Integrated Matrix Extension (IME)

Task Group Meeting

Guido Araujo
Jose Moreira

05/06/24

Agenda

- Update on TG Chair/Vice-chair selection
- Update on TG schedule
- Kick-off on matrix data-type and geometry configuration

Agenda

- Update on TG Chair/Vice-chair selection
- Update on TG schedule
- Kick-off on matrix data-type and geometry configuration

TG schedule for 2024

Task	Del.	Task Description	Meetings											
			1	2	3	4	5	6	7	8	9	10	11	12
1		Architectural features												
3		a. uArch: Overall analysis												
6		b. uArch: Memory access analysis												
2		c. ISA: Matrix data encoding												
4		d. ISA: Register usage and mapping												
6		e. ISA: Data type and geometry configuration												
5		f. ISA: Binary compatibility												
6		g. ISA: Computation operations definition												
7		h. ISA: Instruction encoding												
8		Workloads and bechmarking												
9		a. ML: T-Head profiling and ConvBench												
10		b. HPC: Polybench												
11		c. ML: POWER10 MMA transfers												
12		d. Workload analysis												
13		Quantitative analysis												
14		a. QEMU modelling												
15		b. Performace evaluation												
16		Definition of the final architecture												
17		a. RVM ISA v0												
18		b. RVM ISA v1												
19		RVM Spec writing												

TG schedule for 2024

Task	Del.	Task Description	Meetings											
			1	2	3	4	5	6	7	8	9	10	11	12
1		Architectural features												
3		a. uArch: Overall analysis												
6		b. uArch: Memory access analysis												
2		c. ISA: Matrix data encoding												
4		d. ISA: Register usage and mapping												
6		e. ISA: Data type and geometry configuration												
5		f. ISA: Binary compatibility												
6		g. ISA: Computation operations definition												
7		h. ISA: Instruction encoding												
8		Workloads and bechmarking												
9		a. ML: T-Head profiling and ConvBench												
10		b. HPC: Polybench												
11		c. ML: POWER10 MMA transfers												
12		d. Workload analysis												
13		Quantitative analysis												
14		a. QEMU modelling												
15		b. Performace evaluation												
16		Definition of the final architecture												
17		a. RVM ISA v0												
18		b. RVM ISA v1												
19		RVM Spec writing												

TG schedule for 2024

Task	Del.	Task Description	Meetings											
			1	2	3	4	5	6	7	8	9	10	11	12
1		Architectural features												
3		a. uArch: Overall analysis												
6		b. uArch: Memory access analysis												
2		c. ISA: Matrix data encoding												
4		d. ISA: Register usage and mapping												
6		e. ISA: Data type and geometry configuration												
5		f. ISA: Binary compatibility												
6		g. ISA: Computation operations definition												
7		h. ISA: Instruction encoding												
8		Workloads and bechmarking												
9		a. ML: T-Head profiling and ConvBench												
10		b. HPC: Polybench												
11		c. ML: POWER10 MMA transfers												
12		d. Workload analysis												
13		Quantitative analysis												
14		a. QEMU modelling												
15		b. Performace evaluation												
16		Definition of the final architecture												
17		a. RVM ISA v0												
18		b. RVM ISA v1												
19		RVM Spec writing												

TG schedule for 2024

Task	Del.	Task Description	Meetings											
			1	2	3	4	5	6	7	8	9	10	11	12
1		Architectural features												
3		a. uArch: Overall analysis												
6		b. uArch: Memory access analysis												
2		c. ISA: Matrix data encoding												
4		d. ISA: Register usage and mapping												
6		e. ISA: Data type and geometry configuration												
5		f. ISA: Binary compatibility												
6		g. ISA: Computation operations definition												
7		h. ISA: Instruction encoding												
8		Workloads and bechmarking												
9		a. ML: T-Head profiling and ConvBench												
10		b. HPC: Polybench												
11		c. ML: POWER10 MMA transfers												
12		d. Workload analysis												
13		Quantitative analysis												
14		a. QEMU modelling												
15		b. Performace evaluation												
16		Definition of the final architecture												
17		a. RVM ISA v0												
18		b. RVM ISA v1												
19		RVM Spec writing												

