Integrated Matrix Extension Option C
(Common-Type Variant)

José E. Moreira (IBM)

May 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 document. 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_i^j. $\mathbf{A}(i,:)$ 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* $\lambda \times \lambda$, where λ is the number of rows and columns of the tile. All tiles supported in this variant of Option C are square.

Notation: The tile A_i^j corresponds to the matrix section $\mathbf{A}(i\lambda:+\lambda, j\lambda:+\lambda)$.

The shape ($\lambda \times \lambda$) and number (L) of tiles stored in a vector register is an implementation choice for each elemental data type, but it mus follow the following constraint. Given a vector length (VLEN) and a *matrix element width* (MEW), the vector length must match the element width times the tile size ($\lambda \times \lambda$ elements) times the number of tiles in the vector register. In other words, the following equality must hold:

$$\mathsf{VLEN} = \mathsf{MEW} \cdot \lambda^2 \cdot L.$$

Table 1 shows valid values of the $\langle \lambda, L \rangle$ pair for different combinations of VLEN and MEW. We will show below how to generate binary code that is agnostic to the choice of VLEN, λ, and L, and therefore portable, in both functionality and performance, across any implementation that follows the specification outlined in this document.

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 $\mathbf{A}[0], \mathbf{A}[1], \ldots, \mathbf{A}[L-1]$. Additional arguments to matrix instructions can include *index registers*, containing unsigned integer values, and/or vector masks.

$\langle \lambda, L \rangle$

	MEW			
VLEN	8	16	32	64
32	$\langle 2,1 \rangle$	–	–	–
64	$\langle 2,2 \rangle$	$\langle 2,1 \rangle$	–	–
128	$\langle 2,4 \rangle$, $\langle 4,1 \rangle$	$\langle 2,2 \rangle$	$\langle 2,1 \rangle$	–
256	$\langle 2,8 \rangle$, $\langle 4,2 \rangle$	$\langle 2,4 \rangle$, $\langle 4,1 \rangle$	$\langle 2,2 \rangle$	$\langle 2,1 \rangle$
512	$\langle 2,16 \rangle$, $\langle 4,4 \rangle$, $\langle 8,1 \rangle$	$\langle 2,8 \rangle$, $\langle 4,2 \rangle$	$\langle 2,4 \rangle$, $\langle 4,1 \rangle$	$\langle 2,2 \rangle$
1024	$\langle 2,32 \rangle$, $\langle 4,8 \rangle$, $\langle 8,2 \rangle$	$\langle 2,16 \rangle$, $\langle 4,4 \rangle$, $\langle 8,1 \rangle$	$\langle 2,8 \rangle$, $\langle 4,2 \rangle$	$\langle 2,4 \rangle$, $\langle 4,1 \rangle$
2048	$\langle 2,64 \rangle$, $\langle 4,16 \rangle$, $\langle 8,4 \rangle$, $\langle 16,1 \rangle$	$\langle 2,32 \rangle$, $\langle 4,8 \rangle$, $\langle 8,2 \rangle$	$\langle 2,16 \rangle$, $\langle 4,4 \rangle$, $\langle 8,1 \rangle$	$\langle 2,8 \rangle$, $\langle 4,2 \rangle$

Table 1: Valid values of the $\langle \lambda, L \rangle$ pair for different combinations of matrix element width (MEW) and vector length (VLEN). An implementation must choose one of the valid pairs. For example, when processing single-precision floating-point data (MEW = 32), 512-bit vectors (VLEN = 512) can contain either 4 tiles of 2×2 elements ($\langle 2,4 \rangle$), or 1 tile of 4×4 elements ($\langle 4,1 \rangle$).

3.1 Matrix load instructions

Matrix load instructions have the general form

$$\mathsf{mload}\{\mathsf{variant}\}\langle T, \ldots \rangle(\mathsf{vd}, \mathbf{A}(i:+h, j:+w))$$

where mload identifies this as a matrix load instruction and variant is a specific variant. T is the matrix element data type, which specifies MEW. $\mathbf{A}(i:+h, j:+w)$ is the section of matrix $\mathbf{A}$ to be loaded in the destination register group beginning with vd. Matrix load instructions support different forms of group multipliers, to be specified for each variant.

3.1.1 mload – matrix load (vanilla version)

Syntax: $\mathsf{mload}\langle T, \mathsf{RMUL}, \mathsf{maxrows}, \mathsf{CMUL}, \mathsf{maxcols} \rangle(\mathsf{vd}, \mathbf{A}(i,j), \mathsf{lda})$

Arguments: $\mathbf{A}(i,j)$ is the address of the first element of the section of matrix $\mathbf{A}$ to be loaded in the register group beginning with register vd. The leading dimension of $\mathbf{A}$, necessary to compute the address of the other elements in the section, is passed as lda.

Semantics: This instruction takes the $\mathsf{LMUL} = \mathsf{RMUL} \cdot \mathsf{CMUL}$ vector registers starting at register vd and organizes them as a $\mathsf{RMUL} \times \mathsf{CMUL}$ two-dimensional array of registers, capable of holding up to $\mathsf{RMUL} \cdot \mathsf{CMUL} \cdot \lambda^2 L$ elements of type T. It then loads the array section $\mathbf{A}(i:+\max(\mathsf{maxrows}, \mathsf{RMUL} \cdot \lambda), j:+\max(\mathsf{maxcols}, \mathsf{CMUL} \cdot \lambda\mathsf{L}))$ into those registers. Matrix $\mathbf{A}$ is interpreted as being stored in row-major order, with a leading dimension of lda. Any elements of the target register group that are not loaded from $\mathbf{A}$ are set to 0 (or, in the general case, a specified identity value).

