow \mathbf{B}(k : \kappa, 2 \times \kappa \times \mathsf{vlene} : \kappa \times \mathsf{vlene})$
            $(\mathsf{v14}, \mathsf{v15}) \leftarrow \mathbf{B}(k : \kappa, 3 \times \kappa \times \mathsf{vlene} : \kappa \times \mathsf{vlene})$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v04}, (\mathsf{v08}, \mathsf{v09}), \mathsf{v16}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v04}, (\mathsf{v10}, \mathsf{v11}), \mathsf{v17}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v04}, (\mathsf{v12}, \mathsf{v13}), \mathsf{v18}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v04}, (\mathsf{v14}, \mathsf{v15}), \mathsf{v19}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v05}, (\mathsf{v08}, \mathsf{v09}), \mathsf{v20}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v05}, (\mathsf{v10}, \mathsf{v11}), \mathsf{v21}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v05}, (\mathsf{v12}, \mathsf{v13}), \mathsf{v22}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v05}, (\mathsf{v14}, \mathsf{v15}), \mathsf{v23}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v06}, (\mathsf{v08}, \mathsf{v09}), \mathsf{v24}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v06}, (\mathsf{v10}, \mathsf{v11}), \mathsf{v25}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v06}, (\mathsf{v12}, \mathsf{v13}), \mathsf{v26}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v06}, (\mathsf{v14}, \mathsf{v15}), \mathsf{v27}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v07}, (\mathsf{v08}, \mathsf{v09}), \mathsf{v28}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v07}, (\mathsf{v10}, \mathsf{v11}), \mathsf{v29}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v07}, (\mathsf{v12}, \mathsf{v13}), \mathsf{v30}, x)$
            $\mathsf{mgemmx}\langle \mathsf{fp64}, \mathsf{SEW}, \times, + \rangle(\mathsf{v07}, (\mathsf{v14}, \mathsf{v15}), \mathsf{v31}, x)$
        end
    end
    $(\mathsf{v16}, \mathsf{v17}, \ldots, \mathsf{v31}) \leftarrow \alpha \times (\mathsf{v16}, \mathsf{v17}, \ldots, \mathsf{v31})$
    $\mathbf{C} \leftarrow \begin{bmatrix} \mathsf{v16} & \mathsf{v17} & \mathsf{v18} & \mathsf{v19} \\ \mathsf{v20} & \mathsf{v21} & \mathsf{v22} & \mathsf{v23} \\ \mathsf{v24} & \mathsf{v25} & \mathsf{v26} & \mathsf{v27} \\ \mathsf{v28} & \mathsf{v29} & \mathsf{v30} & \mathsf{v31} \end{bmatrix} + \beta \times \mathbf{C}$
end
```

Figure 2: The micro-kernel of the BLAS routine dgemm computes a $4\lambda \times 4\kappa\mathsf{vlene}$ panel of the result matrix **C**. The body of the innermost loop is executed once for each tile of the **A** registers. In the body, it updates $16 \times \mathsf{vlen}$ tiles of **C**, each with either one or two products of a tile of **A** and a tile of **B**.

$$
\begin{array}{c|cccc}
 & \overset{\text{v08}}{\begin{bmatrix} B_{0,0} & B_{0,1} \end{bmatrix}} & \overset{\text{v10}}{\begin{bmatrix} B_{0,4} & B_{0,5} \end{bmatrix}} & \overset{\text{v12}}{\begin{bmatrix} B_{0,8} & B_{0,9} \end{bmatrix}} & \overset{\text{v14}}{\begin{bmatrix} B_{0,12} & B_{0,13} \end{bmatrix}} \\
 & \overset{\text{v09}}{\begin{bmatrix} B_{0,2} & B_{0,3} \end{bmatrix}} & \overset{\text{v11}}{\begin{bmatrix} B_{0,6} & B_{0,7} \end{bmatrix}} & \overset{\text{v13}}{\begin{bmatrix} B_{0,10} & B_{0,11} \end{bmatrix}} & \overset{\text{v15}}{\begin{bmatrix} B_{0,14} & B_{0,15} \end{bmatrix}} \\
\hline
\overset{\text{v04}}{\begin{bmatrix} A_{0,0} & A_{0,1} \end{bmatrix}} & \overset{\text{v16}}{\begin{bmatrix} C_{0,0} & C_{0,1} \end{bmatrix}} & \overset{\text{v17}}{\begin{bmatrix} C_{0,2} & C_{0,3} \end{bmatrix}} & \overset{\text{v18}}{\begin{bmatrix} C_{0,4} & C_{0,5} \end{bmatrix}} & \overset{\text{v19}}{\begin{bmatrix} C_{0,6} & C_{0,7} \end{bmatrix}} \\
\overset{\text{v05}}{\begin{bmatrix} A_{1,0} & A_{1,1} \end{bmatrix}} & \overset{\text{v20}}{\begin{bmatrix} C_{1,0} & C_{1,1} \end{bmatrix}} & \overset{\text{v21}}{\begin{bmatrix} C_{1,2} & C_{1,3} \end{bmatrix}} & \overset{\text{v22}}{\begin{bmatrix} C_{1,4} & C_{1,5} \end{bmatrix}} & \overset{\text{v23}}{\begin{bmatrix} C_{1,6} & C_{1,7} \end{bmatrix}} \\
\overset{\text{v06}}{\begin{bmatrix} A_{2,0} & A_{2,1} \end{bmatrix}} & \overset{\text{v24}}{\begin{bmatrix} C_{2,0} & C_{2,1} \end{bmatrix}} & \overset{\text{v25}}{\begin{bmatrix} C_{2,2} & C_{2,3} \end{bmatrix}} & \overset{\text{v26}}{\begin{bmatrix} C_{2,4} & C_{2,5} \end{bmatrix}} & \overset{\text{v27}}{\begin{bmatrix} C_{2,6} & C_{2,7} \end{bmatrix}} \\
\overset{\text{v07}}{\begin{bmatrix} A_{3,0} & A_{3,1} \end{bmatrix}} & \overset{\text{v28}}{\begin{bmatrix} C_{3,0} & C_{3,1} \end{bmatrix}} & \overset{\text{v29}}{\begin{bmatrix} C_{3,2} & C_{3,3} \end{bmatrix}} & \overset{\text{v30}}{\begin{bmatrix} C_{3,4} & C_{3,5} \end{bmatrix}} & \overset{\text{v31}}{\begin{bmatrix} C_{3,6} & C_{3,7} \end{bmatrix}}
\end{array}
$$

Figure 3: Register layout for the dgemm micro-kernel. Each inner-most iteration of the micro-kernel updates a $4\lambda \times 4\kappa\mathsf{vlene}$ panel of **C** with the product of a $4\lambda \times \kappa$ panel of **A** by a $\kappa \times 4\kappa\mathsf{vlene}$ panel of **B**. There are 8 vectors assigned to **B** (v08–v15), 4 vectors assigned to **A** (v04–v07), and 16 registers assigned to **C** (v16–v31). If $\kappa = 2\lambda$, all 8 registers assigned to **B** are used. Otherwise, only 4 of those registers are used. The subscripted A, B, and C represent $\lambda \times \kappa$ tiles of matrices **A**, **B**, and **C**, respectively.