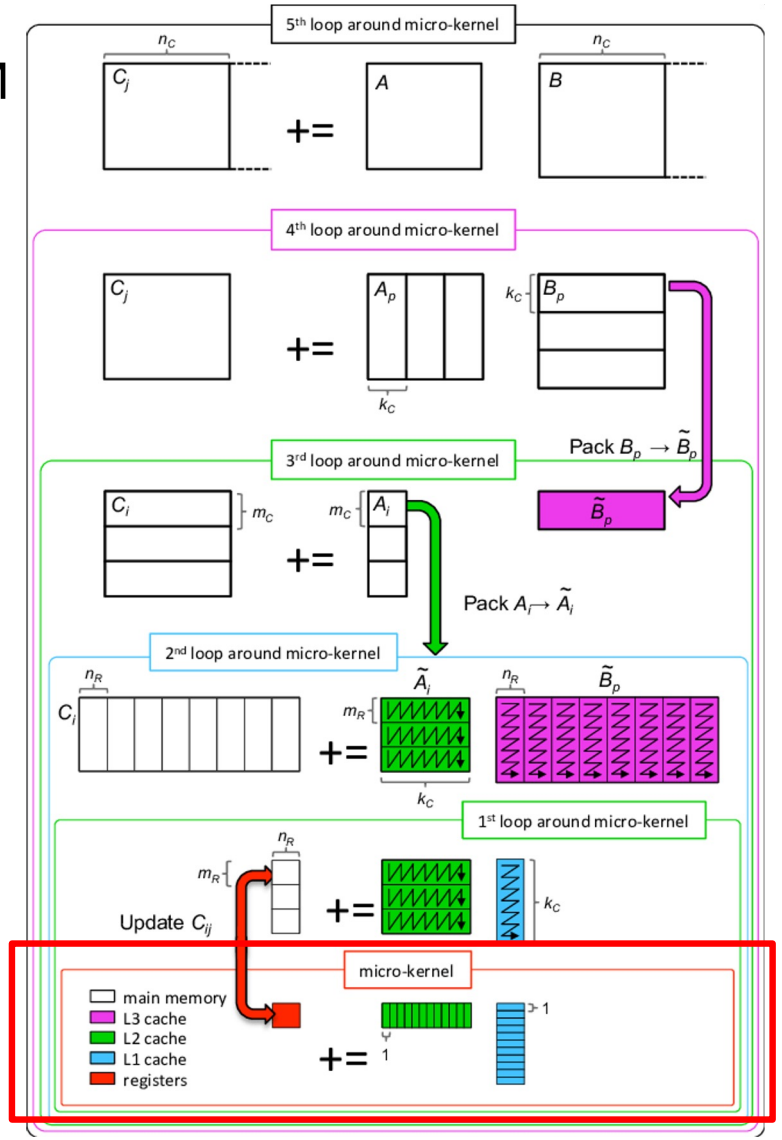
TG schedule for 2024

Task	Del.	Task Description	Meetings											
			1	2	3	4	5	6	7	8	9	10	11	12
1		Architectural features												
3		a. uArch: Overall analysis												
6		b. uArch: Memory access analysis												
2		c. ISA: Matrix data encoding												
4		d. ISA: Register usage and mapping												
6		e. ISA: Data type and geometry configuration												
5		f. ISA: Binary compatibility												
6		g. ISA: Computation operations definition												
7		h. ISA: Instruction encoding												
8		Workloads and bechmarking												
9		a. ML: T-Head profiling and ConvBench												
10		b. HPC: Polybench												
11		c. ML: POWER10 MMA transfers												
12		d. Workload analysis												
13		Quantitative analysis												
14		a. QEMU modelling												
15		b. Performace evaluation												
16		Definition of the final architecture												
17		a. RVM ISA v0												
18		b. RVM ISA v1												
19		RVM Spec writing												