Example: Let VLEN = 512 and MEW = 64. Consider the instruction

$$\mathsf{mload}\langle \mathsf{fp64}, 2, \infty, 2, \infty \rangle(\mathsf{v00}, \mathbf{A}(0,0), \mathsf{lda}).$$

In this case, $\langle \lambda, L \rangle$ must be $\langle 2,2 \rangle$ and vector registers v00, v01, v02, v03 are loaded with a 4×8 section of matrix $\mathbf{A}$ as follows:

$$\mathsf{v00} = \left[\begin{bmatrix} a_0^0 & a_0^1 \\ a_1^0 & a_1^1 \end{bmatrix} \begin{bmatrix} a_0^2 & a_0^3 \\ a_1^2 & a_1^3 \end{bmatrix} \right] \quad \mathsf{v01} = \left[\begin{bmatrix} a_0^4 & a_0^5 \\ a_1^4 & a_1^5 \end{bmatrix} \begin{bmatrix} a_0^6 & a_0^7 \\ a_1^6 & a_1^7 \end{bmatrix} \right]$$

$$\mathsf{v02} = \left[\begin{bmatrix} a_2^0 & a_2^1 \\ a_3^0 & a_3^1 \end{bmatrix} \begin{bmatrix} a_2^2 & a_2^3 \\ a_3^2 & a_3^3 \end{bmatrix} \right] \quad \mathsf{v03} = \left[\begin{bmatrix} a_2^4 & a_2^5 \\ a_3^4 & a_3^5 \end{bmatrix} \begin{bmatrix} a_2^6 & a_2^7 \\ a_3^6 & a_3^7 \end{bmatrix} \right]$$

3.2 Matrix computation instructions

Matrix computation instructions have the general form

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

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 implementation selects a valid $\langle \lambda, L\rangle$ pair compatible with its VLEN. The code will be agnostic to this choice. 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 $\mathsf{bf16}$ 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 common-type variant $\mathsf{EMUL}(\mathbf{A}) = \mathsf{EMUL}(\mathbf{B}) = \mathsf{EMUL}(\mathbf{C}) = \mathsf{LMUL}$. By default, $\mathsf{LMUL} = 1$.

3.2.1 mgemm – matrix multiplication

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

Arguments: $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are vector registers, each with L matrix tiles of shape $\lambda \times \lambda$.

Semantics: This instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \mathbf{A}[i] {}^{\oplus}_{\otimes} \mathbf{B}[i], \forall i \in [0, L)$.

Example: Let $\mathsf{VLEN} = 512$ and $\mathsf{MEW} = 64$. In this case, $\langle \lambda, L\rangle$ must be $\langle 2, 2\rangle$ and vector registers $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ each contain two 2×2 tiles of 64-bit elements. The computation performed is illustrated as follows:

$$\begin{array}{c|c}
 & \mathbf{B}[0] : \begin{bmatrix} b_0^0 & b_0^1 \\ b_1^0 & b_1^1 \end{bmatrix} \quad \mathbf{B}[1] : \begin{bmatrix} b_0^2 & b_0^3 \\ b_1^2 & b_1^3 \end{bmatrix} \\
\hline
\mathbf{A}[0] : \begin{bmatrix} a_0^0 & a_0^1 \\ a_1^0 & a_1^1 \end{bmatrix} & \mathbf{C}[0] : \begin{bmatrix} c_0^0 + a_0^0 b_0^0 + a_0^1 b_1^0 & c_0^1 + a_0^0 b_0^1 + a_0^1 b_1^1 \\ c_1^0 + a_1^0 b_0^0 + a_1^1 b_1^0 & c_1^1 + a_1^0 b_0^1 + a_1^1 b_1^1 \end{bmatrix} \\
\mathbf{A}[1] : \begin{bmatrix} a_0^2 & a_0^3 \\ a_1^2 & a_1^3 \end{bmatrix} & \mathbf{C}[1] : \begin{bmatrix} c_0^2 + a_0^2 b_0^2 + a_0^3 b_1^2 & c_0^3 + a_0^2 b_0^3 + a_0^3 b_1^3 \\ c_1^2 + a_1^2 b_0^2 + a_1^3 b_1^2 & c_0^3 + a_1^2 b_0^3 + a_1^3 b_1^3 \end{bmatrix}
\end{array}$$

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

Syntax: $\mathsf{mgemm0}\langle T, \otimes, \oplus\rangle(\mathbf{A}, \mathbf{B}, \mathbf{C})$

Arguments: $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are vector registers, each with L matrix tiles of shape $\lambda \times \lambda$.

Semantics: This instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \mathbf{A}[0] {}^{\oplus}_{\otimes} \mathbf{B}[i], \forall i \in [0, L)$.

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

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

Arguments: $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are vector registers, each with L matrix tiles of shape $\lambda \times \lambda$. The index register x identifies the matrix tile $\mathbf{A}[x], x \in [0, L)$.

Semantics: This instruction computes $\mathbf{C}[i] \leftarrow \mathbf{C}[i] \oplus \mathbf{A}[x]{}^{\oplus}_{\otimes}\mathbf{B}[i], \forall i \in [0, L)$.

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 $\langle \mathsf{VLEN}, \lambda, L\rangle$ 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 $n = 4\lambda L$.

```
procedure dgemm(M, N, K, α, A, B, β, C)
    for(I ← 0; I < M; I ← I + ml)
        ml ← min(M − I, 4λ)
        for(J ← 0; J < N; J ← J + nl)
            nl ← min(N − J, 4λL)
            μdgemm(K, α, A(I : 4λ. :), B(:, J : 4λL), β, C(I : 4λ, J : 4λL))
        end
    end
end
```

Figure 1: The (M, N, K) BLAS routine dgemm can be implemented as a two-dimensional loop, each iteration computing a $4\lambda \times 4\lambda L$ 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 4λ rows of $\mathbf{A}$ and a block of $4\lambda L$ columns of $\mathbf{B}$.

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, λ, and L chosen by an implementation, as long as they are consistent. We call this code *geometry 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 λL, while the inner loop proceeds in chunks of λ. At each iteration of the outer (k) loop, we load $4L$ tiles of matrix $\mathbf{A}$ in registers v08–v11. This is accomplished with the matrix load instruction discussed above. 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 $\lambda \times 4\lambda L$ panel of $\mathbf{B}$. We then multiply each of the 4 tiles of shape $\lambda \times \lambda$ in position x of registers v08–v11 by the $4L$ tiles of shape $\lambda \times \lambda$ in registers v12–v15. This updates the $16L$ tiles of $\mathbf{C}$ in registers v16–v31. At the end of the outer loop we will have computed $\mathbf{A} \times \mathbf{B}$. The scaling by scalar α can be accomplished with existing RISC-V vector instructions, as can the addition of $\beta\mathbf{C}$. The loading and storing of the panel of $\mathbf{C}$ can again use matrix load and store (not discussed, but similar to load) instructions.