Agenda

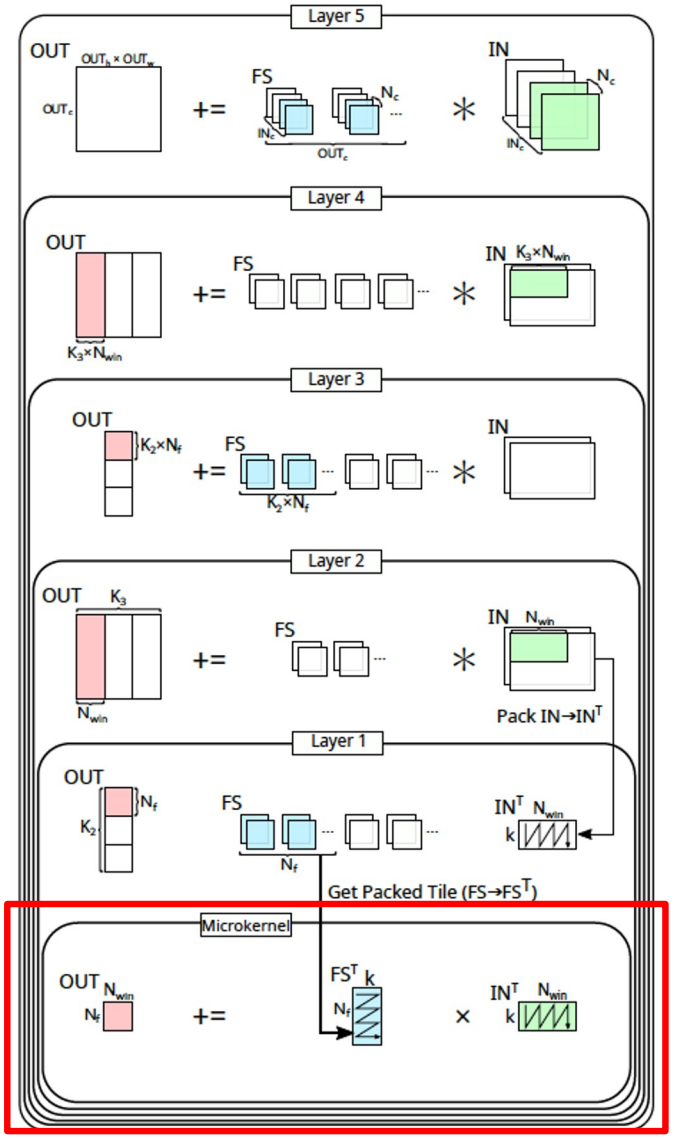
- Update on TG Chair/Vice-chair selection
- Update on TG schedule
- Kick-off on matrix data-type and geometry configuration

GEMM



m, n, k

CONV

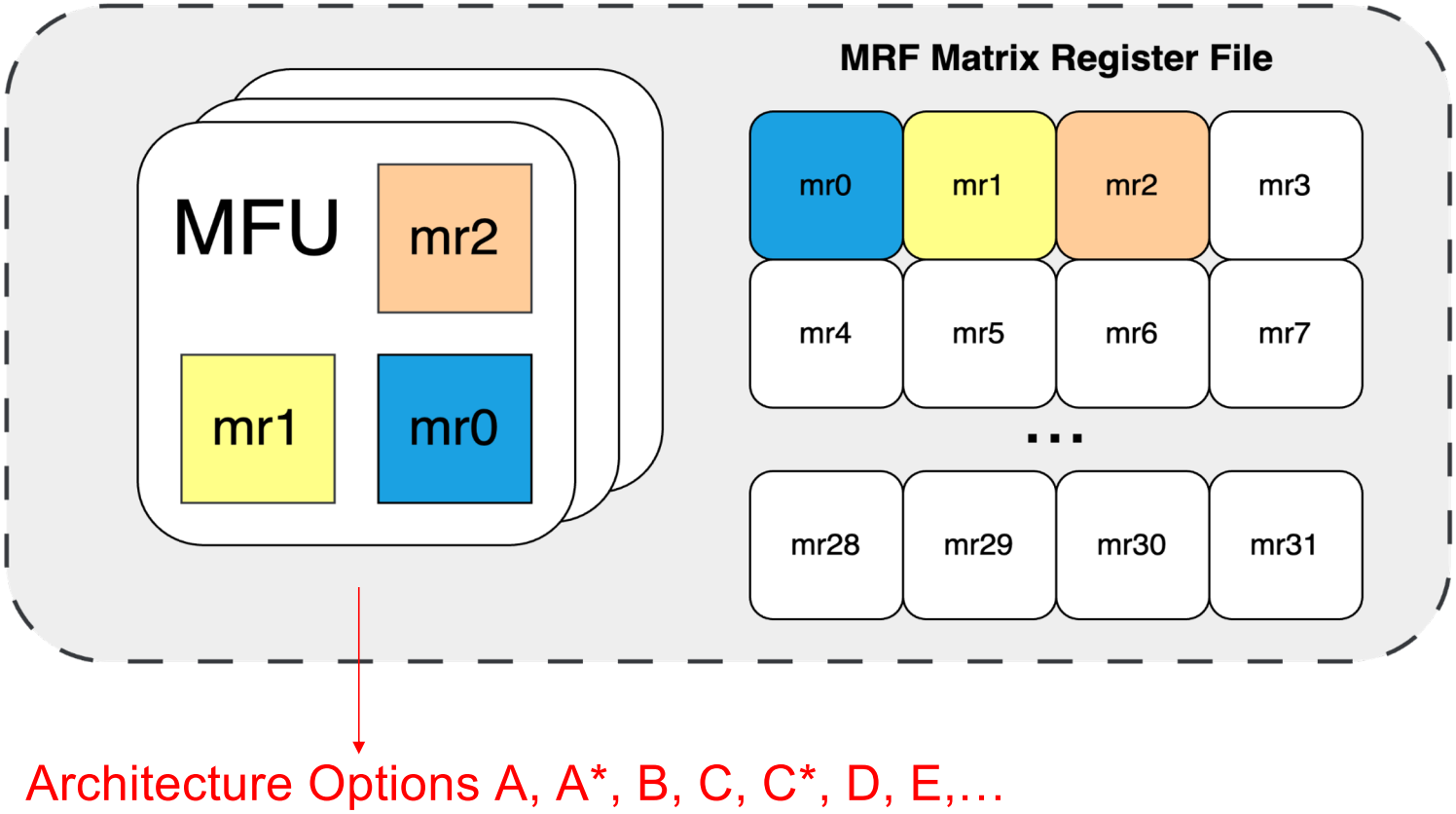


m, n, k

Desirable Features

- Support for operations with varying data types and data widths
  - int8, int16, int32, float32, etc.
  - Including mixed data lengths and widening
- Support for operations with varying number of elements
- Scalable/portable binary code
- Support for grouping many physical registers into one logical register (as in LMUL for RVV)
- Should easy kernel's loop code generation (C += A * B)
  - Determine trip counts for loops m, n and k
  - Determine how to update memory access to A, B, and C

Matrix Functional Units (MFUs) and Registers (MRF)



Matrix configuration registers (MCSR)

- Matrix type csr (mt)
  - Stores types and sizes
  - Information captured at compile time
  - int8, int16, int32, float32, BF16, etc.
- Operation shape csr (ms)
  - Stores dimensions m, n and k
  - Information captured at execution time
  - Three dimensions to enable shapes in matrix operations