The binary code in Figure 2 is agnostic to the values of VLEN, λ, and L, 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 $4\lambda^2 L$ elements of matrix $\mathbf{A}$. Each iteration of the inner loop of Figure 2 loads $4\lambda^2 L$ elements of matrix $\mathbf{B}$. Therefore, the total number of elements loaded in each iteration of the outer loop is $4\lambda^2 L(1 + L)$. Each mgemmx instruction in the inner loop performs λ^3 multiply-adds for each tile of matrix $\mathbf{C}$ produced, or $\lambda^3 L$ multiply-adds per instruction. Since there are 16 instructions in the inner loop body and that body is executed L time per iteration of the outer loop, we perform $16\lambda^3 L^2$ multiply-adds per iteration of the outer loop. Therefore, the computational intensity is

$$\eta = \frac{16\lambda^3 L^2}{4\lambda^2 L(1 + L)} = 4\frac{\lambda L}{1 + L} = \begin{cases} 2\lambda & (L = 1) \\ \frac{8}{3}\lambda & (L = 2) \end{cases} .$$

The computational intensity scales with λ, which is proportional to the square root of VLEN. Furthermore, the computational intensity is highest $L = 1$ or $L = 2$, which are the optimal choices for a given VLEN.

```
procedure μdgemm(K, α, A, B, β, C)
    [ v16 v17 v18 v19
      v20 v21 v22 v23
      v24 v25 v26 v27
      v28 v29 v30 v31 ] ← 0
    for(k ← 0; k < K; k ← k + λL)
        kl ← min(K − k, λL)
        mload⟨fp64, 4, ml, 1, kl⟩(v08, A(0, k), lda)      ( v08 ← A(0 × λ : +λ, k : +λL)
                                                          v09 ← A(1 × λ : +λ, k : +λL)
                                                          v10 ← A(2 × λ : +λ, k : +λL)
                                                          v11 ← A(3 × λ : +λ, k : +λL) )
        for(x ← 0; x < L; x ← x + 1)
            mload⟨fp64, 1, kl, 4, nl⟩(v12, B(k, 0), ldb)  ( v12 ← B(k : +λ, 0 × λL : +λL)
                                                          v13 ← B(k : +λ, 1 × λL : +λL)
                                                          v14 ← B(k : +λ, 2 × λL : +λL)
                                                          v15 ← B(k : +λ, 3 × λL : +λL) )
            mgemmx⟨fp64, ×, +⟩(v08, v12, v16, x)
            mgemmx⟨fp64, ×, +⟩(v08, v13, v17, x)
            mgemmx⟨fp64, ×, +⟩(v08, v14, v18, x)
            mgemmx⟨fp64, ×, +⟩(v08, v15, v19, x)
            mgemmx⟨fp64, ×, +⟩(v09, v12, v20, x)
            mgemmx⟨fp64, ×, +⟩(v09, v13, v21, x)
            mgemmx⟨fp64, ×, +⟩(v09, v14, v22, x)
            mgemmx⟨fp64, ×, +⟩(v09, v15, v23, x)
            mgemmx⟨fp64, ×, +⟩(v10, v12, v24, x)
            mgemmx⟨fp64, ×, +⟩(v10, v13, v25, x)
            mgemmx⟨fp64, ×, +⟩(v10, v14, v26, x)
            mgemmx⟨fp64, ×, +⟩(v10, v15, v27, x)
            mgemmx⟨fp64, ×, +⟩(v11, v12, v28, x)
            mgemmx⟨fp64, ×, +⟩(v11, v13, v29, x)
            mgemmx⟨fp64, ×, +⟩(v11, v14, v30, x)
            mgemmx⟨fp64, ×, +⟩(v11, v15, v31, x)
            kl ← kl − λ
        end
    end
    (v16, v17, ..., v31) ← α × (v16, v17, ..., v31)
    C ← [ v16 v17 v18 v19
          v20 v21 v22 v23
          v24 v25 v26 v27
          v28 v29 v30 v31 ] + β × C
end
```

Figure 2: The micro-kernel of the BLAS routine dgemm computes a $4\lambda \times 4\lambda L$ panel of the result matrix $\mathbf{C}$. The body of the innermost loop is executed once for each tile of the $\mathbf{A}$ registers. In the body, it updates $16 \times L$ tiles of $\mathbf{C}$, each with a product of a tile of $\mathbf{A}$ and a tile of $\mathbf{B}$. The code is agnostic to the values of λ and L.

	v12 $[\,B_0^0 \;\; B_0^1\,]$	v13 $[\,B_0^2 \;\; B_0^3\,]$	v14 $[\,B_0^4 \;\; B_0^5\,]$	v15 $[\,B_0^6 \;\; B_0^7\,]$
v08 $[\,A_0^0 \;\; A_0^1\,]$	v16 $[\,0 \;\; 0\,]$	v17 $[\,0 \;\; 0\,]$	v18 $[\,0 \;\; 0\,]$	v19 $[\,0 \;\; 0\,]$
v09 $[\,A_1^0 \;\; A_1^1\,]$	v20 $[\,0 \;\; 0\,]$	v21 $[\,0 \;\; 0\,]$	v22 $[\,0 \;\; 0\,]$	v23 $[\,0 \;\; 0\,]$
v10 $[\,A_2^0 \;\; A_2^1\,]$	v24 $[\,0 \;\; 0\,]$	v25 $[\,0 \;\; 0\,]$	v26 $[\,0 \;\; 0\,]$	v27 $[\,0 \;\; 0\,]$
v11 $[\,A_3^0 \;\; A_3^1\,]$	v28 $[\,0 \;\; 0\,]$	v29 $[\,0 \;\; 0\,]$	v30 $[\,0 \;\; 0\,]$	v31 $[\,0 \;\; 0\,]$

$$\mathsf{v08} = \left[\begin{bmatrix} a_0^0 & a_0^1 & a_0^2 & a_0^3 \\ a_1^0 & a_1^1 & a_1^2 & a_1^3 \\ a_2^0 & a_2^1 & a_2^2 & a_2^3 \\ a_3^0 & a_3^1 & a_3^2 & a_3^3 \end{bmatrix} \begin{bmatrix} a_0^4 & a_0^5 & a_0^6 & a_0^7 \\ a_1^4 & a_1^5 & a_1^6 & a_1^7 \\ a_2^4 & a_2^5 & a_2^6 & a_2^7 \\ a_3^4 & a_3^5 & a_3^6 & a_3^7 \end{bmatrix} \right] \quad \mathsf{v12} = \left[\begin{bmatrix} b_0^0 & b_0^1 & b_0^2 & b_0^3 \\ b_1^0 & b_1^1 & b_1^2 & b_1^3 \\ b_2^0 & b_2^1 & b_2^2 & b_2^3 \\ b_3^0 & b_3^1 & b_3^2 & b_3^3 \end{bmatrix} \begin{bmatrix} b_0^4 & b_0^5 & b_0^6 & b_0^7 \\ b_1^4 & b_1^5 & b_1^6 & b_1^7 \\ b_2^4 & b_2^5 & b_2^6 & b_2^7 \\ b_3^4 & b_3^5 & b_3^6 & b_3^7 \end{bmatrix} \right]$$