Setting mt and ms CSRs

- Matrix type (mt)

```
setmt  t0, etypei, esizei   # t0 = matrix register size
                            # etypei, esizei = new imm
```

Used in stripminig memory accesses

- Matrix shape (ms)

```
setms.m t1, r1      # t1 = actual number of m iterations, r1 = m
setms.n  t2, r2     # t2 = actual number of n iterations, r2 = n
setms.k  t3, r3     # t3 = actual number of k iterations, r3 = k
```

Used as trip-count in m, n, k nested loops

Assign type and shape to matrices

- Desired operation

  C[m,n] += A[m,k] * B[k,n] -- C → m3, A→ m1 and → m2

- Assign ms and mt to each matrix (out of the inner loop)

```
setmst.mn  m3        # m1 = result matrix register
setmst.mk  m1        # m2 = first matrix operand
setmst.kn  m2        # m3 = second matrix operand
```

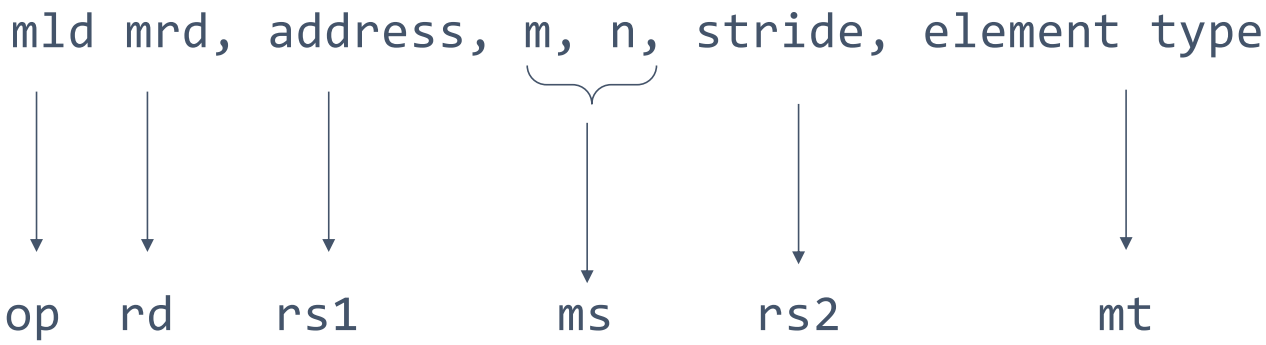
Basic Operations

- Load
- MMAC

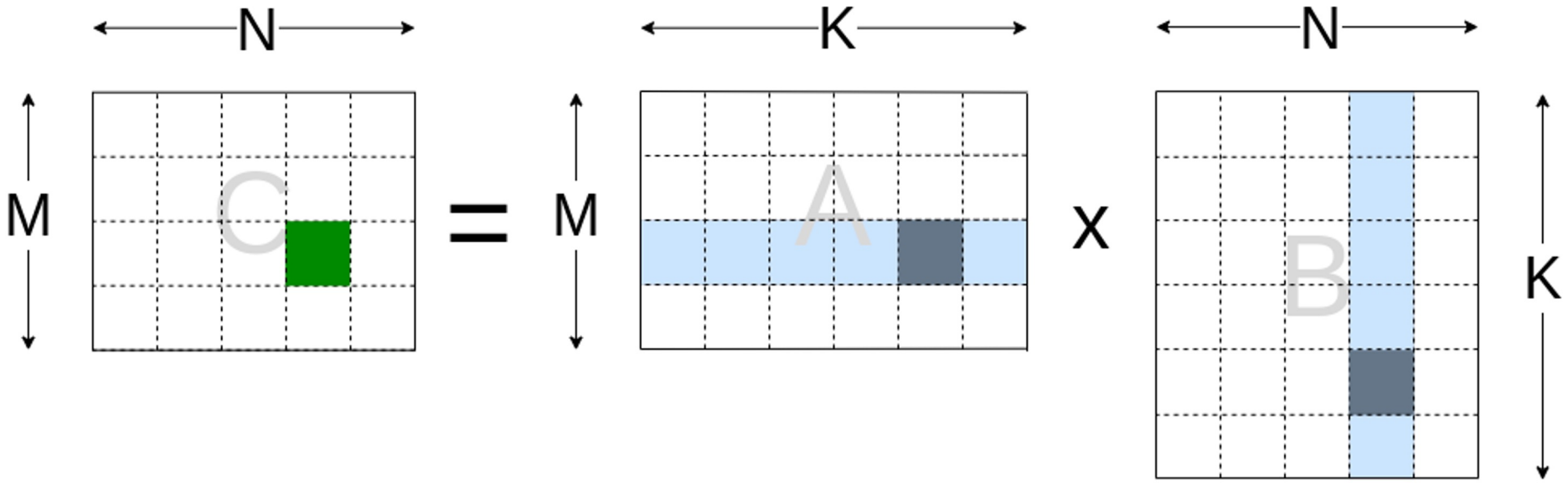