Figure 3: State of the computation at the beginning (line 08) of iteration $\langle k = 0, x = 0\rangle$, with VLEN $= 2048$. Each of the registers contain 32 fp64 elements of either matrix **A** (v08-v11), **B** (v12-v15), or **C** (v16-v31). The 32 elements are organized as two 4×4 tiles in each register. The contents of v08 and v12 are shown explicitly. This particular configuration of $\lambda = 4$ and $L = 2$ is only one of the valid configurations for VLEN $= 2048$ shown in Table 1.

	v12 $[\,B_0^0 \;\; B_0^1\,]$	v13 $[\,B_0^2 \;\; B_0^3\,]$	v14 $[\,B_0^4 \;\; B_0^5\,]$	v15 $[\,B_0^6 \;\; B_0^7\,]$
v08 $[\,A_0^0 \;\; A_0^1\,]$	v16 $[\,A_0^0B_0^0 \;\; A_0^0B_0^1\,]$	v17 $[\,A_0^0B_0^2 \;\; A_0^0B_0^3\,]$	v18 $[\,A_0^0B_0^4 \;\; A_0^0B_0^5\,]$	v19 $[\,A_0^0B_0^6 \;\; A_0^0B_0^7\,]$
v09 $[\,A_1^0 \;\; A_1^1\,]$	v20 $[\,A_1^0B_0^0 \;\; A_1^0B_0^1\,]$	v21 $[\,A_1^0B_0^2 \;\; A_1^0B_0^3\,]$	v22 $[\,A_1^0B_0^4 \;\; A_1^0B_0^5\,]$	v23 $[\,A_1^0B_0^6 \;\; A_1^0B_0^7\,]$
v10 $[\,A_2^0 \;\; A_2^1\,]$	v24 $[\,A_2^0B_0^0 \;\; A_2^0B_0^1\,]$	v25 $[\,A_2^0B_0^2 \;\; A_2^0B_0^3\,]$	v26 $[\,A_2^0B_0^4 \;\; A_2^0B_0^5\,]$	v27 $[\,A_2^0B_0^6 \;\; A_2^0B_0^7\,]$
v11 $[\,A_3^0 \;\; A_3^1\,]$	v28 $[\,A_3^0B_0^0 \;\; A_3^0B_0^1\,]$	v29 $[\,A_3^0B_0^2 \;\; A_3^0B_0^3\,]$	v30 $[\,A_3^0B_0^4 \;\; A_3^0B_0^5\,]$	v31 $[\,A_3^0B_0^6 \;\; A_3^0B_0^7\,]$

Figure 4: State of the computation at the end (line 24) of iteration $\langle k = 0, x = 0\rangle$.

	v12 $\begin{bmatrix} B^0_1 & B^1_1 \end{bmatrix}$	v13 $\begin{bmatrix} B^2_1 & B^3_1 \end{bmatrix}$	v14 $\begin{bmatrix} B^4_1 & B^5_1 \end{bmatrix}$	v15 $\begin{bmatrix} B^6_1 & B^7_1 \end{bmatrix}$
v08 $\begin{bmatrix} A^0_0 & A^1_0 \end{bmatrix}$	v16 $\begin{bmatrix} A^0_0B^0_0 & A^0_0B^1_0 \\ + & + \\ A^1_0B^0_1 & A^1_0B^1_1 \end{bmatrix}$	v17 $\begin{bmatrix} A^0_0B^2_0 & A^0_0B^3_0 \\ + & + \\ A^1_0B^2_1 & A^1_0B^3_1 \end{bmatrix}$	v18 $\begin{bmatrix} A^0_0B^4_0 & A^0_0B^5_0 \\ + & + \\ A^1_0B^4_1 & A^1_0B^5_1 \end{bmatrix}$	v19 $\begin{bmatrix} A^0_0B^6_0 & A^0_0B^7_0 \\ + & + \\ A^1_0B^6_1 & A^1_0B^7_1 \end{bmatrix}$
v09 $\begin{bmatrix} A^0_1 & A^1_1 \end{bmatrix}$	v20 $\begin{bmatrix} A^0_1B^0_0 & A^0_1B^1_0 \\ + & + \\ A^1_1B^0_1 & A^1_1B^1_1 \end{bmatrix}$	v21 $\begin{bmatrix} A^0_1B^2_0 & A^0_1B^3_0 \\ + & + \\ A^1_1B^2_1 & A^1_1B^3_1 \end{bmatrix}$	v22 $\begin{bmatrix} A^0_1B^4_0 & A^0_1B^5_0 \\ + & + \\ A^1_1B^4_1 & A^1_1B^5_1 \end{bmatrix}$	v23 $\begin{bmatrix} A^0_1B^6_0 & A^0_1B^7_0 \\ + & + \\ A^1_1B^6_1 & A^1_1B^7_1 \end{bmatrix}$
v10 $\begin{bmatrix} A^0_2 & A^1_2 \end{bmatrix}$	v24 $\begin{bmatrix} A^0_2B^0_0 & A^0_2B^1_0 \\ + & + \\ A^1_2B^0_1 & A^1_2B^1_1 \end{bmatrix}$	v25 $\begin{bmatrix} A^0_2B^2_0 & A^0_2B^3_0 \\ + & + \\ A^1_2B^2_1 & A^1_2B^3_1 \end{bmatrix}$	v26 $\begin{bmatrix} A^0_2B^4_0 & A^0_2B^5_0 \\ + & + \\ A^1_2B^4_1 & A^1_2B^5_1 \end{bmatrix}$	v27 $\begin{bmatrix} A^0_2B^6_0 & A^0_2B^7_0 \\ + & + \\ A^1_2B^6_1 & A^1_2B^7_1 \end{bmatrix}$
v11 $\begin{bmatrix} A^0_3 & A^1_3 \end{bmatrix}$	v28 $\begin{bmatrix} A^0_3B^0_0 & A^0_3B^1_0 \\ + & + \\ A^1_3B^0_1 & A^1_3B^1_1 \end{bmatrix}$	v29 $\begin{bmatrix} A^0_3B^2_0 & A^0_3B^3_0 \\ + & + \\ A^1_3B^2_1 & A^1_3B^3_1 \end{bmatrix}$	v30 $\begin{bmatrix} A^0_3B^4_0 & A^0_3B^5_0 \\ + & + \\ A^1_3B^4_1 & A^1_3B^5_1 \end{bmatrix}$	v31 $\begin{bmatrix} A^0_3B^6_0 & A^0_3B^7_0 \\ + & + \\ A^1_3B^6_1 & A^1_3B^7_1 \end{bmatrix}$

Figure 5: State of the computation at the end (line 24) of iteration $\langle k = 0, x = 1 \rangle$.

	v12 $\begin{bmatrix} B_2^0 & B_2^1 \end{bmatrix}$	v13 $\begin{bmatrix} B_2^2 & B_2^3 \end{bmatrix}$	v14 $\begin{bmatrix} B_2^4 & B_2^5 \end{bmatrix}$	v15 $\begin{bmatrix} B_2^6 & B_2^7 \end{bmatrix}$
v08 $\begin{bmatrix} A_0^2 & A_0^3 \end{bmatrix}$	v16 $\begin{bmatrix} A_0^0B_0^0 & A_0^0B_0^1 \\ + & + \\ A_0^1B_1^0 & A_0^1B_1^1 \\ + & + \\ A_0^2B_2^0 & A_0^2B_2^1 \end{bmatrix}$	v17 $\begin{bmatrix} A_0^0B_0^2 & A_0^0B_0^3 \\ + & + \\ A_0^1B_1^2 & A_0^1B_1^3 \\ + & + \\ A_0^2B_2^2 & A_0^2B_2^3 \end{bmatrix}$	v18 $\begin{bmatrix} A_0^0B_0^4 & A_0^0B_0^5 \\ + & + \\ A_0^1B_1^4 & A_0^1B_1^5 \\ + & + \\ A_0^2B_2^4 & A_0^2B_2^5 \end{bmatrix}$	v19 $\begin{bmatrix} A_0^0B_0^6 & A_0^0B_0^7 \\ + & + \\ A_0^1B_1^6 & A_0^1B_1^7 \\ + & + \\ A_0^2B_2^6 & A_0^2B_2^7 \end{bmatrix}$
v09 $\begin{bmatrix} A_1^2 & A_1^3 \end{bmatrix}$	v20 $\begin{bmatrix} A_1^0B_0^0 & A_1^0B_0^1 \\ + & + \\ A_1^1B_1^0 & A_1^1B_1^1 \\ + & + \\ A_1^2B_2^0 & A_1^2B_2^1 \end{bmatrix}$	v21 $\begin{bmatrix} A_1^0B_0^2 & A_1^0B_0^3 \\ + & + \\ A_1^1B_1^2 & A_1^1B_1^3 \\ + & + \\ A_1^2B_2^2 & A_1^2B_2^3 \end{bmatrix}$	v22 $\begin{bmatrix} A_1^0B_0^4 & A_1^0B_0^5 \\ + & + \\ A_1^1B_1^4 & A_1^1B_1^5 \\ + & + \\ A_1^2B_2^4 & A_1^2B_2^5 \end{bmatrix}$	v23 $\begin{bmatrix} A_1^0B_0^6 & A_1^0B_0^7 \\ + & + \\ A_1^1B_1^6 & A_1^1B_1^7 \\ + & + \\ A_1^2B_2^6 & A_1^2B_2^7 \end{bmatrix}$
v10 $\begin{bmatrix} A_2^2 & A_2^3 \end{bmatrix}$	v24 $\begin{bmatrix} A_2^0B_0^0 & A_2^0B_0^1 \\ + & + \\ A_2^1B_1^0 & A_2^1B_1^1 \\ + & + \\ A_2^2B_2^0 & A_2^2B_2^1 \end{bmatrix}$	v25 $\begin{bmatrix} A_2^0B_0^2 & A_2^0B_0^3 \\ + & + \\ A_2^1B_1^2 & A_2^1B_1^3 \\ + & + \\ A_2^2B_2^2 & A_2^2B_2^3 \end{bmatrix}$	v26 $\begin{bmatrix} A_2^0B_0^4 & A_2^0B_0^5 \\ + & + \\ A_2^1B_1^4 & A_2^1B_1^5 \\ + & + \\ A_2^2B_2^4 & A_2^2B_2^5 \end{bmatrix}$	v27 $\begin{bmatrix} A_2^0B_0^6 & A_2^0B_0^7 \\ + & + \\ A_2^1B_1^6 & A_2^1B_1^7 \\ + & + \\ A_2^2B_2^6 & A_2^2B_2^7 \end{bmatrix}$
v11 $\begin{bmatrix} A_3^2 & A_3^3 \end{bmatrix}$	v28 $\begin{bmatrix} A_3^0B_0^0 & A_3^0B_0^1 \\ + & + \\ A_3^1B_1^0 & A_3^1B_1^1 \\ + & + \\ A_3^2B_2^0 & A_3^2B_2^1 \end{bmatrix}$	v29 $\begin{bmatrix} A_3^0B_0^2 & A_3^0B_0^3 \\ + & + \\ A_3^1B_1^2 & A_3^1B_1^3 \\ + & + \\ A_3^2B_2^2 & A_3^2B_2^3 \end{bmatrix}$	v30 $\begin{bmatrix} A_3^0B_0^4 & A_3^0B_0^5 \\ + & + \\ A_3^1B_1^4 & A_3^1B_1^5 \\ + & + \\ A_3^2B_2^4 & A_3^2B_2^5 \end{bmatrix}$	v31 $\begin{bmatrix} A_3^0B_0^6 & A_3^0B_0^7 \\ + & + \\ A_3^1B_1^6 & A_3^1B_1^7 \\ + & + \\ A_3^2B_2^6 & A_3^2B_2^7 \end{bmatrix}$