Matrix Load Operation

- Load matrix from memory into matrix register mrd
- Required information
  - matrix register destination
  - address
  - number of rows
  - number of columns
  - row stride
  - element type
- Prototype:
  - mld mrd, address, rows, columns, stride, element type

Proposed encoding for Load



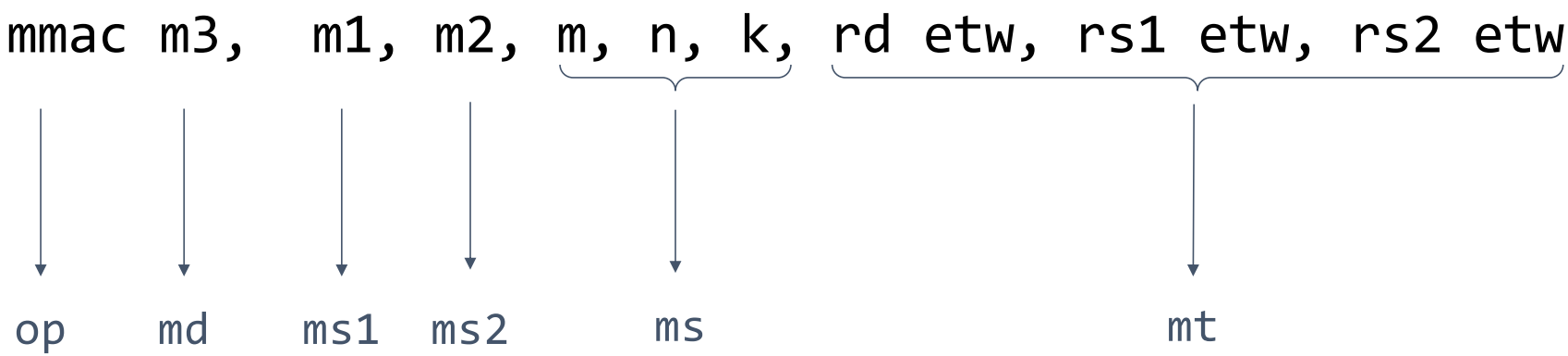
MMAC Operation



MMAC Operation

- Multiplies matrix registers m1 by m2 and accumulates the result in m3
- Required information
  destination matrix register
  source matrix register 1
  source matrix register 2
  dimension m
  dimension n
  dimension k
  element type/width (etw) for rd (mn)
  element type/width (etw) for rs1 (mk)
  element type/width (etw) for rs2 (kn)
- Prototype:
  mmac m3, m1, m2, m, n, k, rd etw, rs1 etw, rs2 etw

Proposed Encoding for mmac



- Issue: validation of the dimension correctness is a problem here.

Non-agnostic vs. Agnostic ISA

- Non-agnostic
  - Target architecture is exposed to the compiler
  - Compiler knows size of matrix registers
  - Compiler knows types of matrix elements
  - Compiler has to take care of matrix register allocation
  - Compiler can stripmine loops based on the number of matrix registers
  - Could eventually recycle LMUL for matrices (e.g. MMUL)
  - Need to re-compile code to leverage future architectures

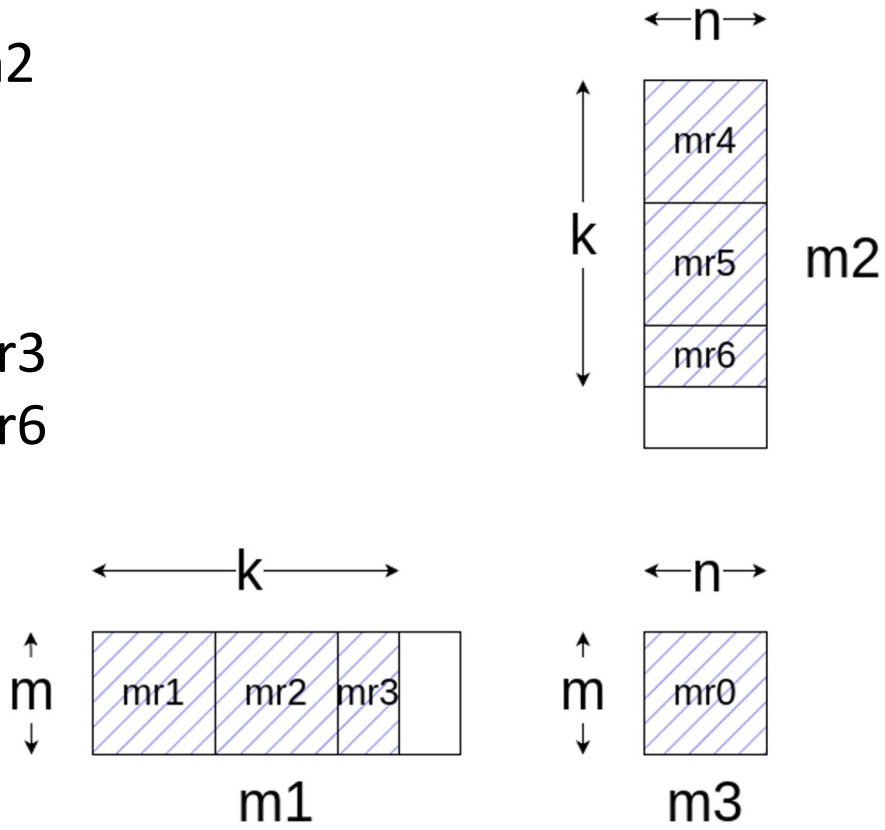
Non-agnostic vs. Agnostic ISA

- Agnostic
  - mt and ms MCSRs store all the information required for kernel
  - There are only three logical matrix registers available in the ISA (m[1-3])
  - There are 32 physical matrix registers available (mr[0-31])
  - Logical registers are assigned (at runtime) to physical registers on demand
  - This continues until all physical registers or Matrix Functional Units (MFUs) are used, when the dispatch of new operations holds
  - mr[0-31] are integral in AME or mapped to vector registers in IME
  - No need to re-compile code to leverage future architectures