Figure 6: State of the computation at the end (line 24) of iteration $\langle k = \lambda L, x = 0 \rangle$.

	v12 $\begin{bmatrix} B_3^0 & B_3^1 \end{bmatrix}$	v13 $\begin{bmatrix} B_3^2 & B_3^3 \end{bmatrix}$	v14 $\begin{bmatrix} B_3^4 & B_3^5 \end{bmatrix}$	v15 $\begin{bmatrix} B_3^6 & B_3^7 \end{bmatrix}$
v08 $\begin{bmatrix} A_0^2 & A_0^3 \end{bmatrix}$	v16 $\begin{bmatrix} A_0^0B_0^0 & A_0^0B_0^1 \\ + & + \\ A_0^1B_1^0 & A_0^1B_1^1 \\ + & + \\ A_0^2B_2^0 & A_0^2B_2^1 \\ + & + \\ A_0^3B_3^0 & A_0^3B_3^1 \end{bmatrix}$	v17 $\begin{bmatrix} A_0^0B_0^2 & A_0^0B_0^3 \\ + & + \\ A_0^1B_1^2 & A_0^1B_1^3 \\ + & + \\ A_0^2B_2^2 & A_0^2B_2^3 \\ + & + \\ A_0^3B_3^2 & A_0^3B_3^3 \end{bmatrix}$	v18 $\begin{bmatrix} A_0^0B_0^4 & A_0^0B_0^5 \\ + & + \\ A_0^1B_1^4 & A_0^1B_1^5 \\ + & + \\ A_0^2B_2^4 & A_0^2B_2^5 \\ + & + \\ A_0^3B_3^4 & A_0^3B_3^5 \end{bmatrix}$	v19 $\begin{bmatrix} A_0^0B_0^6 & A_0^0B_0^7 \\ + & + \\ A_0^1B_1^6 & A_0^1B_1^7 \\ + & + \\ A_0^2B_2^6 & A_0^2B_2^7 \\ + & + \\ A_0^3B_3^6 & A_0^3B_3^7 \end{bmatrix}$
v09 $\begin{bmatrix} A_1^2 & A_1^3 \end{bmatrix}$	v20 $\begin{bmatrix} A_1^0B_0^0 & A_1^0B_0^1 \\ + & + \\ A_1^1B_1^0 & A_1^1B_1^1 \\ + & + \\ A_1^2B_2^0 & A_1^2B_2^1 \\ + & + \\ A_1^3B_3^0 & A_1^3B_3^1 \end{bmatrix}$	v21 $\begin{bmatrix} A_1^0B_0^2 & A_1^0B_0^3 \\ + & + \\ A_1^1B_1^2 & A_1^1B_1^3 \\ + & + \\ A_1^2B_2^2 & A_1^2B_2^3 \\ + & + \\ A_1^3B_3^2 & A_1^3B_3^3 \end{bmatrix}$	v22 $\begin{bmatrix} A_1^0B_0^4 & A_1^0B_0^5 \\ + & + \\ A_1^1B_1^4 & A_1^1B_1^5 \\ + & + \\ A_1^2B_2^4 & A_1^2B_2^5 \\ + & + \\ A_1^3B_3^4 & A_1^3B_3^5 \end{bmatrix}$	v23 $\begin{bmatrix} A_1^0B_0^6 & A_1^0B_0^7 \\ + & + \\ A_1^1B_1^6 & A_1^1B_1^7 \\ + & + \\ A_1^2B_2^6 & A_1^2B_2^7 \\ + & + \\ A_1^3B_3^6 & A_1^3B_3^7 \end{bmatrix}$
v10 $\begin{bmatrix} A_2^2 & A_2^3 \end{bmatrix}$	v24 $\begin{bmatrix} A_2^0B_0^0 & A_2^0B_0^1 \\ + & + \\ A_2^1B_1^0 & A_2^1B_1^1 \\ + & + \\ A_2^2B_2^0 & A_2^2B_2^1 \\ + & + \\ A_2^3B_3^0 & A_2^3B_3^1 \end{bmatrix}$	v25 $\begin{bmatrix} A_2^0B_0^2 & A_2^0B_0^3 \\ + & + \\ A_2^1B_1^2 & A_2^1B_1^3 \\ + & + \\ A_2^2B_2^2 & A_2^2B_2^3 \\ + & + \\ A_2^3B_3^2 & A_2^3B_3^3 \end{bmatrix}$	v26 $\begin{bmatrix} A_2^0B_0^4 & A_2^0B_0^5 \\ + & + \\ A_2^1B_1^4 & A_2^1B_1^5 \\ + & + \\ A_2^2B_2^4 & A_2^2B_2^5 \\ + & + \\ A_2^3B_3^4 & A_2^3B_3^5 \end{bmatrix}$	v27 $\begin{bmatrix} A_2^0B_0^6 & A_2^0B_0^7 \\ + & + \\ A_2^1B_1^6 & A_2^1B_1^7 \\ + & + \\ A_2^2B_2^6 & A_2^2B_2^7 \\ + & + \\ A_2^3B_3^6 & A_2^3B_3^7 \end{bmatrix}$
v11 $\begin{bmatrix} A_3^2 & A_3^3 \end{bmatrix}$	v28 $\begin{bmatrix} A_3^0B_0^0 & A_3^0B_0^1 \\ + & + \\ A_3^1B_1^0 & A_3^1B_1^1 \\ + & + \\ A_3^2B_2^0 & A_3^2B_2^1 \\ + & + \\ A_3^3B_3^0 & A_3^3B_3^1 \end{bmatrix}$	v29 $\begin{bmatrix} A_3^0B_0^2 & A_3^0B_0^3 \\ + & + \\ A_3^1B_1^2 & A_3^1B_1^3 \\ + & + \\ A_3^2B_2^2 & A_3^2B_2^3 \\ + & + \\ A_3^3B_3^2 & A_3^3B_3^3 \end{bmatrix}$	v30 $\begin{bmatrix} A_3^0B_0^4 & A_3^0B_0^5 \\ + & + \\ A_3^1B_1^4 & A_3^1B_1^5 \\ + & + \\ A_3^2B_2^4 & A_3^2B_2^5 \\ + & + \\ A_3^3B_3^4 & A_3^3B_3^5 \end{bmatrix}$	v31 $\begin{bmatrix} A_3^0B_0^6 & A_3^0B_0^7 \\ + & + \\ A_3^1B_1^6 & A_3^1B_1^7 \\ + & + \\ A_3^2B_2^6 & A_3^2B_2^7 \\ + & + \\ A_3^3B_3^6 & A_3^3B_3^7 \end{bmatrix}$

Figure 7: State of the computation at the end (line 24) of iteration $\langle k = \lambda L, x = 1 \rangle$.