Agnostic

Logical
m3 += m1 * m2

Physical
m3 = mr0
m1 = mr1 – mr3
m2 = mr4 – mr6



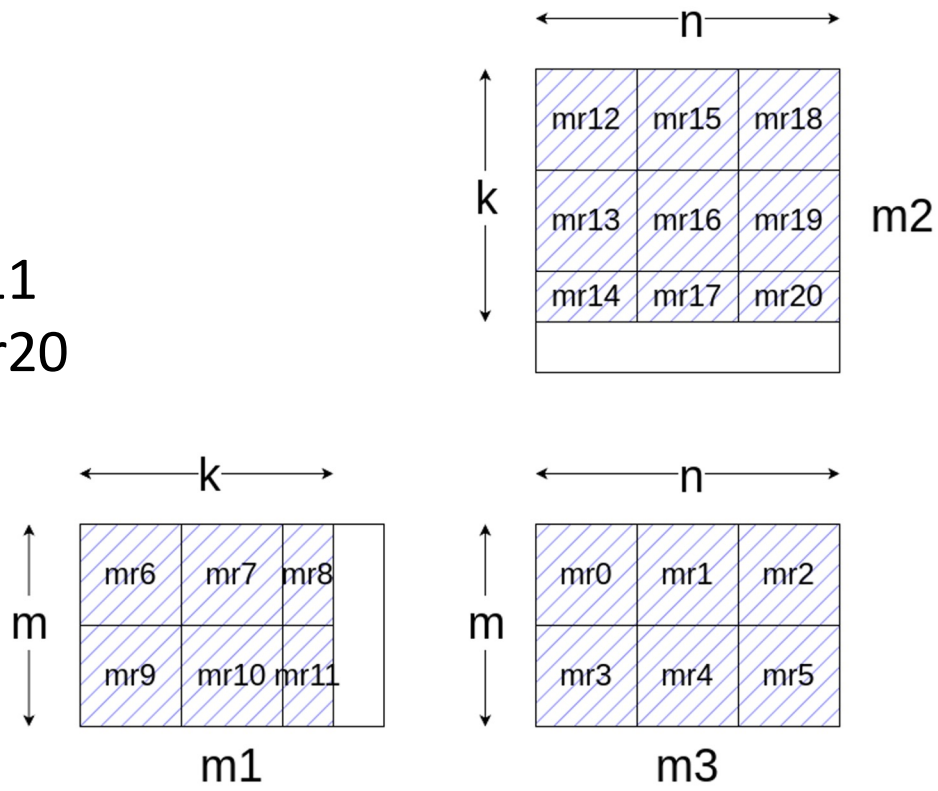

```
Virtual
m3 += m1 * m2

mr0 += mr1 * mr4
mr0 += mr2 * mr5
mr0 += mr3 * mr6
```

Agnostic

Logical
m3 += m1 * m2

Physical
m3 = m0 – mr5
m1 = mr6 – mr11
m2 = mr12 – mr20




```
mr0 += mr6 * mr12
mr0 += mr7 * mr13
mr0 += mr8 * mr14

mr1 += mr6 * mr15
mr1 += mr7 * mr16
mr1 += mr8 * mr17

mr2 += mr6 * mr18
mr2 += mr7 * mr19
mr2 += mr8 * mr20

mr3 += mr9 * mr12
mr3 += mr10 * mr13
mr3 += mr11 * mr14

mr4 += mr9 * mr15
mr4 += mr10 * mr16
mr4 += mr11 * mr17

mr5 += mr9 * mr18
mr5 += mr10 * mr19
mr5 += mr11 * mr20
```

Kernel information to perfom stripmined mmac

- Number of iterations for loops m, n and k
  - Instruction setms returns number of iterations
  - How about edge cases? Treat them outside the loop nesting
- Pointers to update ms1 (A) and ms2 (B) and store md (C)
  - Instruction setmt returns size of matrix register
  - Can use that to update pointers of A and B into memory
  - Need to take care of storing C back into memory (edge case)

Prelim kernel

```
loop_M = setmsm(M);
loop_N = setmsn(N);
loop_K = setmsk(K);

for(int _M = 0; _M<loop_M; _M++){
    for(int _N = 0; _N<loop_N; _N++){
        mld(m3, C_ptr);
        for(int _K = 0; _K<loop_K; _K++){
            mld(m1, A_ptr);
            mld(m2, B_ptr);
            mmac(m3, m1, m2);
            // ptr updates
        }
        mst(m3, C_ptr, ldc);
        // ptr updates
    }
    // ptr updates
}
```