5 Trivial extension to mixed data types

Let $T(\mathbf{A})$, $T(\mathbf{B})$, and $T(\mathbf{C})$ denote the data types of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively. If $\mathsf{sizeof}(T(\mathbf{A})) = \mathsf{sizeof}(T(\mathbf{B})) = \mathsf{sizeof}(T(\mathbf{C}))/n$, we can treat all three matrices as having $\mathsf{MEW} = \mathsf{sizeof}(T(\mathbf{C}))$ and define the elements of $\mathbf{A}$ and $\mathbf{B}$ to be n-vectors of their natural data types ($T(\mathbf{A})$ and $T(\mathbf{B})$, respectively). The $\otimes$ operator becomes the dot-product of two n-vectors of type $T(\mathbf{A})$ and $T(\mathbf{B})$. The $\oplus$ operator adds this dot-product to an element of $\mathbf{C}$. With these definitions it is trivial to extend the common-type variant to treat mixed types for the three matrices, subject to the constraints outlined above.

6 General mixed data types

$$
\begin{array}{c|cccc}
 & \text{v12} & \text{v13} & \text{v14} & \text{v15} \\
 & \begin{bmatrix} B_0^0 \\ B_1^0 \end{bmatrix} & \begin{bmatrix} B_0^1 \\ B_1^1 \end{bmatrix} & \begin{bmatrix} B_0^2 \\ B_1^2 \end{bmatrix} & \begin{bmatrix} B_0^3 \\ B_1^3 \end{bmatrix} \\
\hline
\text{v08} & \text{v16} & \text{v17} & \text{v18} & \text{v19} \\
\begin{bmatrix} A_0^0 & A_0^1 & A_0^2 & A_0^3 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \\
\text{v09} & \text{v20} & \text{v21} & \text{v22} & \text{v23} \\
\begin{bmatrix} A_1^0 & A_1^1 & A_1^2 & A_1^3 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \\
\text{v10} & \text{v24} & \text{v25} & \text{v26} & \text{v27} \\
\begin{bmatrix} A_2^0 & A_2^1 & A_2^2 & A_2^3 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \\
\text{v11} & \text{v28} & \text{v29} & \text{v30} & \text{v31} \\
\begin{bmatrix} A_3^0 & A_3^1 & A_3^2 & A_3^3 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix}
\end{array}
$$

$$
\text{v08} = \begin{bmatrix}
\begin{bmatrix} a_0^0 & a_0^1 & a_0^2 & a_0^3 \\ a_1^0 & a_1^1 & a_1^2 & a_1^3 \\ a_2^0 & a_2^1 & a_2^2 & a_2^3 \\ a_3^0 & a_3^1 & a_3^2 & a_3^3 \end{bmatrix}
\begin{bmatrix} a_0^4 & a_0^5 & a_0^6 & a_0^7 \\ a_1^4 & a_1^5 & a_1^6 & a_1^7 \\ a_2^4 & a_2^5 & a_2^6 & a_2^7 \\ a_3^4 & a_3^5 & a_3^6 & a_3^7 \end{bmatrix}
\begin{bmatrix} a_0^8 & a_0^9 & a_0^{10} & a_0^{11} \\ a_1^8 & a_1^9 & a_1^{10} & a_1^{11} \\ a_2^8 & a_2^9 & a_2^{10} & a_2^{11} \\ a_3^8 & a_3^9 & a_3^{10} & a_3^{11} \end{bmatrix}
\begin{bmatrix} a_0^{12} & a_0^{13} & a_0^{14} & a_0^{15} \\ a_1^{12} & a_1^{13} & a_1^{14} & a_1^{15} \\ a_2^{12} & a_2^{13} & a_2^{14} & a_2^{15} \\ a_3^{12} & a_3^{13} & a_3^{14} & a_3^{15} \end{bmatrix}
\end{bmatrix}
$$

$$
\text{v12} = \begin{bmatrix}
\begin{bmatrix} b_0^0 & b_0^1 & b_0^2 & b_0^3 \\ b_1^0 & b_1^1 & b_1^2 & b_1^3 \\ b_2^0 & b_2^1 & b_2^2 & b_2^3 \\ b_3^0 & b_3^1 & b_3^2 & b_3^3 \end{bmatrix} \\
\begin{bmatrix} b_4^0 & b_4^1 & b_4^2 & b_4^3 \\ b_5^0 & b_5^1 & b_5^2 & b_5^3 \\ b_6^0 & b_6^1 & b_6^2 & b_6^3 \\ b_7^0 & b_7^1 & b_7^2 & b_7^3 \end{bmatrix}
\end{bmatrix}
$$

Figure 8: State of the computation at the beginning of iteration $\langle k = 0, x = 0 \rangle$, with VLEN $= 1024$, $T(\mathbf{C}) =$ fp64, $T(\mathbf{B}) =$ fp32, and $T(\mathbf{A}) =$ fp16. This particular configuration of $\lambda = 4$, $L(\mathbf{C}) = 1$, $L(\mathbf{B}) = 2$, and $L(\mathbf{A}) = 4$ is only one of the valid configurations for VLEN $= 1024$ shown in Table 1.

	v12 $\begin{bmatrix} B_0^0 \\ B_1^0 \end{bmatrix}$	v13 $\begin{bmatrix} B_0^1 \\ B_1^1 \end{bmatrix}$	v14 $\begin{bmatrix} B_0^2 \\ B_1^2 \end{bmatrix}$	v15 $\begin{bmatrix} B_0^3 \\ B_1^3 \end{bmatrix}$
v08 $\begin{bmatrix} A_0^0 & A_0^1 & A_0^2 & A_0^3 \end{bmatrix}$	v16 $\begin{bmatrix} A_0^0B_0^0 + A_0^1B_1^0 \end{bmatrix}$	v17 $\begin{bmatrix} A_0^0B_0^1 + A_0^1B_1^1 \end{bmatrix}$	v18 $\begin{bmatrix} A_0^0B_0^2 + A_0^1B_1^2 \end{bmatrix}$	v19 $\begin{bmatrix} A_0^0B_0^3 + A_0^1B_1^3 \end{bmatrix}$
v09 $\begin{bmatrix} A_1^0 & A_1^1 & A_1^2 & A_1^3 \end{bmatrix}$	v20 $\begin{bmatrix} A_1^0B_0^0 + A_1^1B_1^0 \end{bmatrix}$	v21 $\begin{bmatrix} A_1^0B_0^1 + A_1^1B_1^1 \end{bmatrix}$	v22 $\begin{bmatrix} A_1^0B_0^2 + A_1^1B_1^2 \end{bmatrix}$	v23 $\begin{bmatrix} A_1^0B_0^3 + A_1^1B_1^3 \end{bmatrix}$
v10 $\begin{bmatrix} A_2^0 & A_2^1 & A_2^2 & A_2^3 \end{bmatrix}$	v24 $\begin{bmatrix} A_2^0B_0^0 + A_2^1B_1^0 \end{bmatrix}$	v25 $\begin{bmatrix} A_2^0B_0^1 + A_2^1B_1^1 \end{bmatrix}$	v26 $\begin{bmatrix} A_2^0B_0^2 + A_2^1B_1^2 \end{bmatrix}$	v27 $\begin{bmatrix} A_2^0B_0^3 + A_2^1B_1^3 \end{bmatrix}$
v11 $\begin{bmatrix} A_3^0 & A_3^1 & A_3^2 & A_3^3 \end{bmatrix}$	v28 $\begin{bmatrix} A_3^0B_0^0 + A_3^1B_1^0 \end{bmatrix}$	v29 $\begin{bmatrix} A_3^0B_0^1 + A_3^1B_1^1 \end{bmatrix}$	v30 $\begin{bmatrix} A_3^0B_0^2 + A_3^1B_1^2 \end{bmatrix}$	v31 $\begin{bmatrix} A_3^0B_0^3 + A_3^1B_1^3 \end{bmatrix}$

Figure 9: State of the computation at the end of iteration $\langle k = 0, x = 0 \rangle$.

	v12 $\begin{bmatrix} B_2^0 \\ B_3^0 \end{bmatrix}$	v13 $\begin{bmatrix} B_2^1 \\ B_3^1 \end{bmatrix}$	v14 $\begin{bmatrix} B_2^2 \\ B_3^2 \end{bmatrix}$	v15 $\begin{bmatrix} B_2^3 \\ B_3^3 \end{bmatrix}$
v08 $\begin{bmatrix} A_0^0 & A_0^1 & A_0^2 & A_0^3 \end{bmatrix}$	v16 $\begin{bmatrix} A_0^0B_0^0 + A_0^1B_1^0 \\ + \\ A_0^2B_2^0 + A_0^3B_3^0 \end{bmatrix}$	v17 $\begin{bmatrix} A_0^0B_0^1 + A_0^1B_1^1 \\ + \\ A_0^2B_2^1 + A_0^3B_3^1 \end{bmatrix}$	v18 $\begin{bmatrix} A_0^0B_0^2 + A_0^1B_1^2 \\ + \\ A_0^2B_2^2 + A_0^3B_3^2 \end{bmatrix}$	v19 $\begin{bmatrix} A_0^0B_0^3 + A_0^1B_1^3 \\ + \\ A_0^2B_2^3 + A_0^3B_3^3 \end{bmatrix}$
v09 $\begin{bmatrix} A_1^0 & A_1^1 & A_1^2 & A_1^3 \end{bmatrix}$	v20 $\begin{bmatrix} A_1^0B_0^0 + A_1^1B_1^0 \\ + \\ A_1^2B_2^0 + A_1^3B_3^0 \end{bmatrix}$	v21 $\begin{bmatrix} A_1^0B_0^1 + A_1^1B_1^1 \\ + \\ A_1^2B_2^1 + A_1^3B_3^1 \end{bmatrix}$	v22 $\begin{bmatrix} A_1^0B_0^2 + A_1^1B_1^2 \\ + \\ A_1^2B_2^2 + A_1^3B_3^2 \end{bmatrix}$	v23 $\begin{bmatrix} A_1^0B_0^3 + A_1^1B_1^3 \\ + \\ A_1^2B_2^3 + A_1^3B_3^3 \end{bmatrix}$
v10 $\begin{bmatrix} A_2^0 & A_2^1 & A_2^2 & A_2^3 \end{bmatrix}$	v24 $\begin{bmatrix} A_2^0B_0^0 + A_2^1B_1^0 \\ + \\ A_2^2B_2^0 + A_2^3B_3^0 \end{bmatrix}$	v25 $\begin{bmatrix} A_2^0B_0^1 + A_2^1B_1^1 \\ + \\ A_2^2B_2^1 + A_2^3B_3^1 \end{bmatrix}$	v26 $\begin{bmatrix} A_2^0B_0^2 + A_2^1B_1^2 \\ + \\ A_2^2B_2^2 + A_2^3B_3^2 \end{bmatrix}$	v27 $\begin{bmatrix} A_2^0B_0^3 + A_2^1B_1^3 \\ + \\ A_2^2B_2^3 + A_2^3B_3^3 \end{bmatrix}$
v11 $\begin{bmatrix} A_3^0 & A_3^1 & A_3^2 & A_3^3 \end{bmatrix}$	v28 $\begin{bmatrix} A_3^0B_0^0 + A_3^1B_1^0 \\ + \\ A_3^2B_2^0 + A_3^3B_3^0 \end{bmatrix}$	v29 $\begin{bmatrix} A_3^0B_0^1 + A_3^1B_1^1 \\ + \\ A_3^2B_2^1 + A_3^3B_3^1 \end{bmatrix}$	v30 $\begin{bmatrix} A_3^0B_0^2 + A_3^1B_1^2 \\ + \\ A_3^2B_2^2 + A_3^3B_3^2 \end{bmatrix}$	v31 $\begin{bmatrix} A_3^0B_0^3 + A_3^1B_1^3 \\ + \\ A_3^2B_2^3 + A_3^3B_3^3 \end{bmatrix}$

Figure 10: State of the computation at the end (line 24) of iteration $\langle k = 0, x = 1 \rangle$.